## CONTENTS

1 User Guide  
1.1 Quick Start ................................................................. 3  
1.2 Introduction ................................................................. 4  
1.3 Building Models ............................................................. 5  
1.4 Fitting Models ............................................................... 5  
1.5 Fitting Result ................................................................. 7  
1.6 Visualizing Objects ......................................................... 8  
1.7 References ................................................................. 9  

2 The MenpoFit API  
2.1 Deformable Models ....................................................... 11  
2.2 Internal API ............................................................... 295  

Index  

461
Welcome to the MenpoFit documentation!

MenpoFit is a Python package for building, fitting and manipulating deformable models. It includes state-of-the-art deformable modelling techniques implemented on top of the Menpo project. Currently, the techniques that have been implemented include:

- **Active Appearance Model (AAM)**
  - Holistic, Patch-based, Masked, Linear, Linear Masked
  - Lucas-Kanade Optimisation
  - Cascaded-Regression Optimisation

- **Active Pictorial Structures (APS)**
  - Weighted Gauss-Newton Optimisation with fixed Jacobian and Hessian

- **Active Template Model (ATM)**
  - Holistic, Patch-based, Masked, Linear, Linear Masked
  - Lucas-Kanade Optimisation

- **Lucas-Kanade Image Alignment (LK)**
  - Forward Additive, Forward Compositional, Inverse Compositional
  - Residuals: SSD, Fourier SSD, ECC, Gradient Correlation, Gradient Images

- **Unified Active Appearance Model and Constrained Local Model (Unified AAM-CLM)**
  - Alternating/Project Out with Regularised Landmark Mean Shift

- **Constrained Local Model (CLM)**
  - Active Shape Model
  - Regularised Landmark Mean Shift

- **Ensemble of Regression Trees (ERT)** [provided by DLib]

- **Supervised Descent Method (SDM)**
  - Non Parametric
  - Parametric Shape
  - Parametric Appearance
  - Fully Parametric

Please see the to References for an indicative list of papers that are relevant to the methods implemented in MenpoFit.
The User Guide is designed to give you an overview of the key concepts within MenpoFit. In particular, we want to try and explain some of the design decisions that we made and demonstrate why we think they are powerful concepts for building, fitting and analysing deformable models.

1.1 Quick Start

Here we give a very quick rundown of the basic links and information sources for the project.

1.1.1 Basic Installation

In the Menpo Team, we strongly advocate the usage of conda for scientific Python, as it makes installation of compiled binaries much more simple. In particular, if you wish to use any of the related Menpo projects such as menpofit, menpo3d or menpodetect, you will not be able to easily do so without using conda. The installation of MenpoFit using conda is as easy as

```
$ conda install -c menpo menpofit
```

Conda is able to work out all the requirements/dependencies of MenpoFit. You may for example notice that menpo is one of them. Please see the thorough installation instructions for each platform on the Menpo website.

1.1.2 API Documentation

Visit API Documentation

MenpoFit is extensively documented on a per-method/class level and much of this documentation is reflected in the API Documentation. If any functions or classes are missing, please bring it to the attention of the developers on Github.

1.1.3 Notebooks

Explore the Menpo and MenpoFit Notebooks

For a more thorough set of examples, we provide a set of Jupyter notebooks that demonstrate common use cases of MenpoFit. The notebooks include extensive examples regarding all the state-of-the-art deformable models that we provide. You may need to have a look at the Menpo notebooks in order to get an overview of the basic functionalities required by MenpoFit.
1.1.4 User Group and Issues

If you wish to get in contact with the Menpo developers, you can do so via various channels. If you have found a bug, or if any part of MenpoFit behaves in a way you do not expect, please raise an issue on Github.

If you want to ask a theoretical question, or are having problems installing or setting up MenpoFit, please visit the user group.

1.2 Introduction

This user guide is a general introduction to MenpoFit, aiming to provide a bird’s eye of MenpoFit’s design. After reading this guide you should be able to go explore MenpoFit’s extensive Notebooks and not be too surprised by what you see.

1.2.1 What makes MenpoFit better?

The vast majority of existing deformable modeling software suffers from one or more of the following important issues:

- It is released in binary closed-source format
- It does not come with training code; only pre-trained models
- It is not well-structured which makes it very difficult to tweak and alter
- It only focuses on a single method/model

MenpoFit overcomes the above issues by providing open-source training and fitting code for multiple state-of-the-art deformable models under a unified protocol. We strongly believe that this is the only way towards reproducible and high-quality research.

1.2.2 Core Interfaces

MenpoFit is an object oriented framework for building and fitting deformable models. It makes some basic assumptions that are common for all the implemented methods. For example, all deformable models are trained in multiple scales and the fitting procedure is, in most cases, iterative. MenpoFit’s key interfaces are:

- `MultiScaleNonParametricFitter` - multi-scale fitting class
- `MultiScaleParametricFitter` - multi-scale fitting class that uses a parametric shape model
- `MultiScaleNonParametricIterativeResult` - multi-scale result of an iterative fitting
- `MultiScaleParametricIterativeResult` - multi-scale result of an iterative fitting using a parametric shape model

1.2.3 Deformable Models

- `AAM, LucasKanadeAAMFitter, SupervisedDescentAAMFitter` - Active Appearance Model builder and fitters
- `ATM, LucasKanadeATMFitter` - Active Template Model builder and fitter
- `GenerativeAPS, GaussNewtonAPSFitter` - Active Pictorial Structures builder and fitter
- `CLM, GradientDescentCLMFitter` - Constrained Local Model builder and fitter
1.3 Building Models

All MenpoFit’s models are built in a multi-scale manner, i.e. in multiple resolutions. In all our core classes, this is controlled using the following three parameters:

- **reference_shape** (*PointCloud*) First, the size of the training images is normalized by rescaling them so that the scale of their ground truth shapes matches the scale of this reference shape. In case no reference shape is provided, then the mean of the ground shapes is used. This step is essential in order to ensure consistency between the extracted features of the images.

- **diagonal** (*int*) This parameter is used to rescale the reference shape so that the diagonal of its bounding box matches the provided value. This rescaling takes place before normalizing the training images’ size. Thus, *diagonal* controls the size of the model at the highest scale.

- **scales** (*tuple of float*) A *tuple* with the scale value at each level, provided in ascending order, i.e. from lowest to highest scale. These values are proportional to the final resolution achieved through the reference shape normalization.

Additionally, all models have a **holistic_features** argument which expects the *callable* that will be used for extracting features from the training images.

Given the above assumptions, an example of a typical call for building a deformable model using *HolisticAAM* is:

```python
from menpofit.aam import HolisticAAM
from menpo.feature import fast_dsift

aam = HolisticAAM(training_images, group='PTS', reference_shape=None,
                   diagonal=200, scales=(0.25, 0.5, 1.0),
                   holistic_features=fast_dsift, verbose=True)
```

Information about any kind of model can be retrieved by:

```python
print(aam)
```

The next section (*Fitting*) explains the basics of fitting such a deformable model.

1.4 Fitting Models

1.4.1 Fitter Objects

MenpoFit has specialised classes for performing a fitting process that are called *Fitters*. All *Fitter* objects are subclasses of *MultiScaleNonParametricFitter* and *MultiScaleParametricFitter*. The main difference between those two is that a *MultiScaleParametricFitter* optimises over the parameters of a statistical shape model, whereas *MultiScaleNonParametricFitter* optimises directly the coordinates of a shape.

Their behaviour can differ depending on the deformable model. For example, a Lucas-Kanade AAM fitter (*LucasKanadeAAMFitter*) assumes that you have trained an AAM model (assume the *aam* we trained in the *Building* section) and can be created as:

```python
from menpofit.aam import LucasKanadeAAMFitter
from menpo.shape import ExtendedShape

lkaam = LucasKanadeAAMFitter(aam, shape=ExtendedShape())
```
from menpofit.aam import LucasKanadeAAMFitter, WibergInverseCompositional

fitter = LucasKanadeAAMFitter(aam,
    lk_algorithm_cls=WibergInverseCompositional,
    n_shape=[5, 10, 15], n_appearance=150)

The constructor of the `Fitter` will set the active shape and appearance components based on `n_shape` and `n_appearance` respectively, and will also perform all the necessary pre-computations based on the selected algorithm.

However, there are deformable models that are directly defined through a `Fitter` object, which is responsible for training the model as well. `SupervisedDescentFitter` is a good example. The reason for that is that the fitting process is utilised during the building procedure, thus the functionality of a `Fitter` is required. Such models can be built as:

from menpofit.sdm import SupervisedDescentFitter, NonParametricNewton

fitter = SupervisedDescentFitter(training_images, group='PTS',
    sd_algorithm_cls=NonParametricNewton,
    verbose=True)

Information about a `Fitter` can be retrieved by:

print(fitter)

### 1.4.2 Fitting Methods

All the deformable models that are currently implemented in MenpoFit, which are the state-of-the-art approaches in current literature, aim to find a local optimum of the cost function that they try to optimise, given an initialisation. The initialisation can be seen as an initial estimation of the target shape. MenpoFit’s `Fitter` objects provide two functions for fitting the model to an image:

result = fitter.fit_from_shape(image, initial_shape, max_iters=20, gt_shape=None,
    return_costs=False, **kwargs)

or

result = fitter.fit_from_bb(image, bounding_box, max_iters=20, gt_shape=None,
    return_costs=False, **kwargs)

They only differ on the type of initialisation. `fit_from_shape` expects a `PointCloud` as the `initial_shape`. On the other hand, the `bounding_box` argument of `fit_from_bb` is a `PointDirectedGraph` of 4 vertices that represents the initial bounding box. The bounding box is used in order to align the model’s reference shape and use the resulting `PointCloud` as the initial shape. Such a bounding box can be retrieved using the detection methods of `menpodetect`. The rest of the options are:

**max_iters** (*int or list of int*) Defines the maximum number of iterations. If *int*, then it specifies the maximum number of iterations over all scales. If *list of int*, then it specifies the maximum number of iterations per scale. Note that this does not apply on all deformable models. For example, it can control the number of iterations of a Lucas-Kanade optimisation algorithm, but it does not affect the fitting of a cascaded-regression method (e.g. SDM) which has a predefined number of cascades (iterations).

**gt_shape** (*PointCloud or None*) The ground truth shape associated to the image. This is only useful to compute the final fitting error. It is not used, of course, at any internal stage of the optimisation.

**return_costs** (*bool*) If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned `fitting_result`. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Thus, this
option should only be used for research purposes. Finally, this argument does not apply to all deformable models.

**kwargs (`dict`) Additional keyword arguments that can be passed to specific models.

The next section (*Result*) presents the basics of the fitting result.

## 1.5 Fitting Result

### 1.5.1 Objects

The fitting methods of the *Fitters* presented in the previous section return a result object. MenpoFit has three basic fitting result objects:

- **Result**: Basic fitting result object that holds the final shape, and optionally, the initial shape, ground truth shape and the image.
- **MultiScaleNonParametricIterativeResult**: The result of a multi-scale iterative fitting procedure. Apart from the final shape, it also stores the shapes acquired at each fitting iteration.
- **MultiScaleParametricIterativeResult**: The same as `MultiScaleNonParametricIterativeResult` with the difference that the optimisation was performed over the parameters of a statistical parametric shape model. Thus, apart from the actual shapes, it also stores the shape parameters acquired per iteration. *Note that in this case, the initial shape that was provided by the user gets reconstructed using the shape model, i.e. it first gets projected in order to get the initial estimation of the shape parameters, and then gets reconstructed with those*. The resulting shape is then used as initialisation for the iterative fitting process.

### 1.5.2 Attributes

The above result objects can provide some very useful information regarding the fitting procedure. For example, the various shapes can be retrieved as:

- `result.final_shape` The final shape of the fitting procedure.
- `result.initial_shape` The initial shape of the fitting procedure that was provided by the user.
- `result.reconstructed_initial_shape` The reconstruction of the initial shape that was used to initialise the fitting procedure. It only applies for `MultiScaleParametricIterativeResult`.
- `result.image` The image on which the fitting procedure was applied.
- `result.gt_shape` The ground truth shape associated to the image.
- `result.shapes` The list of shapes acquired at each fitting iteration. It only applies on `MultiScaleNonParametricIterativeResult` and `MultiScaleParametricIterativeResult`.
- `result.costs()` The cost values per iteration, if they were computed during fitting.

Also, a result can compute some error metrics, in case the `gt_shape` of the image exists:

- `result.final_error()` The final fitting error.
- `result.initial_error()` The initial fitting error.
- `result.errors()` The list of errors acquired at each fitting iteration. It only applies on `MultiScaleNonParametricIterativeResult` and `MultiScaleParametricIterativeResult`.

1.5. Fitting Result
1.6 Visualizing Objects

In Menpo, we take an opinionated stance that visualization is a key part of generating research on deformable models. Therefore, we tried to make the mental overhead of visualizing objects as low as possible.

We also took a strong step towards simple visualization by integrating some of our objects with visualization widgets for the Jupyter notebook. Remember that our widgets live on their own repository, called menpowidgets.

1.6.1 Visualizing Models

Without further ado, a quick example of visualising the AAM trained in the Building section with an interactive widget:

```python
%matplotlib inline  # This is only needed if viewing in a Jupyter notebook
aam.view_aam_widget()
```

**Fig. 1: Figure 1:** Example of visualizing an AAM using an interactive widget.

One can visualize the only the multi-scale shape models:

```python
%matplotlib inline
aam.view_shape_models_widget()
```

or the appearance models:

```python
%matplotlib inline
import menpo.io as mio
aam.view_appearance_models_widget()
```

The same visualization widgets can be found in other models, such as ATM, CLM etc.

1.6.2 Visualizing Fitting Result

The fitting result objects shown in Building can be easily visualized. Specifically, the initial and final shapes can be rendered as:

```python
%matplotlib inline
result.view(render_initial_shape=True)
```

Similarly, the shapes acquired at each iteration can be visualized as:

```python
%matplotlib inline
fr.view_iterations()
```

and the corresponding errors as:

```python
%matplotlib inline
fr.plot_errors()
```

Finally, a fitting result can also be analysed through an interactive widget as:

```python
%matplotlib inline
fr.view_widget()
```
Fig. 2: **Figure 2**: Example of visualizing the iterations of a fitting procedure using an interactive widget.

### 1.7 References

This is an indicative list of papers relevant to the methods that are implemented in MenpoFit. They are listed in alphabetical order of the first author’s surname.


THE MENPOFIT API

This section attempts to provide a simple browsing experience for the MenpoFit documentation. In MenpoFit, we use legible docstrings, and therefore, all documentation should be easily accessible in any sensible IDE (or IPython) via tab completion. However, this section should make most of the core classes available for viewing online.

2.1 Deformable Models

2.1.1 menpofit.aam

Active Appearance Model

AAM is a generative model that consists of a statistical parametric model of the shape and the appearance of an object. MenpoFit has several AAMs which differ in the manner that they compute the warp (thus represent the appearance features).

AAM

class menpofit.aam.base.AAM(images, group=None, holistic_features=<function no_op>, reference_shape=None, diagonal=None, scales=(0.5, 1.0), transform=<class 'menpofit.transform.piecewiseaffine.DifferentiablePiecewiseAffine'>, shape_model_cls=<class 'menpofit.modelinstance.OrthoPDM'>, max_shape_components=None, max_appearance_components=None, verbose=False, batch_size=None)

Bases: object

Class for training a multi-scale holistic Active Appearance Model. Please see the references for a basic list of relevant papers.

Parameters

- **images** *(list of menpo.image.Image)* – The list of training images.
- **group** *(str or None, optional)* – The landmark group that will be used to train the AAM. If None and the images only have a single landmark group, then that is the one that will be used. Note that all the training images need to have the specified landmark group.
- **holistic_features** *(closure or list of closure, optional)* – The features that will be extracted from the training images. Note that the features are extracted before warping the images to the reference shape. If list, then it must define a feature function per scale. Please refer to menpo.feature for a list of potential features.
• **reference_shape** *(menpo.shape.PointCloud or None, optional)* – The reference shape that will be used for building the AAM. The purpose of the reference shape is to normalise the size of the training images. The normalisation is performed by rescaling all the training images so that the scale of their ground truth shapes matches the scale of the reference shape. Note that the reference shape is rescaled with respect to the *diagonal* before performing the normalisation. If *None*, then the mean shape will be used.

• **diagonal** *(int or None, optional)* – This parameter is used to rescale the reference shape so that the diagonal of its bounding box matches the provided value. In other words, this parameter controls the size of the model at the highest scale. If *None*, then the reference shape does not get rescaled.

• **scales** *(float or tuple of float, optional)* – The scale value of each scale. They must provided in ascending order, i.e. from lowest to highest scale. If *float*, then a single scale is assumed.

• **transform** *(subclass of DL and DX, optional)* – A differential warp transform object, e.g. `DifferentiablePiecewiseAffine` or `DifferentiableThinPlateSplines`.

• **shape_model_cls** *(subclass of PDM, optional)* – The class to be used for building the shape model. The most common choice is `OrthoPDM`.

• **max_shape_components** *(int, float, list of those or None, optional)* – The number of shape components to keep. If *int*, then it sets the exact number of components. If *float*, then it defines the variance percentage that will be kept. If *list*, then it should define a value per scale. If a single number, then this will be applied to all scales. If *None*, then all the components are kept. Note that the unused components will be permanently trimmed.

• **max_appearance_components** *(int, float, list of those or None, optional)* – The number of appearance components to keep. If *int*, then it sets the exact number of components. If *float*, then it defines the variance percentage that will be kept. If *list*, then it should define a value per scale. If a single number, then this will be applied to all scales. If *None*, then all the components are kept. Note that the unused components will be permanently trimmed.

• **verbose** *(bool, optional)* – If True, then the progress of building the AAM will be printed.

• **batch_size** *(int or None, optional)* – If an *int* is provided, then the training is performed in an incremental fashion on image batches of size equal to the provided value. If *None*, then the training is performed directly on the all the images.

References

**appearance_reconstructions** *(appearance_parameters, n_iters_per_scale)*
Method that generates the appearance reconstructions given a set of appearance parameters. This is to be combined with a `AAMResult` object, in order to generate the appearance reconstructions of a fitting procedure.

**Parameters**

• **appearance_parameters** *(list of (n_params,) ndarray)* – A set of appearance parameters per fitting iteration. It can be retrieved as a property of an `AAMResult` object.

• **n_iters_per_scale** *(list of int)* – The number of iterations per scale. This is necessary in order to figure out which appearance parameters correspond to the model of each scale. It can be retrieved as a property of a `AAMResult` object.

**Returns** **appearance_reconstructions** *(list of menpo.image.Image)* – List of the appearance reconstructions that correspond to the provided parameters.
**build_fitter_interfaces** (*sampling*)
Method that builds the correct Lucas-Kanade fitting interface. It only applies in case you wish to fit the AAM with a Lucas-Kanade algorithm (i.e. *LucasKanadeAAMFitter*).

**Parameters**
- **sampling** *(list of int or ndarray or None)* – It defines a sampling mask per scale. If *int*, then it defines the sub-sampling step of the sampling mask. If *ndarray*, then it explicitly defines the sampling mask. If *None*, then no sub-sampling is applied.

**Returns**
- **fitter_interfaces** *(list)* – The list of Lucas-Kanade interface per scale.

**increment** *(images, group=None, shape_forgetting_factor=1.0, appearance_forgetting_factor=1.0, verbose=False, batch_size=None)*
Method to increment the trained AAM with a new set of training images.

**Parameters**
- **images** *(list of menpo.image.Image)* – The list of training images.
- **group** *(str or None, optional)* – The landmark group that will be used to train the AAM. If *None* and the images only have a single landmark group, then that is the one that will be used. Note that all the training images need to have the specified landmark group.
- **shape_forgetting_factor** *(float, optional)* – Forgetting factor that weights the relative contribution of new samples vs old samples for the shape model. If *1.0*, all samples are weighted equally and, hence, the result is the exact same as performing batch PCA on the concatenated list of old and new samples. If *<1.0*, more emphasis is put on the new samples.
- **appearance_forgetting_factor** *(float, optional)* – Forgetting factor that weights the relative contribution of new samples vs old samples for the appearance model. If *1.0*, all samples are weighted equally and, hence, the result is the exact same as performing batch PCA on the concatenated list of old and new samples. If *<1.0*, more emphasis is put on the new samples.
- **verbose** *(bool, optional)* – If *True*, then the progress of building the AAM will be printed.
- **batch_size** *(int or None, optional)* – If an *int* is provided, then the training is performed in an incremental fashion on image batches of size equal to the provided value. If *None*, then the training is performed directly on all the images.

**instance** *(shape_weights=None, appearance_weights=None, scale_index=-1)*
Generates a novel AAM instance given a set of shape and appearance weights. If no weights are provided, then the mean AAM instance is returned.

**Parameters**
- **shape_weights** *(ndarray or list or None, optional)* – The weights of the shape model that will be used to create a novel shape instance. If *None*, the weights are assumed to be zero, thus the mean shape is used.
- **appearance_weights** *(ndarray or list or None, optional)* – The weights of the appearance model that will be used to create a novel appearance instance. If *None*, the weights are assumed to be zero, thus the mean appearance is used.
- **scale_index** *(int, optional)* – The scale to be used.

**Returns**
- **image** *(menpo.image.Image)* – The AAM instance.

**random_instance** *(scale_index=-1)*
Generates a random instance of the AAM.

**Parameters**
- **scale_index** *(int, optional)* – The scale to be used.
**view_aam_widget** *(n_shape_parameters=5, n_appearance_parameters=5, parameters_bounds=(-3.0, 3.0), mode='multiple', figure_size=(7, 7))*

Visualizes the AAM using an interactive widget.

**Parameters**

- **n_shape_parameters** *(int or list of int or None, optional)* – The number of shape principal components to be used for the parameters sliders. If *int*, then the number of sliders per scale is the minimum between *n_parameters* and the number of active components per scale. If *list* of *int*, then a number of sliders is defined per scale. If *None*, all the active components per scale will have a slider.

- **n_appearance_parameters** *(int or list of int or None, optional)* – The number of appearance principal components to be used for the parameters sliders. If *int*, then the number of sliders per scale is the minimum between *n_parameters* and the number of active components per scale. If *list* of *int*, then a number of sliders is defined per scale. If *None*, all the active components per scale will have a slider.

- **parameters_bounds** *((float, float), optional)* – The minimum and maximum bounds, in std units, for the sliders.

- **mode** *((single, multiple), optional)* – If *'single'* , only a single slider is constructed along with a drop down menu. If *'multiple'* , a slider is constructed for each parameter.

- **figure_size** *((int, int), optional)* – The size of the rendered figure.

**view_appearance_models_widget** *(n_parameters=5, parameters_bounds=(-3.0, 3.0), mode='multiple', figure_size=(7, 7))*

Visualizes the appearance models of the AAM object using an interactive widget.

**Parameters**

- **n_parameters** *(int or list of int or None, optional)* – The number of appearance principal components to be used for the parameters sliders. If *int*, then the number of sliders per scale is the minimum between *n_parameters* and the number of active components per scale. If *list* of *int*, then a number of sliders is defined per scale. If *None*, all the active components per scale will have a slider.

- **parameters_bounds** *((float, float), optional)* – The minimum and maximum bounds, in std units, for the sliders.

- **mode** *((single, multiple), optional)* – If *'single'* , only a single slider is constructed along with a drop down menu. If *'multiple'* , a slider is constructed for each parameter.

- **figure_size** *((int, int), optional)* – The size of the rendered figure.

**view_shape_models_widget** *(n_parameters=5, parameters_bounds=(-3.0, 3.0), mode='multiple', figure_size=(7, 7))*

Visualizes the shape models of the AAM object using an interactive widget.

**Parameters**

- **n_parameters** *(int or list of int or None, optional)* – The number of shape principal components to be used for the parameters sliders. If *int*, then the number of sliders per scale is the minimum between *n_parameters* and the number of active components per scale. If *list* of *int*, then a number of sliders is defined per scale. If *None*, all the active components per scale will have a slider.
• **parameters_bounds** ((float, float), optional) – The minimum and maximum bounds, in std units, for the sliders.

• **mode** ({'single', 'multiple'}, optional) – If 'single', only a single slider is constructed along with a drop down menu. If 'multiple', a slider is constructed for each parameter.

• **figure_size** ((int, int), optional) – The size of the rendered figure.

**property n_scales**

Returns the number of scales.

Type int

### HolisticAAM

```python
menpofit.aam.HolisticAAM
```

alias of AAM

### MaskedAAM

```python
class menpofit.aam.MaskedAAM(images, group=None, holistic_features=<function no_op>, reference_shape=None, diagonal=None, scales=(0.5, 1.0), patch_shape=(17, 17), shape_model_cls=<class 'menpo.modelinstance.OrthoPDM'>, max_shape_components=None, max_appearance_components=None, verbose=False, batch_size=None)
```

Bases: AAM

Class for training a multi-scale patch-based Masked Active Appearance Model. The appearance of this model is formulated by simply masking an image with a patch-based mask.

**Parameters**

- **images** ([menpo.image.Image](#)) – The list of training images.

- **group** ([str](https://www.python.org/doc/2.7/library/stdtypes.html#str) or None, optional) – The landmark group that will be used to train the AAM. If None and the images only have a single landmark group, then that is the one that will be used. Note that all the training images need to have the specified landmark group.

- **holistic_features** ([closure](https://www.python.org/doc/2.7/library/functional.html#types-of-function-objects) or list of closure, optional) – The features that will be extracted from the training images. Note that the features are extracted before warping the images to the reference shape. If list, then it must define a feature function per scale. Please refer to [menpo.feature](#) for a list of potential features.

- **reference_shape** ([menpo.shape.PointCloud](#) or None, optional) – The reference shape that will be used for building the AAM. The purpose of the reference shape is to normalise the size of the training images. The normalization is performed by rescaling all the training images so that the scale of their ground truth shapes matches the scale of the reference shape. Note that the reference shape is rescaled with respect to the **diagonal** before performing the normalisation. If None, then the mean shape will be used.

- **diagonal** ([int](https://www.python.org/doc/2.7/library/stdtypes.html#int) or None, optional) – This parameter is used to rescale the reference shape so that the diagonal of its bounding box matches the provided value. In other words, this parameter controls the size of the model at the highest scale. If None, then the reference shape does not get rescaled.
• **scales** *(float or tuple of float, optional)* – The scale value of each scale. They must provided in ascending order, i.e. from lowest to highest scale. If float, then a single scale is assumed.

• **patch_shape** *((int, int), optional)* – The size of the patches of the mask that is used to sample the appearance vectors.

• **shape_model_cls** *(subclass of PDM, optional)* – The class to be used for building the shape model. The most common choice is OrthoPDM.

• **max_shape_components** *(int, float, list of those or None, optional)* – The number of shape components to keep. If int, then it sets the exact number of components. If float, then it defines the variance percentage that will be kept. If list, then it should define a value per scale. If a single number, then this will be applied to all scales. If None, then all the components are kept. Note that the unused components will be permanently trimmed.

• **max_appearance_components** *(int, float, list of those or None, optional)* – The number of appearance components to keep. If int, then it sets the exact number of components. If float, then it defines the variance percentage that will be kept. If list, then it should define a value per scale. If a single number, then this will be applied to all scales. If None, then all the components are kept. Note that the unused components will be permanently trimmed.

• **verbose** *(bool, optional)* – If True, then the progress of building the AAM will be printed.

• **batch_size** *(int or None, optional)* – If an int is provided, then the training is performed in an incremental fashion on image batches of size equal to the provided value. If None, then the training is performed directly on the all the images.

---

**appearance_reconstructions**(appearance_parameters, n_iters_per_scale)

Method that generates the appearance reconstructions given a set of appearance parameters. This is to be combined with a AAMResult object, in order to generate the appearance reconstructions of a fitting procedure.

**Parameters**

• **appearance_parameters** *(list of (n_params,) ndarray)* – A set of appearance parameters per fitting iteration. It can be retrieved as a property of an AAMResult object.

• **n_iters_per_scale** *(list of int)* – The number of iterations per scale. This is necessary in order to figure out which appearance parameters correspond to the model of each scale. It can be retrieved as a property of a AAMResult object.

**Returns** appearance_reconstructions *(list of menpo.image.Image)* – List of the appearance reconstructions that correspond to the provided parameters.

---

**build_fitter_interfaces**(sampling)

Method that builds the correct Lucas-Kanade fitting interface. It only applies in case you wish to fit the AAM with a Lucas-Kanade algorithm (i.e. LucasKanadeAAMFitter).

**Parameters** sampling *(list of int or ndarray or None)* – It defines a sampling mask per scale.

If int, then it defines the sub-sampling step of the sampling mask. If ndarray, then it explicitly defines the sampling mask. If None, then no sub-sampling is applied.

**Returns** fitter_interfaces *(list)* – The list of Lucas-Kanade interface per scale.

---

**increment**(images, group=None, shape_forgetting_factor=1.0, appearance_forgetting_factor=1.0, verbose=False, batch_size=None)

Method to increment the trained AAM with a new set of training images.

**Parameters**

• **images** *(list of menpo.image.Image)* – The list of training images.
• **group** *(str or None, optional)* – The landmark group that will be used to train the AAM. If `None` and the images only have a single landmark group, then that is the one that will be used. Note that all the training images need to have the specified landmark group.

• **shape_forgetting_factor** *(0.0, 1.0] float, optional)* – Forgetting factor that weights the relative contribution of new samples vs old samples for the shape model. If 1.0, all samples are weighted equally and, hence, the result is the exact same as performing batch PCA on the concatenated list of old and new simples. If <1.0, more emphasis is put on the new samples.

• **appearance_forgetting_factor** *(0.0, 1.0] float, optional)* – Forgetting factor that weights the relative contribution of new samples vs old samples for the appearance model. If 1.0, all samples are weighted equally and, hence, the result is the exact same as performing batch PCA on the concatenated list of old and new simples. If <1.0, more emphasis is put on the new samples.

• **verbose** *(bool, optional)* – If `True`, then the progress of building the AAM will be printed.

• **batch_size** *(int or None, optional)* – If an `int` is provided, then the training is performed in an incremental fashion on image batches of size equal to the provided value. If `None`, then the training is performed directly on all the images.

**instance** *(shape_weights=None, appearance_weights=None, scale_index=-1)*

Generates a novel AAM instance given a set of shape and appearance weights. If no weights are provided, then the mean AAM instance is returned.

**Parameters**

• **shape_weights** *(n_weights,) ndarray or list or None, optional)* – The weights of the shape model that will be used to create a novel shape instance. If `None`, the weights are assumed to be zero, thus the mean shape is used.

• **appearance_weights** *(n_weights,) ndarray or list or None, optional)* – The weights of the appearance model that will be used to create a novel appearance instance. If `None`, the weights are assumed to be zero, thus the mean appearance is used.

• **scale_index** *(int, optional)* – The scale to be used.

**Returns** `image` *(menpo.image.Image)* – The AAM instance.

**random_instance** *(scale_index=-1)*

Generates a random instance of the AAM.

**Parameters**

• **scale_index** *(int, optional)* – The scale to be used.

**Returns** `image` *(menpo.image.Image)* – The AAM instance.

**view_aam_widget** *(n_shape_parameters=5, n_appearance_parameters=5, parameters_bounds=(-3.0, 3.0), mode='multiple', figure_size=(7, 7))*

Visualizes the AAM using an interactive widget.

**Parameters**

• **n_shape_parameters** *(int or list of int or None, optional)* – The number of shape principal components to be used for the parameters sliders. If `int`, then the number of sliders per scale is the minimum between `n_parameters` and the number of active components per scale. If `list` of `int`, then a number of sliders is defined per scale. If `None`, all the active components per scale will have a slider.

• **n_appearance_parameters** *(int or list of int or None, optional)* – The number of appearance principal components to be used for the parameters sliders. If `int`, then the number of sliders per scale is the minimum between `n_parameters` and the number of
active components per scale. If list of int, then a number of sliders is defined per scale. If None, all the active components per scale will have a slider.

- **parameters_bounds** ((float, float), optional) – The minimum and maximum bounds, in std units, for the sliders.

- **mode** ([single, multiple], optional) – If 'single', only a single slider is constructed along with a drop down menu. If 'multiple', a slider is constructed for each parameter.

- **figure_size** ((int, int), optional) – The size of the rendered figure.

```python
view_appearance_models_widget(n_parameters=5, parameters_bounds=(-3.0, 3.0), mode='multiple', figure_size=(7, 7))
```

Visualizes the appearance models of the AAM object using an interactive widget.

**Parameters**

- **n_parameters** (int or list of int or None, optional) – The number of appearance principal components to be used for the parameters sliders. If int, then the number of sliders per scale is the minimum between n_parameters and the number of active components per scale. If list of int, then a number of sliders is defined per scale. If None, all the active components per scale will have a slider.

- **parameters_bounds** ((float, float), optional) – The minimum and maximum bounds, in std units, for the sliders.

- **mode** ([single, multiple], optional) – If 'single', only a single slider is constructed along with a drop down menu. If 'multiple', a slider is constructed for each parameter.

- **figure_size** ((int, int), optional) – The size of the rendered figure.

```python
view_shape_models_widget(n_parameters=5, parameters_bounds=(-3.0, 3.0), mode='multiple', figure_size=(7, 7))
```

Visualizes the shape models of the AAM object using an interactive widget.

**Parameters**

- **n_parameters** (int or list of int or None, optional) – The number of shape principal components to be used for the parameters sliders. If int, then the number of sliders per scale is the minimum between n_parameters and the number of active components per scale. If list of int, then a number of sliders is defined per scale. If None, all the active components per scale will have a slider.

- **parameters_bounds** ((float, float), optional) – The minimum and maximum bounds, in std units, for the sliders.

- **mode** ([single, multiple], optional) – If 'single', only a single slider is constructed along with a drop down menu. If 'multiple', a slider is constructed for each parameter.

- **figure_size** ((int, int), optional) – The size of the rendered figure.

**property n_scales**

Returns the number of scales.

**Type** int
LinearAAM

class menpofit.aam.LinearAAM(images, group=None, holistic_features=<function no_op>, reference_shape=None, diagonal=None, scales=(0.5, 1.0), transform=<class 'menpofit.transform.thinsplatesplines.DifferentiableThinPlateSplines'>, shape_model_cls=<class 'menpofit.modelinstance.OrthoPDM'>, max_shape_components=None, max_appearance_components=None, verbose=False, batch_size=None)

Bases: AAM

Class for training a multi-scale Linear Active Appearance Model.

Parameters

- **images** (list of menpo.image.Image) – The list of training images.
- **group** (str or None, optional) – The landmark group that will be used to train the AAM. If None and the images only have a single landmark group, then that is the one that will be used. Note that all the training images need to have the specified landmark group.
- **holistic_features** (closure or list of closure, optional) – The features that will be extracted from the training images. Note that the features are extracted before warping the images to the reference shape. If list, then it must define a feature function per scale. Please refer to menpo.feature for a list of potential features.
- **reference_shape** (menpo.shape.PointCloud or None, optional) – The reference shape that will be used for building the AAM. The purpose of the reference shape is to normalise the size of the training images. The normalization is performed by rescaling all the training images so that the scale of their ground truth shapes matches the scale of the reference shape. Note that the reference shape is rescaled with respect to the diagonal before performing the normalisation. If None, then the mean shape will be used.
- **diagonal** (int or None, optional) – This parameter is used to rescale the reference shape so that the diagonal of its bounding box matches the provided value. In other words, this parameter controls the size of the model at the highest scale. If None, then the reference shape does not get rescaled.
- **scales** (float or tuple of float, optional) – The scale value of each scale. They must provided in ascending order, i.e. from lowest to highest scale. If float, then a single scale is assumed.
- **transform** (subclass of DL and DX, optional) – A differential warp transform object, e.g. DifferentiablePiecewiseAffine or DifferentiableThinPlateSplines.
- **shape_model_cls** (subclass of PDM, optional) – The class to be used for building the shape model. The most common choice is OrthoPDM.
- **max_shape_components** (int, float, list of those or None, optional) – The number of shape components to keep. If int, then it sets the exact number of components. If float, then it defines the variance percentage that will be kept. If list, then it should define a value per scale. If a single number, then this will be applied to all scales. If None, then all the components are kept. Note that the unused components will be permanently trimmed.
- **max_appearance_components** (int, float, list of those or None, optional) – The number of appearance components to keep. If int, then it sets the exact number of components. If float, then it defines the variance percentage that will be kept. If list, then it should define a value per scale. If a single number, then this will be applied to all scales. If None, then all the components are kept. Note that the unused components will be permanently trimmed.
• **verbose** *(bool, optional)* – If `True`, then the progress of building the AAM will be printed.

• **batch_size** *(int or None, optional)* – If an `int` is provided, then the training is performed in an incremental fashion on image batches of size equal to the provided value. If `None`, then the training is performed directly on all the images.

**appearance_reconstructions** *(appearance_parameters, n_iters_per_scale)*

Method that generates the appearance reconstructions given a set of appearance parameters. This is to be combined with a `AAMResult` object, in order to generate the appearance reconstructions of a fitting procedure.

**Parameters**

• **appearance_parameters** *(list of (n_params,) ndarray)* – A set of appearance parameters per fitting iteration. It can be retrieved as a property of an `AAMResult` object.

• **n_iters_per_scale** *(list of int)* – The number of iterations per scale. This is necessary in order to figure out which appearance parameters correspond to the model of each scale. It can be retrieved as a property of a `AAMResult` object.

**Returns** **appearance_reconstructions** *(list of menpo.image.Image)* – List of the appearance reconstructions that correspond to the provided parameters.

**build_fitter_interfaces** *(sampling)*

Method that builds the correct Lucas-Kanade fitting interface. It only applies in case you wish to fit the AAM with a Lucas-Kanade algorithm (i.e. `LucasKanadeAAMFitter`).

**Parameters** **sampling** *(list of int or ndarray or None)* – It defines a sampling mask per scale. If `int`, then it defines the sub-sampling step of the sampling mask. If `ndarray`, then it explicitly defines the sampling mask. If `None`, then no sub-sampling is applied.

**Returns** **fitter_interfaces** *(list)* – The list of Lucas-Kanade interface per scale.

**increment** *(images, group=None, shape_forgetting_factor=1.0, appearance_forgetting_factor=1.0, verbose=False, batch_size=None)*

Method to increment the trained AAM with a new set of training images.

**Parameters**

• **images** *(list of menpo.image.Image)* – The list of training images.

• **group** *(str or None, optional)* – The landmark group that will be used to train the AAM. If `None` and the images only have a single landmark group, then that is the one that will be used. Note that all the training images need to have the specified landmark group.

• **shape_forgetting_factor** *(0.0, 1.0] float, optional)* – Forgetting factor that weights the relative contribution of new samples vs old samples for the shape model. If `1.0`, all samples are weighted equally and, hence, the result is the exact same as performing batch PCA on the concatenated list of old and new simples. If `<1.0`, more emphasis is put on the new samples.

• **appearance_forgetting_factor** *(0.0, 1.0] float, optional)* – Forgetting factor that weights the relative contribution of new samples vs old samples for the appearance model. If `1.0`, all samples are weighted equally and, hence, the result is the exact same as performing batch PCA on the concatenated list of old and new simples. If `<1.0`, more emphasis is put on the new samples.

• **verbose** *(bool, optional)* – If `True`, then the progress of building the AAM will be printed.

• **batch_size** *(int or None, optional)* – If an `int` is provided, then the training is performed in an incremental fashion on image batches of size equal to the provided value. If `None`, then the training is performed directly on all the images.
instance (shape_weights=None, appearance_weights=None, scale_index=-1)
Generates a novel AAM instance given a set of shape and appearance weights. If no weights are provided, then the mean AAM instance is returned.

Parameters

- **shape_weights** ((n_weights,) ndarray or list or None, optional) – The weights of the shape model that will be used to create a novel shape instance. If None, the weights are assumed to be zero, thus the mean shape is used.

- **appearance_weights** ((n_weights,) ndarray or list or None, optional) – The weights of the appearance model that will be used to create a novel appearance instance. If None, the weights are assumed to be zero, thus the mean appearance is used.

- **scale_index** (int, optional) – The scale to be used.

Returns **image** (menpo.image.Image) – The AAM instance.

random_instance (scale_index=-1)
Generates a random instance of the AAM.

Parameters **scale_index** (int, optional) – The scale to be used.

Returns **image** (menpo.image.Image) – The AAM instance.

view_aam_widget (n_shape_parameters=5, n_appearance_parameters=5, parameters_bounds=(-3.0, 3.0), mode='multiple', figure_size=(7, 7))
Visualizes the AAM using an interactive widget.

Parameters

- **n_shape_parameters** (int or list of int or None, optional) – The number of shape principal components to be used for the parameters sliders. If int, then the number of sliders per scale is the minimum between n_parameters and the number of active components per scale. If list of int, then a number of sliders is defined per scale. If None, all the active components per scale will have a slider.

- **n_appearance_parameters** (int or list of int or None, optional) – The number of appearance principal components to be used for the parameters sliders. If int, then the number of sliders per scale is the minimum between n_parameters and the number of active components per scale. If list of int, then a number of sliders is defined per scale. If None, all the active components per scale will have a slider.

- **parameters_bounds** ((float, float), optional) – The minimum and maximum bounds, in std units, for the sliders.

- **mode** ([single, multiple], optional) – If 'single', only a single slider is constructed along with a drop down menu. If 'multiple', a slider is constructed for each parameter.

- **figure_size** ((int, int), optional) – The size of the rendered figure.

view_appearance_models_widget (n_parameters=5, parameters_bounds=(-3.0, 3.0), mode='multiple', figure_size=(7, 7))
Visualizes the appearance models of the AAM object using an interactive widget.

Parameters

- **n_parameters** (int or list of int or None, optional) – The number of appearance principal components to be used for the parameters sliders. If int, then the number of sliders per scale is the minimum between n_parameters and the number of active components per scale. If list of int, then a number of sliders is defined per scale. If None, all the active components per scale will have a slider.
• **parameters_bounds** ((float, float), optional) – The minimum and maximum bounds, in std units, for the sliders.

• **mode** ([single, multiple], optional) – If 'single', only a single slider is constructed along with a drop down menu. If 'multiple', a slider is constructed for each parameter.

• **figure_size** ((int, int), optional) – The size of the rendered figure.

**view_shape_models_widget** (n_parameters=5, parameters_bounds=(-3.0, 3.0), mode='multiple', figure_size=(7, 7))

Visualizes the shape models of the AAM object using an interactive widget.

**Parameters**

• **n_parameters** (int or list of int or None, optional) – The number of shape principal components to be used for the parameters sliders. If int, then the number of sliders per scale is the minimum between n_parameters and the number of active components per scale. If list of int, then a number of sliders is defined per scale. If None, all the active components per scale will have a slider.

• **parameters_bounds** ((float, float), optional) – The minimum and maximum bounds, in std units, for the sliders.

• **mode** ([single, multiple], optional) – If 'single', only a single slider is constructed along with a drop down menu. If 'multiple', a slider is constructed for each parameter.

• **figure_size** ((int, int), optional) – The size of the rendered figure.

**property n_scales**

Returns the number of scales.

Type  int

**LinearMaskedAAM**

class menpofit.aam.LinearMaskedAAM(images, group=None, holistic_features=<function no_op>, reference_shape=None, diagonal=None, scales=(0.5, 1.0), patch_shape=(17, 17), shape_model_cls=<class 'menpo.fit.modelinstance.OrthoPDM'>, max_shape_components=None, max_appearance_components=None, verbose=False, batch_size=None)

Bases: AAM

Class for training a multi-scale Linear Masked Active Appearance Model.

**Parameters**

• **images** (list of menpo.image.Image) – The list of training images.

• **group** (str or None, optional) – The landmark group that will be used to train the AAM. If None and the images only have a single landmark group, then that is the one that will be used. Note that all the training images need to have the specified landmark group.

• **holistic_features** (closure or list of closure, optional) – The features that will be extracted from the training images. Note that the features are extracted before warping the images to the reference shape. If list, then it must define a feature function per scale. Please refer to menpo.feature for a list of potential features.
**MenpoFit Documentation, Release 0.6.1**

- **reference_shape** *(menpo.shape.PointCloud or None, optional)* – The reference shape that will be used for building the AAM. The purpose of the reference shape is to normalise the size of the training images. The normalization is performed by rescaling all the training images so that the scale of their ground truth shapes matches the scale of the reference shape. Note that the reference shape is rescaled with respect to the *diagonal* before performing the normalisation. If None, then the mean shape will be used.

- **diagonal** *(int or None, optional)* – This parameter is used to rescale the reference shape so that the diagonal of its bounding box matches the provided value. In other words, this parameter controls the size of the model at the highest scale. If None, then the reference shape does not get rescaled.

- **scales** *(float or tuple of float, optional)* – The scale value of each scale. They must provided in ascending order, i.e. from lowest to highest scale. If float, then a single scale is assumed.

- **patch_shape** *(int, int), optional* – The size of the patches of the mask that is used to sample the appearance vectors.

- **shape_model_cls** *(subclass of PDM, optional)* – The class to be used for building the shape model. The most common choice is OrthoPDM.

- **max_shape_components** *(int, float, list of those or None, optional)* – The number of shape components to keep. If int, then it sets the exact number of components. If float, then it defines the variance percentage that will be kept. If list, then it should define a value per scale. If a single number, then this will be applied to all scales. If None, then all the components are kept. Note that the unused components will be permanently trimmed.

- **max_appearance_components** *(int, float, list of those or None, optional)* – The number of appearance components to keep. If int, then it sets the exact number of components. If float, then it defines the variance percentage that will be kept. If list, then it should define a value per scale. If a single number, then this will be applied to all scales. If None, then all the components are kept. Note that the unused components will be permanently trimmed.

- **verbose** *(bool, optional)* – If True, then the progress of building the AAM will be printed.

- **batch_size** *(int or None, optional)* – If an int is provided, then the training is performed in an incremental fashion on image batches of size equal to the provided value. If None, then the training is performed directly on the all the images.

**appearance_reconstructions** *(appearance_parameters, n_iters_per_scale)*

Method that generates the appearance reconstructions given a set of appearance parameters. This is to be combined with a AAMResult object, in order to generate the appearance reconstructions of a fitting procedure.

**Parameters**

- **appearance_parameters** *(list of (n_params,) ndarray)* – A set of appearance parameters per fitting iteration. It can be retrieved as a property of an AAMResult object.

- **n_iters_per_scale** *(list of int)* – The number of iterations per scale. This is necessary in order to figure out which appearance parameters correspond to the model of each scale. It can be retrieved as a property of a AAMResult object.

**Returns** appearance_reconstructions *(list of menpo.image.Image)* – List of the appearance reconstructions that correspond to the provided parameters.

**build_fitter_interfaces** *(sampling)*

Method that builds the correct Lucas-Kanade fitting interface. It only applies in case you wish to fit the AAM with a Lucas-Kanade algorithm (i.e. LucasKanadeAAMFitter).

### 2.1. Deformable Models
Parameters `sampling` *(list of int or ndarray or None)* – It defines a sampling mask per scale. If `int`, then it defines the sub-sampling step of the sampling mask. If `ndarray`, then it explicitly defines the sampling mask. If `None`, then no sub-sampling is applied.

Returns `fitter_interfaces` *(list)* – The list of Lucas-Kanade interface per scale.

`increment` *(images, group=None, shape_forgetting_factor=1.0, appearance_forgetting_factor=1.0, verbose=False, batch_size=None)*

Method to increment the trained AAM with a new set of training images.

Parameters

- `images` *(list of menpo.image.Image)* – The list of training images.
- `group` *(str or None, optional)* – The landmark group that will be used to train the AAM. If `None` and the images only have a single landmark group, then that is the one that will be used. Note that all the training images need to have the specified landmark group.
- `shape_forgetting_factor` *(float, optional)* – Forgetting factor that weights the relative contribution of new samples vs old samples for the shape model. If 1.0, all samples are weighted equally and, hence, the result is the exact same as performing batch PCA on the concatenated list of old and new simples. If <1.0, more emphasis is put on the new samples.
- `appearance_forgetting_factor` *(float, optional)* – Forgetting factor that weights the relative contribution of new samples vs old samples for the appearance model. If 1.0, all samples are weighted equally and, hence, the result is the exact same as performing batch PCA on the concatenated list of old and new simples. If <1.0, more emphasis is put on the new samples.
- `verbose` *(bool, optional)* – If `True`, then the progress of building the AAM will be printed.
- `batch_size` *(int or None, optional)* – If an `int` is provided, then the training is performed in an incremental fashion on image batches of size equal to the provided value. If `None`, then the training is performed directly on the all the images.

`instance` *(shape_weights=None, appearance_weights=None, scale_index=-1)*

Generates a novel AAM instance given a set of shape and appearance weights. If no weights are provided, then the mean AAM instance is returned.

Parameters

- `shape_weights` *(ndarray or list or None, optional)* – The weights of the shape model that will be used to create a novel shape instance. If `None`, the weights are assumed to be zero, thus the mean shape is used.
- `appearance_weights` *(ndarray or list or None, optional)* – The weights of the appearance model that will be used to create a novel appearance instance. If `None`, the weights are assumed to be zero, thus the mean appearance is used.
- `scale_index` *(int, optional)* – The scale to be used.


`random_instance` *(scale_index=-1)*

Generates a random instance of the AAM.

Parameters `scale_index` *(int, optional)* – The scale to be used.

view_aam_widget (n_shape_parameters=5, n_appearance_parameters=5, parameters_bounds=(-3.0, 3.0), mode='multiple', figure_size=(7, 7))
Visualizes the AAM using an interactive widget.

Parameters

• **n_shape_parameters** *(int or list of int or None, optional)* – The number of shape principal components to be used for the parameters sliders. If *int*, then the number of sliders per scale is the minimum between *n_parameters* and the number of active components per scale. If *list* of *int*, then a number of sliders is defined per scale. If *None*, all the active components per scale will have a slider.

• **n_appearance_parameters** *(int or list of int or None, optional)* – The number of appearance principal components to be used for the parameters sliders. If *int*, then the number of sliders per scale is the minimum between *n_parameters* and the number of active components per scale. If *list* of *int*, then a number of sliders is defined per scale. If *None*, all the active components per scale will have a slider.

• **parameters_bounds** *(float, float), optional)* – The minimum and maximum bounds, in std units, for the sliders.

• **mode** *(single, multiple), optional)* – If 'single', only a single slider is constructed along with a drop down menu. If 'multiple', a slider is constructed for each parameter.

• **figure_size** *(int, int), optional)* – The size of the rendered figure.

view_appearance_models_widget (n_parameters=5, parameters_bounds=(-3.0, 3.0), mode='multiple', figure_size=(7, 7))
Visualizes the appearance models of the AAM object using an interactive widget.

Parameters

• **n_parameters** *(int or list of int or None, optional)* – The number of appearance principal components to be used for the parameters sliders. If *int*, then the number of sliders per scale is the minimum between *n_parameters* and the number of active components per scale. If *list* of *int*, then a number of sliders is defined per scale. If *None*, all the active components per scale will have a slider.

• **parameters_bounds** *(float, float), optional)* – The minimum and maximum bounds, in std units, for the sliders.

• **mode** *(single, multiple), optional)* – If 'single', only a single slider is constructed along with a drop down menu. If 'multiple', a slider is constructed for each parameter.

• **figure_size** *(int, int), optional)* – The size of the rendered figure.

view_shape_models_widget (n_parameters=5, parameters_bounds=(-3.0, 3.0), mode='multiple', figure_size=(7, 7))
Visualizes the shape models of the AAM object using an interactive widget.

Parameters

• **n_parameters** *(int or list of int or None, optional)* – The number of shape principal components to be used for the parameters sliders. If *int*, then the number of sliders per scale is the minimum between *n_parameters* and the number of active components per scale. If *list* of *int*, then a number of sliders is defined per scale. If *None*, all the active components per scale will have a slider.

• **parameters_bounds** *(float, float), optional)* – The minimum and maximum bounds, in std units, for the sliders.
• **mode** ([`single`, `multiple`], optional) – If 'single', only a single slider is constructed along with a drop down menu. If 'multiple', a slider is constructed for each parameter.

• **figure_size** ([`int`, `int`], optional) – The size of the rendered figure.

**property n_scales**

Returns the number of scales.

Type `int`

---

### PatchAAM

**class** `menpo.fit.aam.PatchAAM`([`images`, `group=None`, `holistic_features=<function no_op>`, `reference_shape=None`, `diagonal=None`, `scales=(0.5, 1.0)`, `patch_shape=(17, 17)`, `patch_normalisation=<function no_op>`, `shape_model_cls=<class 'menpo.fit.modelinstance.OrthoPDM'>`, `max_shape_components=None`, `max_appearance_components=None`, `verbose=False`, `batch_size=None`)**

**Bases:** `AAM`

Class for training a multi-scale Patch-Based Active Appearance Model. The appearance of this model is formulated by simply sampling patches around the image’s landmarks.

**Parameters**

• **images** ([`list of menpo.image.Image`]) – The list of training images.

• **group** ([`str` or `None`], optional) – The landmark group that will be used to train the AAM. If `None` and the images only have a single landmark group, then that is the one that will be used. Note that all the training images need to have the specified landmark group.

• **holistic_features** ([`closure` or `list of closure`], optional) – The features that will be extracted from the training images. Note that the features are extracted before warping the images to the reference shape. If `list`, then it must define a feature function per scale. Please refer to `menpo.feature` for a list of potential features.

• **reference_shape** ([`menpo.shape.PointCloud` or `None`], optional) – The reference shape that will be used for building the AAM. The purpose of the reference shape is to normalise the size of the training images. The normalization is performed by rescaling all the training images so that the scale of their ground truth shapes matches the scale of the reference shape. Note that the reference shape is rescaled with respect to the `diagonal` before performing the normalisation. If `None`, then the mean shape will be used.

• **diagonal** ([`int` or `None`], optional) – This parameter is used to rescale the reference shape so that the diagonal of its bounding box matches the provided value. In other words, this parameter controls the size of the model at the highest scale. If `None`, then the reference shape does not get rescaled.

• **scales** ([`float` or `tuple of float`], optional) – The scale value of each scale. They must be provided in ascending order, i.e. from lowest to highest scale. If `float`, then a single scale is assumed.

• **patch_shape** ([`int`, `int`] or `list of (int, int)`], optional) – The shape of the patches to be extracted. If a `list` is provided, then it defines a patch shape per scale.

• **patch_normalisation** ([`list of callable` or `single callable`], optional) – The normalisation function to be applied on the extracted patches. If `list`, then it must have length equal
to the number of scales. If a single patch normalization callable, then this is the one applied to all scales.

• **shape_model_cls** *(subclass of :class:`PDM`*, optional*) – The class to be used for building the shape model. The most common choice is `OrthoPDM`.

• **max_shape_components** *(int, float, list of those or None, optional)* – The number of shape components to keep. If int, then it sets the exact number of components. If float, then it defines the variance percentage that will be kept. If list, then it should define a value per scale. If a single number, then this will be applied to all scales. If None, then all the components are kept. Note that the unused components will be permanently trimmed.

• **max_appearance_components** *(int, float, list of those or None, optional)* – The number of appearance components to keep. If int, then it sets the exact number of components. If float, then it defines the variance percentage that will be kept. If list, then it should define a value per scale. If a single number, then this will be applied to all scales. If None, then all the components are kept. Note that the unused components will be permanently trimmed.

• **verbose** *(bool, optional)* – If True, then the progress of building the AAM will be printed.

• **batch_size** *(int or None, optional)* – If an int is provided, then the training is performed in an incremental fashion on image batches of size equal to the provided value. If None, then the training is performed directly on all the images.

**appearance_reconstructions** *(appearance_parameters, n_iters_per_scale)*

Method that generates the appearance reconstructions given a set of appearance parameters. This is to be combined with a :class:`AAMResult` object, in order to generate the appearance reconstructions of a fitting procedure.

**Parameters**

• **appearance_parameters** *(list of ndarray)* – A set of appearance parameters per fitting iteration. It can be retrieved as a property of a :class:`AAMResult` object.

• **n_iters_per_scale** *(list of int)* – The number of iterations per scale. This is necessary in order to figure out which appearance parameters correspond to the model of each scale. It can be retrieved as a property of a :class:`AAMResult` object.

**Returns** appearance_reconstructions *(list of ndarray)* – List of the appearance reconstructions that correspond to the provided parameters.

**build_fitter_interfaces** *(sampling)*

Method that builds the correct Lucas-Kanade fitting interface. It only applies in case you wish to fit the AAM with a Lucas-Kanade algorithm (i.e. :class:`LucasKanadeAAMFitter`).

**Parameters** sampling *(list of int or ndarray or None)* – It defines a sampling mask per scale. If int, then it defines the sub-sampling step of the sampling mask. If ndarray, then it explicitly defines the sampling mask. If None, then no sub-sampling is applied.

**Returns** fitter_interfaces *(list)* – The list of Lucas-Kanade interface per scale.

**increment** *(images, group=None, shape_forgetting_factor=1.0, appearance_forgetting_factor=1.0, verbose=False, batch_size=None)*

Method to increment the trained AAM with a new set of training images.

**Parameters**

• **images** *(list of menpo.image.Image)* – The list of training images.

• **group** *(str or None, optional)* – The landmark group that will be used to train the AAM. If None and the images only have a single landmark group, then that is the one that will be used. Note that all the training images need to have the specified landmark group.

2.1. Deformable Models
• `shape_forgetting_factor` ([0.0, 1.0] float, optional) – Forgetting factor that weights the relative contribution of new samples vs old samples for the shape model. If 1.0, all samples are weighted equally and, hence, the result is the exact same as performing batch PCA on the concatenated list of old and new simples. If <1.0, more emphasis is put on the new samples.

• `appearance_forgetting_factor` ([0.0, 1.0] float, optional) – Forgetting factor that weights the relative contribution of new samples vs old samples for the appearance model. If 1.0, all samples are weighted equally and, hence, the result is the exact same as performing batch PCA on the concatenated list of old and new simples. If <1.0, more emphasis is put on the new samples.

• `verbose` (bool, optional) – If True, then the progress of building the AAM will be printed.

• `batch_size` (int or None, optional) – If an int is provided, then the training is performed in an incremental fashion on image batches of size equal to the provided value. If None, then the training is performed directly on all the images.

`instance` (shape_weights=None, appearance_weights=None, scale_index=-1)
Generates a novel AAM instance given a set of shape and appearance weights. If no weights are provided, then the mean AAM instance is returned.

Parameters

• `shape_weights` ((n_weights,) ndarray or list or None, optional) – The weights of the shape model that will be used to create a novel shape instance. If None, the weights are assumed to be zero, thus the mean shape is used.

• `appearance_weights` ((n_weights,) ndarray or list or None, optional) – The weights of the appearance model that will be used to create a novel appearance instance. If None, the weights are assumed to be zero, thus the mean appearance is used.

• `scale_index` (int, optional) – The scale to be used.

Returns `image` (menpo.image.Image) – The AAM instance.

`random_instance` (scale_index=-1)
Generates a random instance of the AAM.

Parameters `scale_index` (int, optional) – The scale to be used.

Returns `image` (menpo.image.Image) – The AAM instance.

`view_aam_widget` (n_shape_parameters=5, n_appearance_parameters=5, parameters_bounds=(-3.0, 3.0), mode='multiple', figure_size=(7, 7))
Visualizes the AAM using an interactive widget.

Parameters

• `n_shape_parameters` (int or list of int or None, optional) – The number of shape principal components to be used for the parameters sliders. If int, then the number of sliders per scale is the minimum between n_parameters and the number of active components per scale. If list of int, then a number of sliders is defined per scale. If None, all the active components per scale will have a slider.

• `n_appearance_parameters` (int or list of int or None, optional) – The number of appearance principal components to be used for the parameters sliders. If int, then the number of sliders per scale is the minimum between n_parameters and the number of active components per scale. If list of int, then a number of sliders is defined per scale. If None, all the active components per scale will have a slider.
MenpoFit Documentation, Release 0.6.1

- **parameters_bounds** ((float, float), optional) – The minimum and maximum bounds, in std units, for the sliders.

- **mode** ([single, multiple], optional) – If 'single', only a single slider is constructed along with a drop down menu. If 'multiple', a slider is constructed for each parameter.

- **figure_size** ((int, int), optional) – The size of the rendered figure.

```python
view.AppearanceModelsWidget(n_parameters=5, parameters_bounds=(-3.0, 3.0), mode='multiple', figure_size=(7, 7))
```

Visualizes the appearance models of the AAM object using an interactive widget.

**Parameters**

- **n_parameters** (int or list of int or None, optional) – The number of appearance principal components to be used for the parameters sliders. If int, then the number of sliders per scale is the minimum between n_parameters and the number of active components per scale. If list of int, then a number of sliders is defined per scale. If None, all the active components per scale will have a slider.

- **parameters_bounds** ((float, float), optional) – The minimum and maximum bounds, in std units, for the sliders.

- **mode** ([single, multiple], optional) – If 'single', only a single slider is constructed along with a drop down menu. If 'multiple', a slider is constructed for each parameter.

- **figure_size** ((int, int), optional) – The size of the rendered figure.

```python
view.ShapeModelsWidget(n_parameters=5, parameters_bounds=(-3.0, 3.0), mode='multiple', figure_size=(7, 7))
```

Visualizes the shape models of the AAM object using an interactive widget.

**Parameters**

- **n_parameters** (int or list of int or None, optional) – The number of shape principal components to be used for the parameters sliders. If int, then the number of sliders per scale is the minimum between n_parameters and the number of active components per scale. If list of int, then a number of sliders is defined per scale. If None, all the active components per scale will have a slider.

- **parameters_bounds** ((float, float), optional) – The minimum and maximum bounds, in std units, for the sliders.

- **mode** ([single, multiple], optional) – If 'single', only a single slider is constructed along with a drop down menu. If 'multiple', a slider is constructed for each parameter.

- **figure_size** ((int, int), optional) – The size of the rendered figure.

**property n_scales**

Returns the number of scales.

Type **int**

**Fitters**

An AAM can be optimised either in a gradient descent manner (Lucas-Kanade) or using cascaded regression (Supervised Descent).

2.1. Deformable Models
LucasKanadeAAMFitter

class menpofit.aam.LucasKanadeAAMFitter(aam,  lk_algorithm_cls=<class 'menpofit.aam.algorithm.lk.WibergInverseCompositional'>,  n_shape=None,  n_appearance=None,  sampling=None)

Bases: AAMFitter

Class for defining an AAM fitter using the Lucas-Kanade optimisation.

Note: When using a method with a parametric shape model, the first step is to **reconstruct the initial shape** using the shape model. The generated reconstructed shape is then used as initialisation for the iterative optimisation. This step takes place at each scale and it is not considered as an iteration, thus it is not counted for the provided max_iters.

Parameters

- **aam (AAM or subclass)** – The trained AAM model.
- **lk_algorithm_cls (class, optional)** – The Lukas-Kanade optimisation algorithm that will get applied. The possible algorithms are:

<table>
<thead>
<tr>
<th>Class</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlternatingForwardCompositional</td>
<td>Alternating</td>
</tr>
<tr>
<td>AlternatingInverseCompositional</td>
<td></td>
</tr>
<tr>
<td>ModifiedAlternatingForwardCompositional</td>
<td>Modified Alternating</td>
</tr>
<tr>
<td>ModifiedAlternatingInverseCompositional</td>
<td></td>
</tr>
<tr>
<td>ProjectOutForwardCompositional</td>
<td>Project-Out</td>
</tr>
<tr>
<td>ProjectOutInverseCompositional</td>
<td></td>
</tr>
<tr>
<td>SimultaneousForwardCompositional</td>
<td>Simultaneous</td>
</tr>
<tr>
<td>SimultaneousInverseCompositional</td>
<td></td>
</tr>
<tr>
<td>WibergForwardCompositional</td>
<td>Wiberg</td>
</tr>
<tr>
<td>WibergInverseCompositional</td>
<td></td>
</tr>
</tbody>
</table>

- **n_shape (int or float or list of those or None, optional)** – The number of shape components that will be used. If int, then it defines the exact number of active components. If float, then it defines the percentage of variance to keep. If int or float, then the provided value will be applied for all scales. If list, then it defines a value per scale. If None, then all the available components will be used. Note that this simply sets the active components without trimming the unused ones. Also, the available components may have already been trimmed to max_shape_components during training.

- **n_appearance (int or float or list of those or None, optional)** – The number of appearance components that will be used. If int, then it defines the exact number of active components. If float, then it defines the percentage of variance to keep. If int or float, then the provided value will be applied for all scales. If list, then it defines a value per scale. If None, then all the available components will be used. Note that this simply sets the active components without trimming the unused ones. Also, the available components may have already been trimmed to max_appearance_components during training.

- **sampling (list of int or ndarray or None)** – It defines a sampling mask per scale. If int, then it defines the sub-sampling step of the sampling mask. If ndarray, then it explicitly defines the sampling mask. If None, then no sub-sampling is applied.
**appearance_reconstructions** *(appearance_parameters, n_iters_per_scale)*

Method that generates the appearance reconstructions given a set of appearance parameters. This is to be combined with a `AAMResult` object, in order to generate the appearance reconstructions of a fitting procedure.

**Parameters**

- **appearance_parameters** *(list of (n_params,) ndarray)* – A set of appearance parameters per fitting iteration. It can be retrieved as a property of an `AAMResult` object.

- **n_iters_per_scale** *(list of int)* – The number of iterations per scale. This is necessary in order to figure out which appearance parameters correspond to the model of each scale. It can be retrieved as a property of a `AAMResult` object.

**Returns** *appearance_reconstructions* *(list of menpo.image.Image)* – List of the appearance reconstructions that correspond to the provided parameters.

**fit_from_bb** *(image, bounding_box, max_iters=20, gt_shape=None, return_costs=False, **kwargs)*

Fits the multi-scale fitter to an image given an initial bounding box.

**Parameters**

- **image** *(menpo.image.Image or subclass)* – The image to be fitted.

- **bounding_box** *(menpo.shape.PointDirectedGraph)* – The initial bounding box from which the fitting procedure will start. Note that the bounding box is used in order to align the model’s reference shape.

- **max_iters** *(int or list of int, optional)* – The maximum number of iterations. If `int`, then it specifies the maximum number of iterations over all scales. If `list of int`, then specifies the maximum number of iterations per scale.

- **gt_shape** *(menpo.shape.PointCloud, optional)* – The ground truth shape associated to the image.

- **return_costs** *(bool, optional)* – If `True`, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned `fitting_result`. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.

- **kwargs** *(dict, optional)* – Additional keyword arguments that can be passed to specific implementations.

**Returns** *fitting_result* *(MultiScaleNonParametricIterativeResult or subclass)* – The multi-scale fitting result containing the result of the fitting procedure.

**fit_from_shape** *(image, initial_shape, max_iters=20, gt_shape=None, return_costs=False, **kwargs)*

Fits the multi-scale fitter to an image given an initial shape.

**Parameters**

- **image** *(menpo.image.Image or subclass)* – The image to be fitted.

- **initial_shape** *(menpo.shape.PointCloud)* – The initial shape estimate from which the fitting procedure will start.

- **max_iters** *(int or list of int, optional)* – The maximum number of iterations. If `int`, then it specifies the maximum number of iterations over all scales. If `list of int`, then specifies the maximum number of iterations per scale.

- **gt_shape** *(menpo.shape.PointCloud, optional)* – The ground truth shape associated to the image.
• **return_costs** *(bool, optional)* – If `True`, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned `fitting_result`. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.

• **kwargs** *(dict, optional)* – Additional keyword arguments that can be passed to specific implementations.

**Returns** *fitting_result* *(MultiScaleNonParametricIterativeResult or subclass)* – The multi-scale fitting result containing the result of the fitting procedure.

**warped_images** *(image, shapes)*

Given an input test image and a list of shapes, it warps the image into the shapes. This is useful for generating the warped images of a fitting procedure stored within an `AAMResult`.

**Parameters**

• **image** *(menpo.image.Image or subclass)* – The input image to be warped.

• **shapes** *(list of menpo.shape.PointCloud)* – The list of shapes in which the image will be warped. The shapes are obtained during the iterations of a fitting procedure.

**Returns** *warped_images* *(list of menpo.image.MaskedImage or ndarray)* – The warped images.

**property aam**
The trained AAM model.

*Type* `AAM or subclass`

**property holistic_features**
The features that are extracted from the input image at each scale in ascending order, i.e. from lowest to highest scale.

*Type* `list of closure`

**property n_scales**
Returns the number of scales.

*Type* `int`

**property reference_shape**
The reference shape that is used to normalise the size of an input image so that the scale of its initial fitting shape matches the scale of this reference shape.

*Type* `menpo.shape.PointCloud`

**property scales**
The scale value of each scale in ascending order, i.e. from lowest to highest scale.

*Type* `list of int or float`
SupervisedDescentAAMFitter

class menpofit.aam.SupervisedDescentAAMFitter(images, aam, group=None, bounding_box_group_glob=None, n_shape=None, n_appearance=None, sampling=None, sd_algorithm_cls=<class 'menpofit.aam.algorithm.sd.ProjectOutNewton'>, n_iterations=6, n_perturbations=30, perturb_from_gt_bounding_box=<function noisy_shape_from_bounding_box>, batch_size=None, verbose=False)

Bases: SupervisedDescentFitter

Class for training a multi-scale cascaded-regression Supervised Descent AAM fitter.

Parameters

- **images** (list of menpo.image.Image) – The list of training images.

- **aam** (AAM or subclass) – The trained AAM model.

- **group** (str or None, optional) – The landmark group that will be used to train the fitter. If None and the images only have a single landmark group, then that is the one that will be used. Note that all the training images need to have the specified landmark group.

- **bounding_box_group_glob** (glob or None, optional) – Glob that defines the bounding boxes to be used for training. If None, then the bounding boxes of the ground truth shapes are used.

- **n_shape** (int or float or list of those or None, optional) – The number of shape components that will be used. If int, then it defines the exact number of active components. If float, then it defines the percentage of variance to keep. If int or float, then the provided value will be applied for all scales. If list, then it defines a value per scale. If None, then all the available components will be used. Note that this simply sets the active components without trimming the unused ones. Also, the available components may have already been trimmed to max_shape_components during training.

- **n_appearance** (int or float or list of those or None, optional) – The number of appearance components that will be used. If int, then it defines the exact number of active components. If float, then it defines the percentage of variance to keep. If int or float, then the provided value will be applied for all scales. If list, then it defines a value per scale. If None, then all the available components will be used. Note that this simply sets the active components without trimming the unused ones. Also, the available components may have already been trimmed to max_appearance_components during training.

- **sampling** (list of int or ndarray or None) – It defines a sampling mask per scale. If int, then it defines the sub-sampling step of the sampling mask. If ndarray, then it explicitly defines the sampling mask. If None, then no sub-sampling is applied.

- **sd_algorithm_cls** (class, optional) – The Supervised Descent algorithm to be used. The possible algorithms are:
<table>
<thead>
<tr>
<th>Class</th>
<th>Features</th>
<th>Regression</th>
</tr>
</thead>
<tbody>
<tr>
<td>MeanTemplateNewton</td>
<td>Mean Template</td>
<td>IRLRegression</td>
</tr>
<tr>
<td>MeanTemplateGaussNewton</td>
<td></td>
<td>IRLRegression</td>
</tr>
<tr>
<td>ProjectOutNewton</td>
<td>Project-Out</td>
<td>IRLRegression</td>
</tr>
<tr>
<td>ProjectOutGaussNewton</td>
<td></td>
<td>IRLRegression</td>
</tr>
<tr>
<td>AppearanceWeightsNewton</td>
<td>App. Weights</td>
<td>IRLRegression</td>
</tr>
<tr>
<td>AppearanceWeightsGaussNewton</td>
<td></td>
<td>IRLRegression</td>
</tr>
</tbody>
</table>

- **n_iterations** *(int or list of int, optional)* – The number of iterations (cascades) of each level. If *list*, it must specify a value per scale. If *int*, then it defines the total number of iterations (cascades) over all scales.

- **n_perturbations** *(int or None, optional)* – The number of perturbations to be generated from the provided bounding boxes.

- **perturb_from_gt_bounding_box** *(callable, optional)* – The function that will be used to generate the perturbations.

- **batch_size** *(int or None, optional)* – If an *int* is provided, then the training is performed in an incremental fashion on image batches of size equal to the provided value. If *None*, then the training is performed directly on all the images.

- **verbose** *(bool, optional)* – If True, then the progress of training will be printed.

### fit_from_bb.*(image, bounding_box, max_iters=20, gt_shape=None, return_costs=False, **kwargs)*

Fits the multi-scale fitter to an image given an initial bounding box.

**Parameters**

- **image** *(menpo.image.Image or subclass)* – The image to be fitted.

- **bounding_box** *(menpo.shape.PointDirectedGraph)* – The initial bounding box from which the fitting procedure will start. Note that the bounding box is used in order to align the model’s reference shape.

- **max_iters** *(int or list of int, optional)* – The maximum number of iterations. If *int*, then it specifies the maximum number of iterations over all scales. If *list* of *int*, then specifies the maximum number of iterations per scale.

- **gt_shape** *(menpo.shape.PointCloud, optional)* – The ground truth shape associated to the image.

- **return_costs** *(bool, optional)* – If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting result. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.

- **kwargs** *(dict, optional)* – Additional keyword arguments that can be passed to specific implementations.

**Returns** *fitting_result*(MultiScaleNonParametricIterativeResult or subclass) – The multi-scale fitting result containing the result of the fitting procedure.

### fit_from_shape.*(image, initial_shape, max_iters=20, gt_shape=None, return_costs=False, **kwargs)*

Fits the multi-scale fitter to an image given an initial shape.

**Parameters**

- **image** *(menpo.image.Image or subclass)* – The image to be fitted.
• **initial_shape** (*menpo.shape.PointCloud*) – The initial shape estimate from which the fitting procedure will start.

• **max_iters** (int or list of int, optional) – The maximum number of iterations. If int, then it specifies the maximum number of iterations over all scales. If list of int, then specifies the maximum number of iterations per scale.

• **gt_shape** (*menpo.shape.PointCloud*, optional) – The ground truth shape associated to the image.

• **return_costs** (bool, optional) – If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.

• **kwargs** (dict, optional) – Additional keyword arguments that can be passed to specific implementations.

Returns **fitting_result** (*MultiScaleNonParametricIterativeResult* or subclass) – The multi-scale fitting result containing the result of the fitting procedure.

### increment

**increment** (*images*, **group=None**, **bounding_box_group_glob=None**, **verbose=False**, **batch_size=None**)

Method to increment the trained SDM with a new set of training images.

Parameters

• **images** (list of *menpo.image.Image*) – The list of training images.

• **group** (str or None, optional) – The landmark group that corresponds to the ground truth shape of each image. If None and the images only have a single landmark group, then that is the one that will be used. Note that all the training images need to have the specified landmark group.

• **bounding_box_group_glob** (glob or None, optional) – Glob that defines the bounding boxes to be used for training. If None, then the bounding boxes of the ground truth shapes are used.

• **verbose** (bool, optional) – If True, then the progress of training will be printed.

• **batch_size** (int or None, optional) – If an int is provided, then the training is performed in an incremental fashion on image batches of size equal to the provided value. If None, then the training is performed directly on all the images.

**warped_images** (*image*, *shapes*)

Given an input test image and a list of shapes, it warps the image into the shapes. This is useful for generating the warped images of a fitting procedure stored within a *MultiScaleParametricIterativeResult*.

Parameters

• **image** (*menpo.image.Image* or subclass) – The input image to be warped.

• **shapes** (list of *menpo.shape.PointCloud*) – The list of shapes in which the image will be warped. The shapes are obtained during the iterations of a fitting procedure.

Returns **warped_images** (list of *menpo.image.MaskedImage* or *ndarray*) – The warped images.

**property holistic_features**

The features that are extracted from the input image at each scale in ascending order, i.e. from lowest to highest scale.

**Type** list of closure
property n_scales
    Returns the number of scales.
    Type int

property reference_shape
    The reference shape that is used to normalise the size of an input image so that the scale of its initial fitting shape matches the scale of this reference shape.
    Type menpo.shape.PointCloud

property scales
    The scale value of each scale in ascending order, i.e. from lowest to highest scale.
    Type list of int or float

Lucas-Kanade Optimisation Algorithms

AlternatingForwardCompositional

class menpofit.aam.AlternatingForwardCompositional (aam_interface, eps=1e-05)
    Bases: Alternating

Alternating Forward Compositional (AFC) Gauss-Newton algorithm.

run (image, initial_shape, gt_shape=None, max_iters=20, return_costs=False, map_inference=False)
    Execute the optimization algorithm.

Parameters

• image (menpo.image.Image) – The input test image.

• initial_shape (menpo.shape.PointCloud) – The initial shape from which the optimization will start.

• gt_shape (menpo.shape.PointCloud or None, optional) – The ground truth shape of the image. It is only needed in order to get passed in the optimization result object, which has the ability to compute the fitting error.

• max_iters (int, optional) – The maximum number of iterations. Note that the algorithm may converge, and thus stop, earlier.

• return_costs (bool, optional) – If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.

• map_inference (bool, optional) – If True, then the solution will be given after performing MAP inference.

Returns fitting_result (AAMAlgorithmResult) – The parametric iterative fitting result.

property appearance_model
    Returns the appearance model of the AAM.
    Type menpo.model.PCAModel

property template
    Returns the template of the AAM (usually the mean of the appearance model).
    Type menpo.image.Image or subclass
property transform

Returns the model driven differential transform object of the AAM, e.g.
DifferentiablePiecewiseAffine or DifferentiableThinPlateSplines.

Type subclass of DL and DX

AlternatingInverseCompositional

class menpofit.aam.AlternatingInverseCompositional (aam_interface, eps=1e-05)
Bases: Alternating

Alternating Inverse Compositional (AIC) Gauss-Newton algorithm.

run (image, initial_shape, gt_shape=None, max_iters=20, return_costs=False, map_inference=False)

Execute the optimization algorithm.

Parameters

• image (menpo.image.Image) – The input test image.
• initial_shape (menpo.shape.PointCloud) – The initial shape from which the optimization will start.
• gt_shape (menpo.shape.PointCloud or None, optional) – The ground truth shape of the image. It is only needed in order to get passed in the optimization result object, which has the ability to compute the fitting error.
• max_iters (int, optional) – The maximum number of iterations. Note that the algorithm may converge, and thus stop, earlier.
• return_costs (bool, optional) – If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.
• map_inference (bool, optional) – If True, then the solution will be given after performing MAP inference.

Returns fitting_result (AAMAlgorithmResult) – The parametric iterative fitting result.

property appearance_model

Returns the appearance model of the AAM.

Type menpo.model.PCAModel

property template

Returns the template of the AAM (usually the mean of the appearance model).

Type menpo.image.Image or subclass

property transform

Returns the model driven differential transform object of the AAM, e.g.
DifferentiablePiecewiseAffine or DifferentiableThinPlateSplines.

Type subclass of DL and DX
ModifiedAlternatingForwardCompositional

class menpofit.aam.ModifiedAlternatingForwardCompositional(aam_interface, eps=1e-05)
Bases: ModifiedAlternating
Modified Alternating Forward Compositional (MAFC) Gauss-Newton algorithm

run(image, initial_shape, gt_shape=None, max_iters=20, return_costs=False, map_inference=False)
  Execute the optimization algorithm.

Parameters
  • image (menpo.image.Image) – The input test image.
  • initial_shape (menpo.shape.PointCloud) – The initial shape from which the optimization will start.
  • gt_shape (menpo.shape.PointCloud or None, optional) – The ground truth shape of the image. It is only needed in order to get passed in the optimization result object, which has the ability to compute the fitting error.
  • max_iters (int, optional) – The maximum number of iterations. Note that the algorithm may converge, and thus stop, earlier.
  • return_costs (bool, optional) – If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.
  • map_inference (bool, optional) – If True, then the solution will be given after performing MAP inference.

Returns fitting_result (AAMAlgorithmResult) – The parametric iterative fitting result.

property appearance_model
  Returns the appearance model of the AAM.
  
  Type menpo.model.PCAModel

property template
  Returns the template of the AAM (usually the mean of the appearance model).
  
  Type menpo.image.Image or subclass

property transform
  Returns the model driven differential transform object of the AAM, e.g. DifferentiablePiecewiseAffine or DifferentiableThinPlateSplines.
  
  Type subclass of DL and DX

ModifiedAlternatingInverseCompositional

class menpofit.aam.ModifiedAlternatingInverseCompositional(aam_interface, eps=1e-05)
Bases: ModifiedAlternating
Modified Alternating Inverse Compositional (MAIC) Gauss-Newton algorithm

run(image, initial_shape, gt_shape=None, max_iters=20, return_costs=False, map_inference=False)
  Execute the optimization algorithm.
Parameters

• **image** (menpo.image.Image) – The input test image.

• **initial_shape** (menpo.shape.PointCloud) – The initial shape from which the optimization will start.

• **gt_shape** (menpo.shape.PointCloud or None, optional) – The ground truth shape of the image. It is only needed in order to get passed in the optimization result object, which has the ability to compute the fitting error.

• **max_iters** (int, optional) – The maximum number of iterations. Note that the algorithm may converge, and thus stop, earlier.

• **return_costs** (bool, optional) – If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting result. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.

• **map_inference** (bool, optional) – If True, then the solution will be given after performing MAP inference.

Returns **fitting_result** (AAMAlgorithmResult) – The parametric iterative fitting result.

**property appearance_model**

Returns the appearance model of the AAM.

Type menpo.model.PCAModel

**property template**

Returns the template of the AAM (usually the mean of the appearance model).

Type menpo.image.Image or subclass

**property transform**

Returns the model driven differential transform object of the AAM, e.g. DifferentiablePiecewiseAffine or DifferentiableThinPlateSplines.

Type subclass of DL and DX

ProjectOutForwardCompositional

class menpofit.aam.ProjectOutForwardCompositional (aam_interface, eps=1e-05)

Bases: ProjectOut

Project-out Forward Compositional (POFC) Gauss-Newton algorithm.

project_out (J)

Projects-out the appearance subspace from a given vector or matrix.

Type ndarray

run (image, initial_shape, gt_shape=None, max_iters=20, return_costs=False, map_inference=False)

Execute the optimization algorithm.

Parameters

• **image** (menpo.image.Image) – The input test image.

• **initial_shape** (menpo.shape.PointCloud) – The initial shape from which the optimization will start.
• `gt_shape` *(menpo.shape<PointCloud> or None, optional)* – The ground truth shape of the image. It is only needed in order to get passed in the optimization result object, which has the ability to compute the fitting error.

• `max_iters` *(int, optional)* – The maximum number of iterations. Note that the algorithm may converge, and thus stop, earlier.

• `return_costs` *(bool, optional)* – If `True`, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned `fitting_result`. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.

• `map_inference` *(bool, optional)* – If `True`, then the solution will be given after performing MAP inference.

Returns `fitting_result` *(AAMAlgorithmResult)* – The parametric iterative fitting result.

**property appearance_model**

Returns the appearance model of the AAM.

  Type `menpo.model.PCAModel`

**property template**

Returns the template of the AAM (usually the mean of the appearance model).

  Type `menpo.image.Image` or subclass

**property transform**

Returns the model driven differential transform object of the AAM, e.g. `DifferentiablePiecewiseAffine` or `DifferentiableThinPlateSplines`.

  Type subclass of `DL` and `DX`

### ProjectOutInverseCompositional

class `menpofit.aam.ProjectOutInverseCompositional(aam_interface, eps=1e-05)`

Bases: `ProjectOut`

Project-out Inverse Compositional (POIC) Gauss-Newton algorithm.

**project_out** *(J)*

Projects-out the appearance subspace from a given vector or matrix.

  Type `ndarray`

**run** *(image, initial_shape, gt_shape=None, max_iters=20, return_costs=False, map_inference=False)*

Execute the optimization algorithm.

**Parameters**

• `image` *(menpo.image.Image)* – The input test image.

• `initial_shape` *(menpo.shape<PointCloud})* – The initial shape from which the optimization will start.

• `gt_shape` *(menpo.shape<PointCloud} or None, optional)* – The ground truth shape of the image. It is only needed in order to get passed in the optimization result object, which has the ability to compute the fitting error.

• `max_iters` *(int, optional)* – The maximum number of iterations. Note that the algorithm may converge, and thus stop, earlier.
• **return_costs** *(bool, optional)* – If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned *fitting_result*. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.

• **map_inference** *(bool, optional)* – If True, then the solution will be given after performing MAP inference.

Returns *fitting_result* *(AAMAlgorithmResult)* – The parametric iterative fitting result.

**property appearance_model**

Returns the appearance model of the AAM.

Type *menpo.model.PCAModel*

**property template**

Returns the template of the AAM (usually the mean of the appearance model).

Type *menpo.image.Image* or subclass

**property transform**

Returns the model driven differential transform object of the AAM, e.g. *DifferentiablePiecewiseAffine* or *DifferentiableThinPlateSplines*.

Type subclass of *DL* and *DX*

### SimultaneousForwardCompositional

class *menpofit.aam.SimultaneousForwardCompositional* *(aam_interface, eps=1e-05)*

Bases: Simultaneous

Simultaneous Forward Compositional (SFC) Gauss-Newton algorithm.

**run** *(image, initial_shape, gt_shape=None, max_iters=20, return_costs=False, map_inference=False)*

Execute the optimization algorithm.

**Parameters**

• **image** *(menpo.image.Image)* – The input test image.

• **initial_shape** *(menpo.shape.PointCloud)* – The initial shape from which the optimization will start.

• **gt_shape** *(menpo.shape.PointCloud or None, optional)* – The ground truth shape of the image. It is only needed in order to get passed in the optimization result object, which has the ability to compute the fitting error.

• **max_iters** *(int, optional)* – The maximum number of iterations. Note that the algorithm may converge, and thus stop, earlier.

• **return_costs** *(bool, optional)* – If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned *fitting_result*. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.

• **map_inference** *(bool, optional)* – If True, then the solution will be given after performing MAP inference.

Returns *fitting_result* *(AAMAlgorithmResult)* – The parametric iterative fitting result.
**property appearance_model**

Returns the appearance model of the AAM.

Type `menpo.model.PCAModel`

**property template**

Returns the template of the AAM (usually the mean of the appearance model).

Type `menpo.image.Image` or subclass

**property transform**

Returns the model driven differential transform object of the AAM, e.g. `DifferentiablePiecewiseAffine` or `DifferentiableThinPlateSplines`.

Type `subclass` of `DL` and `DX`

---

### SimultaneousInverseCompositional

**class** `menpofit.aam.SimultaneousInverseCompositional` *(aam_interface, eps=1e-05)*

Bases: `Simultaneous`

Simultaneous Inverse Compositional (SIC) Gauss-Newton algorithm.

**run** *(image, initial_shape, gt_shape=None, max_iters=20, return_costs=False, map_inference=False)*

Execute the optimization algorithm.

**Parameters**

- **image** *(menpo.image.Image)* – The input test image.

- **initial_shape** *(menpo.shape.PointCloud)* – The initial shape from which the optimization will start.

- **gt_shape** *(menpo.shape.PointCloud or None, optional)* – The ground truth shape of the image. It is only needed in order to get passed in the optimization result object, which has the ability to compute the fitting error.

- **max_iters** *(int, optional)* – The maximum number of iterations. Note that the algorithm may converge, and thus stop, earlier.

- **return_costs** *(bool, optional)* – If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.

- **map_inference** *(bool, optional)* – If True, then the solution will be given after performing MAP inference.

**Returns** `fitting_result` *(AAMAlgorithmResult)* – The parametric iterative fitting result.

**property appearance_model**

Returns the appearance model of the AAM.

Type `menpo.model.PCAModel`

**property template**

Returns the template of the AAM (usually the mean of the appearance model).

Type `menpo.image.Image` or subclass
**property transform**

Returns the model driven differential transform object of the AAM, e.g. `DifferentiablePiecewiseAffine` or `DifferentiableThinPlateSplines`.

*Type* subclass of DL and DX

---

**WibergForwardCompositional**

```python
class menpofit.aam.WibergForwardCompositional(aam_interface, eps=1e-05)
Bases: Wiberg

Wiberg Forward Compositional (WFC) Gauss-Newton algorithm.

run(image, initial_shape, gt_shape=None, max_iters=20, return_costs=False, map_inference=False)
Execute the optimization algorithm.
```

**Parameters**

- **image** *(menpo.image.Image)* – The input test image.
- **initial_shape** *(menpo.shape.PointCloud)* – The initial shape from which the optimization will start.
- **gt_shape** *(menpo.shape.PointCloud or None, optional)* – The ground truth shape of the image. It is only needed in order to get passed in the optimization result object, which has the ability to compute the fitting error.
- **max_iters** *(int, optional)* – The maximum number of iterations. Note that the algorithm may converge, and thus stop, earlier.
- **return_costs** *(bool, optional)* – If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.
- **map_inference** *(bool, optional)* – If True, then the solution will be given after performing MAP inference.

*Returns* fitting_result *(AAMAlgorithmResult)* – The parametric iterative fitting result.

---

**property appearance_model**

Returns the appearance model of the AAM.

*Type* menpo.model.PCAModel

**property template**

Returns the template of the AAM (usually the mean of the appearance model).

*Type* menpo.image.Image or subclass

---

**property transform**

Returns the model driven differential transform object of the AAM, e.g. `DifferentiablePiecewiseAffine` or `DifferentiableThinPlateSplines`.

*Type* subclass of DL and DX

---

**WibergInverseCompositional**

```python
class menpofit.aam.WibergInverseCompositional(aam_interface, eps=1e-05)
Bases: Wiberg

Wiberg Inverse Compositional (WIC) Gauss-Newton algorithm.
```
Wiberg Inverse Compositional (WIC) Gauss-Newton algorithm.

run \( \text{image, initial\_shape, gt\_shape=None, max\_iters=20, return\_costs=False, map\_inference=False} \)

Execute the optimization algorithm.

**Parameters**

- **image** *(menpo.image.Image)* – The input test image.
- **initial\_shape** *(menpo.shape.PointCloud)* – The initial shape from which the optimization will start.
- **gt\_shape** *(menpo.shape.PointCloud or None, optional)* – The ground truth shape of the image. It is only needed in order to get passed in the optimization result object, which has the ability to compute the fitting error.
- **max\_iters** *(int, optional)* – The maximum number of iterations. Note that the algorithm may converge, and thus stop, earlier.
- **return\_costs** *(bool, optional)* – If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting\_result. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.
- **map\_inference** *(bool, optional)* – If True, then the solution will be given after performing MAP inference.

**Returns** fitting\_result *(AAMAlgorithmResult)* – The parametric iterative fitting result.

**property appearance\_model**

Returns the appearance model of the AAM.

Type *menpo.model.PCAModel*

**property template**

Returns the template of the AAM (usually the mean of the appearance model).

Type *menpo.image.Image or subclass*

**property transform**

Returns the model driven differential transform object of the AAM, e.g. *DifferentiablePiecewiseAffine* or *DifferentiableThinPlateSplines*.

Type *subclass of DL and DX*

**Supervised Descent Optimisation Algorithms**

**AppearanceWeightsNewton**

class menpofit.aam.AppearanceWeightsNewton *(aam\_interface, n\_iterations=3, compute\_error=<function euclidean\_bb\_normalised\_error>, alpha=0, bias=True)*

**Bases:** AppearanceWeights

Class for training a cascaded-regression Newton algorithm using Incremental Regularized Linear Regression (*IRLRegression*) given a trained AAM model. The algorithm uses the projection weights of the appearance vectors as features in the regression.

**Parameters**
MenpoFit Documentation, Release 0.6.1

- **aam_interface** (The AAM interface class from menpofit.aam.algorithm.lk.) – Existing interfaces include:

<table>
<thead>
<tr>
<th>Interface</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'LucasKanadeStandardInterface'</td>
<td>Suitable for holistic AAMs</td>
</tr>
<tr>
<td>'LucasKanadeLinearInterface'</td>
<td>Suitable for linear AAMs</td>
</tr>
<tr>
<td>'LucasKanadePatchInterface'</td>
<td>Suitable for patch-based AAMs</td>
</tr>
</tbody>
</table>

- **n_iterations** *(int, optional)* – The number of iterations (cascades).
- **compute_error** *(callable, optional)* – The function to be used for computing the fitting error when training each cascade.
- **alpha** *(float, optional)* – The regularization parameter.
- **bias** *(bool, optional)* – Flag that controls whether to use a bias term.

**increment** *(images, gt_shapes, current_shapes, prefix='', verbose=False)*
Method to increment the model with the set of current shapes.

**Parameters**
- **images** *(list of menpo.image.Image)* – The list of training images.
- **gt_shapes** *(list of menpo.shape.PointCloud)* – The list of ground truth shapes that correspond to the images.
- **current_shapes** *(list of menpo.shape.PointCloud)* – The list of current shapes that correspond to the images.
- **prefix** *(str, optional)* – The prefix to use when printing information.
- **verbose** *(bool, optional)* – If True, then information is printed during training.

**Returns** **current_shapes** *(list of menpo.shape.PointCloud)* – The list of current shapes that correspond to the images.

**project** *(J)*
Projects a given vector or matrix onto the appearance subspace.

**Type** *ndarray*

**run** *(image, initial_shape, gt_shape=None, return_costs=False, **kwargs)*
Run the algorithm to an image given an initial shape.

**Parameters**
- **image** *(menpo.image.Image or subclass)* – The image to be fitted.
- **initial_shape** *(menpo.shape.PointCloud)* – The initial shape from which the fitting procedure will start.
- **gt_shape** *(menpo.shape.PointCloud or None, optional)* – The ground truth shape associated to the image.
- **return_costs** *(bool, optional)* – If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that this argument currently has no effect and will raise a warning if set to ‘‘True’’. This is because it is not possible to evaluate the cost function of this algorithm.

**Returns** **fitting_result** *(AAMAlgorithmResult)* – The parametric iterative fitting result.

**train** *(images, gt_shapes, current_shapes, prefix='', verbose=False)*
Method to train the model given a set of initial shapes.
Parameters

- **images** *(list of menpo.image.Image)* – The list of training images.
- **gt_shapes** *(list of menpo.shape.PointCloud)* – The list of ground truth shapes that correspond to the images.
- **current_shapes** *(list of menpo.shape.PointCloud)* – The list of current shapes that correspond to the images, which will be used as initial shapes.
- **prefix** *(str, optional)* – The prefix to use when printing information.
- **verbose** *(bool, optional)* – If True, then information is printed during training.

Returns **current_shapes** *(list of menpo.shape.PointCloud)* – The list of current shapes that correspond to the images.

**property appearance_model**
Returns the appearance model of the AAM.

Type *menpo.model.PCAModel*

**property transform**
Returns the model driven differential transform object of the AAM, e.g. `DifferentiablePiecewiseAffine` or `DifferentiableThinPlateSplines`.

Type *subclass of DL and DX*

### AppearanceWeightsGaussNewton

**class menpofit.aam.AppearanceWeightsGaussNewton**(aam_interface, n_iterations=3, compute_error=<function euclidean_bb_normalised_error>, alpha=0, alpha2=0, bias=True)

Bases: AppearanceWeights

Class for training a cascaded-regression Gauss-Newton algorithm using Indirect Incremental Regularized Linear Regression (*IIRLRegression*) given a trained AAM model. The algorithm uses the projection weights of the appearance vectors as features in the regression.

Parameters

- **aam_interface** *(The AAM interface class from menpofit.aam.algorithm.lk.)* – Existing interfaces include:
  - `'LucasKanadeStandardInterface'` Suitable for holistic AAMs
  - `'LucasKanadeLinearInterface'` Suitable for linear AAMs
  - `'LucasKanadePatchInterface'` Suitable for patch-based AAMs

- **n_iterations** *(int, optional)* – The number of iterations (cascades).
- **compute_error** *(callable, optional)* – The function to be used for computing the fitting error when training each cascade.
- **alpha** *(float, optional)* – The regularization parameter.
- **alpha2** *(float, optional)* – The regularization parameter of the Hessian matrix.
- **bias** *(bool, optional)* – Flag that controls whether to use a bias term.

**increment** *(images, gt_shapes, current_shapes, prefix='', verbose=False)*

Method to increment the model with the set of current shapes.
Parameters

- **images** (list of menpo.image.Image) – The list of training images.
- **gt_shapes** (list of menpo.shape.PointCloud) – The list of ground truth shapes that correspond to the images.
- **current_shapes** (list of menpo.shape.PointCloud) – The list of current shapes that correspond to the images.
- **prefix** (str, optional) – The prefix to use when printing information.
- **verbose** (bool, optional) – If True, then information is printed during training.

Returns **current_shapes** (list of menpo.shape.PointCloud) – The list of current shapes that correspond to the images.

**project** (*J*)
Projects a given vector or matrix onto the appearance subspace.

Type **ndarray**

**run** (*image, initial_shape, gt_shape=None, return_costs=False, **kwargs*)
Run the algorithm to an image given an initial shape.

Parameters

- **image** (menpo.image.Image or subclass) – The image to be fitted.
- **initial_shape** (menpo.shape.PointCloud) – The initial shape from which the fitting procedure will start.
- **gt_shape** (menpo.shape.PointCloud or None, optional) – The ground truth shape associated to the image.
- **return_costs** (bool, optional) – If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned **fitting_result**. Note that this argument currently has no effect and will raise a warning if set to ‘True’. This is because it is not possible to evaluate the cost function of this algorithm.

Returns **fitting_result** (AAMAlgorithmResult) – The parametric iterative fitting result.

**train** (*images, gt_shapes, current_shapes, prefix='', verbose=False*)
Method to train the model given a set of initial shapes.

Parameters

- **images** (list of menpo.image.Image) – The list of training images.
- **gt_shapes** (list of menpo.shape.PointCloud) – The list of ground truth shapes that correspond to the images.
- **current_shapes** (list of menpo.shape.PointCloud) – The list of current shapes that correspond to the images, which will be used as initial shapes.
- **prefix** (str, optional) – The prefix to use when printing information.
- **verbose** (bool, optional) – If True, then information is printed during training.

Returns **current_shapes** (list of menpo.shape.PointCloud) – The list of current shapes that correspond to the images.

**property appearance_model**
Returns the appearance model of the AAM.
Type `menpo.model.PCAModel`

**property transform**
Returns the model driven differential transform object of the AAM, e.g. `DifferentiablePiecewiseAffine` or `DifferentiableThinPlateSplines`.

Type subclass of `DL` and `DX`

### MeanTemplateNewton

class `menpofit.aam.MeanTemplateNewton`(aam_interface, n_iterations=3, compute_error=<function euclidean_bb_normalised_error>, alpha=0, bias=True)

Bases: `MeanTemplate`

Class for training a cascaded-regression Newton algorithm using Incremental Regularized Linear Regression (IRLRegression) given a trained AAM model. The algorithm uses the centered appearance vectors as features in the regression.

**Parameters**

- **aam_interface** (The AAM interface class from `menpofit.aam.algorithm.lk`) – Existing interfaces include:

<table>
<thead>
<tr>
<th>Interface Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'LucasKanadeStandardInterface'</td>
<td>Suitable for holistic AAMs</td>
</tr>
<tr>
<td>'LucasKanadeLinearInterface'</td>
<td>Suitable for linear AAMs</td>
</tr>
<tr>
<td>'LucasKanadePatchInterface'</td>
<td>Suitable for patch-based AAMs</td>
</tr>
</tbody>
</table>

- **n_iterations** (int, optional) – The number of iterations (cascades).

- **compute_error** (callable, optional) – The function to be used for computing the fitting error when training each cascade.

- **alpha** (float, optional) – The regularization parameter.

- **bias** (bool, optional) – Flag that controls whether to use a bias term.

**increment** (images, gt_shapes, current_shapes, prefix=", verbose=False)

Method to increment the model with the set of current shapes.

**Parameters**

- **images** (list of `menpo.image.Image`) – The list of training images.

- **gt_shapes** (list of `menpo.shape.PointCloud`) – The list of ground truth shapes that correspond to the images.

- **current_shapes** (list of `menpo.shape.PointCloud`) – The list of current shapes that correspond to the images.

- **prefix** (str, optional) – The prefix to use when printing information.

- **verbose** (bool, optional) – If True, then information is printed during training.

**Returns** current_shapes (list of `menpo.shape.PointCloud`) – The list of current shapes that correspond to the images.

**run** (image, initial_shape, gt_shape=None, return_costs=False, **kwargs)

Run the algorithm to an image given an initial shape.

**Parameters**
• **image** (*menpo.image.Image* or subclass) – The image to be fitted.

• **initial_shape** (*menpo.shape.PointCloud*) – The initial shape from which the fitting procedure will start.

• **gt_shape** (*menpo.shape.PointCloud* or *None*, optional) – The ground truth shape associated to the image.

• **return_costs** (*bool*, optional) – If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that this argument currently has no effect and will raise a warning if set to ‘True’. This is because it is not possible to evaluate the cost function of this algorithm.

**Returns** fitting_result (*AAMAlgorithmResult*) – The parametric iterative fitting result.

**train** (*images*, *gt_shapes*, *current_shapes*, *prefix=", verbose=False*)

Method to train the model given a set of initial shapes.

**Parameters**

• **images** (*list of menpo.image.Image*) – The list of training images.

• **gt_shapes** (*list of menpo.shape.PointCloud*) – The list of ground truth shapes that correspond to the images.

• **current_shapes** (*list of menpo.shape.PointCloud*) – The list of current shapes that correspond to the images, which will be used as initial shapes.

• **prefix** (*str*, optional) – The prefix to use when printing information.

• **verbose** (*bool*, optional) – If True, then information is printed during training.

**Returns** current_shapes (*list of menpo.shape.PointCloud*) – The list of current shapes that correspond to the images.

**property appearance_model**

Returns the appearance model of the AAM.

**Type** *menpo.model.PCAModel*

**property transform**

Returns the model driven differential transform object of the AAM, e.g. *DifferentiablePiecewiseAffine* or *DifferentiableThinPlateSplines*.

**Type** *subclass of DL and DX*

**MeanTemplateGaussNewton**

**Class** *menpofit.aam.MeanTemplateGaussNewton* (*aam_interface*,

• **compute_error** = *<function euclidean_bb_normalised_error>*),

• **alpha** = *0*,

• **alpha2** = *0*,

• **bias** = *True*)

**Bases:** *MeanTemplate*

Class for training a cascaded-regression Gauss-Newton algorithm using Indirect Incremental Regularized Linear Regression (*IIRLRegression*) given a trained AAM model. The algorithm uses the centered appearance vectors as features in the regression.

**Parameters**

• **aam_interface** (The AAM interface class from *menpofit.aam.algorithm.lk*) – Existing interfaces include:
• **n_iterations** (*int*, optional) – The number of iterations (cascades).

• **compute_error** (*callable*, optional) – The function to be used for computing the fitting error when training each cascade.

• **alpha** (*float*, optional) – The regularization parameter.

• **alpha2** (*float*, optional) – The regularization parameter of the Hessian matrix.

• **bias** (*bool*, optional) – Flag that controls whether to use a bias term.

**increment** (*images, gt_shapes, current_shapes, prefix=”, verbose=False*)
Method to increment the model with the set of current shapes.

**Parameters**

• **images** (*list of menpo.image.Image*) – The list of training images.

• **gt_shapes** (*list of menpo.shape.PointCloud*) – The list of ground truth shapes that correspond to the images.

• **current_shapes** (*list of menpo.shape.PointCloud*) – The list of current shapes that correspond to the images.

• **prefix** (*str*, optional) – The prefix to use when printing information.

• **verbose** (*bool*, optional) – If True, then information is printed during training.

**Returns** **current_shapes** (*list of menpo.shape.PointCloud*) – The list of current shapes that correspond to the images.

**run** (*image, initial_shape, gt_shape=None, return_costs=False, **kwargs*)
Run the algorithm to an image given an initial shape.

**Parameters**

• **image** (*menpo.image.Image or subclass*) – The image to be fitted.

• **initial_shape** (*menpo.shape.PointCloud*) – The initial shape from which the fitting procedure will start.

• **gt_shape** (*menpo.shape.PointCloud or None*, optional) – The ground truth shape associated to the image.

• **return_costs** (*bool*, optional) – If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that this argument currently has no effect and will raise a warning if set to “True”. This is because it is not possible to evaluate the cost function of this algorithm.

**Returns** **fitting_result** (*AAMAlgorithmResult*) – The parametric iterative fitting result.

**train** (*images, gt_shapes, current_shapes, prefix=”, verbose=False*)
Method to train the model given a set of initial shapes.

**Parameters**

• **images** (*list of menpo.image.Image*) – The list of training images.
• **gt_shapes** (list of *menpo.shape.PointCloud*) – The list of ground truth shapes that correspond to the images.

• **current_shapes** (list of *menpo.shape<PointCloud]*) – The list of current shapes that correspond to the images, which will be used as initial shapes.

• **prefix** (*str*, optional) – The prefix to use when printing information.

• **verbose** (*bool*, optional) – If True, then information is printed during training.

**Returns** current_shapes (list of *menpo.shape.PointCloud*) – The list of current shapes that correspond to the images.

**property appearance_model**

Returns the appearance model of the AAM.

Type *menpo.model.PCAModel*

**property transform**

Returns the model driven differential transform object of the AAM, e.g. *DifferentiablePiecewiseAffine* or *DifferentiableThinPlateSplines*.

Type *subclass of DL and DX*

### ProjectOutNewton

**class menpofit.aam.ProjectOutNewton**(aam_interface, n_iterations=3, compute_error=<function euclidean_bb_normalised_error>, alpha=0, bias=True)

Bases: ProjectOut

Class for training a cascaded-regression Newton algorithm using Incremental Regularized Linear Regression (IRLRegression) given a trained AAM model. The algorithm uses the projected-out appearance vectors as features in the regression.

**Parameters**

• **aam_interface** (The AAM interface class from *menpofit.aam.algorithm.lk.*) – Existing interfaces include:

<table>
<thead>
<tr>
<th>Class</th>
<th>AAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>‘LucasKanadeStandardInterface’</td>
<td>Suitable for holistic AAMs</td>
</tr>
<tr>
<td>‘LucasKanadeLinearInterface’</td>
<td>Suitable for linear AAMs</td>
</tr>
<tr>
<td>‘LucasKanadePatchInterface’</td>
<td>Suitable for patch-based AAMs</td>
</tr>
</tbody>
</table>

• **n_iterations** (*int*, optional) – The number of iterations (cascades).

• **compute_error** (*callable*, optional) – The function to be used for computing the fitting error when training each cascade.

• **alpha** (*float*, optional) – The regularization parameter.

• **bias** (*bool*, optional) – Flag that controls whether to use a bias term.

**increment**(images, gt_shapes, current_shapes, prefix=",", verbose=False)

Method to increment the model with the set of current shapes.

**Parameters**

• **images** (list of *menpo.image.Image*) – The list of training images.
• **gt_shapes** *(list of menpo.shape.PointCloud)* – The list of ground truth shapes that correspond to the images.

• **current_shapes** *(list of menpo.shape.PointCloud)* – The list of current shapes that correspond to the images.

• **prefix** *(str, optional)* – The prefix to use when printing information.

• **verbose** *(bool, optional)* – If True, then information is printed during training.

**Returns** **current_shapes** *(list of menpo.shape.PointCloud)* – The list of current shapes that correspond to the images.

**project_out** *(J)*

Projects-out the appearance subspace from a given vector or matrix.

**Type** *ndarray*

**run** *(image, initial_shape, gt_shape=None, return_costs=False, **kwargs)*

Run the algorithm to an image given an initial shape.

**Parameters**

• **image** *(menpo.image.Image or subclass)* – The image to be fitted.

• **initial_shape** *(menpo.shape.PointCloud)* – The initial shape from which the fitting procedure will start.

• **gt_shape** *(menpo.shape.PointCloud or None, optional)* – The ground truth shape associated to the image.

• **return_costs** *(bool, optional)* – If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. *Note that this argument currently has no effect and will raise a warning if set to ‘‘True‘*. This is because it is not possible to evaluate the cost function of this algorithm.

**Returns** **fitting_result** *(AAMAlgorithmResult)* – The parametric iterative fitting result.

**train** *(images, gt_shapes, current_shapes, prefix=”, verbose=False)*

Method to train the model given a set of initial shapes.

**Parameters**

• **images** *(list of menpo.image.Image)* – The list of training images.

• **gt_shapes** *(list of menpo.shape.PointCloud)* – The list of ground truth shapes that correspond to the images.

• **current_shapes** *(list of menpo.shape.PointCloud)* – The list of current shapes that correspond to the images, which will be used as initial shapes.

• **prefix** *(str, optional)* – The prefix to use when printing information.

• **verbose** *(bool, optional)* – If True, then information is printed during training.

**Returns** **current_shapes** *(list of menpo.shape.PointCloud)* – The list of current shapes that correspond to the images.

**property appearance_model**

Returns the appearance model of the AAM.

**Type** *menpo.model.PCAModel*
property transform
Returns the model driven differential transform object of the AAM, e.g. DifferentiablePiecewiseAffine or DifferentiableThinPlateSplines.

Type subclass of DL and DX

ProjectOutGaussNewton
class menpofit.aam.ProjectOutGaussNewton(aam_interface, n_iterations=3,
compute_error=\texttt{euclidean_bb_normalised_error}, alpha=0,
alpha2=0, bias=True)

Bases: ProjectOut

Class for training a cascaded-regression Gauss-Newton algorithm using Indirect Incremental Regularized Linear Regression (IIRLRegression) given a trained AAM model. The algorithm uses the projected-out appearance vectors as features in the regression.

Parameters

• aam_interface (The AAM interface class from menpofit.aam.algorithm.lk.) – Existing interfaces include:

<table>
<thead>
<tr>
<th>Interface</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'LucasKanadeStandardInterface'</td>
<td>Suitable for holistic AAMs</td>
</tr>
<tr>
<td>'LucasKanadeLinearInterface'</td>
<td>Suitable for linear AAMs</td>
</tr>
<tr>
<td>'LucasKanadePatchInterface'</td>
<td>Suitable for patch-based AAMs</td>
</tr>
</tbody>
</table>

• n_iterations (int, optional) – The number of iterations (cascades).

• compute_error (callable, optional) – The function to be used for computing the fitting error when training each cascade.

• alpha (float, optional) – The regularization parameter.

• alpha2 (float, optional) – The regularization parameter of the Hessian matrix.

• bias (bool, optional) – Flag that controls whether to use a bias term.

increment(images, gt_shapes, current_shapes, prefix=", verbose=False)

Method to increment the model with the set of current shapes.

Parameters

• images (list of menpo.image.Image) – The list of training images.

• gt_shapes (list of menpo.shape.PointCloud) – The list of ground truth shapes that correspond to the images.

• current_shapes (list of menpo.shape.PointCloud) – The list of current shapes that correspond to the images.

• prefix (str, optional) – The prefix to use when printing information.

• verbose (bool, optional) – If True, then information is printed during training.

Returns current_shapes (list of menpo.shape.PointCloud) – The list of current shapes that correspond to the images.

project_out(J)
Projects-out the appearance subspace from a given vector or matrix.

Type ndarray

2.1. Deformable Models
run\((\text{image, initial\_shape, gt\_shape}=\text{None, return\_costs}=\text{False, **kwargs})\)

Run the algorithm to an image given an initial shape.

**Parameters**

- **image** (menpo.image.Image or subclass) – The image to be fitted.
- **initial\_shape** (menpo.shape.PointCloud) – The initial shape from which the fitting procedure will start.
- **gt\_shape** (menpo.shape.PointCloud or None, optional) – The ground truth shape associated to the image.
- **return\_costs** (bool, optional) – If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting\_result. Note that this argument currently has no effect and will raise a warning if set to ‘True’. This is because it is not possible to evaluate the cost function of this algorithm.

**Returns** fitting\_result (AAMAlgorithmResult) – The parametric iterative fitting result.

train\((\text{images, gt\_shapes, current\_shapes, prefix}=\text{”, verbose}=\text{False})\)

Method to train the model given a set of initial shapes.

**Parameters**

- **images** (list of menpo.image.Image) – The list of training images.
- **gt\_shapes** (list of menpo.shape.PointCloud) – The list of ground truth shapes that correspond to the images.
- **current\_shapes** (list of menpo.shape.PointCloud) – The list of current shapes that correspond to the images, which will be used as initial shapes.
- **prefix** (str, optional) – The prefix to use when printing information.
- **verbose** (bool, optional) – If True, then information is printed during training.

**Returns** current\_shapes (list of menpo.shape.PointCloud) – The list of current shapes that correspond to the images.

**property appearance\_model**

Returns the appearance model of the AAM.

**Type** menpo.model.PCAModel

**property transform**

Returns the model driven differential transform object of the AAM, e.g. DifferentiablePiecewiseAffine or DifferentiableThinPlateSplines.

**Type** subclass of DL and DX

### Fitting Result

**AAMResult**

**class** menpofit.aam.result.AAMResult (results, scales, affine\_transforms, scale\_transforms, image=\text{None, gt\_shape}=\text{None})

**Bases:** MultiScaleParametricIterativeResult

Class for storing the multi-scale iterative fitting result of an AAM. It holds the shapes, shape parameters, appearance parameters and costs per iteration.
When using a method with a parametric shape model, the first step is to **reconstruct the initial shape** using the shape model. The generated reconstructed shape is then used as initialisation for the iterative optimisation. This step is not counted in the number of iterations.

### Parameters

- **results** (list of `AAMAlgorithmResult`) – The list of optimization results per scale.
- **scales** (list or tuple) – The list of scale values per scale (low to high).
- **affine_transforms** (list of `menpo.transform.Affine`) – The list of affine transforms per scale that transform the shapes into the original image space.
- **scale_transforms** (list of `menpo.shape.Scale`) – The list of scaling transforms per scale.
- **image** (`menpo.image.Image` or subclass or `None`, optional) – The image on which the fitting process was applied. Note that a copy of the image will be assigned as an attribute. If `None`, then no image is assigned.
- **gt_shape** (`menpo.shape.PointCloud` or `None`, optional) – The ground truth shape associated with the image. If `None`, then no ground truth shape is assigned.

#### displacements()

A list containing the displacement between the shape of each iteration and the shape of the previous one.

**Type** list of `ndarray`

#### displacements_stats(stat_type='mean')

A list containing a statistical metric on the displacements between the shape of each iteration and the shape of the previous one.

- **Parameters**
  - `stat_type` ({'mean', 'median', 'min', 'max'}, optional) – Specifies a statistic metric to be extracted from the displacements.

- **Returns**
  - `displacements_stat` (list of float) – The statistical metric on the points displacements for each iteration.

- **Raises**
  - `ValueError` – type must be ‘mean’, ‘median’, ‘min’ or ‘max’

#### errors(compute_error=None)

Returns a list containing the error at each fitting iteration, if the ground truth shape exists.

- **Parameters**
  - `compute_error` (callable, optional) – Callable that computes the error between the shape at each iteration and the ground truth shape.

- **Returns**
  - `errors` (list of float) – The error at each iteration of the fitting process.

- **Raises**
  - `ValueError` – Ground truth shape has not been set, so the final error cannot be computed

#### final_error(compute_error=None)

Returns the final error of the fitting process, if the ground truth shape exists. This is the error computed based on the `final_shape`.

- **Parameters**
  - `compute_error` (callable, optional) – Callable that computes the error between the fitted and ground truth shapes.

- **Returns**
  - `final_error` (float) – The final error at the end of the fitting process.
Raises `ValueError` – Ground truth shape has not been set, so the final error cannot be computed

`initial_error` (**compute_error=None**)

Returns the initial error of the fitting process, if the ground truth shape and initial shape exist. This is the error computed based on the `initial_shape`.

**Parameters** `compute_error` (**callable, optional**) – Callable that computes the error between the initial and ground truth shapes.

**Returns** `initial_error` (**float**) – The initial error at the beginning of the fitting process.

Raises

- `ValueError` – Initial shape has not been set, so the initial error cannot be computed
- `ValueError` – Ground truth shape has not been set, so the initial error cannot be computed

`plot_costs` (**figure_id=None**, **new_figure=False**, **render_lines=True**, **line_colour='b'**, **line_style='-'**, **line_width=2**, **render_markers=True**, **marker_style='o'**, **marker_size=4**, **marker_face_colour='b'**, **marker_edge_colour='k'**, **marker_edge_width=1.0**, **render_axes=True**, **axes_font_name='sans-serif'**, **axes_font_size=10**, **axes_font_style='normal'**, **axes_font_weight='normal'**, **axes_x_limits=0.0**, **axes_y_limits=None**, **axes_x_ticks=None**, **axes_y_ticks=None**, **figure_size=(10, 6)**, **render_grid=True**, **grid_line_style='–'**, **grid_line_width=0.5**)

Plot of the cost function evolution at each fitting iteration.

**Parameters**

- `figure_id` (**object, optional**) – The id of the figure to be used.
- `new_figure` (**bool, optional**) – If True, a new figure is created.
- `render_lines` (**bool, optional**) – If True, the line will be rendered.
- `line_colour` (**colour or None, optional**) – The colour of the line. If None, the colour is sampled from the jet colormap. Example `colour` options are

  ```python
  ['r', 'g', 'b', 'c', 'm', 'k', 'w']
  or (3, ) ndarray
  ```

- `line_style` (**{'-', '--', '-.', ':'}, optional**) – The style of the lines.
- `line_width` (**float, optional**) – The width of the lines.
- `render_markers` (**bool, optional**) – If True, the markers will be rendered.
- `marker_style` (**marker, optional**) – The style of the markers. Example `marker` options

  ```python
  ['.', ',', 'o', 'v', '^', '<', '>', '+', 'x', 'D', 'd', 's', 'p', '*', 'h', 'H', '1', '2', '3', '4', '8']
  ```

- `marker_size` (**int, optional**) – The size of the markers in points.
- `marker_face_colour` (**colour or None, optional**) – The face (filling) colour of the markers. If None, the colour is sampled from the jet colormap. Example `colour` options are

  ```python
  ['r', 'g', 'b', 'c', 'm', 'k', 'w']
  or (3, ) ndarray
  ```
MenpoFit Documentation, Release 0.6.1

- **marker_edge_colour** *(colour or None, optional)* – The edge colour of the markers. If None, the colour is sampled from the jet colormap. Example **colour** options are

  ```python
  ['r', 'g', 'b', 'c', 'm', 'k', 'w']
  or
  (3, ) ndarray
  ```

- **marker_edge_width** *(float, optional)* – The width of the markers’ edge.

- **render_axes** *(bool, optional)* – If True, the axes will be rendered.

- **axes_font_name** *(See below, optional)* – The font of the axes. Example options

  ```python
  {'serif', 'sans-serif', 'cursive', 'fantasy', 'monospace'}
  ```

- **axes_font_size** *(int, optional)* – The font size of the axes.

- **axes_font_style** *(See below, optional)* – The font style of the axes.

  ```python
  {'normal', 'italic', 'oblique'}
  ```

- **axes_font_weight** *(See below, optional)* – The font weight of the axes. Example options

  ```python
  {'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'}
  ```

- **axes_x_limits** *(float or (float, float) or None, optional)* – The limits of the x axis. If float, then it sets padding on the right and left of the graph as a percentage of the curves’ width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.

- **axes_y_limits** *(float or (float, float) or None, optional)* – The limits of the y axis. If float, then it sets padding on the top and bottom of the graph as a percentage of the curves’ height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.

- **axes_x_ticks** *(list or tuple or None, optional)* – The ticks of the x axis.

- **axes_y_ticks** *(list or tuple or None, optional)* – The ticks of the y axis.

- **figure_size** *((float, float) or None, optional)* – The size of the figure in inches.

- **render_grid** *(bool, optional)* – If True, the grid will be rendered.

  ```python
  {'-', '--', '-.', ':'}
  ```

- **grid_line_style** *(optional)* – The style of the grid lines.

- **grid_line_width** *(optional)* – The width of the grid lines.

**plot_displacements** *(stat_type='mean', figure_id=None, new_figure=False, render_lines=True, line_colour='b', line_style='-', line_width=2, render_markers=True, marker_style='o', marker_size=4, marker_face_colour='b', marker_edge_colour='k', marker_edge_width=1.0, render_axes=True, axes_font_name='sans-serif', axes_font_size=10, axes_font_style='normal', axes_font_weight='normal', axes_x_limits=0.0, axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None, figure_size=(10, 6), render_grid=True, grid_line_style='-', grid_line_width=0.5)

Plot of a statistical metric of the displacement between the shape of each iteration and the shape of the
Parameters

- **stat_type** ([mean, median, min, max], optional) – Specifies a statistic metric to be extracted from the displacements (see also displacements_stats() method).
- **figure_id** (object, optional) – The id of the figure to be used.
- **new_figure** (bool, optional) – If True, a new figure is created.
- **render_lines** (bool, optional) – If True, the line will be rendered.
- **line_colour** (colour or None (See below), optional) – The colour of the line. If None, the colour is sampled from the jet colormap. Example colour options are

```python
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- **line_style** (str (See below), optional) – The style of the lines. Example options:

```python
{-, --, -, :, }
```

- **line_width** (float, optional) – The width of the lines.
- **render_markers** (bool, optional) – If True, the markers will be rendered.
- **marker_style** (str (See below), optional) – The style of the markers. Example marker options

```python
{., ,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
```

- **marker_size** (int, optional) – The size of the markers in points.
- **marker_face_colour** (colour or None, optional) – The face (filling) colour of the markers. If None, the colour is sampled from the jet colormap. Example colour options are

```python
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- **marker_edge_colour** (colour or None, optional) – The edge colour of the markers. If None, the colour is sampled from the jet colormap. Example colour options are

```python
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- **marker_edge_width** (float, optional) – The width of the markers’ edge.
- **render_axes** (bool, optional) – If True, the axes will be rendered.
- **axes_font_name** (str (See below), optional) – The font of the axes. Example options

```python
{serif, sans-serif, cursive, fantasy, monospace}
```

- **axes_font_size** (int, optional) – The font size of the axes.
- **axes_font_style** (str (See below), optional) – The font style of the axes. Example options
• **axes_font_weight** *(str (See below), optional)* – The font weight of the axes. Example options:

   \{(ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black)\}

• **axes_x_limits** *(float or (float, float) or None, optional)* – The limits of the x axis. If `float`, then it sets padding on the right and left of the graph as a percentage of the curves’ width. If `tuple` or `list`, then it defines the axis limits. If None, then the limits are set automatically.

• **axes_y_limits** *(float or (float, float) or None, optional)* – The limits of the y axis. If `float`, then it sets padding on the top and bottom of the graph as a percentage of the curves’ height. If `tuple` or `list`, then it defines the axis limits. If None, then the limits are set automatically.

• **axes_x_ticks** *(list or tuple or None, optional)* – The ticks of the x axis.

• **axes_y_ticks** *(list or tuple or None, optional)* – The ticks of the y axis.

• **figure_size** *(float or (float, float) or None, optional)* – The size of the figure in inches.

• **render_grid** *(bool, optional)* – If True, the grid will be rendered.

• **grid_line_style** *(\{‘-‘, ‘--’, ‘-.’, ‘:’\}, optional)* – The style of the grid lines.

• **grid_line_width** *(float, optional)* – The width of the grid lines.

Returns *renderer* *(menpo.visualize.GraphPlotter)* – The renderer object.

**plot_errors** *(compute_error=None, figure_id=None, new_figure=False, render_lines=True, line_colour='b', line_style='-', line_width=2, render_markers=True, marker_style='o', marker_size=4, marker_face_colour='b', marker_edge_colour='k', marker_edge_width=1.0, render_axes=True, axes_font_name='sans-serif', axes_font_size=10, axes_font_style='normal', axes_font_weight='normal', axes_x_limits=0.0, axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None, figure_size=(10, 6), render_grid=True, grid_line_style='-', grid_line_width=0.5)*

Plot of the error evolution at each fitting iteration.

**Parameters**

• **compute_error** *(callable, optional)* – Callable that computes the error between the shape at each iteration and the ground truth shape.

• **figure_id** *(object, optional)* – The id of the figure to be used.

• **new_figure** *(bool, optional)* – If True, a new figure is created.

• **render_lines** *(bool, optional)* – If True, the line will be rendered.

• **line_colour** *(colour or None (See below), optional)* – The colour of the line. If None, the colour is sampled from the jet colormap. Example colour options are:

   \{r, g, b, c, m, k, w\}

   **or**

   *(3, ) ndarray*

• **line_style** *(str (See below), optional)* – The style of the lines. Example options:
- line_width (float, optional) – The width of the lines.
- render_markers (bool, optional) – If True, the markers will be rendered.
- marker_style (str (See below), optional) – The style of the markers. Example marker options

\{., --, -., :\}

- marker_size (int, optional) – The size of the markers in points.
- marker_face_colour (colour or None, optional) – The face (filling) colour of the markers. If None, the colour is sampled from the jet colormap. Example colour options are

\{r, g, b, c, m, k, w\}

or

(3, ) ndarray

- marker_edge_colour (colour or None, optional) – The edge colour of the markers. If None, the colour is sampled from the jet colormap. Example colour options are

\{r, g, b, c, m, k, w\}

or

(3, ) ndarray

- marker_edge_width (float, optional) – The width of the markers’ edge.
- render_axes (bool, optional) – If True, the axes will be rendered.
- axes_font_name (str (See below), optional) – The font of the axes. Example options

\{serif, sans-serif, cursive, fantasy, monospace\}

- axes_font_size (int, optional) – The font size of the axes.
- axes_font_style (str (See below), optional) – The font style of the axes. Example options

\{normal, italic, oblique\}

- axes_font_weight (str (See below), optional) – The font weight of the axes. Example options

\{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black\}

- axes_x_limits (float or (float, float) or None, optional) – The limits of the x axis. If float, then it sets padding on the right and left of the graph as a percentage of the curves’ width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.

- axes_y_limits (float or (float, float) or None, optional) – The limits of the y axis. If float, then it sets padding on the top and bottom of the graph as a percentage of the curves’ height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.

- axes_x_ticks (list or tuple or None, optional) – The ticks of the x axis.
• **axes_y_ticks** (*list* or *tuple* or *None*, optional) – The ticks of the y axis.
• **figure_size** (*float*, *float*) or *None*, optional) – The size of the figure in inches.
• **render_grid** (*bool*, optional) – If *True*, the grid will be rendered.
• **grid_line_style** ({'-', '--', '-.', ':'}, optional) – The style of the grid lines.
• **grid_line_width** (*float*, optional) – The width of the grid lines.

**Returns** *renderer* (*menpo.visualize.GraphPlotter*) – The renderer object.

**reconstructed_initial_error** (*compute_error=None*)

Returns the error of the reconstructed initial shape of the fitting process, if the ground truth shape exists. This is the error computed based on the `reconstructed_initial_shapes[0]`.

**Parameters**

* **compute_error** (*callable*, optional) – Callable that computes the error between the reconstructed initial and ground truth shapes.

**Returns** *reconstructed_initial_error* (*float*) – The error that corresponds to the initial shape’s reconstruction.

**Raises** *ValueError* – Ground truth shape has not been set, so the reconstructed initial error cannot be computed.

**to_result** (*pass_image=True, pass_initial_shape=True, pass_gt_shape=True*)

Returns a *Result* instance of the object, i.e. a fitting result object that does not store the iterations. This can be useful for reducing the size of saved fitting results.

**Parameters**

• **pass_image** (*bool*, optional) – If *True*, then the image will get passed (if it exists).
• **pass_initial_shape** (*bool*, optional) – If *True*, then the initial shape will get passed (if it exists).
• **pass_gt_shape** (*bool*, optional) – If *True*, then the ground truth shape will get passed (if it exists).

**Returns** *result* (*Result*) – The final “lightweight” fitting result.

**view** (*figure_id=None, new_figure=False, render_image=True, render_final_shape=True, render_initial_shape=False, render_gt_shape=False, subplots_enabled=True, channels=None, interpolation='bilinear', cmap_name=None, alpha=1.0, masked=True, final_marker_face_colour='r', final_marker_edge_colour='k', final_line_colour='r', initial_marker_face_colour='b', initial_marker_edge_colour='k', initial_line_colour='b', gt_marker_face_colour='y', gt_marker_edge_colour='k', gt_line_colour='y', render_lines=True, line_style='-', line_width=2, render_markers=True, marker_style='o', marker_size=4, marker_edge_width=1.0, render_numbering=False, numbers_horizontal_align='center', numbers_vertical_align='bottom', numbers_font_name='sans-serif', numbers_font_size=10, numbers_font_style='normal', numbers_font_weight='normal', render_legend=True, legend_title='', legend_font_name='sans-serif', legend_font_style='normal', legend_font_size=10, legend_font_weight='normal', legend_marker_scale=None, legend_location=2, legend_bbox_to_anchor=(1.05, 1.0), legend_border_axes_pad=None, legend_n_columns=1, legend_horizontal_spacing=None, legend_vertical_spacing=None, legend_border=True, legend_border_padding=None, legend_shadow=False, legend_rounded_corners=False, render_axes=False, axes_font_name='sans-serif', axes_font_size=10, axes_font_style='normal', axes_font_weight='normal', axes_x_limits=None, axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None, figure_size=(7, 7))

Visualize the fitting result. The method renders the final fitted shape and optionally the initial shape,
ground truth shape and the image, if they were provided.

**Parameters**

- **figure_id** *(object, optional)* – The id of the figure to be used.
- **new_figure** *(bool, optional)* – If True, a new figure is created.
- **render_image** *(bool, optional)* – If True and the image exists, then it gets rendered.
- **render_final_shape** *(bool, optional)* – If True, then the final fitting shape gets rendered.
- **render_initial_shape** *(bool, optional)* – If True and the initial fitting shape exists, then it gets rendered.
- **render_gt_shape** *(bool, optional)* – If True and the ground truth shape exists, then it gets rendered.
- **subplots_enabled** *(bool, optional)* – If True, then the requested final, initial and ground truth shapes get rendered on separate subplots.
- **channels** *(int or list of int or all or None)* – If int or list of int, the specified channel(s) will be rendered. If all, all the channels will be rendered in subplots. If None and the image is RGB, it will be rendered in RGB mode. If None and the image is not RGB, it is equivalent to all.
- **interpolation** *(See Below, optional)* – The interpolation used to render the image. For example, if bilinear, the image will be smooth and if nearest, the image will be pixelated. Example options

```python
{none, nearest, bilinear, bicubic, spline16, spline36, hanning, hamming, hermite, kaiser, catrom, gaussian, bessel, mitchell, sinc, lanczos}
```

- **cmap_name** *(str, optional)* – If None, single channel and three channel images default to greyscale and rgb colormaps respectively.
- **alpha** *(float, optional)* – The alpha blending value, between 0 (transparent) and 1 (opaque).
- **masked** *(bool, optional)* – If True, then the image is rendered as masked.
- **final_marker_face_colour** *(See Below, optional)* – The face (filling) colour of the markers of the final fitting shape. Example options

```python
[r, g, b, c, m, k, w]
or
(3, ) ndarray
```

- **final_marker_edge_colour** *(See Below, optional)* – The edge colour of the markers of the final fitting shape. Example options

```python
[r, g, b, c, m, k, w]
or
(3, ) ndarray
```

- **final_line_colour** *(See Below, optional)* – The line colour of the final fitting shape. Example options
• **initial_marker_face_colour** *(See Below, optional)* – The face (filling) colour of the markers of the initial shape. Example options

```python
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• **initial_marker_edge_colour** *(See Below, optional)* – The edge colour of the markers of the initial shape. Example options

```python
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• **initial_line_colour** *(See Below, optional)* – The line colour of the initial shape. Example options

```python
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• **gt_marker_face_colour** *(See Below, optional)* – The face (filling) colour of the markers of the ground truth shape. Example options

```python
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• **gt_marker_edge_colour** *(See Below, optional)* – The edge colour of the markers of the ground truth shape. Example options

```python
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• **gt_line_colour** *(See Below, optional)* – The line colour of the ground truth shape. Example options

```python
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• **render_lines** *(bool or list of bool, optional)* – If True, the lines will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per shape in `(final, initial, groundtruth)` order.

• **line_style** *(str or list of str, optional)* – The style of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per shape in `(final, initial, groundtruth)` order. Example options:

```python
[‘-‘, ‘--‘, ‘-.’, ‘:’]
```
- **line_width** *(float or list of float, optional)* – The width of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per shape in *(final, initial, groundtruth)* order.

- **render_markers** *(bool or list of bool, optional)* – If True, the markers will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per shape in *(final, initial, groundtruth)* order.

- **marker_style** *(str or list of str, optional)* – The style of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per shape in *(final, initial, groundtruth)* order. Example options:

  ```
  {.,,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
  ```

- **marker_size** *(int or list of int, optional)* – The size of the markers in points. You can either provide a single value that will be used for all shapes or a list with a different value per shape in *(final, initial, groundtruth)* order.

- **marker_edge_width** *(float or list of float, optional)* – The width of the markers’ edge. You can either provide a single value that will be used for all shapes or a list with a different value per shape in *(final, initial, groundtruth)* order.

- **render_numbering** *(bool, optional)* – If True, the landmarks will be numbered.

- **numbers_horizontal_align** *(center, right, left, optional)* – The horizontal alignment of the numbers’ texts.

- **numbers_vertical_align** *(center, top, bottom, baseline, optional)* – The vertical alignment of the numbers’ texts.

- **numbers_font_name** *(See Below, optional)* – The font of the numbers. Example options

  ```
  {serif, sans-serif, cursive, fantasy, monospace}
  ```

- **numbers_font_size** *(int, optional)* – The font size of the numbers.

- **numbers_font_style** *(normal, italic, oblique, optional)* – The font style of the numbers.

- **numbers_font_weight** *(See Below, optional)* – The font weight of the numbers. Example options

  ```
  {ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
  ```

- **numbers_font_colour** *(See Below, optional)* – The font colour of the numbers. Example options

  ```
  {r, g, b, c, m, k, w} 
  or 
  (3, ) ndarray
  ```

- **render_legend** *(bool, optional)* – If True, the legend will be rendered.

- **legend_title** *(str, optional)* – The title of the legend.

- **legend_font_name** *(See below, optional)* – The font of the legend. Example options
• `legend_font_style ((normal, italic, oblique), optional)` – The font style of the legend.

• `legend_font_size (int, optional)` – The font size of the legend.

• `legend_font_weight (See Below, optional)` – The font weight of the legend. Example options

{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}

• `legend_marker_scale (float, optional)` – The relative size of the legend markers with respect to the original

• `legend_location (int, optional)` – The location of the legend. The predefined values are:

<table>
<thead>
<tr>
<th>Legend Location</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>best</td>
<td>0</td>
</tr>
<tr>
<td>upper right</td>
<td>1</td>
</tr>
<tr>
<td>upper left</td>
<td>2</td>
</tr>
<tr>
<td>lower left</td>
<td>3</td>
</tr>
<tr>
<td>lower right</td>
<td>4</td>
</tr>
<tr>
<td>right</td>
<td>5</td>
</tr>
<tr>
<td>center left</td>
<td>6</td>
</tr>
<tr>
<td>center right</td>
<td>7</td>
</tr>
<tr>
<td>lower center</td>
<td>8</td>
</tr>
<tr>
<td>upper center</td>
<td>9</td>
</tr>
<tr>
<td>center</td>
<td>10</td>
</tr>
</tbody>
</table>

• `legend_bbox_to_anchor ((float, float) tuple, optional)` – The bbox that the legend will be anchored.

• `legend_border_axes_pad (float, optional)` – The pad between the axes and legend border.

• `legend_n_columns (int, optional)` – The number of the legend’s columns.

• `legend_horizontal_spacing (float, optional)` – The spacing between the columns.

• `legend_vertical_spacing (float, optional)` – The vertical space between the legend entries.

• `legend_border (bool, optional)` – If True, a frame will be drawn around the legend.

• `legend_border_padding (float, optional)` – The fractional whitespace inside the legend border.

• `legend_shadow (bool, optional)` – If True, a shadow will be drawn behind legend.

• `legend_rounded_corners (bool, optional)` – If True, the frame’s corners will be rounded (fancybox).

• `render_axes (bool, optional)` – If True, the axes will be rendered.

• `axes_font_name (See Below, optional)` – The font of the axes. Example options
• **axes_font_size** *(int, optional)* – The font size of the axes.

• **axes_font_style** *(normal, italic, oblique, optional)* – The font style of the axes.

• **axes_font_weight** *(See Below, optional)* – The font weight of the axes. Example options

  (ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black)

• **axes_x_limits** *(float or (float, float) or None, optional)* – The limits of the x axis. If float, then it sets padding on the right and left of the Image as a percentage of the Image’s width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.

• **axes_y_limits** *(float, float) tuple or None, optional)* – The limits of the y axis. If float, then it sets padding on the top and bottom of the Image as a percentage of the Image’s height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.

• **axes_x_ticks** *(list or tuple or None, optional)* – The ticks of the x axis.

• **axes_y_ticks** *(list or tuple or None, optional)* – The ticks of the y axis.

• **figure_size** *(float, float) tuple or None optional* – The size of the figure in inches.

  Returns **renderer** *(class)* – The renderer object.

  **view_iterations**(figure_id=None, new_figure=False, iters=None, render_image=True, subplots_enabled=False, channels=None, interpolation='bilinear', cmap_name=None, alpha=1.0, masked=True, render_lines=True, line_style='-', line_width=2, line_colour=None, render_markers=True, marker_edge_colour=None, marker_face_colour=None, marker_style='o', marker_size=4, marker_edge_width=1.0, render_numbering=False, numbers_horizontal_align='center', numbers_vertical_align='bottom', numbers_font_name='sans-serif', numbers_font_size=10, numbers_font_style='normal', numbers_font_weight='normal', numbers_font_colour='k', render_legend=True, legend_title='', legend_font_name='sans-serif', legend_font_style='normal', legend_font_weight='normal', legend_font_size=10, legend_location=2, legend_bbox_to_anchor=(1.05, 1.0), legend_border_axes_pad=None, legend_n_columns=1, legend_horizontal_spacing=None, legend_vertical_spacing=None, legend_border=True, legend_border_padding=None, legend_shadow=False, legend_rounded_corners=False, render_axes=False, axes_font_name='sans-serif', axes_font_size=10, axes_font_style='normal', axes_font_weight='normal', axes_x_limits=None, axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None, figure_size=(7, 7))

  Visualize the iterations of the fitting process.

  **Parameters**

  • **figure_id** *(object, optional)* – The id of the figure to be used.

  • **new_figure** *(bool, optional)* – If True, a new figure is created.
• **iters** (*int or list of int or None*, optional) – The iterations to be visualized. If `None`, then all the iterations are rendered.

<table>
<thead>
<tr>
<th>No.</th>
<th>Visualised shape</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td><code>self.initial_shape</code></td>
<td>Initial shape</td>
</tr>
<tr>
<td>1</td>
<td><code>self.reconstructed_initial_shape</code></td>
<td>Reconstructed initial</td>
</tr>
<tr>
<td>2</td>
<td><code>self.shapes[2]</code></td>
<td>Iteration 1</td>
</tr>
<tr>
<td>i</td>
<td><code>self.shapes[i]</code></td>
<td>Iteration i-1</td>
</tr>
<tr>
<td>n_iters+1</td>
<td><code>self.final_shape</code></td>
<td>Final shape</td>
</tr>
</tbody>
</table>

• **render_image** (*bool*, optional) – If `True` and the image exists, then it gets rendered.

• **subplots_enabled** (*bool*, optional) – If `True`, then the requested final, initial and ground truth shapes get rendered on separate subplots.

• **channels** (*int or list of int or all or None*) – If `int or list of int`, the specified channel(s) will be rendered. If `all`, all the channels will be rendered in subplots. If `None` and the image is RGB, it will be rendered in RGB mode. If `None` and the image is not RGB, it is equivalent to `all`.

• **interpolation** (*str* (See Below), optional) – The interpolation used to render the image. For example, if `bilinear`, the image will be smooth and if `nearest`, the image will be pixelated. Example options

```
{none, nearest, bilinear, bicubic, spline16, spline36, hanning, hamming, hermite, kaiser, quadratic, catrom, gaussian, bessel, mitchell, sinc, lanczos}
```

• **cmap_name** (*str*, optional) – If `None`, single channel and three channel images default to greyscale and rgb colormaps respectively.

• **alpha** (*float*, optional) – The alpha blending value, between 0 (transparent) and 1 (opaque).

• **masked** (*bool*, optional) – If `True`, then the image is rendered as masked.

• **render_lines** (*bool or list of bool*, optional) – If `True`, the lines will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.

• **line_style** (*str or list of str* (See below), optional) – The style of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options:

```
{-, --, -, ., :}
```

• **line_width** (*float or list of float*, optional) – The width of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.

• **line_colour** (*colour or list of colour* (See Below), optional) – The colour of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```
{r, g, b, c, m, k, w}
```

or

```
(3, ) ndarray
```
• **render_markers** *(bool or list of bool, optional)* – If True, the markers will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.

• **marker_style** *(str or list of str (See below), optional)* – The style of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```
(., ,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8)
```

• **marker_size** *(int or list of int, optional)* – The size of the markers in points. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.

• **marker_edge_colour** *(colour or list of colour (See Below), optional)* – The edge colour of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```
[r, g, b, c, m, k, w]
or
(3, ) ndarray
```

• **marker_face_colour** *(colour or list of colour (See Below), optional)* – The face (filling) colour of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```
[r, g, b, c, m, k, w]
or
(3, ) ndarray
```

• **marker_edge_width** *(float or list of float, optional)* – The width of the markers’ edge. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.

• **render_numbering** *(bool, optional)* – If True, the landmarks will be numbered.

• **numbers_horizontal_align** *(str (See below), optional)* – The horizontal alignment of the numbers’ texts. Example options

```
{center, right, left}
```

• **numbers_vertical_align** *(str (See below), optional)* – The vertical alignment of the numbers’ texts. Example options

```
{center, top, bottom, baseline}
```

• **numbers_font_name** *(str (See below), optional)* – The font of the numbers. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

• **numbers_font_size** *(int, optional)* – The font size of the numbers.

• **numbers_font_style** *(normal, italic, oblique), optional)* – The font style of the numbers.

• **numbers_font_weight** *(str (See below), optional)* – The font weight of the numbers. Example options
• numbers_font_colour (See Below, optional) – The font colour of the numbers. Example options

{|r, g, b, c, m, k, w| or (3,) ndarray|

• render_legend (bool, optional) – If True, the legend will be rendered.

• legend_title (str, optional) – The title of the legend.

• legend_font_name (See below, optional) – The font of the legend. Example options

|{serif, sans-serif, cursive, fantasy, monospace}|

• legend_font_style (str (See below), optional) – The font style of the legend. Example options

|{normal, italic, oblique}|

• legend_font_size (int, optional) – The font size of the legend.

• legend_font_weight (str (See below), optional) – The font weight of the legend. Example options

|{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}|

• legend_marker_scale (float, optional) – The relative size of the legend markers with respect to the original

• legend_location (int, optional) – The location of the legend. The predefined values are:

|'best' | 0 |
|'upper right' | 1 |
|'upper left' | 2 |
|'lower left' | 3 |
|'lower right' | 4 |
|'right' | 5 |
|'center left' | 6 |
|'center right' | 7 |
|'lower center' | 8 |
|'upper center' | 9 |
|'center' | 10 |

• legend_bbox_to_anchor ((float, float) tuple, optional) – The bbox that the legend will be anchored.

• legend_border_axes_pad (float, optional) – The pad between the axes and legend border.

• legend_n_columns (int, optional) – The number of the legend’s columns.

2.1. Deformable Models
• **legend_horizontal_spacing** *(float, optional)* – The spacing between the columns.

• **legend_vertical_spacing** *(float, optional)* – The vertical space between the legend entries.

• **legend_border** *(bool, optional)* – If True, a frame will be drawn around the legend.

• **legend_border_padding** *(float, optional)* – The fractional whitespace inside the legend border.

• **legend_shadow** *(bool, optional)* – If True, a shadow will be drawn behind legend.

• **legend_rounded_corners** *(bool, optional)* – If True, the frame’s corners will be rounded (fancybox).

• **render_axes** *(bool, optional)* – If True, the axes will be rendered.

• **axes_font_name** *(str (See below), optional)* – The font of the axes. Example options

{serif, sans-serif, cursive, fantasy, monospace}

• **axes_font_size** *(int, optional)* – The font size of the axes.

• **axes_font_style** *( (normal, italic, oblique), optional)* – The font style of the axes.

• **axes_font_weight** *(str (See below), optional)* – The font weight of the axes. Example options

{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}

• **axes_x_limits** *(float or (float, float) or None, optional)* – The limits of the x axis. If float, then it sets padding on the right and left of the Image as a percentage of the Image’s width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.

• **axes_y_limits** *( (float, float) tuple or None, optional)* – The limits of the y axis. If float, then it sets padding on the top and bottom of the Image as a percentage of the Image’s height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.

• **axes_x_ticks** *(list or tuple or None, optional)* – The ticks of the x axis.

• **axes_y_ticks** *(list or tuple or None, optional)* – The ticks of the y axis.

• **figure_size** *( (float, float) tuple or None optional)* – The size of the figure in inches.

Returns **renderer** *(class)* – The renderer object.

**view_widget** *(figure_size=(7, 7))*

Visualizes the result object using an interactive widget.

Parameters **figure_size** *( (int, int), optional)* – The initial size of the rendered figure.

**property appearance_parameters**

Returns the **list** of appearance parameters obtained at each iteration of the fitting process. The list includes the parameters of the **initial_shape** (if it exists) and **final_shape**.

Type **list** of *(n_params,)* *ndarray*

**property costs**

Returns a **list** with the cost per iteration. It returns **None** if the costs are not computed.

Type **list** of *float* or **None**
property final_shape
    Returns the final shape of the fitting process.
    Type menpo.shape.PointCloud

property gt_shape
    Returns the ground truth shape associated with the image. In case there is not an attached ground truth
    shape, then None is returned.
    Type menpo.shape.PointCloud or None

property image
    Returns the image that the fitting was applied on, if it was provided. Otherwise, it returns None.
    Type menpo.shape.Image or subclass or None

property initial_shape
    Returns the initial shape that was provided to the fitting method to initialise the fitting process. In case the
    initial shape does not exist, then None is returned.
    Type menpo.shape.PointCloud or None

property is_iterative
    Flag whether the object is an iterative fitting result.
    Type bool

property n_iters
    Returns the total number of iterations of the fitting process.
    Type int

property n_iters_per_scale
    Returns the number of iterations per scale of the fitting process.
    Type list of int

property n_scales
    Returns the number of scales used during the fitting process.
    Type int

property reconstructed_initial_shapes
    Returns the result of the reconstruction step that takes place at each scale before applying the iterative
    optimisation.
    Type list of menpo.shape.PointCloud

property shape_parameters
    Returns the list of shape parameters obtained at each iteration of the fitting process. The list includes the
    parameters of the initial_shape (if it exists) and final_shape.
    Type list of (n_params,) ndarray

property shapes
    Returns the list of shapes obtained at each iteration of the fitting process. The list includes the initial_shape
    (if it exists) and final_shape.
    Type list of menpo.shape.PointCloud
AAMAlgorithmResult

class menpofit.aam.result.AAMAlgorithmResult(shapes, shape_parameters, appearance_parameters, initial_shape=None, image=None, gt_shape=None, costs=None)

Bases: ParametricIterativeResult

Class for storing the iterative result of an AAM optimisation algorithm.

**Note:** When using a method with a parametric shape model, the first step is to **reconstruct the initial shape** using the shape model. The generated reconstructed shape is then used as initialisation for the iterative optimisation. This step is not counted in the number of iterations.

**Parameters**

- **shapes** *(list of menpo.shape.PointCloud)* – The list of shapes per iteration. The first and last members correspond to the initial and final shapes, respectively.

- **shape_parameters** *(list of (n_shape_parameters,) ndarray)* – The list of shape parameters per iteration. The first and last members correspond to the initial and final shapes, respectively.

- **appearance_parameters** *(list of (n_appearance_parameters,) ndarray)* – The list of appearance parameters per iteration. The first and last members correspond to the initial and final shapes, respectively.

- **initial_shape** *(menpo.shape.PointCloud or None, optional)* – The initial shape from which the fitting process started. If None, then no initial shape is assigned.

- **image** *(menpo.image.Image or subclass or None, optional)* – The image on which the fitting process was applied. Note that a copy of the image will be assigned as an attribute. If None, then no image is assigned.

- **gt_shape** *(menpo.shape.PointCloud or None, optional)* – The ground truth shape associated with the image. If None, then no ground truth shape is assigned.

- **costs** *(list of float or None, optional)* – The list of cost per iteration. If None, then it is assumed that the cost function cannot be computed for the specific algorithm.

**displacements ()**

A list containing the displacement between the shape of each iteration and the shape of the previous one.

**Type** list of ndarray

**displacements_stats** *(stat_type='mean')*

A list containing a statistical metric on the displacements between the shape of each iteration and the shape of the previous one.

**Parameters** stat_type *({'mean', 'median', 'min', 'max'}, optional)* – Specifies a statistic metric to be extracted from the displacements.

**Returns** displacements_stat *(list of float)* – The statistical metric on the points displacements for each iteration.

**Raises** ValueError – type must be ‘mean’, ‘median’, ‘min’ or ‘max’

**errors** *(compute_error=None)*

Returns a list containing the error at each fitting iteration, if the ground truth shape exists.
Parameters `compute_error` *(callable, optional)* – Callable that computes the error between the shape at each iteration and the ground truth shape.

Returns `errors` *(list of float)* – The error at each iteration of the fitting process.

Raises `ValueError` – Ground truth shape has not been set, so the final error cannot be computed.

`final_error` *(compute_error=None)*

Returns the final error of the fitting process, if the ground truth shape exists. This is the error computed based on the `final_shape`.

Parameters `compute_error` *(callable, optional)* – Callable that computes the error between the fitted and ground truth shapes.

Returns `final_error` *(float)* – The final error at the end of the fitting process.

Raises `ValueError` – Ground truth shape has not been set, so the final error cannot be computed.

`initial_error` *(compute_error=None)*

Returns the initial error of the fitting process, if the ground truth shape and initial shape exist. This is the error computed based on the `initial_shape`.

Parameters `compute_error` *(callable, optional)* – Callable that computes the error between the initial and ground truth shapes.

Returns `initial_error` *(float)* – The initial error at the beginning of the fitting process.

Raises

- `ValueError` – Initial shape has not been set, so the initial error cannot be computed
- `ValueError` – Ground truth shape has not been set, so the initial error cannot be computed

`plot_costs` *(figure_id=None, new_figure=False, render_lines=True, line_colour='b', line_style='-', line_width=2, render_markers=True, marker_style='o', marker_size=4, marker_face_colour='b', marker_edge_colour='k', marker_edge_width=1.0, render_axes=True, axes_font_name='sans-serif', axes_font_size=10, axes_font_style='normal', axes_font_weight='normal', axes_x_limits=0.0, axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None, grid_line_style='-', grid_line_width=0.5)*

Plot of the cost function evolution at each fitting iteration.

Parameters

- `figure_id` *(object, optional)* – The id of the figure to be used.
- `new_figure` *(bool, optional)* – If `True`, a new figure is created.
- `render_lines` *(bool, optional)* – If `True`, the line will be rendered.
- `line_colour` *(colour or None, optional)* – The colour of the line. If `None`, the colour is sampled from the jet colormap. Example `colour` options are

```
{'r', 'g', 'b', 'c', 'm', 'k', 'w'}
```

or

```
(3, ) ndarray
```

- `line_style` *(('-', '--', '-', ':'), optional)* – The style of the lines.
- `line_width` *(float, optional)* – The width of the lines.
- `render_markers` *(bool, optional)* – If `True`, the markers will be rendered.
• **marker_style** (marker, optional) – The style of the markers. Example marker options

```
[',', '.', ',', 's', 'o', 'v', '^', '<', '>', '+', 'x', 'D', 'd', '', 'p', '*', 'h', 'H', '1', '2', '3', '4', '8']
```

• **marker_size** (int, optional) – The size of the markers in points.

• **marker_face_colour** (colour or None, optional) – The face (filling) colour of the markers. If None, the colour is sampled from the jet colormap. Example colour options are

```
['r', 'g', 'b', 'c', 'm', 'k', 'w']
```

or

```
(3, ) ndarray
```

• **marker_edge_colour** (colour or None, optional) – The edge colour of the markers. If None, the colour is sampled from the jet colormap. Example colour options are

```
['r', 'g', 'b', 'c', 'm', 'k', 'w']
```

or

```
(3, ) ndarray
```

• **marker_edge_width** (float, optional) – The width of the markers’ edge.

• **render_axes** (bool, optional) – If True, the axes will be rendered.

• **axes_font_name** (See below, optional) – The font of the axes. Example options

```
['serif', 'sans-serif', 'cursive', 'fantasy', 'monospace']
```

• **axes_font_size** (int, optional) – The font size of the axes.

• **axes_font_style** ("normal", 'italic', 'oblique', optional) – The font style of the axes.

• **axes_font_weight** (See below, optional) – The font weight of the axes. Example options

```
['ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black']
```

• **axes_x_limits** (float or (float, float) or None, optional) – The limits of the x axis. If float, then it sets padding on the right and left of the graph as a percentage of the curves’ width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.

• **axes_y_limits** (float or (float, float) or None, optional) – The limits of the y axis. If float, then it sets padding on the top and bottom of the graph as a percentage of the curves’ height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.

• **axes_x_ticks** (list or tuple or None, optional) – The ticks of the x axis.

• **axes_y_ticks** (list or tuple or None, optional) – The ticks of the y axis.

• **figure_size** ((float, float) or None, optional) – The size of the figure in inches.

• **render_grid** (bool, optional) – If True, the grid will be rendered.
• `grid_line_style` (``'-'`, `'--'`, `'.'`, `':'``), optional) – The style of the grid lines.

• `grid_line_width` (``float``), optional) – The width of the grid lines.

Returns `renderer` (``menpo.visualize.GraphPlotter``) – The renderer object.

`plot_displacements` (``stat_type='mean'``, ``figure_id=None``, ``new_figure=False``, ``render_lines=True``, ``line_colour='b'``, ``line_style='-'``, ``line_width=2``, ``render_markers=True``, 
``marker_style='o'``, ``marker_size=4``, ``marker_face_colour='b'``, 
``marker_edge_colour='k'``, ``marker_edge_width=1.0``, 
``render_axes=True``, 
``axes_font_name='sans-serif'``, 
``axes_font_size=10``, 
``axes_font_style='normal'``, 
``axes_font_weight='normal'``, 
``axes_x_limits=0.0``, 
``axes_y_limits=None``, 
``axes_x_ticks=None``, 
``axes_y_ticks=None``, 
``figure_size=(10, 6)``, 
``render_grid=True``, 
``grid_line_style='–'``, 
``grid_line_width=0.5``)

Plot of a statistical metric of the displacement between the shape of each iteration and the shape of the previous one.

Parameters

• `stat_type` (``{mean, median, min, max}``, optional) – Specifies a statistic metric to be extracted from the displacements (see also `displacements_stats()` method).

• `figure_id` (``object``), optional) – The id of the figure to be used.

• `new_figure` (``bool``), optional) – If True, a new figure is created.

• `render_lines` (``bool``), optional) – If True, the line will be rendered.

• `line_colour` (``colour`` or ``None`` (See below), optional) – The colour of the line. If None, the colour is sampled from the jet colormap. Example colour options are

\[{r, g, b, c, m, k, w}\]

or

\[(3, )\] ndarray

• `line_style` (``str`` (See below), optional) – The style of the lines. Example options:

\[-, --, -., :\]

• `line_width` (``float``), optional) – The width of the lines.

• `render_markers` (``bool``), optional) – If True, the markers will be rendered.

• `marker_style` (``str`` (See below), optional) – The style of the markers. Example marker options

\[.\, ,\, ,\, o, v, ^, <, >, ., x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8\]

• `marker_size` (``int``), optional) – The size of the markers in points.

• `marker_face_colour` (``colour`` or ``None`` , optional) – The face (filling) colour of the markers. If None, the colour is sampled from the jet colormap. Example colour options are

\[{r, g, b, c, m, k, w}\]

or

\[(3, )\] ndarray

• `marker_edge_colour` (``colour`` or ``None`` , optional) – The edge colour of the markers. If None, the colour is sampled from the jet colormap. Example colour options are

\[2.1.\ Deformable\ Models\]
\{(r, g, b, c, m, k, w)\} or \{(3, )\} ndarray

- **marker_edge_width** (*float*, optional) – The width of the markers’ edge.
- **render_axes** (*bool*, optional) – If *True*, the axes will be rendered.
- **axes_font_name** (*str* (See below), optional) – The font of the axes. Example options
  \{serif, sans-serif, cursive, fantasy, monospace\}

- **axes_font_size** (*int*, optional) – The font size of the axes.
- **axes_font_style** (*str* (See below), optional) – The font style of the axes. Example options
  \{normal, italic, oblique\}

- **axes_font_weight** (*str* (See below), optional) – The font weight of the axes. Example options
  \{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black\}

- **axes_x_limits** (*float* or (*float*, *float*) or *None*, optional) – The limits of the x axis. If *float*, then it sets padding on the right and left of the graph as a percentage of the curves’ width. If *tuple* or *list*, then it defines the axis limits. If *None*, then the limits are set automatically.

- **axes_y_limits** (*float* or (*float*, *float*) or *None*, optional) – The limits of the y axis. If *float*, then it sets padding on the top and bottom of the graph as a percentage of the curves’ height. If *tuple* or *list*, then it defines the axis limits. If *None*, then the limits are set automatically.

- **axes_x_ticks** (*list* or *tuple* or *None*, optional) – The ticks of the x axis.
- **axes_y_ticks** (*list* or *tuple* or *None*, optional) – The ticks of the y axis.
- **figure_size** (*(*float*, *float*)* or *None*, optional) – The size of the figure in inches.
- **render_grid** (*bool*, optional) – If *True*, the grid will be rendered.

- **grid_line_style** (*\{'-\', '--', '-.', ':', \}'*, optional) – The style of the grid lines.
- **grid_line_width** (*float*, optional) – The width of the grid lines.

**Returns** renderer (*menpo.visualize.GraphPlotter*) – The renderer object.

**plot_errors** (*compute_error=None, figure_id=None, new_figure=False, render_lines=True, line_colour='b', line_style='-', line_width=2, render_markers=True, marker_style='o', marker_size=4, marker_face_colour='b', marker_edge_colour='k', marker_edge_width=1.0, render_axes=True, axes_font_name='sans-serif', axes_font_size=10, axes_font_style='normal', axes_font_weight='normal', axes_x_limits=0.0, axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None, figure_size=(10, 6), render_grid=True, grid_line_style='-', grid_line_width=0.5)

Plot of the error evolution at each fitting iteration.

**Parameters**
- **compute_error** (*callable*, optional) – Callable that computes the error between the shape at each iteration and the ground truth shape.

- **figure_id** (*object*, optional) – The id of the figure to be used.

- **new_figure** (*bool*, optional) – If True, a new figure is created.

- **render_lines** (*bool*, optional) – If True, the line will be rendered.

- **line_colour** (*colour or None* (See below), optional) – The colour of the line. If None, the colour is sampled from the jet colormap. Example *colour* options are

  ```
  {r, g, b, c, m, k, w}
  or
  (3, ) ndarray
  ```

- **line_style** (*str* (See below), optional) – The style of the lines. Example options:

  ```
  {-, --, -., :}
  ```

- **line_width** (*float*, optional) – The width of the lines.

- **render_markers** (*bool*, optional) – If True, the markers will be rendered.

- **marker_style** (*str* (See below), optional) – The style of the markers. Example *marker* options

  ```
  {., ,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
  ```

- **marker_size** (*int*, optional) – The size of the markers in points.

- **marker_face_colour** (*colour or None*, optional) – The face (filling) colour of the markers. If None, the colour is sampled from the jet colormap. Example *colour* options are

  ```
  {r, g, b, c, m, k, w}
  or
  (3, ) ndarray
  ```

- **marker_edge_colour** (*colour or None*, optional) – The edge colour of the markers. If None, the colour is sampled from the jet colormap. Example *colour* options are

  ```
  {r, g, b, c, m, k, w}
  or
  (3, ) ndarray
  ```

- **marker_edge_width** (*float*, optional) – The width of the markers’ edge.

- **render_axes** (*bool*, optional) – If True, the axes will be rendered.

- **axes_font_name** (*str* (See below), optional) – The font of the axes. Example options

  ```
  {serif, sans-serif, cursive, fantasy, monospace}
  ```

- **axes_font_size** (*int*, optional) – The font size of the axes.

- **axes_font_style** (*str* (See below), optional) – The font style of the axes. Example options

  ```
  {normal, italic, oblique}
  ```

### 2.1. Deformable Models
• **axes_font_weight** *(str (See below), optional)* – The font weight of the axes. Example options

```python
(ultralight, light, normal, regular, book, medium, roman,
semibold, demibold, demi, bold, heavy, extra bold, black)
```

• **axes_x_limits** *(float or (float, float) or None, optional)* – The limits of the x axis. If `float`, then it sets padding on the right and left of the graph as a percentage of the curves’ width. If `tuple` or `list`, then it defines the axis limits. If `None`, then the limits are set automatically.

• **axes_y_limits** *(float or (float, float) or None, optional)* – The limits of the y axis. If `float`, then it sets padding on the top and bottom of the graph as a percentage of the curves’ height. If `tuple` or `list`, then it defines the axis limits. If `None`, then the limits are set automatically.

• **axes_x_ticks** *(list or tuple or None, optional)* – The ticks of the x axis.

• **axes_y_ticks** *(list or tuple or None, optional)* – The ticks of the y axis.

• **figure_size** *((float, float) or None, optional)* – The size of the figure in inches.

• **render_grid** *(bool, optional)* – If `True`, the grid will be rendered.

• **grid_line_style** *( {'-', '--', '-.', ':'}, optional)* – The style of the grid lines.

• **grid_line_width** *(float, optional)* – The width of the grid lines.

**Returns** `renderer` *(menpo.visualize.GraphPlotter)* – The renderer object.

**reconstructed_initial_error** *(compute_error=None)*

Returns the error of the reconstructed initial shape of the fitting process, if the ground truth shape exists. This is the error computed based on the `reconstructed_initial_shape`.

**Parameters** compute_error *(callable, optional)* – Callable that computes the error between the reconstructed initial and ground truth shapes.

**Returns** `reconstructed_initial_error` *(float)* – The error that corresponds to the initial shape’s reconstruction.

**Raises** `ValueError` – Ground truth shape has not been set, so the reconstructed initial error cannot be computed.

**to_result** *(pass_image=True, pass_initial_shape=True, pass_gt_shape=True)*

Returns a `Result` instance of the object, i.e. a fitting result object that does not store the iterations. This can be useful for reducing the size of saved fitting results.

**Parameters**

• pass_image *(bool, optional)* – If `True`, then the image will get passed (if it exists).

• pass_initial_shape *(bool, optional)* – If `True`, then the initial shape will get passed (if it exists).

• pass_gt_shape *(bool, optional)* – If `True`, then the ground truth shape will get passed (if it exists).

**Returns** `result` *(Result)* – The final “lightweight” fitting result.
Visualize the fitting result. The method renders the final fitted shape and optionally the initial shape, ground truth shape and the image, if they were provided.

Parameters

- **figure_id** (object, optional) – The id of the figure to be used.
- **new_figure** (bool, optional) – If True, a new figure is created.
- **render_image** (bool, optional) – If True and the image exists, then it gets rendered.
- **render_final_shape** (bool, optional) – If True, then the final fitting shape gets rendered.
- **render_initial_shape** (bool, optional) – If True and the initial fitting shape exists, then it gets rendered.
- **render_gt_shape** (bool, optional) – If True and the ground truth shape exists, then it gets rendered.
- **subplots_enabled** (bool, optional) – If True, then the requested final, initial and ground truth shapes get rendered on separate subplots.
- **channels** (int or list of int or all or None) – If int or list of int, the specified channel(s) will be rendered. If all, all the channels will be rendered in subplots. If None and the image is RGB, it will be rendered in RGB mode. If None and the image is not RGB, it is equivalent to all.
- **interpolation** (See Below, optional) – The interpolation used to render the image. For example, if bilinear, the image will be smooth and if nearest, the image will be pixelated. Example options

```python
{none, nearest, bilinear, bicubic, spline16, spline36, hanning, hamming, hermite, kaiser, quadric, catrom, gaussian, bessel, mitchell, sinc, lanczos}
```

- **cmap_name** (str, optional) – If None, single channel and three channel images default to greyscale and rgb colormaps respectively.
- **alpha** (float, optional) – The alpha blending value, between 0 (transparent) and 1 (opaque).
• **masked** *(bool, optional)* – If True, then the image is rendered as masked.

• **final_marker_face_colour** *(See Below, optional)* – The face (filling) colour of the markers of the final fitting shape. Example options

  \[
  \{r, g, b, c, m, k, w\}
  \text{ or }
  \{3, \text{ ndarray}\}
  \]

• **final_marker_edge_colour** *(See Below, optional)* – The edge colour of the markers of the final fitting shape. Example options

  \[
  \{r, g, b, c, m, k, w\}
  \text{ or }
  \{3, \text{ ndarray}\}
  \]

• **final_line_colour** *(See Below, optional)* – The line colour of the final fitting shape. Example options

  \[
  \{r, g, b, c, m, k, w\}
  \text{ or }
  \{3, \text{ ndarray}\}
  \]

• **initial_marker_face_colour** *(See Below, optional)* – The face (filling) colour of the markers of the initial shape. Example options

  \[
  \{r, g, b, c, m, k, w\}
  \text{ or }
  \{3, \text{ ndarray}\}
  \]

• **initial_marker_edge_colour** *(See Below, optional)* – The edge colour of the markers of the initial shape. Example options

  \[
  \{r, g, b, c, m, k, w\}
  \text{ or }
  \{3, \text{ ndarray}\}
  \]

• **initial_line_colour** *(See Below, optional)* – The line colour of the initial shape. Example options

  \[
  \{r, g, b, c, m, k, w\}
  \text{ or }
  \{3, \text{ ndarray}\}
  \]

• **gt_marker_face_colour** *(See Below, optional)* – The face (filling) colour of the markers of the ground truth shape. Example options

  \[
  \{r, g, b, c, m, k, w\}
  \text{ or }
  \{3, \text{ ndarray}\}
  \]

• **gt_marker_edge_colour** *(See Below, optional)* – The edge colour of the markers of the ground truth shape. Example options

  \[
  \{r, g, b, c, m, k, w\}
  \text{ or }
  \{3, \text{ ndarray}\}
  \]
- **gt_line_colour** (See Below, optional) – The line colour of the ground truth shape. Example options
  
  \[
  \{r, g, b, c, m, k, w\}
  \]
  or
  \[
  (3, ) \text{ndarray}
  \]

- **render_lines** (bool or list of bool, optional) – If True, the lines will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per shape in \((\text{final}, \text{initial}, \text{groundtruth})\) order.

- **line_style** (str or list of str, optional) – The style of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per shape in \((\text{final}, \text{initial}, \text{groundtruth})\) order. Example options:
  
  \[
  \{'-, '--', '-.', ':'\}
  \]

- **line_width** (float or list of float, optional) – The width of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per shape in \((\text{final}, \text{initial}, \text{groundtruth})\) order.

- **render_markers** (bool or list of bool, optional) – If True, the markers will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per shape in \((\text{final}, \text{initial}, \text{groundtruth})\) order.

- **marker_style** (str or list of str, optional) – The style of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per shape in \((\text{final}, \text{initial}, \text{groundtruth})\) order. Example options:
  
  \[
  \{., , , o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8\}
  \]

- **marker_size** (int or list of int, optional) – The size of the markers in points. You can either provide a single value that will be used for all shapes or a list with a different value per shape in \((\text{final}, \text{initial}, \text{groundtruth})\) order.

- **marker_edge_width** (float or list of float, optional) – The width of the markers’ edge. You can either provide a single value that will be used for all shapes or a list with a different value per shape in \((\text{final}, \text{initial}, \text{groundtruth})\) order.

- **render_numbering** (bool, optional) – If True, the landmarks will be numbered.

- **numbers_horizontal_align** (\{center, right, left\}, optional) – The horizontal alignment of the numbers’ texts.

- **numbers_vertical_align** (\{center, top, bottom, baseline\}, optional) – The vertical alignment of the numbers’ texts.

- **numbers_font_name** (See Below, optional) – The font of the numbers. Example options
  
  \[
  \{\text{serif, sans-serif, cursive, fantasy, monospace}\}
  \]

- **numbers_font_size** (int, optional) – The font size of the numbers.

- **numbers_font_style** (\{normal, italic, oblique\}, optional) – The font style of the numbers.

- **numbers_font_weight** (See Below, optional) – The font weight of the numbers. Example options
• **numbers_font_colour** *(See Below, optional)* – The font colour of the numbers. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• **render_legend** *(bool, optional)* – If True, the legend will be rendered.

• **legend_title** *(str, optional)* – The title of the legend.

• **legend_font_name** *(See below, optional)* – The font of the legend. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

• **legend_font_style** *(normal, italic, oblique), optional)* – The font style of the legend.

• **legend_font_size** *(int, optional)* – The font size of the legend.

• **legend_font_weight** *(See Below, optional)* – The font weight of the legend. Example options

```
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```

• **legend_marker_scale** *(float, optional)* – The relative size of the legend markers with respect to the original.

• **legend_location** *(int, optional)* – The location of the legend. The predefined values are:

```
'best'  0
'upper right'  1
'upper left'   2
'lower left'   3
'lower right'  4
'right'       5
'center left'  6
'center right' 7
'lower center' 8
'upper center' 9
'center'      10
```

• **legend_bbox_to_anchor** *((float, float) tuple, optional)* – The bbox that the legend will be anchored.

• **legend_border_axes_pad** *(float, optional)* – The pad between the axes and legend border.

• **legend_n_columns** *(int, optional)* – The number of the legend’s columns.

• **legend_horizontal_spacing** *(float, optional)* – The spacing between the columns.
• **legend_vertical_spacing** (*float*, optional) – The vertical space between the legend entries.

• **legend_border** (*bool*, optional) – If True, a frame will be drawn around the legend.

• **legend_border_padding** (*float*, optional) – The fractional whitespace inside the legend border.

• **legend_shadow** (*bool*, optional) – If True, a shadow will be drawn behind legend.

• **legend_rounded_corners** (*bool*, optional) – If True, the frame’s corners will be rounded (fancybox).

• **render_axes** (*bool*, optional) – If True, the axes will be rendered.

• **axes_font_name** (*See Below, optional*) – The font of the axes. Example options

  {serif, sans-serif, cursive, fantasy, monospace}

• **axes_font_size** (*int*, optional) – The font size of the axes.

• **axes_font_style** (*{normal, italic, oblique}*), optional) – The font style of the axes.

• **axes_font_weight** (*See Below, optional*) – The font weight of the axes. Example options

  {ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}

• **axes_x_limits** (*float or (float, float) or None, optional*) – The limits of the x axis. If float, then it sets padding on the right and left of the Image as a percentage of the Image’s width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.

• **axes_y_limits** (**(float, float) tuple or None, optional*) – The limits of the y axis. If float, then it sets padding on the top and bottom of the Image as a percentage of the Image’s height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.

• **axes_x_ticks** (*list or tuple or None, optional*) – The ticks of the x axis.

• **axes_y_ticks** (*list or tuple or None, optional*) – The ticks of the y axis.

• **figure_size** (**(float, float) tuple or None optional*) – The size of the figure in inches.

**Returns**  **renderer** (*class*) – The renderer object.
view_iterations (figure_id=None, new_figure=False, iters=None, render_image=True, subplots_enabled=False, channels=None, interpolation='bilinear', cmap_name=None, alpha=1.0, masked=True, render_lines=True, line_style='-', line_width=2, line_colour=None, render_markers=True, marker_edge_colour=None, marker_face_colour=None, marker_style='o', marker_size=4, marker_edge_width=1.0, render_numbering=False, numbers_horizontal_align='center', numbers_vertical_align='bottom', numbers_font_name='sans-serif', numbers_font_size=10, numbers_font_style='normal', numbers_font_weight='normal', numbers_font_colour='k', render_legend=True, legend_title='', legend_font_name='sans-serif', legend_font_style='normal', legend_font_size=10, legend_font_weight='normal', legend_marker_scale=None, legend_location=2, legend_bbox_to_anchor=(1.05, 1.0), legend_border_axes_pad=None, legend_n_columns=1, legend_horizontal_spacing=None, legend_vertical_spacing=None, legend_border=True, legend_border_padding=None, legend_shadow=False, legend_rounded_corners=False, render_axes=False, axes_font_name='sans-serif', axes_font_size=10, axes_font_style='normal', axes_font_weight='normal', axes_x_limits=None, axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None, figure_size=(7, 7))

Visualize the iterations of the fitting process.

**Parameters**

- **figure_id** *(object, optional)* – The id of the figure to be used.
- **new_figure** *(bool, optional)* – If True, a new figure is created.
- **iters** *(int or list of int or None, optional)* – The iterations to be visualized. If None, then all the iterations are rendered.

<table>
<thead>
<tr>
<th>No.</th>
<th>Visualised shape</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>self.initial_shape</td>
<td>Initial shape</td>
</tr>
<tr>
<td>1</td>
<td>self.reconstructed_initial_shape</td>
<td>Reconstructed initial</td>
</tr>
<tr>
<td>2</td>
<td>self.shapes[2]</td>
<td>Iteration 1</td>
</tr>
<tr>
<td>i</td>
<td>self.shapes[i]</td>
<td>Iteration i-1</td>
</tr>
<tr>
<td>n_iters+1</td>
<td>self.final_shape</td>
<td>Final shape</td>
</tr>
</tbody>
</table>

- **render_image** *(bool, optional)* – If True and the image exists, then it gets rendered.
- **subplots_enabled** *(bool, optional)* – If True, then the requested final, initial and ground truth shapes get rendered on separate subplots.
- **channels** *(int or list of int or all or None, optional)* – If int or list of int, the specified channel(s) will be rendered. If all, all the channels will be rendered in subplots. If None and the image is RGB, it will be rendered in RGB mode. If None and the image is not RGB, it is equivalent to all.
- **interpolation** *(str (See Below), optional)* – The interpolation used to render the image. For example, if bilinear, the image will be smooth and if nearest, the image will be pixelated. Example options

```
{none, nearest, bilinear, bicubic, spline16, spline36, hanning, hamming, hermite, kaiser, quadric, catrom, gaussian, bessel, mitchell, sinc, lanczos}
```

- **cmap_name** *(str, optional.)* – If None, single channel and three channel images default to greyscale and rgb colormaps respectively.
• **alpha** *(float, optional)* – The alpha blending value, between 0 (transparent) and 1 (opaque).

• **masked** *(bool, optional)* – If True, then the image is rendered as masked.

• **render_lines** *(bool or list of bool, optional)* – If True, the lines will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.

• **line_style** *(str or list of str (See below), optional)* – The style of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options:

  ```
  {-, --, -., :}
  ```

• **line_width** *(float or list of float, optional)* – The width of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.

• **line_colour** *(colour or list of colour (See Below), optional)* – The colour of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

  ```
  {r, g, b, c, m, k, w}
  or
  (3, ) ndarray
  ```

• **render_markers** *(bool or list of bool, optional)* – If True, the markers will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.

• **marker_style** *(str or list of str (See below), optional)* – The style of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

  ```
  {., ,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
  ```

• **marker_size** *(int or list of int, optional)* – The size of the markers in points. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.

• **marker_edge_colour** *(colour or list of colour (See Below), optional)* – The edge colour of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

  ```
  {r, g, b, c, m, k, w}
  or
  (3, ) ndarray
  ```

• **marker_face_colour** *(colour or list of colour (See Below), optional)* – The face (filling) colour of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

  ```
  {r, g, b, c, m, k, w}
  or
  (3, ) ndarray
  ```

• **marker_edge_width** *(float or list of float, optional)* – The width of the markers’ edge. You can either provide a single value that will be used for all shapes or a list with a different
value per iteration shape.

- **render_numbering** *(bool, optional)* – If True, the landmarks will be numbered.

- **numbers_horizontal_align** *(str (See below), optional)* – The horizontal alignment of the numbers’ texts. Example options
  
  ```
  {center, right, left}
  ```

- **numbers_vertical_align** *(str (See below), optional)* – The vertical alignment of the numbers’ texts. Example options
  
  ```
  {center, top, bottom, baseline}
  ```

- **numbers_font_name** *(str (See below), optional)* – The font of the numbers. Example options
  
  ```
  {serif, sans-serif, cursive, fantasy, monospace}
  ```

- **numbers_font_size** *(int, optional)* – The font size of the numbers.

- **numbers_font_style** *(str (See below), optional)* – The font style of the numbers.

- **numbers_font_weight** *(str (See below), optional)* – The font weight of the numbers. Example options
  
  ```
  {ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
  ```

- **numbers_font_colour** *(See Below, optional)* – The font colour of the numbers. Example options
  
  ```
  {r, g, b, c, m, k, w}
  or
  (3, ) ndarray
  ```

- **render_legend** *(bool, optional)* – If True, the legend will be rendered.

- **legend_title** *(str, optional)* – The title of the legend.

- **legend_font_name** *(See below, optional)* – The font of the legend. Example options
  
  ```
  {serif, sans-serif, cursive, fantasy, monospace}
  ```

- **legend_font_style** *(str (See below), optional)* – The font style of the legend. Example options
  
  ```
  {normal, italic, oblique}
  ```

- **legend_font_size** *(int, optional)* – The font size of the legend.

- **legend_font_weight** *(str (See below), optional)* – The font weight of the legend. Example options
  
  ```
  {ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
  ```
- **legend_marker_scale** *(float, optional)* – The relative size of the legend markers with respect to the original

- **legend_location** *(int, optional)* – The location of the legend. The predefined values are:

<table>
<thead>
<tr>
<th>Location</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>'best'</td>
<td>0</td>
</tr>
<tr>
<td>'upper right'</td>
<td>1</td>
</tr>
<tr>
<td>'upper left'</td>
<td>2</td>
</tr>
<tr>
<td>'lower left'</td>
<td>3</td>
</tr>
<tr>
<td>'lower right'</td>
<td>4</td>
</tr>
<tr>
<td>'right'</td>
<td>5</td>
</tr>
<tr>
<td>'center left'</td>
<td>6</td>
</tr>
<tr>
<td>'center right'</td>
<td>7</td>
</tr>
<tr>
<td>'lower center'</td>
<td>8</td>
</tr>
<tr>
<td>'upper center'</td>
<td>9</td>
</tr>
<tr>
<td>'center'</td>
<td>10</td>
</tr>
</tbody>
</table>

- **legend_bbox_to_anchor** *((float, float) tuple, optional)* – The bbox that the legend will be anchored.

- **legend_border_axes_pad** *(float, optional)* – The pad between the axes and legend border.

- **legend_n_columns** *(int, optional)* – The number of the legend’s columns.

- **legend_horizontal_spacing** *(float, optional)* – The spacing between the columns.

- **legend_vertical_spacing** *(float, optional)* – The vertical space between the legend entries.

- **legend_border** *(bool, optional)* – If True, a frame will be drawn around the legend.

- **legend_border_padding** *(float, optional)* – The fractional whitespace inside the legend border.

- **legend_shadow** *(bool, optional)* – If True, a shadow will be drawn behind legend.

- **legend_rounded_corners** *(bool, optional)* – If True, the frame’s corners will be rounded (fancybox).

- **render_axes** *(bool, optional)* – If True, the axes will be rendered.

- **axes_font_name** *(str (See below), optional)* – The font of the axes. Example options:

  `{serif, sans-serif, cursive, fantasy, monospace}`

- **axes_font_size** *(int, optional)* – The font size of the axes.

- **axes_font_style** *(str (normal, italic, oblique), optional)* – The font style of the axes.

- **axes_font_weight** *(str (See below), optional)* – The font weight of the axes. Example options:

  `{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}`

- **axes_x_limits** *(float or (float, float) or None, optional)* – The limits of the x axis. If float, then it sets padding on the right and left of the Image as a percentage of the Image’s...
width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.

- **axes_y_limits** ((float, float) tuple or None, optional) – The limits of the y axis. If float, then it sets padding on the top and bottom of the Image as a percentage of the Image’s height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.

- **axes_x_ticks** (list or tuple or None, optional) – The ticks of the x axis.

- **axes_y_ticks** (list or tuple or None, optional) – The ticks of the y axis.

- **figure_size** ((float, float) tuple or None optional) – The size of the figure in inches.

Returns **renderer** (class) – The renderer object.

**view_widget**(figure_size=(7, 7))

Visualizes the result object using an interactive widget.

Parameters **figure_size** ((int, int), optional) – The initial size of the rendered figure.

**property appearance_parameters**

Returns the list of appearance parameters obtained at each iteration of the fitting process. The list includes the parameters of the initial_shape (if it exists) and final_shape.

Type list of (n_params,) ndarray

**property costs**

Returns a list with the cost per iteration. It returns None if the costs are not computed.

Type list of float or None

**property final_shape**

Returns the final shape of the fitting process.

Type menpo.shape.PointCloud

**property gt_shape**

Returns the ground truth shape associated with the image. In case there is not an attached ground truth shape, then None is returned.

Type menpo.shape.PointCloud or None

**property image**

Returns the image that the fitting was applied on, if it was provided. Otherwise, it returns None.

Type menpo.shape.Image or subclass or None

**property initial_shape**

Returns the initial shape that was provided to the fitting method to initialise the fitting process. In case the initial shape does not exist, then None is returned.

Type menpo.shape.PointCloud or None

**property is_iterative**

Flag whether the object is an iterative fitting result.

Type bool

**property n_iters**

Returns the total number of iterations of the fitting process.

Type int
property `reconstructed_initial_shape`  
Returns the initial shape’s reconstruction with the shape model that was used to initialise the iterative optimisation process.

Type `menpo.shape.PointCloud`

property `shape_parameters`  
Returns the list of shape parameters obtained at each iteration of the fitting process. The list includes the parameters of the `reconstructed_initial_shape` and `final_shape`.

Type `list of (n_params,) ndarray`

property `shapes`  
Returns the list of shapes obtained at each iteration of the fitting process. The list includes the `initial_shape` (if it exists), `reconstructed_initial_shape` and `final_shape`.

Type `list of menpo.shape.PointCloud`

Pre-Trained Model

`load_balanced_frontal_face_fitter`  

`menpofit.aam.load_balanced_frontal_face_fitter()`  
Loads a frontal face patch-based AAM fitter that is a good compromise between model size, fitting time and fitting performance. The model returns 68 facial landmark points (the standard IBUG68 markup).

Note that the first time you invoke this function, menpofit will download the fitter from Menpo’s server. The fitter will then be stored locally for future use.

The model is a PatchAAM trained using the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>diagonal</td>
<td>110</td>
</tr>
<tr>
<td>scales</td>
<td>(0.5, 1.0)</td>
</tr>
<tr>
<td>patch_shape</td>
<td>[(13, 13), (13, 13)]</td>
</tr>
<tr>
<td>holistic_features</td>
<td><code>menpo.feature.fast_dsift()</code></td>
</tr>
<tr>
<td>n_shape</td>
<td>[5, 20]</td>
</tr>
<tr>
<td>n_appearance</td>
<td>[30, 150]</td>
</tr>
<tr>
<td>lk_algorithm_cls</td>
<td>WibergInverseCompositional</td>
</tr>
</tbody>
</table>

It is also using the following sampling grid:

```python
import numpy as np

patch_shape = (13, 13)
sampling_step = 4

sampling_grid = np.zeros(patch_shape, dtype=np.bool)
sampling_grid[:,:,:sampling_step, ::sampling_step] = True
sampling = [sampling_grid, sampling_grid]
```

Additionally, it is trained on LFPW trainset, HELEN trainset, IBUG and AFW datasets (3283 images in total), which are hosted in http://ibug.doc.ic.ac.uk/resources/facial-point-annotations/.

Returns fitter (LucasKanadeAAMFitter) – A pre-trained LucasKanadeAAMFitter based on a PatchAAM that performs facial landmark localization returning 68 points (iBUG68).
2.1.2 menpofit.aps

Active Pictorial Structures

APS is a model that utilises a Gaussian Markov Random Field (GMRF) for learning an appearance model with pairwise distributions based on a graph. It also has a parametric statistical shape model (either using PCA or GMRF), as well as a spring-like deformation prior term. The optimisation is performed using a weighted Gauss-Newton algorithm with fixed Jacobian and Hessian.

GenerativeAPS

class menpofit.aps.GenerativeAPS(images, group=None, appearance_graph=None, shape_graph=None, deformation_graph=None, holistic_features=<function no_op>, reference_shape=None, diagonal=None, scales=(0.5, 1.0), patch_shape=(17, 17), patch_normalisation=<function no_op>, use_procrustes=True, precision_dtype=<class 'numpy.float32'>, max_shape_components=None, n_appearance_components=None, can_be_incremented=False, verbose=False, batch_size=None)

Bases: object

Class for training a multi-scale Generative Active Pictorial Structures model. Please see the references for a basic list of relevant papers.

Parameters

- **images** (list of menpo.image.Image) – The list of training images.
- **group** (str or None, optional) – The landmark group that will be used to train the AAM. If None and the images only have a single landmark group, then that is the one that will be used. Note that all the training images need to have the specified landmark group.
- **appearance_graph** (list of graphs or a single graph or None, optional) – The graph to be used for the appearance menpo.model.GMRFModel training. It must be a menpo.shape.UndirectedGraph. If None, then a menpo.model.PCAModel is used instead.
- **shape_graph** (list of graphs or a single graph or None, optional) – The graph to be used for the shape menpo.model.GMRFModel training. It must be a menpo.shape.UndirectedGraph. If None, then the shape model is built using menpo.model.PCAModel.
- **deformation_graph** (list of graphs or a single graph or None, optional) – The graph to be used for the deformation menpo.model.GMRFModel training. It must be either a menpo.shape.DirectedGraph or a menpo.shape.Tree. If None, then the minimum spanning tree of the data is computed.
- **holistic_features** (closure or list of closure, optional) – The features that will be extracted from the training images. Note that the features are extracted before warping the images to the reference shape. If list, then it must define a feature function per scale. Please refer to menpo.feature for a list of potential features.
- **reference_shape** (menpo.shape.PointCloud or None, optional) – The reference shape that will be used for building the APS. The purpose of the reference shape is to normalise the size of the training images. The normalization is performed by rescaling all the training images so that the scale of their ground truth shapes matches the scale of the reference shape.
Note that the reference shape is rescaled with respect to the diagonal before performing the normalisation. If None, then the mean shape will be used.

- **diagonal** (int or None, optional) – This parameter is used to rescale the reference shape so that the diagonal of its bounding box matches the provided value. In other words, this parameter controls the size of the model at the highest scale. If None, then the reference shape does not get rescaled.

- **scales** (float or tuple of float, optional) – The scale value of each scale. They must provided in ascending order, i.e. from lowest to highest scale. If float, then a single scale is assumed.

- **patch_shape** ((int, int) or list of (int, int), optional) – The shape of the patches to be extracted. If a list is provided, then it defines a patch shape per scale.

- **patch_normalisation** (list of callable or a single callable, optional) – The normalisation function to be applied on the extracted patches. If list, then it must have length equal to the number of scales. If a single patch normalization callable, then this is the one applied to all scales.

- **use_procrustes** (bool, optional) – If True, then Generalized Procrustes Alignment is applied before building the deformation model.

- **precision_dtype** (numpy.dtype, optional) – The data type of the appearance GMRF's precision matrix. For example, it can be set to numpy.float32 for single precision or to numpy.float64 for double precision. Even though the precision matrix is stored as a scipy.sparse matrix, this parameter has a big impact on the amount of memory required by the model.

- **max_shape_components** (int, float, list of those or None, optional) – The number of shape components to keep. If int, then it sets the exact number of components. If float, then it defines the variance percentage that will be kept. If list, then it should define a value per scale. If a single number, then this will be applied to all scales. If None, then all the components are kept. Note that the unused components will be permanently trimmed.

- **n_appearance_components** (list of int or int or None, optional) – The number of appearance components used for building the appearance menpo.shape.GMRFModel. If list, then it must have length equal to the number of scales. If a single int, then this is the one applied to all scales. If None, the covariance matrix of each edge is inverted using np.linalg.inv. If int, it is inverted using truncated SVD using the specified number of components.

- **can_be_incremented** (bool, optional) – In case you intend to incrementally update the model in the future, then this flag must be set to True from the first place. Note that if True, the appearance and deformation menpo.shape.GMRFModel models will occupy double memory.

- **verbose** (bool, optional) – If True, then the progress of building the APS will be printed.

- **batch_size** (int or None, optional) – If an int is provided, then the training is performed in an incremental fashion on image batches of size equal to the provided value. If None, then the training is performed directly on the all the images.

References

increment (images, group=None, batch_size=None, verbose=False)

Method that incrementally updates the APS model with a new batch of training images.
• **images** (list of menpo.image.Image) – The list of training images.

• **group** (str or None, optional) – The landmark group that will be used to train the APS. If None and the images only have a single landmark group, then that is the one that will be used. Note that all the training images need to have the specified landmark group.

• **batch_size** (int or None, optional) – If an int is provided, then the training is performed in an incremental fashion on image batches of size equal to the provided value. If None, then the training is performed directly on all the images.

• **verbose** (bool, optional) – If True, then the progress of building the APS will be printed.

### instance (shape_weights=None, scale_index=-1, as_graph=False)

Generates an instance of the shape model.

**Parameters**

• **shape_weights** ((n_weights,) ndarray or list or None, optional) – The weights of the shape model that will be used to create a novel shape instance. If None, the weights are assumed to be zero, thus the mean shape is used.

• **scale_index** (int, optional) – The scale to be used.

• **as_graph** (bool, optional) – If True, then the instance will be returned as a menpo.shape.PointTree or a menpo.shape.PointDirectedGraph, depending on the type of the deformation graph.

### random_instance (scale_index=-1, as_graph=False)

Generates a random instance of the APS.

**Parameters**

• **scale_index** (int, optional) – The scale to be used.

• **as_graph** (bool, optional) – If True, then the instance will be returned as a menpo.shape.PointTree or a menpo.shape.PointDirectedGraph, depending on the type of the deformation graph.

### view_appearance_graph_widget (scale_index=-1, figure_size=(7, 7))

Visualize the appearance graph using an interactive widget.

**Parameters**

• **scale_index** (int, optional) – The scale to be used.

• **figure_size** ((int, int), optional) – The size of the rendered figure.

**Raises** ValueError – Scale level {scale_index} uses a PCA appearance model, so there is no graph

### view_deformation_graph_widget (scale_index=-1, figure_size=(7, 7))

Visualize the deformation graph using an interactive widget.

**Parameters**

• **scale_index** (int, optional) – The scale to be used.

• **figure_size** ((int, int), optional) – The size of the rendered figure.
view_deformation_model (scale_index=-1, n_std=2, render_colour_bar=False, colour_map='jet', image_view=True, figure_id=None, new_figure=False, render_graph_lines=True, graph_line_colour='b', graph_line_style='-', graph_line_width=1.0, ellipse_line_colour='r', ellipse_line_style='-', ellipse_line_width=1.0, render_markers=True, marker_style='o', marker_size=5, marker_face_colour='k', marker_edge_colour='k', marker_edge_width=1.0, render_axes=False, axes_font_name='sans-serif', axes_font_size=10, axes_font_style='normal', axes_font_weight='normal', crop_proportion=0.1, figure_size=(7, 7))

Visualize the deformation model by plotting a Gaussian ellipsis per graph edge.

Parameters

- **scale_index** *(int, optional)* – The scale to be used.
- **n_std** *(float, optional)* – This defines the size of the ellipses in terms of number of standard deviations.
- **render_colour_bar** *(bool, optional)* – If True, then the ellipses will be coloured based on their normalized standard deviations and a colour bar will also appear on the side. If False, then all the ellipses will have the same colour.
- **colour_map** *(str, optional)* – A valid Matplotlib colour map. For more info, please refer to matplotlib.cm.
- **image_view** *(bool, optional)* – If True the ellipses will be rendered in the image coordinates system.
- **figure_id** *(object, optional)* – The id of the figure to be used.
- **new_figure** *(bool, optional)* – If True, a new figure is created.
- **render_graph_lines** *(bool, optional)* – Defines whether to plot the graph’s edges.
- **graph_line_colour** *(See Below, optional)* – The colour of the lines of the graph’s edges. Example options:

```
[r, g, b, c, m, k, w]
or
(3, ) ndarray
```

- **graph_line_style** *(See Below, optional)* – The style of the lines of the graph’s edges.
- **graph_line_width** *(float, optional)* – The width of the lines of the graph’s edges.
- **ellipse_line_colour** *(See Below, optional)* – The colour of the lines of the ellipses. Example options:

```
[r, g, b, c, m, k, w]
or
(3, ) ndarray
```

- **ellipse_line_style** *(See Below, optional)* – The style of the lines of the ellipses.
- **ellipse_line_width** *(float, optional)* – The width of the lines of the ellipses.
- **render_markers** *(bool, optional)* – If True, the centers of the ellipses will be rendered.
• **marker_style** *(See Below, optional)* – The style of the centers of the ellipses. Example options

```
(., ., o, v, ^, <, >, +, x, d, s, p, x, h, H, 1, 2, 3, 4, 8)
```

• **marker_size** *(int, optional)* – The size of the centers of the ellipses in points.

• **marker_face_colour** *(See Below, optional)* – The face (filling) colour of the centers of the ellipses. Example options

```
[r, g, b, c, m, k, w]
or
(3, ) ndarray
```

• **marker_edge_colour** *(See Below, optional)* – The edge colour of the centers of the ellipses. Example options

```
[r, g, b, c, m, k, w]
or
(3, ) ndarray
```

• **marker_edge_width** *(float, optional)* – The edge width of the centers of the ellipses.

• **render_axes** *(bool, optional)* – If True, the axes will be rendered.

• **axes_font_name** *(See Below, optional)* – The font of the axes. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

• **axes_font_size** *(int, optional)* – The font size of the axes.

• **axes_font_style** *(normal, italic, oblique)* – The font style of the axes.

• **axes_font_weight** *(See Below, optional)* – The font weight of the axes. Example options

```
{ultralight, light, normal, regular, book, medium, roman, semibold, semibold, demi, bold, heavy, extra bold, black}
```

• **crop_proportion** *(float, optional)* – The proportion to be left around the centers’ pointcloud.

• **figure_size** *(((float, float) tuple or None) optional)* – The size of the figure in inches.

view_shape_graph_widget *(scale_index=-1, figure_size=(7, 7))*

Visualize the shape graph using an interactive widget.

**Parameters**

• **scale_index** *(int, optional)* – The scale to be used.

• **figure_size** *(((int, int), optional)* – The size of the rendered figure.

**Raises** ValueError – Scale level [scale_index] uses a PCA shape model, so there is no graph

view_shape_models_widget *(n_parameters=5, parameters_bounds=(-3.0, 3.0), mode='multiple', figure_size=(7, 7))*

Visualizes the shape models of the APS object using an interactive widget.

**Parameters**
• **n_parameters** (*int or list of int* or *None*, optional) – The number of shape principal components to be used for the parameters sliders. If *int*, then the number of sliders per scale is the minimum between *n_parameters* and the number of active components per scale. If *list of int*, then a number of sliders is defined per scale. If *None*, all the active components per scale will have a slider.

• **parameters_bounds** (*tuple of float, float*, optional) – The minimum and maximum bounds, in std units, for the sliders.

• **mode** (*{single, multiple}*, optional) – If 'single', only a single slider is constructed along with a drop down menu. If 'multiple', a slider is constructed for each parameter.

• **figure_size** (*tuple of int, int*, optional) – The size of the rendered figure.

**property n_scales**

Returns the number of scales.

Type: int

### Fitters

#### GaussNewtonAPSFitter

**class** `menpofit.aps.GaussNewtonAPSFitter` *(aps, gn_algorithm_cls=<class 'menpofit.aps.algorithm.gn.Inverse'>, n_shape=None, weight=200.0, sampling=None)*

**Bases:** APSFitter

A class for fitting an APS model with Gauss-Newton optimization.

**Note:** When using a method with a parametric shape model, the first step is to **reconstruct the initial shape** using the shape model. The generated reconstructed shape is then used as initialisation for the iterative optimisation. This step takes place at each scale and it is not considered as an iteration, thus it is not counted for the provided `max_iters`.

**Parameters**

• **aps** (*GenerativeAPS or subclass*) – The trained model.

• **gn_algorithm_cls** (*class*, optional) – The Gauss-Newton optimisation algorithm that will get applied. The possible algorithms are Inverse and Forward. Note that the Forward algorithm is too slow. It is not recommended to be used for fitting an APS and is only included for comparison purposes.

• **n_shape** (*int or float or list of those or None*, optional) – The number of shape components that will be used. If *int*, then it defines the exact number of active components. If *float*, then it defines the percentage of variance to keep. If *int or float*, then the provided value will be applied for all scales. If *list*, then it defines a value per scale. If *None*, then all the available components will be used. Note that this simply sets the active components without trimming the unused ones. Also, the available components may have already been trimmed to `max_shape_components` during training.

• **weight** (*float or list of float*, optional) – The weight between the appearance cost and the deformation cost. The provided value gets multiplied with the deformation cost. If *float*, then the provided value will be used for all scales. If *list*, then it should define a value per scale.
• **sampling** *(list of int or ndarray or None)* – It defines a sampling mask per scale. If int, then it defines the sub-sampling step of the sampling mask. If ndarray, then it explicitly defines the sampling mask. If None, then no sub-sampling is applied. Note that depending on the model and the size of the appearance precision matrix, the sub-sampling may be impossible to be applied due to insufficient memory. This is because the sub-sampling of the appearance precision matrix involves converting it to scipy.sparse.lil_matrix, sub-sampling it and re-convert it back to scipy.sparse.bsr_matrix, which is a memory intensive procedure.

```python
fit_from_bb(image, bounding_box, max_iters=20, gt_shape=None, return_costs=False, **kwargs)
```

Fits the multi-scale fitter to an image given an initial bounding box.

**Parameters**

- **image** *(menpo.image.Image or subclass)* – The image to be fitted.
- **bounding_box** *(menpo.shape.PointDirectedGraph)* – The initial bounding box from which the fitting procedure will start. Note that the bounding box is used in order to align the model’s reference shape.
- **max_iters** *(int or list of int, optional)* – The maximum number of iterations. If int, then it specifies the maximum number of iterations over all scales. If list of int, then specifies the maximum number of iterations per scale.
- **gt_shape** *(menpo.shape.PointCloud, optional)* – The ground truth shape associated to the image.
- **return_costs** *(bool, optional)* – If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.
- **kwargs** *(dict, optional)* – Additional keyword arguments that can be passed to specific implementations.

**Returns**

- **fitting_result** *(MultiScaleNonParametricIterativeResult or subclass)* – The multi-scale fitting result containing the result of the fitting procedure.

```python
fit_from_shape(image, initial_shape, max_iters=20, gt_shape=None, return_costs=False, **kwargs)
```

Fits the multi-scale fitter to an image given an initial shape.

**Parameters**

- **image** *(menpo.image.Image or subclass)* – The image to be fitted.
- **initial_shape** *(menpo.shape.PointCloud)* – The initial shape estimate from which the fitting procedure will start.
- **max_iters** *(int or list of int, optional)* – The maximum number of iterations. If int, then it specifies the maximum number of iterations over all scales. If list of int, then specifies the maximum number of iterations per scale.
- **gt_shape** *(menpo.shape.PointCloud, optional)* – The ground truth shape associated to the image.
- **return_costs** *(bool, optional)* – If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.
• **kwargs** (*dict*, optional) – Additional keyword arguments that can be passed to specific implementations.

**Returns** fitting_result (*MultiScaleNonParametricIterativeResult or subclass*) – The multi-scale fitting result containing the result of the fitting procedure.

**warped_images** (*image, shapes*)

Given an input test image and a list of shapes, it warps the image into the shapes. This is useful for generating the warped images of a fitting procedure stored within an **APSResult**.

**Parameters**

• **image** (*menpo.image.Image or subclass*) – The input image to be warped.

• **shapes** (*list of menpo.shape.PointCloud*) – The list of shapes in which the image will be warped. The shapes are obtained during the iterations of a fitting procedure.

**Returns** warped_images (*list of menpo.image.MaskedImage or ndarray*) – The warped images.

**property** aps

The trained APS model.

**Type** **GenerativeAPS** or subclass

**property** holistic_features

The features that are extracted from the input image at each scale in ascending order, i.e. from lowest to highest scale.

**Type** *list of closure*

**property** n_scales

Returns the number of scales.

**Type** *int*

**property** reference_shape

The reference shape that is used to normalise the size of an input image so that the scale of its initial fitting shape matches the scale of this reference shape.

**Type** *menpo.shape.PointCloud*

**property** scales

The scale value of each scale in ascending order, i.e. from lowest to highest scale.

**Type** *list of int or float*

---

**2.1. Deformable Models**

---

---

**Gauss-Newton Optimisation Algorithms**

**Inverse**

**class** menpofit.aps.Inverse(*aps_interface, eps=1e-05*)

**Bases**: GaussNewton

Inverse Gauss-Newton algorithm for APS.

**run** (*image, initial_shape, gt_shape=None, max_iters=20, return_costs=False*)

Execute the optimization algorithm.

**Parameters**

• **image** (*menpo.image.Image*) – The input test image.
• `initial_shape` ([`menpo.shape.PointCloud`]) – The initial shape from which the optimization will start.

• `gt_shape` ([`menpo.shape.PointCloud` or `None`, optional]) – The ground truth shape of the image. It is only needed in order to get passed in the optimization result object, which has the ability to compute the fitting error.

• `max_iters` (int, optional) – The maximum number of iterations. Note that the algorithm may converge, and thus stop, earlier.

• `return_costs` (bool, optional) – If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned `fitting_result`. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.

Returns `fitting_result` ([`APSAlgorithmResult`]) – The parametric iterative fitting result.

**property appearance_model**

Returns the appearance GMRF model.

Type: `menpo.model.GMRFModel`

**property deformation_model**

Returns the deformation GMRF model.

Type: `menpo.model.GMRFModel`

**property template**

Returns the template (usually the mean appearance).

Type: `menpo.image.Image`

**property transform**

Returns the motion model.

Type: `OrthoPDM`

---

**Forward**

```python
class menpofit.aps.Forward(aps_interface, eps=1e-05)
```

Bases: `GaussNewton`

Forward Gauss-Newton algorithm for APS.

**Note:** The Forward optimization is too slow. It is not recommended to be used for fitting an APS and is only included for comparison purposes. Use `Inverse` instead.

```python
run(image, initial_shape, gt_shape=None, max_iters=20, return_costs=False)
```

Execute the optimization algorithm.

**Parameters**

• `image` ([`menpo.image.Image`]) – The input test image.

• `initial_shape` ([`menpo.shape.PointCloud`]) – The initial shape from which the optimization will start.
• **gt_shape** (*menpo.shape.PointCloud* or *None*, optional) – The ground truth shape of the image. It is only needed in order to get passed in the optimization result object, which has the ability to compute the fitting error.

• **max_iters** (*int*, optional) – The maximum number of iterations. Note that the algorithm may converge, and thus stop, earlier.

• **return_costs** (*bool*, optional) – If *True*, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned *fitting_result*. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.

**Returns** fitting_result (*APSAlgorithmResult*) – The parametric iterative fitting result.

**property appearance_model**

Returns the appearance GMRF model.

Type *menpo.model.GMRFModel*

**property deformation_model**

Returns the deformation GMRF model.

Type *menpo.model.GMRFModel*

**property template**

Returns the template (usually the mean appearance).

Type *menpo.image.Image*

**property transform**

Returns the motion model.

Type *OrthoPDM*

Fitting Result

**APSResult**

**class** menpofit.aps.result.APSResult(*results*, *scales*, *affine_transforms*, *scale_transforms*, *image*=*None*, *gt_shape*=*None*)

**Bases:** *MultiScaleParametricIterativeResult*

Class for storing the multi-scale iterative fitting result of an APS. It holds the shapes, shape parameters, appearance parameters and costs per iteration.

**Note:** When using a method with a parametric shape model, the first step is to reconstruct the initial shape using the shape model. The generated reconstructed shape is then used as initialisation for the iterative optimisation. This step is not counted in the number of iterations.

**Parameters**

• **results** (*list of APSAlgorithmResult*) – The list of optimization results per scale.

• **scales** (*list or tuple*) – The list of scale values per scale (low to high).

• **affine_transforms** (*list of menpo.transform.Affine*) – The list of affine transforms per scale that transform the shapes into the original image space.
- **scale_transforms** (list of *menpo.shape.Scale*) – The list of scaling transforms per scale.

- **image** (*menpo.image.Image* or subclass or *None*, optional) – The image on which the fitting process was applied. Note that a copy of the image will be assigned as an attribute. If *None*, then no image is assigned.

- **gt_shape** (*menpo.shape.PointCloud* or *None*, optional) – The ground truth shape associated with the image. If *None*, then no ground truth shape is assigned.

**displacements** ()
A list containing the displacement between the shape of each iteration and the shape of the previous one.

Type list of ndarray

**displacements_stats** (*stat_type=*'mean'*)
A list containing a statistical metric on the displacements between the shape of each iteration and the shape of the previous one.

Parameters *stat_type* (*{'mean', 'median', 'min', 'max'}*, optional) – Specifies a statistic metric to be extracted from the displacements.

Returns **displacements_stat** (list of float) – The statistical metric on the points displacements for each iteration.

Raises **ValueError** – type must be ‘mean’, ‘median’, ‘min’ or ‘max’

**errors** (*compute_error=*_None_*)
Returns a list containing the error at each fitting iteration, if the ground truth shape exists.

Parameters *compute_error* (callable, optional) – Callable that computes the error between the shape at each iteration and the ground truth shape.

Returns **errors** (list of float) – The error at each iteration of the fitting process.

Raises **ValueError** – Ground truth shape has not been set, so the final error cannot be computed

**final_error** (*compute_error=*_None_*)
Returns the final error of the fitting process, if the ground truth shape exists. This is the error computed based on the **final_shape**.

Parameters *compute_error* (callable, optional) – Callable that computes the error between the fitted and ground truth shapes.

Returns **final_error** (float) – The final error at the end of the fitting process.

Raises **ValueError** – Ground truth shape has not been set, so the final error cannot be computed

**initial_error** (*compute_error=*_None_*)
Returns the initial error of the fitting process, if the ground truth shape and initial shape exist. This is the error computed based on the **initial_shape**.

Parameters *compute_error* (callable, optional) – Callable that computes the error between the initial and ground truth shapes.

Returns **initial_error** (float) – The initial error at the beginning of the fitting process.

Raises

- **ValueError** – Initial shape has not been set, so the initial error cannot be computed
• **ValueError** – Ground truth shape has not been set, so the initial error cannot be computed

```
plot_costs(figure_id=None, new_figure=False, render_lines=True, line_colour='b', line_style='-', line_width=2, render_markers=True, marker_style='o', marker_size=4, marker_face_colour='b', marker_edge_colour='k', marker_edge_width=1.0, render_axes=True, axes_font_name='sans-serif', axes_font_size=10, axes_font_style='normal', axes_font_weight='normal', axes_x_limits=0.0, axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None, figure_size=(10, 6), render_grid=True, grid_line_style='-', grid_line_width=0.5)
```

Plot of the cost function evolution at each fitting iteration.

**Parameters**

- **figure_id** (object, optional) – The id of the figure to be used.
- **new_figure** (bool, optional) – If True, a new figure is created.
- **render_lines** (bool, optional) – If True, the line will be rendered.
- **line_colour** (colour or None, optional) – The colour of the line. If None, the colour is sampled from the jet colormap. Example colour options are

  ```
  ['r', 'g', 'b', 'c', 'm', 'k', 'w]
  or
  (3, ) ndarray
  ```

- **line_style** ('-'|'--'|'-.'|':'), optional) – The style of the lines.
- **line_width** (float, optional) – The width of the lines.
- **render_markers** (bool, optional) – If True, the markers will be rendered.
- **marker_style** (marker, optional) – The style of the markers. Example marker options

  ```
  ['.', ',', 'o', 'v', '^', '<', '>', '+', 'x', 'D', 'd', 's', 'p', '*', 'h', 'H', '1', '2', '3', '4', '8']
  ```

- **marker_size** (int, optional) – The size of the markers in points.
- **marker_face_colour** (colour or None, optional) – The face (filling) colour of the markers. If None, the colour is sampled from the jet colormap. Example colour options are

  ```
  ['r', 'g', 'b', 'c', 'm', 'k', 'w]
  or
  (3, ) ndarray
  ```

- **marker_edge_colour** (colour or None, optional) – The edge colour of the markers. If None, the colour is sampled from the jet colormap. Example colour options are

  ```
  ['r', 'g', 'b', 'c', 'm', 'k', 'w]
  or
  (3, ) ndarray
  ```

- **marker_edge_width** (float, optional) – The width of the markers’ edge.
- **render_axes** (bool, optional) – If True, the axes will be rendered.
- **axes_font_name** (See below, optional) – The font of the axes. Example options
MenpoFit Documentation, Release 0.6.1

```python
('serif', 'sans-serif', 'cursive', 'fantasy', 'monospace')
```

- `axes_font_size (int, optional)` – The font size of the axes.
- `axes_font_style ({'normal', 'italic', 'oblique'}, optional)` – The font style of the axes.
- `axes_font_weight (See below, optional)` – The font weight of the axes. Example options
  ```python
  ('ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black')
  ```
- `axes_x_limits (float or (float, float) or None, optional)` – The limits of the x axis. If `float`, then it sets padding on the right and left of the graph as a percentage of the curves’ width. If `tuple` or `list`, then it defines the axis limits. If `None`, then the limits are set automatically.
- `axes_y_limits (float or (float, float) or None, optional)` – The limits of the y axis. If `float`, then it sets padding on the top and bottom of the graph as a percentage of the curves’ height. If `tuple` or `list`, then it defines the axis limits. If `None`, then the limits are set automatically.
- `axes_x_ticks (list or tuple or None, optional)` – The ticks of the x axis.
- `axes_y_ticks (list or tuple or None, optional)` – The ticks of the y axis.
- `figure_size ((float, float) or None, optional)` – The size of the figure in inches.
- `render_grid (bool, optional)` – If `True`, the grid will be rendered.
- `grid_line_style (('|-', '|--', '-.', ':'), optional)` – The style of the grid lines.
- `grid_line_width (float, optional)` – The width of the grid lines.

Returns `renderer (menpo.visualize.GraphPlotter)` – The renderer object.

```python
plot_displacements (stat_type='mean', figure_id=None, new_figure=False, render_lines=True, line_colour='b', line_style='-', line_width=2, render_markers=True, marker_style='o', marker_size=4, marker_face_colour='b', marker_edge_colour='k', marker_edge_width=1.0, render_axes=True, axes_font_name='sans-serif', axes_font_size=10, axes_font_style='normal', axes_font_weight='normal', axes_x_limits=0.0, axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None, figure_size=(10, 6), render_grid=True, grid_line_style='--', grid_line_width=0.5)
```

Plot of a statistical metric of the displacement between the shape of each iteration and the shape of the previous one.

Parameters

- `stat_type ({'mean', 'median', 'min', 'max'}, optional)` – Specifies a statistic metric to be extracted from the displacements (see also `displacements_stats()` method).
- `figure_id (object, optional)` – The id of the figure to be used.
- `new_figure (bool, optional)` – If `True`, a new figure is created.
- `render_lines (bool, optional)` – If `True`, the line will be rendered.
- `line_colour (colour or None (See below), optional)` – The colour of the line. If `None`, the colour is sampled from the jet colormap. Example `colour` options are
**line_style** *(str, optional)* – The style of the lines. Example options:

```
{-, --, -., :}
```

**line_width** *(float, optional)* – The width of the lines.

**render_markers** *(bool, optional)* – If True, the markers will be rendered.

**marker_style** *(str, optional)* – The style of the markers. Example *marker* options

```
{., ,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
```

**marker_size** *(int, optional)* – The size of the markers in points.

**marker_face_colour** *(colour or None, optional)* – The face (filling) colour of the markers. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

**marker_edge_colour** *(colour or None, optional)* – The edge colour of the markers. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

**marker_edge_width** *(float, optional)* – The width of the markers’ edge.

**render_axes** *(bool, optional)* – If True, the axes will be rendered.

**axes_font_name** *(str, optional)* – The font of the axes. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

**axes_font_size** *(int, optional)* – The font size of the axes.

**axes_font_style** *(str, optional)* – The font style of the axes. Example options

```
{normal, italic, oblique}
```

**axes_font_weight** *(str, optional)* – The font weight of the axes. Example options

```
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```

**axes_x_limits** *(float or (float, float) or None, optional)* – The limits of the x axis. If *float*, then it sets padding on the right and left of the graph as a percentage of the curves’ width. If *tuple* or *list*, then it defines the axis limits. If None, then the limits are set automatically.
MenpoFit Documentation, Release 0.6.1

- **axes_y_limits** *(float or (float, float) or None, optional)* – The limits of the y axis. If float, then it sets padding on the top and bottom of the graph as a percentage of the curves’ height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.

- **axes_x_ticks** *(list or tuple or None, optional)* – The ticks of the x axis.

- **axes_y_ticks** *(list or tuple or None, optional)* – The ticks of the y axis.

- **figure_size** *((float, float) or None, optional)* – The size of the figure in inches.

- **render_grid** *(bool, optional)* – If True, the grid will be rendered.

- **grid_line_style** *(str, optional)* – The style of the grid lines.

- **grid_line_width** *(float, optional)* – The width of the grid lines.

Returns `renderer` *(menpo.visualize.GraphPlotter)* – The renderer object.

`plot_errors` *(compute_error=None, figure_id=None, new_figure=False, render_lines=True, line_colour='b', line_style='-', line_width=2, render_markers=True, marker_style='o', marker_size=4, marker_face_colour='b', marker_edge_colour='k', marker_edge_width=1.0, render_axes=True, axes_font_name='sans-serif', axes_font_size=10, axes_font_style='normal', axes_font_weight='normal', axes_x_limits=0.0, axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None, figure_size=(10, 6), render_grid=True, grid_line_style='--', grid_line_width=0.5)*

Plot of the error evolution at each fitting iteration.

**Parameters**

- **compute_error** *(callable, optional)* – Callable that computes the error between the shape at each iteration and the ground truth shape.

- **figure_id** *(object, optional)* – The id of the figure to be used.

- **new_figure** *(bool, optional)* – If True, a new figure is created.

- **render_lines** *(bool, optional)* – If True, the line will be rendered.

- **line_colour** *(colour or None (See below), optional)* – The colour of the line. If None, the colour is sampled from the jet colormap. Example `colour` options are

  ```
  {r, g, b, c, m, k, w}
  or
  (3, ) ndarray
  ```

- **line_style** *(str (See below), optional)* – The style of the lines. Example options:

  ```
  [-, --, -, :, ]
  ```

- **line_width** *(float, optional)* – The width of the lines.

- **render_markers** *(bool, optional)* – If True, the markers will be rendered.

- **marker_style** *(str (See below), optional)* – The style of the markers. Example `marker` options

  ```
  {.,., o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
  ```

- **marker_size** *(int, optional)* – The size of the markers in points.
• **marker_face_colour** (*colour* or *None*, optional) – The face (filling) colour of the markers. If *None*, the colour is sampled from the jet colormap. Example *colour* options are

```python
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• **marker_edge_colour** (*colour* or *None*, optional) – The edge colour of the markers. If *None*, the colour is sampled from the jet colormap. Example *colour* options are

```python
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• **marker_edge_width** (*float*, optional) – The width of the markers’ edge.

• **render_axes** (*bool*, optional) – If *True*, the axes will be rendered.

• **axes_font_name** (*str* (See below), optional) – The font of the axes. Example options

```python
{serif, sans-serif, cursive, fantasy, monospace}
```

• **axes_font_size** (*int*, optional) – The font size of the axes.

• **axes_font_style** (*str* (See below), optional) – The font style of the axes. Example options

```python
{normal, italic, oblique}
```

• **axes_font_weight** (*str* (See below), optional) – The font weight of the axes. Example options

```python
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```

• **axes_x_limits** (*float* or (*float*, *float*)) or *None*, optional) – The limits of the x axis. If *float*, then it sets padding on the right and left of the graph as a percentage of the curves’ width. If *tuple* or *list*, then it defines the axis limits. If *None*, then the limits are set automatically.

• **axes_y_limits** (*float* or (*float*, *float*)) or *None*, optional) – The limits of the y axis. If *float*, then it sets padding on the top and bottom of the graph as a percentage of the curves’ height. If *tuple* or *list*, then it defines the axis limits. If *None*, then the limits are set automatically.

• **axes_x_ticks** (*list* or *tuple* or *None*, optional) – The ticks of the x axis.

• **axes_y_ticks** (*list* or *tuple* or *None*, optional) – The ticks of the y axis.

• **figure_size** ((*float*, *float*) or *None*, optional) – The size of the figure in inches.

• **render_grid** (*bool*, optional) – If *True*, the grid will be rendered.

• **grid_line_style** (‘-, ‘--’, ‘.-’, ‘:’), optional) – The style of the grid lines.

• **grid_line_width** (*float*, optional) – The width of the grid lines.

Returns **renderer** (*menpo.visualize.GraphPlotter*) – The renderer object.
reconstructed_initial_error (compute_error=None)
Returns the error of the reconstructed initial shape of the fitting process, if the ground truth shape exists. This is the error computed based on the reconstructed_initial_shapes[0].

Parameters compute_error (callable, optional) – Callable that computes the error between the reconstructed initial and ground truth shapes.

Returns reconstructed_initial_error (float) – The error that corresponds to the initial shape’s reconstruction.

Raises ValueError – Ground truth shape has not been set, so the reconstructed initial error cannot be computed
to_result (pass_image=True, pass_initial_shape=True, pass_gt_shape=True)
Returns a Result instance of the object, i.e. a fitting result object that does not store the iterations. This can be useful for reducing the size of saved fitting results.

Parameters

• pass_image (bool, optional) – If True, then the image will get passed (if it exists).

• pass_initial_shape (bool, optional) – If True, then the initial shape will get passed (if it exists).

• pass_gt_shape (bool, optional) – If True, then the ground truth shape will get passed (if it exists).

Returns result (Result) – The final “lightweight” fitting result.
view (figure_id=None, new_figure=False, render_image=True, render_final_shape=True, render_initial_shape=False, render_gt_shape=False, subplots_enabled=True, channels=None, interpolation='bilinear', cmap_name=None, alpha=1.0, masked=True, final_marker_face_colour='r', final_marker_edge_colour='k', final_line_colour='r', initial_marker_face_colour='b', initial_marker_edge_colour='k', initial_line_colour='b', gt_marker_face_colour='y', gt_marker_edge_colour='k', gt_line_colour='y', render_lines=True, line_style='-', line_width=2, render_markers=True, marker_style='o', marker_size=4, marker_edge_width=1.0, render_numbering=False, numbers_horizontal_align='center', numbers_vertical_align='bottom', numbers_font_name='sans-serif', numbers_font_size=10, numbers_font_style='normal', numbers_font_weight='normal', legend_marker_scale=1, legend_title='', legend_font_name='sans-serif', legend_font_style='normal', legend_font_size=10, legend_font_weight='normal', legend_marker_scale=1, legend_location=2, legend_box_to_anchor=(1.05, 1.0), legend_border_axes_pad=1, legend_n_columns=1, legend_horizontal_spacing=0, legend_vertical_spacing=0, legend_border=True, legend_border_padding=1, legend_shadow=False, legend_rounded_corners=False, render_axes=False, axes_font_name='sans-serif', axes_font_size=10, axes_font_style='normal', axes_font_weight='normal', axes_x_limits=None, axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None, figure_size=(7, 7))
Visualize the fitting result. The method renders the final fitted shape and optionally the initial shape, ground truth shape and the image, id they were provided.

Parameters

• figure_id (object, optional) – The id of the figure to be used.

• new_figure (bool, optional) – If True, a new figure is created.

• render_image (bool, optional) – If True and the image exists, then it gets rendered.

• render_final_shape (bool, optional) – If True, then the final fitting shape gets rendered.
• **render_initial_shape** (*bool*, optional) – If True and the initial fitting shape exists, then it gets rendered.

• **render_gt_shape** (*bool*, optional) – If True and the ground truth shape exists, then it gets rendered.

• **subplots_enabled** (*bool*, optional) – If True, then the requested final, initial and ground truth shapes get rendered on separate subplots.

• **channels** (*int* or *list of int* or *all* or *None*) – If *int* or *list of int*, the specified channel(s) will be rendered. If *all*, all the channels will be rendered in subplots. If *None* and the image is RGB, it will be rendered in RGB mode. If *None* and the image is not RGB, it is equivalent to *all*.

• **interpolation** *(See Below, optional)* – The interpolation used to render the image. For example, if *bilinear*, the image will be smooth and if *nearest*, the image will be pixelated. Example options

\[
\text{none, nearest, bilinear, bicubic, spline16, spline36, hanning, hamming, hermite, kaiser, quadric, catrom, gaussian, bessel, mitchell, sinc, lanczos}
\]

• **cmap_name** (*str*, optional.) – If None, single channel and three channel images default to greyscale and rgb colormaps respectively.

• **alpha** (*float*, optional) – The alpha blending value, between 0 (transparent) and 1 (opaque).

• **masked** (*bool*, optional) – If True, then the image is rendered as masked.

• **final_marker_face_colour** *(See Below, optional)* – The face (filling) colour of the markers of the final fitting shape. Example options

\[
\{r, g, b, c, m, k, w\}
\text{or}
\{3, \} \text{ndarray}
\]

• **final_marker_edge_colour** *(See Below, optional)* – The edge colour of the markers of the final fitting shape. Example options

\[
\{r, g, b, c, m, k, w\}
\text{or}
\{3, \} \text{ndarray}
\]

• **final_line_colour** *(See Below, optional)* – The line colour of the final fitting shape. Example options

\[
\{r, g, b, c, m, k, w\}
\text{or}
\{3, \} \text{ndarray}
\]

• **initial_marker_face_colour** *(See Below, optional)* – The face (filling) colour of the markers of the initial shape. Example options

\[
\{r, g, b, c, m, k, w\}
\text{or}
\{3, \} \text{ndarray}
\]

• **initial_marker_edge_colour** *(See Below, optional)* – The edge colour of the markers of the initial shape. Example options

\[
\{r, g, b, c, m, k, w\}
\text{or}
\{3, \} \text{ndarray}
\]
• initial_line_colour (See Below, optional) – The line colour of the initial shape. Example options

{r, g, b, c, m, k, w} or (3, ) ndarray

• gt_marker_face_colour (See Below, optional) – The face (filling) colour of the markers of the ground truth shape. Example options

{r, g, b, c, m, k, w} or (3, ) ndarray

• gt_marker_edge_colour (See Below, optional) – The edge colour of the markers of the ground truth shape. Example options

{r, g, b, c, m, k, w} or (3, ) ndarray

• gt_line_colour (See Below, optional) – The line colour of the ground truth shape. Example options

{r, g, b, c, m, k, w} or (3, ) ndarray

• render_lines (bool or list of bool, optional) – If True, the lines will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (final, initial, groundtruth) order.

• line_style (str or list of str, optional) – The style of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (final, initial, groundtruth) order. Example options:

['-', '--', '-.', ':']

• line_width (float or list of float, optional) – The width of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (final, initial, groundtruth) order.

• render_markers (bool or list of bool, optional) – If True, the markers will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (final, initial, groundtruth) order.

• marker_style (str or list of str, optional) – The style of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (final, initial, groundtruth) order. Example options:

(., ., o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8)
• **marker_size** (*int* or list of *int*, optional) – The size of the markers in points. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (*final, initial, groundtruth*) order.

• **marker_edge_width** (*float* or list of *float*, optional) – The width of the markers’ edge. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (*final, initial, groundtruth*) order.

• **render_numbering** (*bool*, optional) – If True, the landmarks will be numbered.

• **numbers_horizontal_align** (*{center, right, left}*), optional) – The horizontal alignment of the numbers’ texts.

• **numbers_vertical_align** (*{center, top, bottom, baseline}*), optional) – The vertical alignment of the numbers’ texts.

• **numbers_font_name** (*See Below, optional*) – The font of the numbers. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

• **numbers_font_size** (*int*, optional) – The font size of the numbers.

• **numbers_font_style** (*{normal, italic, oblique}*), optional) – The font style of the numbers.

• **numbers_font_weight** (*See Below, optional*) – The font weight of the numbers. Example options

```
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```

• **numbers_font_colour** (*See Below, optional*) – The font colour of the numbers. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• **render_legend** (*bool*, optional) – If True, the legend will be rendered.

• **legend_title** (*str*, optional) – The title of the legend.

• **legend_font_name** (*See below, optional*) – The font of the legend. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

• **legend_font_style** (*{normal, italic, oblique}*), optional) – The font style of the legend.

• **legend_font_size** (*int*, optional) – The font size of the legend.

• **legend_font_weight** (*See Below, optional*) – The font weight of the legend. Example options

```
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```

• **legend_marker_scale** (*float*, optional) – The relative size of the legend markers with respect to the original
• **legend_location** (*int*, optional) – The location of the legend. The predefined values are:

<table>
<thead>
<tr>
<th>Legend Location</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>best</td>
<td>0</td>
</tr>
<tr>
<td>upper right</td>
<td>1</td>
</tr>
<tr>
<td>upper left</td>
<td>2</td>
</tr>
<tr>
<td>lower left</td>
<td>3</td>
</tr>
<tr>
<td>lower right</td>
<td>4</td>
</tr>
<tr>
<td>right</td>
<td>5</td>
</tr>
<tr>
<td>center left</td>
<td>6</td>
</tr>
<tr>
<td>center right</td>
<td>7</td>
</tr>
<tr>
<td>lower center</td>
<td>8</td>
</tr>
<tr>
<td>upper center</td>
<td>9</td>
</tr>
<tr>
<td>center</td>
<td>10</td>
</tr>
</tbody>
</table>

• **legend_bbox_to_anchor** ((*float*, *float*) *tuple*, optional) – The bbox that the legend will be anchored.

• **legend_border_axes_pad** (*float*, optional) – The pad between the axes and legend border.

• **legend_n_columns** (*int*, optional) – The number of the legend’s columns.

• **legend_horizontal_spacing** (*float*, optional) – The spacing between the columns.

• **legend_vertical_spacing** (*float*, optional) – The vertical space between the legend entries.

• **legend_border** (*bool*, optional) – If `True`, a frame will be drawn around the legend.

• **legend_border_padding** (*float*, optional) – The fractional whitespace inside the legend border.

• **legend_shadow** (*bool*, optional) – If `True`, a shadow will be drawn behind legend.

• **legend_rounded_corners** (*bool*, optional) – If `True`, the frame’s corners will be rounded (fancybox).

• **render_axes** (*bool*, optional) – If `True`, the axes will be rendered.

• **axes_font_name** (*See Below, optional*) – The font of the axes. Example options

```latex
{serif, sans-serif, cursive, fantasy, monospace}
```

• **axes_font_size** (*int*, optional) – The font size of the axes.

• **axes_font_style** (*{normal, italic, oblique}*), optional) – The font style of the axes.

• **axes_font_weight** (*See Below, optional*) – The font weight of the axes. Example options

```latex
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```

• **axes_x_limits** (*float* or (*float*, *float*) or *None*, optional) – The limits of the x axis. If `float`, then it sets padding on the right and left of the Image as a percentage of the Image’s width. If *tuple* or *list*, then it defines the axis limits. If *None*, then the limits are set automatically.
MenpoFit Documentation, Release 0.6.1

- **axes_y_limits** ((float, float) tuple or None, optional) – The limits of the y axis. If float, then it sets padding on the top and bottom of the Image as a percentage of the Image’s height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.

- **axes_x_ticks** (list or tuple or None, optional) – The ticks of the x axis.

- **axes_y_ticks** (list or tuple or None, optional) – The ticks of the y axis.

- **figure_size** ((float, float) tuple or None optional) – The size of the figure in inches.

**Returns** renderer (class) – The renderer object.

**view_iterations** (figure_id=None, new_figure=False, iters=None, render_image=True, subplots_enabled=False, channels=None, interpolation='bilinear', cmap_name=None, alpha=1.0, masked=True, render_lines=True, line_style='-', line_width=2, line_colour=None, render_markers=True, marker_edge_colour=None, marker_face_colour=None, marker_style='o', marker_size=4, marker_edge_width=1.0, render_numbering=False, numbers_horizontal_align='center', numbers_vertical_align='bottom', numbers_font_name='sans-serif', numbers_font_size=10, numbers_font_style='normal', numbers_font_weight='normal', numbers_font_colour='k', render_legend=True, legend_title='', legend_font_name='sans-serif', legend_font_style='normal', legend_font_size=10, legend_font_weight='normal', legend_location=2, legend_bbox_to_anchor=(1.05, 1.0), legend_focus_axes_pad=None, legend_n_columns=1, legend_horizontal_spacing=None, legend_spacing=None, legend_border=True, legend_border_padding=None, legend_shadow=False, legend_rounded_corners=False, render_axes=True, axes_face_colour=None, axes_edge_colour=None, axes_x_limits=None, axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None, figure_size=(7, 7))

Visualize the iterations of the fitting process.

**Parameters**

- **figure_id** (object, optional) – The id of the figure to be used.

- **new_figure** (bool, optional) – If True, a new figure is created.

- **iters** (int or list of int or None, optional) – The iterations to be visualized. If None, then all the iterations are rendered.

<table>
<thead>
<tr>
<th>No.</th>
<th>Visualised shape</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>self.initial_shape</td>
<td>Initial shape</td>
</tr>
<tr>
<td>1</td>
<td>self.reconstructed_initial_shape</td>
<td>Reconstructed initial</td>
</tr>
<tr>
<td>2</td>
<td>self.shapes[2]</td>
<td>Iteration 1</td>
</tr>
<tr>
<td>n_iters+1</td>
<td>self.final_shape</td>
<td>Final shape</td>
</tr>
</tbody>
</table>

- **render_image** (bool, optional) – If True and the image exists, then it gets rendered.

- **subplots_enabled** (bool, optional) – If True, then the requested final, initial and ground truth shapes get rendered on separate subplots.

- **channels** (int or list of int or all or None) – If int or list of int, the specified channel(s) will be rendered. If all, all the channels will be rendered in subplots. If None and the

### 2.1. Deformable Models
image is RGB, it will be rendered in RGB mode. If None and the image is not RGB, it is equivalent to all.

- **interpolation** *(str (See Below), optional)* – The interpolation used to render the image. For example, if **bilinear**, the image will be smooth and if **nearest**, the image will be pixelated. Example options

```python
{none, nearest, bilinear, bicubic, spline16, spline36, hanning, hamming, hermite, kaiser, quadric, catrom, gaussian, bessel, mitchell, sinc, lanczos}
```

- **cmap_name** *(str, optional)* – If None, single channel and three channel images default to greyscale and rgb colormaps respectively.

- **alpha** *(float, optional)* – The alpha blending value, between 0 (transparent) and 1 (opaque).

- **masked** *(bool, optional)* – If True, then the image is rendered as masked.

- **render_lines** *(bool or list of bool, optional)* – If True, the lines will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.

- **line_style** *(str or list of str (See below), optional)* – The style of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options:

```python
{-, --, -., :}
```

- **line_width** *(float or list of float, optional)* – The width of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.

- **line_colour** *(colour or list of colour (See Below), optional)* – The colour of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```python
[r, g, b, c, m, k, w]
```

- **render_markers** *(bool or list of bool, optional)* – If True, the markers will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.

- **marker_style** *(str or 'list of str (See below), optional)* – The style of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```python
{., ,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
```

- **marker_size** *(int or list of int, optional)* – The size of the markers in points. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.

- **marker_edge_colour** *(colour or list of colour (See Below), optional)* – The edge colour of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options
MenpoFit Documentation, Release 0.6.1

- `marker_face_colour` (colour or list of colour (See Below), optional) – The face (filling) colour of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

  \[ \{ \text{r, g, b, c, m, k, w} \} \]
  \[ \text{or} \]
  \[ (3, \) ndarray \]

- `marker_edge_width` (float or list of float, optional) – The width of the markers’ edge. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.

- `render_numbering` (bool, optional) – If True, the landmarks will be numbered.

- `numbers_horizontal_align` (str (See below), optional) – The horizontal alignment of the numbers’ texts. Example options

  \[ \{ \text{center, right, left} \} \]

- `numbers_vertical_align` (str (See below), optional) – The vertical alignment of the numbers’ texts. Example options

  \[ \{ \text{center, top, bottom, baseline} \} \]

- `numbers_font_name` (str (See below), optional) – The font of the numbers. Example options

  \[ \{ \text{serif, sans-serif, cursive, fantasy, monospace} \} \]

- `numbers_font_size` (int, optional) – The font size of the numbers.

- `numbers_font_style` (\{(normal, italic, oblique)\}, optional) – The font style of the numbers.

- `numbers_font_weight` (str (See below), optional) – The font weight of the numbers. Example options

  \[ \{ \text{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black} \} \]

- `numbers_font_colour`(See Below, optional) – The font colour of the numbers. Example options

  \[ \{ \text{r, g, b, c, m, k, w} \} \]
  \[ \text{or} \]
  \[ (3, \) ndarray \]

- `render_legend` (bool, optional) – If True, the legend will be rendered.

- `legend_title` (str, optional) – The title of the legend.

- `legend_font_name`(See below, optional) – The font of the legend. Example options

2.1. Deformable Models
• **legend_font_style** *(str, optional)* – The font style of the legend. Example options

{serif, sans-serif, cursive, fantasy, monospace}

• **legend_font_size** *(int, optional)* – The font size of the legend.

• **legend_font_weight** *(str, optional)* – The font weight of the legend. Example options

{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}

• **legend_marker_scale** *(float, optional)* – The relative size of the legend markers with respect to the original

• **legend_location** *(int, optional)* – The location of the legend. The predefined values are:

```
<table>
<thead>
<tr>
<th>Location</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>best</td>
<td>0</td>
</tr>
<tr>
<td>upper right</td>
<td>1</td>
</tr>
<tr>
<td>upper left</td>
<td>2</td>
</tr>
<tr>
<td>lower left</td>
<td>3</td>
</tr>
<tr>
<td>lower right</td>
<td>4</td>
</tr>
<tr>
<td>right</td>
<td>5</td>
</tr>
<tr>
<td>center left</td>
<td>6</td>
</tr>
<tr>
<td>center right</td>
<td>7</td>
</tr>
<tr>
<td>lower center</td>
<td>8</td>
</tr>
<tr>
<td>upper center</td>
<td>9</td>
</tr>
<tr>
<td>center</td>
<td>10</td>
</tr>
</tbody>
</table>
```

• **legend_bbox_to_anchor** *((float, float) tuple, optional)* – The bbox that the legend will be anchored.

• **legend_border_axes_pad** *(float, optional)* – The pad between the axes and legend border.

• **legend_n_columns** *(int, optional)* – The number of the legend’s columns.

• **legend_horizontal_spacing** *(float, optional)* – The spacing between the columns.

• **legend_vertical_spacing** *(float, optional)* – The vertical space between the legend entries.

• **legend_border** *(bool, optional)* – If True, a frame will be drawn around the legend.

• **legend_border_padding** *(float, optional)* – The fractional whitespace inside the legend border.

• **legend_shadow** *(bool, optional)* – If True, a shadow will be drawn behind legend.

• **legend_rounded_corners** *(bool, optional)* – If True, the frame’s corners will be rounded (fancybox).

• **render_axes** *(bool, optional)* – If True, the axes will be rendered.

• **axes_font_name** *(str, See below), optional)* – The font of the axes. Example options
• **axes_font_size** *(int, optional)* – The font size of the axes.

• **axes_font_style** *(normal, italic, oblique, optional)* – The font style of the axes.

• **axes_font_weight** *(str (See below), optional)* – The font weight of the axes. Example options

   {ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}

• **axes_x_limits** *(float or (float, float) or None, optional)* – The limits of the x axis. If float, then it sets padding on the right and left of the Image as a percentage of the Image’s width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.

• **axes_y_limits** *(float, float) tuple or None, optional)* – The limits of the y axis. If float, then it sets padding on the top and bottom of the Image as a percentage of the Image’s height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.

• **axes_x_ticks** *(list or tuple or None, optional)* – The ticks of the x axis.

• **axes_y_ticks** *(list or tuple or None, optional)* – The ticks of the y axis.

• **figure_size** *(float, float) tuple or None optional)* – The size of the figure in inches.

    Returns **renderer** *(class)* – The renderer object.

    **view_widget** *(figure_size=(7, 7))*

    Visualizes the result object using an interactive widget.

    Parameters **figure_size** *(int, int, optional)* – The initial size of the rendered figure.

    **property appearance_costs**

    Returns a *list* with the appearance cost per iteration. It returns *None* if the costs are not computed.

    Type *list* of *float* or *None*

    **property costs**

    Returns a *list* with the cost per iteration. It returns *None* if the costs are not computed.

    Type *list* of *float* or *None*

    **property deformation_costs**

    Returns a *list* with the deformation cost per iteration. It returns *None* if the costs are not computed.

    Type *list* of *float* or *None*

    **property final_shape**

    Returns the final shape of the fitting process.

    Type *menpo.shape.PointCloud*

    **property gt_shape**

    Returns the ground truth shape associated with the image. In case there is not an attached ground truth shape, then *None* is returned.

    Type *menpo.shape.PointCloud* or *None*

    **property image**

    Returns the image that the fitting was applied on, if it was provided. Otherwise, it returns *None*.
**Type**  `menpo.shape.Image` or subclass or None

**property initial_shape**  
Returns the initial shape that was provided to the fitting method to initialise the fitting process. In case the initial shape does not exist, then `None` is returned.

**Type**  `menpo.shape.PointCloud` or None

**property is_iterative**  
Flag whether the object is an iterative fitting result.

**Type**  `bool`

**property n_iters**  
Returns the total number of iterations of the fitting process.

**Type**  `int`

**property n_iters_per_scale**  
Returns the number of iterations per scale of the fitting process.

**Type**  `list` of `int`

**property n_scales**  
Returns the number of scales used during the fitting process.

**Type**  `int`

**property reconstructed_initial_shapes**  
Returns the result of the reconstruction step that takes place at each scale before applying the iterative optimisation.

**Type**  `list` of `menpo.shape.PointCloud`

**property shape_parameters**  
Returns the list of shape parameters obtained at each iteration of the fitting process. The list includes the parameters of the `initial_shape` (if it exists) and `final_shape`.

**Type**  `list` of `(n_params,)` `ndarray`

**property shapes**  
Returns the list of shapes obtained at each iteration of the fitting process. The list includes the `initial_shape` (if it exists) and `final_shape`.

**Type**  `list` of `menpo.shape.PointCloud`

### APSAlgorithmResult

**class** `menpo.fit.aps.result.APSAlgorithmResult` *(shapes, shape_parameters, initial_shape=None, image=None, gt_shape=None, appearance_costs=None, deformation_costs=None, costs=None)*

**Bases:** `ParametricIterativeResult`

Class for storing the iterative result of an APS optimisation algorithm.

**Note:** When using a method with a parametric shape model, the first step is to **reconstruct the initial shape** using the shape model. The generated reconstructed shape is then used as initialisation for the iterative optimisation. This step is not counted in the number of iterations.
Parameters

- **shapes** *(list of menpo.shape.PointCloud)* – The list of shapes per iteration. The first and last members correspond to the initial and final shapes, respectively.

- **shape_parameters** *(list of (n_shape_parameters,) ndarray)* – The list of shape parameters per iteration. The first and last members correspond to the initial and final shapes, respectively.

- **initial_shape** *(menpo.shape.PointCloud or None, optional)* – The initial shape from which the fitting process started. If None, then no initial shape is assigned.

- **image** *(menpo.image.Image or subclass or None, optional)* – The image on which the fitting process was applied. Note that a copy of the image will be assigned as an attribute. If None, then no image is assigned.

- **gt_shape** *(menpo.shape.PointCloud or None, optional)* – The ground truth shape associated with the image. If None, then no ground truth shape is assigned.

- **appearance_costs** *(list of float or None, optional)* – The list of the appearance cost per iteration. If None, then it is assumed that the cost function cannot be computed for the specific algorithm.

- **deformation_costs** *(list of float or None, optional)* – The list of the deformation cost per iteration. If None, then it is assumed that the cost function cannot be computed for the specific algorithm.

- **costs** *(list of float or None, optional)* – The list of the total cost per iteration. If None, then it is assumed that the cost function cannot be computed for the specific algorithm.

- **displacements** ()
  A list containing the displacement between the shape of each iteration and the shape of the previous one.

  **Type** list of ndarray

- **displacements_stats**(stat_type='mean')
  A list containing a statistical metric on the displacements between the shape of each iteration and the shape of the previous one.

  **Parameters** stat_type={'mean', 'median', 'min', 'max'}, optional) – Specifies a statistic metric to be extracted from the displacements.

  **Returns** displacements_stat *(list of float)* – The statistical metric on the points displacements for each iteration.

  **Raises** ValueError – type must be ‘mean’, ‘median’, ‘min’ or ‘max’

- **errors**(compute_error=None)
  Returns a list containing the error at each fitting iteration, if the ground truth shape exists.

  **Parameters** compute_error(callable, optional) – Callable that computes the error between the shape at each iteration and the ground truth shape.

  **Returns** errors *(list of float)* – The error at each iteration of the fitting process.

  **Raises** ValueError – Ground truth shape has not been set, so the final error cannot be computed

- **final_error**(compute_error=None)
  Returns the final error of the fitting process, if the ground truth shape exists. This is the error computed based on the final_shape.
Parameters `compute_error` (callable, optional) – Callable that computes the error between the fitted and ground truth shapes.

Returns `final_error` (float) – The final error at the end of the fitting process.

Raises `ValueError` – Ground truth shape has not been set, so the final error cannot be computed

`initial_error` (compute_error=None)

Returns the initial error of the fitting process, if the ground truth shape and initial shape exist. This is the error computed based on the `initial_shape`.

Parameters `compute_error` (callable, optional) – Callable that computes the error between the initial and ground truth shapes.

Returns `initial_error` (float) – The initial error at the beginning of the fitting process.

Raises

- `ValueError` – Initial shape has not been set, so the initial error cannot be computed
- `ValueError` – Ground truth shape has not been set, so the initial error cannot be computed

`plot_costs` (figure_id=None, new_figure=False, render_lines=True, line_colour='b', line_style='-', line_width=2, render_markers=True, marker_style='o', marker_size=4, marker_face_colour='b', marker_edge_colour='k', marker_edge_width=1.0, render_axes=True, axes_font_name='sans-serif', axes_font_size=10, axes_font_style='normal', axes_font_weight='normal', axes_x_limits=0.0, axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None, figure_size=(10, 6), render_grid=True, grid_line_style='–', grid_line_width=0.5)

Plot of the cost function evolution at each fitting iteration.

Parameters

- `figure_id` (object, optional) – The id of the figure to be used.
- `new_figure` (bool, optional) – If True, a new figure is created.
- `render_lines` (bool, optional) – If True, the line will be rendered.
- `line_colour` (colour or None, optional) – The colour of the line. If None, the colour is sampled from the jet colormap. Example colour options are
  
  ```
  ['r', 'g', 'b', 'c', 'm', 'k', 'w']
  or
  (3, ) ndarray
  ```

- `line_style` (('-', '--', '-.', ':'), optional) – The style of the lines.
- `line_width` (float, optional) – The width of the lines.
- `render_markers` (bool, optional) – If True, the markers will be rendered.
- `marker_style` (marker, optional) – If True, the markers will be rendered. Example marker options
  
  ```
  ['.', ',', 'o', 'v', '^', '<', '>', '+', 'x', 'D', 'd', 's', 'p', '*', 'h', 'H', '1', '2', '3', '4', '8']
  ```

- `marker_size` (int, optional) – The size of the markers in points.
- `marker_face_colour` (colour or None, optional) – The face (filling) colour of the markers. If None, the colour is sampled from the jet colormap. Example colour options are
• **marker_edge_colour** *(colour or None, optional)* – The edge colour of the markers. If None, the colour is sampled from the jet colormap. Example *colour* options are

\[
\{\text{‘r’, ‘g’, ‘b’, ‘c’, ‘m’, ‘k’, ‘w’}\}
\text{or}
\{(3, ) \text{ndarray}\}
\]

• **marker_edge_width** *(float, optional)* – The width of the markers’ edge.

• **render_axes** *(bool, optional)* – If True, the axes will be rendered.

• **axes_font_name** *(See below, optional)* – The font of the axes. Example options

\[
\]

• **axes_font_size** *(int, optional)* – The font size of the axes.


• **axes_font_weight** *(See below, optional)* – The font weight of the axes. Example options

\[
\]

• **axes_x_limits** *(float or (float, float) or None, optional)* – The limits of the x axis. If float, then it sets padding on the right and left of the graph as a percentage of the curves’ width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.

• **axes_y_limits** *(float or (float, float) or None, optional)* – The limits of the y axis. If float, then it sets padding on the top and bottom of the graph as a percentage of the curves’ height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.

• **axes_x_ticks** *(list or tuple or None, optional)* – The ticks of the x axis.

• **axes_y_ticks** *(list or tuple or None, optional)* – The ticks of the y axis.

• **figure_size** *(((float, float) or None, optional)* – The size of the figure in inches.

• **render_grid** *(bool, optional)* – If True, the grid will be rendered.

• **grid_line_style** *(list or tuple or None, optional)* – The style of the grid lines.

• **grid_line_width** *(float, optional)* – The width of the grid lines.

Returns **renderer** *(menpo.visualize.GraphPlotter)* – The renderer object.

2.1. Deformable Models
plot_displacements(stat_type='mean', figure_id=None, new_figure=False, render_lines=True, line_colour='b', line_style='-', line_width=2, render_markers=True, marker_style='o', marker_size=4, marker_face_colour='b', marker_edge_colour='k', marker_edge_width=1.0, render_axes=True, axes_font_name='sans-serif', axes_font_size=10, axes_font_style='normal', axes_font_weight='normal', axes_x_limits=0.0, axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None, figure_size=(10, 6), render_grid=True, grid_line_style='–', grid_line_width=0.5)

Plot of a statistical metric of the displacement between the shape of each iteration and the shape of the previous one.

Parameters

- stat_type ({mean, median, min, max}, optional) – Specifies a statistic metric to be extracted from the displacements (see also displacements_stats() method).
- figure_id (object, optional) – The id of the figure to be used.
- new_figure (bool, optional) – If True, a new figure is created.
- render_lines (bool, optional) – If True, the line will be rendered.
- line_colour (colour or None (See below), optional) – The colour of the line. If None, the colour is sampled from the jet colormap. Example colour options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- line_style (str (See below), optional) – The style of the lines. Example options:

```
[–, ––, -, ., :]
```

- line_width (float, optional) – The width of the lines.
- render_markers (bool, optional) – If True, the markers will be rendered.
- marker_style (str (See below), optional) – The style of the markers. Example marker options

```
{., ,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
```

- marker_size (int, optional) – The size of the markers in points.
- marker_face_colour (colour or None, optional) – The face (filling) colour of the markers. If None, the colour is sampled from the jet colormap. Example colour options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- marker_edge_colour (colour or None, optional) – The edge colour of the markers. If None, the colour is sampled from the jet colormap. Example colour options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- marker_edge_width (float, optional) – The width of the markers’ edge.
• **render_axes** *(bool, optional)* – If True, the axes will be rendered.

• **axes_font_name** *(str (See below), optional)* – The font of the axes. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

• **axes_font_size** *(int, optional)* – The font size of the axes.

• **axes_font_style** *(str (See below), optional)* – The font style of the axes. Example options

```
{normal, italic, oblique}
```

• **axes_font_weight** *(str (See below), optional)* – The font weight of the axes. Example options

```
{ultralight, light, normal, regular, book, medium, roman, 
semibold, demibold, demi, bold, heavy, extra bold, black}
```

• **axes_x_limits** *(float or (float, float) or None, optional)* – The limits of the x axis. If float, then it sets padding on the right and left of the graph as a percentage of the curves’ width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.

• **axes_y_limits** *(float or (float, float) or None, optional)* – The limits of the y axis. If float, then it sets padding on the top and bottom of the graph as a percentage of the curves’ height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.

• **axes_x_ticks** *(list or tuple or None, optional)* – The ticks of the x axis.

• **axes_y_ticks** *(list or tuple or None, optional)* – The ticks of the y axis.

• **figure_size** *((float, float) or None, optional)* – The size of the figure in inches.

• **render_grid** *(bool, optional)* – If True, the grid will be rendered.

• **grid_line_style** *(\{‘-‘, ‘--’, ‘-.’, ‘:’\}, optional)* – The style of the grid lines.

• **grid_line_width** *(float, optional)* – The width of the grid lines.

Returns **renderer** *(menpo.visualize.GraphPlotter)* – The renderer object.

### plot_errors

**plot_errors** *(compute_error=None, figure_id=None, new_figure=False, render_lines=True, line_colour='b', line_style='-', line_width=2, render_markers=True, marker_style='o', marker_size=4, marker_face_colour='b', marker_edge_colour='k', marker_edge_width=1.0, render_axes=True, axes_font_name='sans-serif', axes_font_size=10, axes_font_weight='normal', axes_x_limits=0.0, axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None, figure_size=(10, 6), render_grid=True, grid_line_style='-', grid_line_width=0.5)*

Plot of the error evolution at each fitting iteration.

#### Parameters

• **compute_error** *(callable, optional)* – Callable that computes the error between the shape at each iteration and the ground truth shape.

• **figure_id** *(object, optional)* – The id of the figure to be used.

• **new_figure** *(bool, optional)* – If True, a new figure is created.

• **render_lines** *(bool, optional)* – If True, the line will be rendered.
• **lineColour** (*colour* or *None* (See below), optional) – The colour of the line. If *None*, the colour is sampled from the jet colormap. Example *colour* options are

```plaintext
[r, g, b, c, m, k, w]
```


• **lineStyle** (*str* (See below), optional) – The style of the lines. Example options:

```plaintext
[-, --, -. , :]
```

• **lineWidth** (*float*, optional) – The width of the lines.

• **renderMarkers** (*bool*, optional) – If *True*, the markers will be rendered.

• **markerStyle** (*str* (See below), optional) – The style of the markers. Example *marker* options:

```plaintext
[., ,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8]
```

• **markerSize** (*int*, optional) – The size of the markers in points.

• **markerFaceColour** (*colour* or *None*, optional) – The face (filling) colour of the markers. If *None*, the colour is sampled from the jet colormap. Example *colour* options are

```plaintext
[r, g, b, c, m, k, w]
```

• **markerEdgeColour** (*colour* or *None*, optional) – The edge colour of the markers. If *None*, the colour is sampled from the jet colormap. Example *colour* options are

```plaintext
[r, g, b, c, m, k, w]
```

• **markerEdgeWidth** (*float*, optional) – The width of the markers’ edge.

• **renderAxes** (*bool*, optional) – If *True*, the axes will be rendered.

• **axesFontName** (*str* (See below), optional) – The font of the axes. Example options:

```plaintext
[serif, sans-serif, cursive, fantasy, monospace]
```

• **axesFontSize** (*int*, optional) – The font size of the axes.

• **axesFontStyle** (*str* (See below), optional) – The font style of the axes. Example options:

```plaintext
[normal, italic, oblique]
```

• **axesFontWeight** (*str* (See below), optional) – The font weight of the axes. Example options:

```plaintext
[ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black]
```

• **axesXLimits** (*float* or (*float*, *float*) or *None*, optional) – The limits of the x axis. If *float*, then it sets padding on the right and left of the graph as a percentage of the curves’
width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.

- **axes_y_limits** (*float or (float, float) or None, optional*) – The limits of the y axis. If float, then it sets padding on the top and bottom of the graph as a percentage of the curves’ height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.

- **axes_x_ticks** (*list or tuple or None, optional*) – The ticks of the x axis.

- **axes_y_ticks** (*list or tuple or None, optional*) – The ticks of the y axis.

- **figure_size** (*((float, float) or None, optional*) – The size of the figure in inches.

- **render_grid** (*bool, optional*) – If True, the grid will be rendered.

- **grid_line_style** (*{'-', '--', '-.', ':'}, optional*) – The style of the grid lines.

- **grid_line_width** (*float, optional*) – The width of the grid lines.

Returns **renderer** (*menpo.visualize.GraphPlotter*) – The renderer object.

**reconstructed_initial_error**(compute_error=None)

Returns the error of the reconstructed initial shape of the fitting process, if the ground truth shape exists. This is the error computed based on the reconstructed initial shape.

Parameters **compute_error** (*callable, optional*) – Callable that computes the error between the reconstructed initial and ground truth shapes.

Returns **reconstructed_initial_error** (*float*) – The error that corresponds to the initial shape’s reconstruction.

Raises **ValueError** – Ground truth shape has not been set, so the reconstructed initial error cannot be computed.

**to_result**(pass_image=True, pass_initial_shape=True, pass_gt_shape=True)

Returns a **Result** instance of the object, i.e. a fitting result object that does not store the iterations. This can be useful for reducing the size of saved fitting results.

Parameters

- **pass_image** (*bool, optional*) – If True, then the image will get passed (if it exists).

- **pass_initial_shape** (*bool, optional*) – If True, then the initial shape will get passed (if it exists).

- **pass_gt_shape** (*bool, optional*) – If True, then the ground truth shape will get passed (if it exists).

Returns **result** (*Result*) – The final “lightweight” fitting result.
Visualize the fitting result. The method renders the final fitted shape and optionally the initial shape, ground truth shape and the image, if they were provided.

**Parameters**

- `figure_id` *(object, optional)* – The id of the figure to be used.
- `new_figure` *(bool, optional)* – If `True`, a new figure is created.
- `render_image` *(bool, optional)* – If `True` and the image exists, then it gets rendered.
- `render_final_shape` *(bool, optional)* – If `True`, then the final fitting shape gets rendered.
- `render_initial_shape` *(bool, optional)* – If `True` and the initial fitting shape exists, then it gets rendered.
- `render_gt_shape` *(bool, optional)* – If `True` and the ground truth shape exists, then it gets rendered.
- `subplots_enabled` *(bool, optional)* – If `True`, then the requested final, initial and ground truth shapes get rendered on separate subplots.
- `channels` *(int or list of int or all or None)* – If `int` or list of `int`, the specified channel(s) will be rendered. If `all`, all the channels will be rendered in subplots. If `None` and the image is RGB, it will be rendered in RGB mode. If `None` and the image is not RGB, it is equivalent to `all`.
- `interpolation` *(See Below, optional)* – The interpolation used to render the image. For example, if `bilinear`, the image will be smooth and if `nearest`, the image will be pixelated. Example options

```python
{none, nearest, bilinear, bicubic, spline16, spline36, hanning, hamming, hermite, kaiser, quadric, catrom, gaussian, bessel, mitchell, sinc, lanczos}
```

- `cmap_name` *(str, optional)* – If `None`, single channel and three channel images default to greyscale and rgb colormaps respectively.
- `alpha` *(float, optional)* – The alpha blending value, between 0 (transparent) and 1 (opaque).
• **masked** *(bool, optional)* – If True, then the image is rendered as masked.

• **final_marker_face_colour** *(See Below, optional)* – The face (filling) colour of the markers of the final fitting shape. Example options

```
{r, g, b, c, m, k, w}  
or  
(3, ) ndarray
```

• **final_marker_edge_colour** *(See Below, optional)* – The edge colour of the markers of the final fitting shape. Example options

```
{r, g, b, c, m, k, w}  
or  
(3, ) ndarray
```

• **final_line_colour** *(See Below, optional)* – The line colour of the final fitting shape. Example options

```
{r, g, b, c, m, k, w}  
or  
(3, ) ndarray
```

• **initial_marker_face_colour** *(See Below, optional)* – The face (filling) colour of the markers of the initial shape. Example options

```
{r, g, b, c, m, k, w}  
or  
(3, ) ndarray
```

• **initial_marker_edge_colour** *(See Below, optional)* – The edge colour of the markers of the initial shape. Example options

```
{r, g, b, c, m, k, w}  
or  
(3, ) ndarray
```

• **initial_line_colour** *(See Below, optional)* – The line colour of the initial shape. Example options

```
{r, g, b, c, m, k, w}  
or  
(3, ) ndarray
```

• **gt_marker_face_colour** *(See Below, optional)* – The face (filling) colour of the markers of the ground truth shape. Example options

```
{r, g, b, c, m, k, w}  
or  
(3, ) ndarray
```

• **gt_marker_edge_colour** *(See Below, optional)* – The edge colour of the markers of the ground truth shape. Example options

```
{r, g, b, c, m, k, w}  
or  
(3, ) ndarray
```
• `gt_line_colour (See Below, optional)` – The line colour of the ground truth shape. Example options

```
{r, g, b, c, m, k, w}
```

or

```
(3, ) ndarray
```

• `render_lines (bool or list of bool, optional)` – If True, the lines will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per shape in `(final, initial, groundtruth)` order.

• `line_style (str or list of str, optional)` – The style of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per shape in `(final, initial, groundtruth)` order. Example options:

```
{'-', '--', '-.', ':'}
```

• `line_width (float or list of float, optional)` – The width of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per shape in `(final, initial, groundtruth)` order.

• `render_markers (bool or list of bool, optional)` – If True, the markers will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per shape in `(final, initial, groundtruth)` order.

• `marker_style (str or list of str, optional)` – The style of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per shape in `(final, initial, groundtruth)` order. Example options:

```
{.,,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
```

• `marker_size (int or list of int, optional)` – The size of the markers in points. You can either provide a single value that will be used for all shapes or a list with a different value per shape in `(final, initial, groundtruth)` order.

• `marker_edge_width (float or list of float, optional)` – The width of the markers’ edge. You can either provide a single value that will be used for all shapes or a list with a different value per shape in `(final, initial, groundtruth)` order.

• `render_numbering (bool, optional)` – If True, the landmarks will be numbered.

• `numbers_horizontal_align ({center, right, left}, optional)` – The horizontal alignment of the numbers’ texts.

• `numbers_vertical_align ({center, top, bottom, baseline}, optional)` – The vertical alignment of the numbers’ texts.

• `numbers_font_name (See Below, optional)` – The font of the numbers. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

• `numbers_font_size (int, optional)` – The font size of the numbers.

• `numbers_font_style ({normal, italic, oblique}, optional)` – The font style of the numbers.

• `numbers_font_weight (See Below, optional)` – The font weight of the numbers. Example options
• **numbers_font_colour** *(See Below, optional)* – The font colour of the numbers. Example options

\{r, g, b, c, m, k, w\}

or

\((3, )\) ndarray

• **render_legend** *(bool, optional)* – If True, the legend will be rendered.

• **legend_title** *(str, optional)* – The title of the legend.

• **legend_font_name** *(See below, optional)* – The font of the legend. Example options

\{serif, sans-serif, cursive, fantasy, monospace\}

• **legend_font_style** *(\{normal, italic, oblique\}, optional)* – The font style of the legend.

• **legend_font_size** *(int, optional)* – The font size of the legend.

• **legend_font_weight** *(See Below, optional)* – The font weight of the legend. Example options

\{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black\}

• **legend_marker_scale** *(float, optional)* – The relative size of the legend markers with respect to the original.

• **legend_location** *(int, optional)* – The location of the legend. The predefined values are:

<table>
<thead>
<tr>
<th>Location</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>'best'</td>
<td>0</td>
</tr>
<tr>
<td>'upper right'</td>
<td>1</td>
</tr>
<tr>
<td>'upper left'</td>
<td>2</td>
</tr>
<tr>
<td>'lower left'</td>
<td>3</td>
</tr>
<tr>
<td>'lower right'</td>
<td>4</td>
</tr>
<tr>
<td>'right'</td>
<td>5</td>
</tr>
<tr>
<td>'center left'</td>
<td>6</td>
</tr>
<tr>
<td>'center right'</td>
<td>7</td>
</tr>
<tr>
<td>'lower center'</td>
<td>8</td>
</tr>
<tr>
<td>'upper center'</td>
<td>9</td>
</tr>
<tr>
<td>'center'</td>
<td>10</td>
</tr>
</tbody>
</table>

• **legend_bbox_to_anchor** *(\{float, float\} tuple, optional)* – The bbox that the legend will be anchored.

• **legend_border_axes_pad** *(float, optional)* – The pad between the axes and legend border.

• **legend_n_columns** *(int, optional)* – The number of the legend’s columns.

• **legend_horizontal_spacing** *(float, optional)* – The spacing between the columns.
• **legend_vertical_spacing** (*float*, optional) – The vertical space between the legend entries.

• **legend_border** (*bool*, optional) – If *True*, a frame will be drawn around the legend.

• **legend_border_padding** (*float*, optional) – The fractional whitespace inside the legend border.

• **legend_shadow** (*bool*, optional) – If *True*, a shadow will be drawn behind legend.

• **legend_rounded_corners** (*bool*, optional) – If *True*, the frame’s corners will be rounded (fancybox).

• **render_axes** (*bool*, optional) – If *True*, the axes will be rendered.

• **axes_font_name** (*See Below*, optional) – The font of the axes. Example options

  ```
  [serif, sans-serif, cursive, fantasy, monospace]
  ```

• **axes_font_size** (*int*, optional) – The font size of the axes.

• **axes_font_style** (*{normal, italic, oblique}*), optional) – The font style of the axes.

• **axes_font_weight** (*See Below*, optional) – The font weight of the axes. Example options

  ```
  [ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black]
  ```

• **axes_x_limits** (*float* or (*float*, *float*) or *None*, optional) – The limits of the x axis. If *float*, then it sets padding on the right and left of the Image as a percentage of the Image’s width. If *tuple* or *list*, then it defines the axis limits. If *None*, then the limits are set automatically.

• **axes_y_limits** (*(*float*, *float*)* tuple or *None*, optional) – The limits of the y axis. If *float*, then it sets padding on the top and bottom of the Image as a percentage of the Image’s height. If *tuple* or *list*, then it defines the axis limits. If *None*, then the limits are set automatically.

• **axes_x_ticks** (*list* or *tuple* or *None*, optional) – The ticks of the x axis.

• **axes_y_ticks** (*list* or *tuple* or *None*, optional) – The ticks of the y axis.

• **figure_size** (*(*float*, *float*)* tuple or *None* optional) – The size of the figure in inches.

Returns **renderer** (*class*) – The renderer object.
`view_iterations` (figure_id=None, new_figure=False, iters=None, render_image=True, subplots_enabled=False, channels=None, interpolation='bilinear', cmap_name=None, alpha=1.0, masked=True, render_lines=True, line_style='-', line_width=2, line_colour=None, render_markers=True, marker_edge_colour=None, marker_face_colour=None, marker_style='o', marker_size=4, marker_edge_width=1.0, render_numbering=False, numbers_horizontal_align='center', numbers_vertical_align='bottom', numbers_font_name='sans-serif', numbers_font_size=10, numbers_font_style='normal', numbers_font_weight='normal', numbers_font_colour='k', render_legend=True, legend_title='', legend_font_name='sans-serif', legend_font_style='normal', legend_font_size=10, legend_font_weight='normal', legend_marker_scale=None, legend_location=2, legend_bbox_to_anchor=(1.05, 1.0), legend_border_axes_pad=None, legend_n_columns=1, legend_horizontal_spacing=None, legend_vertical_spacing=None, legend_border=True, legend_border_padding=None, legend_shadow=False, legend_rounded_corners=False, render_axes=False, axes_font_name='sans-serif', axes_font_size=10, axes_font_style='normal', axes_font_weight='normal', axes_x_limits=None, axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None, figure_size=(7, 7))

Visualize the iterations of the fitting process.

**Parameters**

- **figure_id** (object, optional) – The id of the figure to be used.
- **new_figure** (bool, optional) – If True, a new figure is created.
- **iters** (int or list of int or None, optional) – The iterations to be visualized. If None, then all the iterations are rendered.

<table>
<thead>
<tr>
<th>No.</th>
<th>Visualised shape</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>self.initial_shape</td>
<td>Initial shape</td>
</tr>
<tr>
<td>1</td>
<td>self.reconstructed_initial_shape</td>
<td>Reconstructed initial</td>
</tr>
<tr>
<td>2</td>
<td>self.shapes[2]</td>
<td>Iteration 1</td>
</tr>
<tr>
<td>i</td>
<td>self.shapes[i]</td>
<td>Iteration i</td>
</tr>
<tr>
<td>n_iters+1</td>
<td>self.final_shape</td>
<td>Final shape</td>
</tr>
</tbody>
</table>

- **render_image** (bool, optional) – If True and the image exists, then it gets rendered.
- **subplots_enabled** (bool, optional) – If True, then the requested final, initial and ground truth shapes get rendered on separate subplots.
- **channels** (int or list of int or all or None) – If int or list of int, the specified channel(s) will be rendered. If all, all the channels will be rendered in subplots. If None and the image is RGB, it will be rendered in RGB mode. If None and the image is not RGB, it is equivalent to all.
- **interpolation** (str (See Below), optional) – The interpolation used to render the image. For example, if bilinear, the image will be smooth and if nearest, the image will be pixelated. Example options

```python
{none, nearest, bilinear, bicubic, spline16, spline36, hanning, hamming, hermite, kaiser, quadric, catrom, gaussian, bessel, mitchell, sinc, lanczos}
```

- **cmap_name** (str, optional.) – If None, single channel and three channel images default to greyscale and rgb colormaps respectively.

2.1. Deformable Models 129
• **alpha** *(float, optional)* – The alpha blending value, between 0 (transparent) and 1 (opaque).

• **masked** *(bool, optional)* – If True, then the image is rendered as masked.

• **render_lines** *(bool or list of bool, optional)* – If True, the lines will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.

• **line_style** *(str or list of str)* (See below), optional – The style of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options:

  
  ```
  (-, --, -., :)
  ```

• **line_width** *(float or list of float, optional)* – The width of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.

• **line_colour** *(colour or list of colour)* (See Below), optional – The colour of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

  ```
  {r, g, b, c, m, k, w}
  or
  (3, ) ndarray
  ```

• **render_markers** *(bool or list of bool, optional)* – If True, the markers will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.

• **marker_style** *(str or list of str)* (See below), optional – The style of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

  ```
  {., ,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
  ```

• **marker_size** *(int or list of int, optional)* – The size of the markers in points. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.

• **marker_edge_colour** *(colour or list of colour)* (See Below), optional – The edge colour of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

  ```
  {r, g, b, c, m, k, w}
  or
  (3, ) ndarray
  ```

• **marker_face_colour** *(colour or list of colour)* (See Below), optional – The face (filling) colour of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

  ```
  {r, g, b, c, m, k, w}
  or
  (3, ) ndarray
  ```

• **marker_edge_width** *(float or list of float, optional)* – The width of the markers’ edge. You can either provide a single value that will be used for all shapes or a list with a different
value per iteration shape.

- **render_numbering** (*bool*, optional) – If True, the landmarks will be numbered.

- **numbers_horizontal_align** (*str* (See below), optional) – The horizontal alignment of the numbers’ texts. Example options
  
  ```
  [center, right, left]
  ```

- **numbers_vertical_align** (*str* (See below), optional) – The vertical alignment of the numbers’ texts. Example options
  
  ```
  [center, top, bottom, baseline]
  ```

- **numbers_font_name** (*str* (See below), optional) – The font of the numbers. Example options
  
  ```
  [serif, sans-serif, cursive, fantasy, monospace]
  ```

- **numbers_font_size** (*int*, optional) – The font size of the numbers.

- **numbers_font_style** (*str* (See below), optional) – The font style of the numbers.

- **numbers_font_weight** (*str* (See below), optional) – The font weight of the numbers. Example options
  
  ```
  [ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black]
  ```

- **numbers_font_colour** (*See Below, optional*) – The font colour of the numbers. Example options
  
  ```
  [r, g, b, c, m, k, w] or (3, ) ndarray
  ```

- **render_legend** (*bool*, optional) – If True, the legend will be rendered.

- **legend_title** (*str*, optional) – The title of the legend.

- **legend_font_name** (*See below, optional*) – The font of the legend. Example options
  
  ```
  [serif, sans-serif, cursive, fantasy, monospace]
  ```

- **legend_font_style** (*str* (See below), optional) – The font style of the legend. Example options
  
  ```
  [normal, italic, oblique]
  ```

- **legend_font_size** (*int*, optional) – The font size of the legend.

- **legend_font_weight** (*str* (See below), optional) – The font weight of the legend. Example options
  
  ```
  [ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black]
  ```

2.1. Deformable Models
• **legend_marker_scale** *(float, optional)* – The relative size of the legend markers with respect to the original.

• **legend_location** *(int, optional)* – The location of the legend. The predefined values are:

<table>
<thead>
<tr>
<th>Location</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>'best'</td>
<td>0</td>
</tr>
<tr>
<td>'upper right'</td>
<td>1</td>
</tr>
<tr>
<td>'upper left'</td>
<td>2</td>
</tr>
<tr>
<td>'lower left'</td>
<td>3</td>
</tr>
<tr>
<td>'lower right'</td>
<td>4</td>
</tr>
<tr>
<td>'right'</td>
<td>5</td>
</tr>
<tr>
<td>'center left'</td>
<td>6</td>
</tr>
<tr>
<td>'center right'</td>
<td>7</td>
</tr>
<tr>
<td>'lower center'</td>
<td>8</td>
</tr>
<tr>
<td>'upper center'</td>
<td>9</td>
</tr>
<tr>
<td>'center'</td>
<td>10</td>
</tr>
</tbody>
</table>

• **legend_bbox_to_anchor** *((float, float) tuple, optional)* – The bbox that the legend will be anchored.

• **legend_border_axes_pad** *(float, optional)* – The pad between the axes and legend border.

• **legend_n_columns** *(int, optional)* – The number of the legend’s columns.

• **legend_horizontal_spacing** *(float, optional)* – The spacing between the columns.

• **legend_vertical_spacing** *(float, optional)* – The vertical space between the legend entries.

• **legend_border** *(bool, optional)* – If True, a frame will be drawn around the legend.

• **legend_border_padding** *(float, optional)* – The fractional whitespace inside the legend border.

• **legend_shadow** *(bool, optional)* – If True, a shadow will be drawn behind legend.

• **legend_rounded_corners** *(bool, optional)* – If True, the frame’s corners will be rounded (fancybox).

• **render_axes** *(bool, optional)* – If True, the axes will be rendered.

• **axes_font_name** *(str (See below), optional)* – The font of the axes. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

• **axes_font_size** *(int, optional)* – The font size of the axes.

• **axes_font_style** *((normal, italic, oblique), optional)* – The font style of the axes.

• **axes_font_weight** *(str (See below), optional)* – The font weight of the axes. Example options

```
{ultralight, light, normal, regular, book, medium, roman,
   semibold, demibold, demi, bold, heavy, extra bold, black}
```

• **axes_x_limits** *(float or (float, float) or None, optional)* – The limits of the x axis. If *float*, then it sets padding on the right and left of the Image as a percentage of the Image’s
width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.

- **axes_y_limits** ((float, float) tuple or None, optional) – The limits of the y axis. If float, then it sets padding on the top and bottom of the Image as a percentage of the Image’s height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.

- **axes_x_ticks** (list or tuple or None, optional) – The ticks of the x axis.

- **axes_y_ticks** (list or tuple or None, optional) – The ticks of the y axis.

- **figure_size** ((float, float) tuple or None optional) – The size of the figure in inches.

    Returns **renderer** (class) – The renderer object.

**view_widget** (figure_size=(7, 7))

Visualizes the result object using an interactive widget.

**Parameters** figure_size ((int, int), optional) – The initial size of the rendered figure.

**property appearance_costs**

Returns a list with the appearance cost per iteration. It returns None if the costs are not computed.

**Type** list of float or None

**property costs**

Returns a list with the cost per iteration. It returns None if the costs are not computed.

**Type** list of float or None

**property deformation_costs**

Returns a list with the deformation cost per iteration. It returns None if the costs are not computed.

**Type** list of float or None

**property final_shape**

Returns the final shape of the fitting process.

**Type** menpo.shape.PointCloud

**property gt_shape**

Returns the ground truth shape associated with the image. In case there is not an attached ground truth shape, then None is returned.

**Type** menpo.shape.PointCloud or None

**property image**

Returns the image that the fitting was applied on, if it was provided. Otherwise, it returns None.

**Type** menpo.shape.Image or subclass or None

**property initial_shape**

Returns the initial shape that was provided to the fitting method to initialise the fitting process. In case the initial shape does not exist, then None is returned.

**Type** menpo.shape.PointCloud or None

**property is_iterative**

Flag whether the object is an iterative fitting result.

**Type** bool

**property n_iters**

Returns the total number of iterations of the fitting process.
Type int

property reconstructed_initial_shape
Returns the initial shape’s reconstruction with the shape model that was used to initialise the iterative optimisation process.

Type menpo.shape.PointCloud

property shape_parameters
Returns the list of shape parameters obtained at each iteration of the fitting process. The list includes the parameters of the reconstructed_initial_shape and final_shape.

Type list of (n_params,) ndarray

property shapes
Returns the list of shapes obtained at each iteration of the fitting process. The list includes the initial_shape (if it exists), reconstructed_initial_shape and final_shape.

Type list of menpo.shape.PointCloud

2.1.3 menpofit.atm

Active Template Model

ATM is a generative model that performs deformable alignment between a template image and a test image with respect to a statistical parametric shape model. MenpoFit has several ATMs which differ in the manner that they compute the warp (thus represent the appearance features).

ATM

class menpofit.atm.base.ATM(template, shapes, group=None, holistic_features=<function no_op>, reference_shape=None, diagonal=None, scales=(0.5, 1.0), transform=class 'menpofit.transform.piecewiseaffine.DifferentiablePiecewiseAffine', shape_model_cls=class 'menpofit.modelinstance.OrthoPDM', max_shape_components=None, verbose=False, batch_size=None)

Bases: object

Class for training a multi-scale holistic Active Template Model.

Parameters

• template (menpo.image.Image) – The template image.

• shapes (list of menpo.shape.PointCloud) – The list of training shapes.

• group (str or None, optional) – The landmark group of the template that will be used to train the ATM. If None and the template only has a single landmark group, then that is the one that will be used.

• holistic_features (closure or list of closure, optional) – The features that will be extracted from the training images. Note that the features are extracted before warping the images to the reference shape. If list, then it must define a feature function per scale. Please refer to menpo.feature for a list of potential features.

• reference_shape (menpo.shape.PointCloud or None, optional) – The reference shape that will be used for building the ATM. The purpose of the reference shape is to normalise the size of the training images. The normalization is performed by rescaling all the training images so that the scale of their ground truth shapes matches the scale of the reference shape.
Note that the reference shape is rescaled with respect to the diagonal before performing the normalisation. If None, then the mean shape will be used.

- **diagonal** (int or None, optional) – This parameter is used to rescale the reference shape so that the diagonal of its bounding box matches the provided value. In other words, this parameter controls the size of the model at the highest scale. If None, then the reference shape does not get rescaled.

- **scales** (float or tuple of float, optional) – The scale value of each scale. They must be provided in ascending order, i.e. from lowest to highest scale. If float, then a single scale is assumed.

- **transform** (subclass of DL and DX, optional) – A differential warp transform object, e.g. DifferentiablePiecewiseAffine or DifferentiableThinPlateSplines.

- **shape_model_cls** (subclass of PDM, optional) – The class to be used for building the shape model. The most common choice is OrthoPDM.

- **max_shape_components** (int, float, list of those or None, optional) – The number of shape components to keep. If int, then it sets the exact number of components. If float, then it defines the variance percentage that will be kept. If list, then it should define a value per scale. If a single number, then this will be applied to all scales. If None, then all the components are kept. Note that the unused components will be permanently trimmed.

- **verbose** (bool, optional) – If True, then the progress of building the ATM will be printed.

- **batch_size** (int or None, optional) – If an int is provided, then the training is performed in an incremental fashion on image batches of size equal to the provided value. If None, then the training is performed directly on all the images.

**References**

**build_fitter_interfaces** *(sampling)*

Method that builds the correct Lucas-Kanade fitting interface.

**Parameters**

- **sampling** *(list of int or ndarray or None)* – It defines a sampling mask per scale. If int, then it defines the sub-sampling step of the sampling mask. If ndarray, then it explicitly defines the sampling mask. If None, then no sub-sampling is applied.

**Returns**

- **fitter_interfaces** *(list)* – The list of Lucas-Kanade interface per scale.

**increment** *(template, shapes, group=None, shape_forgetting_factor=1.0, verbose=False, batch_size=None)*

Method to increment the trained ATM with a new set of training shapes and a new template.

**Parameters**

- **template** *(menpo.image.Image)* – The template image.

- **shapes** *(list of menpo.shape.PointCloud)* – The list of training shapes.

- **group** *(str or None, optional)* – The landmark group of the template that will be used to train the ATM. If None and the template only has a single landmark group, then that is the one that will be used.

- **shape_forgetting_factor** *(list of float, optional)* – Forgetting factor that weights the relative contribution of new samples vs old samples for the shape model. If 1.0, all samples are weighted equally and, hence, the result is the exact same as performing batch PCA on the concatenated list of old and new samples. If <1.0, more emphasis is put on the new samples.
• **verbose** *(bool, optional)* – If True, then the progress of building the ATM will be printed.

• **batch_size** *(int or None, optional)* – If an int is provided, then the training is performed in an incremental fashion on image batches of size equal to the provided value. If None, then the training is performed directly on the all the images.

**instance** *(shape_weights=None, scale_index=-1)*

Generates a novel ATM instance given a set of shape weights. If no weights are provided, the mean ATM instance is returned.

Parameters

• **shape_weights** *(ndarray or list or None, optional)* – The weights of the shape model that will be used to create a novel shape instance. If None, the weights are assumed to be zero, thus the mean shape is used.

• **scale_index** *(int, optional)* – The scale to be used.

Returns *image* *(menpo.image.Image)* – The ATM instance.

**random_instance** *(scale_index=-1)*

Generates a random instance of the ATM.

Parameters **scale_index** *(int, optional)* – The scale to be used.

Returns *image* *(menpo.image.Image)* – The ATM instance.

**view_atm_widget** *(n_shape_parameters=5, parameters_bounds=(-3.0, 3.0), mode='multiple', figure_size=(7, 7))*

Visualizes the ATM using an interactive widget.

Parameters

• **n_shape_parameters** *(int or list of int or None, optional)* – The number of shape principal components to be used for the parameters sliders. If int, then the number of sliders per scale is the minimum between n_parameters and the number of active components per scale. If list of int, then a number of sliders is defined per scale. If None, all the active components per scale will have a slider.

• **parameters_bounds** *(float, float), optional)* – The minimum and maximum bounds, in std units, for the sliders.

• **mode** *(single, multiple, optional)* – If 'single', only a single slider is constructed along with a drop down menu. If 'multiple', a slider is constructed for each parameter.

• **figure_size** *(int, int), optional)* – The size of the rendered figure.

**view_shape_models_widget** *(n_parameters=5, parameters_bounds=(-3.0, 3.0), mode='multiple', figure_size=(7, 7))*

Visualizes the shape models of the ATM object using an interactive widget.

Parameters

• **n_parameters** *(int or list of int or None, optional)* – The number of shape principal components to be used for the parameters sliders. If int, then the number of sliders per scale is the minimum between n_parameters and the number of active components per scale. If list of int, then a number of sliders is defined per scale. If None, all the active components per scale will have a slider.

• **parameters_bounds** *(float, float), optional)* – The minimum and maximum bounds, in std units, for the sliders.
• **mode** ({'single', 'multiple'}, optional) – If 'single', only a single slider is constructed along with a drop down menu. If 'multiple', a slider is constructed for each parameter.

• **figure_size** ((int, int), optional) – The size of the rendered figure.

**property n_scales**

Returns the number of scales.

**Type** int

---

**HolisticATM**

```python
menpofit.atm.HolisticATM
alias of ATM
```

---

**MaskedATM**

```python
class menpofit.atm.MaskedATM(template, shapes, group=None, holistic_features=<function no_op>, reference_shape=None, diagonal=None, scales=(0.5, 1.0), patch_shape=(17, 17), max_shape_components=None, verbose=False, batch_size=None)
```

**Bases:** ATM

Class for training a multi-scale patch-based Masked Active Template Model. The appearance of this model is formulated by simply masking an image with a patch-based mask.

**Parameters**

- **template** (*menpo.image.Image*) – The template image.
- **shapes** (*list of menpo.shape.PointCloud*) – The list of training shapes.
- **group** (*str or None*, optional) – The landmark group of the template that will be used to train the ATM. If None and the template only has a single landmark group, then that is the one that will be used.
- **holistic_features** (*closure or list of closure*, optional) – The features that will be extracted from the training images. Note that the features are extracted before warping the images to the reference shape. If list, then it must define a feature function per scale. Please refer to menpo.feature for a list of potential features.
- **reference_shape** (*menpo.shape.PointCloud or None*, optional) – The reference shape that will be used for building the ATM. The purpose of the reference shape is to normalise the size of the training images. The normalization is performed by rescaling all the training images so that the scale of their ground truth shapes matches the scale of the reference shape. Note that the reference shape is rescaled with respect to the `diagonal` before performing the normalisation. If None, then the mean shape will be used.
- **diagonal** (*int or None*, optional) – This parameter is used to rescale the reference shape so that the diagonal of its bounding box matches the provided value. In other words, this parameter controls the size of the model at the highest scale. If None, then the reference shape does not get rescaled.
- **scales** (*float or tuple of float*, optional) – The scale value of each scale. They must provided in ascending order, i.e. from lowest to highest scale. If float, then a single scale is assumed.
• **patch_shape**
  
  `((int, int), optional)` – The size of the patches of the mask that is used to sample the appearance vectors.

• **max_shape_components**

  `(int, float, list of those or None, optional)` – The number of shape components to keep. If `int`, then it sets the exact number of components. If `float`, then it defines the variance percentage that will be kept. If `list`, then it should define a value per scale. If a single number, then this will be applied to all scales. If `None`, then all the components are kept. Note that the unused components will be permanently trimmed.

• **verbose**

  `(bool, optional)` – If `True`, then the progress of building the ATM will be printed.

• **batch_size**

  `(int or None, optional)` – If an `int` is provided, then the training is performed in an incremental fashion on image batches of size equal to the provided value. If `None`, then the training is performed directly on the all the images.

---

### build_fitter_interfaces

Method that builds the correct Lucas-Kanade fitting interface.

**Parameters**

- **sampling**

  `(list of int or ndarray or None)` – It defines a sampling mask per scale. If `int`, then it defines the sub-sampling step of the sampling mask. If `ndarray`, then it explicitly defines the sampling mask. If `None`, then no sub-sampling is applied.

**Returns**

- **fitter_interfaces**

  `(list)` – The list of Lucas-Kanade interface per scale.

---

### increment

Method to increment the trained ATM with a new set of training shapes and a new template.

**Parameters**

- **template**

  `(menpo.image.Image)` – The template image.

- **shapes**

  `(list of menpo.shape.PointCloud)` – The list of training shapes.

- **group**

  `(str or None, optional)` – The landmark group of the `template` that will be used to train the ATM. If `None` and the `template` only has a single landmark group, then that is the one that will be used.

- **shape_forgetting_factor**

  `([0.0, 1.0], float, optional)` – Forgetting factor that weights the relative contribution of new samples vs old samples for the shape model. If `1.0`, all samples are weighted equally and, hence, the result is the exact same as performing batch PCA on the concatenated list of old and new simples. If `<1.0`, more emphasis is put on the new samples.

- **verbose**

  `(bool, optional)` – If `True`, then the progress of building the ATM will be printed.

- **batch_size**

  `(int or None, optional)` – If an `int` is provided, then the training is performed in an incremental fashion on image batches of size equal to the provided value. If `None`, then the training is performed directly on the all the images.

---

### instance

Generates a novel ATM instance given a set of shape weights. If no weights are provided, the mean ATM instance is returned.

**Parameters**

- **shape_weights**

  `((n_weights,)) ndarray or list or None, optional)` – The weights of the shape model that will be used to create a novel shape instance. If `None`, the weights are assumed to be zero, thus the mean shape is used.

- **scale_index**

  `(int, optional)` – The scale to be used.

**Returns**

- **image**

  `(menpo.image.Image)` – The ATM instance.
random_instance (scale_index=-1)
Generates a random instance of the ATM.

Parameters

scale_index (int, optional) – The scale to be used.

Returns

image (menpo.image.Image) – The ATM instance.

view_atm_widget (n_shape_parameters=5, parameters_bounds=(-3.0, 3.0), mode='multiple', figure_size=(7, 7))
Visualizes the ATM using an interactive widget.

Parameters

• n_shape_parameters (int or list of int or None, optional) – The number of shape principal components to be used for the parameters sliders. If int, then the number of sliders per scale is the minimum between n_parameters and the number of active components per scale. If list of int, then a number of sliders is defined per scale. If None, all the active components per scale will have a slider.

• parameters_bounds ((float, float), optional) – The minimum and maximum bounds, in std units, for the sliders.

• mode ({'single', 'multiple'}, optional) – If 'single', only a single slider is constructed along with a drop down menu. If 'multiple', a slider is constructed for each parameter.

• figure_size ((int, int), optional) – The size of the rendered figure.

view_shape_models_widget (n_parameters=5, parameters_bounds=(-3.0, 3.0), mode='multiple', figure_size=(7, 7))
Visualizes the shape models of the ATM object using an interactive widget.

Parameters

• n_parameters (int or list of int or None, optional) – The number of shape principal components to be used for the parameters sliders. If int, then the number of sliders per scale is the minimum between n_parameters and the number of active components per scale. If list of int, then a number of sliders is defined per scale. If None, all the active components per scale will have a slider.

• parameters_bounds ((float, float), optional) – The minimum and maximum bounds, in std units, for the sliders.

• mode ({'single', 'multiple'}, optional) – If 'single', only a single slider is constructed along with a drop down menu. If 'multiple', a slider is constructed for each parameter.

• figure_size ((int, int), optional) – The size of the rendered figure.

property n_scales
Returns the number of scales.

Type int

LinearATM

class menpofit.atm.LinearATM (template, shapes, group=None, holistic_features=<function no_op>, reference_shape=None, diagonal=None, scales=(0.5, 1.0), transform=<class 'menpofit.transform.thinsplatesplines.DifferentiableThinPlateSplines'>, max_shape_components=None, verbose=False, batch_size=None)

Bases: ATM
Class for training a multi-scale Linear Active Template Model.

Parameters

- **template** *(menpo.image.Image)* – The template image.
- **shapes** *(list of menpo.shape.PointCloud)* – The list of training shapes.
- **group** *(str or None, optional)* – The landmark group of the template that will be used to train the ATM. If None and the template only has a single landmark group, then that is the one that will be used.
- **holistic_features** *(closure or list of closure, optional)* – The features that will be extracted from the training images. Note that the features are extracted before warping the images to the reference shape. If list, then it must define a feature function per scale. Please refer to menpo.feature for a list of potential features.
- **reference_shape** *(menpo.shape.PointCloud or None, optional)* – The reference shape that will be used for building the ATM. The purpose of the reference shape is to normalise the size of the training images. The normalization is performed by rescaling all the training images so that the scale of their ground truth shapes matches the scale of the reference shape. Note that the reference shape is rescaled with respect to the **diagonal** before performing the normalisation. If None, then the mean shape will be used.
- **diagonal** *(int or None, optional)* – This parameter is used to rescale the reference shape so that the diagonal of its bounding box matches the provided value. In other words, this parameter controls the size of the model at the highest scale. If None, then the reference shape does not get rescaled.
- **scales** *(float or tuple of float, optional)* – The scale value of each scale. They must provided in ascending order, i.e. from lowest to highest scale. If float, then a single scale is assumed.
- **transform** *(subclass of DL and DX, optional)* – A differential warp transform object, e.g. DifferentiablePiecewiseAffine or DifferentiableThinPlateSplines.
- **max_shape_components** *(int, float, list of those or None, optional)* – The number of shape components to keep. If int, then it sets the exact number of components. If float, then it defines the variance percentage that will be kept. If list, then it should define a value per scale. If a single number, then this will be applied to all scales. If None, then all the components are kept. Note that the unused components will be permanently trimmed.
- **verbose** *(bool, optional)* – If True, then the progress of building the ATM will be printed.
- **batch_size** *(int or None, optional)* – If an int is provided, then the training is performed in an incremental fashion on image batches of size equal to the provided value. If None, then the training is performed directly on the all the images.

**build_fitter_interfaces**(sampling)

Method that builds the correct Lucas-Kanade fitting interface.

Parameters **sampling** *(list of int or ndarray or None)* – It defines a sampling mask per scale. If int, then it defines the sub-sampling step of the sampling mask. If ndarray, then it explicitly defines the sampling mask. If None, then no sub-sampling is applied.

Returns **fitter_interfaces** *(list)* – The list of Lucas-Kanade interface per scale.

**increment**(template, shapes, group=None, shape_forgetting_factor=1.0, verbose=False, batch_size=None)

Method to increment the trained ATM with a new set of training shapes and a new template.

Parameters
• **template** (*menpo.image.Image*) – The template image.

• **shapes** (*list of menpo.shape.PointCloud*) – The list of training shapes.

• **group** (*str or None, optional*) – The landmark group of the template that will be used to train the ATM. If None and the template only has a single landmark group, then that is the one that will be used.

• **shape_forgetting_factor** (*[0.0, 1.0] float, optional*) – Forgetting factor that weights the relative contribution of new samples vs old samples for the shape model. If 1.0, all samples are weighted equally and, hence, the result is the exact same as performing batch PCA on the concatenated list of old and new samples. If < 1.0, more emphasis is put on the new samples.

• **verbose** (*bool, optional*) – If True, then the progress of building the ATM will be printed.

• **batch_size** (*int or None, optional*) – If an int is provided, then the training is performed in an incremental fashion on image batches of size equal to the provided value. If None, then the training is performed directly on all the images.

**instance** (*shape_weights=None, scale_index=-1*)
Generates a novel ATM instance given a set of shape weights. If no weights are provided, the mean ATM instance is returned.

**Parameters**

• **shape_weights** (*ndarray or list or None, optional*) – The weights of the shape model that will be used to create a novel shape instance. If None, the weights are assumed to be zero, thus the mean shape is used.

• **scale_index** (*int, optional*) – The scale to be used.


**random_instance** (*scale_index=-1*)
Generates a random instance of the ATM.

**Parameters** scale_index (*int, optional*) – The scale to be used.


**view_atm_widget** (*n_shape_parameters=5, parameters_bounds=(-3.0, 3.0), mode='multiple', figure_size=(7, 7]*)
Visualizes the ATM using an interactive widget.

**Parameters**

• **n_shape_parameters** (*int or list of int or None, optional*) – The number of shape principal components to be used for the parameters sliders. If int, then the number of sliders per scale is the minimum between n_parameters and the number of active components per scale. If list of int, then a number of sliders is defined per scale. If None, all the active components per scale will have a slider.

• **parameters_bounds** (*((float, float), optional*) – The minimum and maximum bounds, in std units, for the sliders.

• **mode** (*{single, multiple}, optional*) – If 'single', only a single slider is constructed along with a drop down menu. If 'multiple', a slider is constructed for each parameter.

• **figure_size** (*((int, int), optional*) – The size of the rendered figure.
view_shape_models_widget (n_parameters=5, parameters_bounds=(-3.0, 3.0), mode='multiple', figure_size=(7, 7))

Visualizes the shape models of the ATM object using an interactive widget.

Parameters

• **n_parameters** *(int or list of int or None, optional)* — The number of shape principal components to be used for the parameters sliders. If *int*, then the number of sliders per scale is the minimum between *n_parameters* and the number of active components per scale. If *list of int*, then a number of sliders is defined per scale. If *None*, all the active components per scale will have a slider.

• **parameters_bounds** *((float, float), optional)* — The minimum and maximum bounds, in std units, for the sliders.

• **mode** *(single, multiple, optional)* — If *'single'*, only a single slider is constructed along with a drop down menu. If *'multiple'*, a slider is constructed for each parameter.

• **figure_size** *((int, int), optional)* — The size of the rendered figure.

property n_scales

Returns the number of scales.

Type  *int*

LinearMaskedATM

class menpo.fit.LinearMaskedATM (template, shapes, group=None, holistic_features=<function no_op>, reference_shape=None, diagonal=None, scales=(0.5, 1.0), patch_shape=(17, 17), max_shape_components=None, verbose=False, batch_size=None)

Bases: *ATM*

Class for training a multi-scale Linear Masked Active Template Model.

Parameters

• **template** *(menpo.image.Image)* — The template image.

• **shapes** *(list of menpo.shape.PointCloud)* — The list of training shapes.

• **group** *(str or None, optional)* — The landmark group of the *template* that will be used to train the ATM. If *None* and the *template* only has a single landmark group, then that is the one that will be used.

• **holistic_features** *(closure or list of closure, optional)* — The features that will be extracted from the training images. Note that the features are extracted before warping the images to the reference shape. If *list*, then it must define a feature function per scale. Please refer to *menpo.feature* for a list of potential features.

• **reference_shape** *(menpo.shape.PointCloud or None, optional)* — The reference shape that will be used for building the ATM. The purpose of the reference shape is to normalise the size of the training images. The normalization is performed by rescaling all the training images so that the scale of their ground truth shapes matches the scale of the reference shape. Note that the reference shape is rescaled with respect to the *diagonal* before performing the normalisation. If *None*, then the mean shape will be used.

• **diagonal** *(int or None, optional)* — This parameter is used to rescale the reference shape so that the diagonal of its bounding box matches the provided value. In other words, this
parameter controls the size of the model at the highest scale. If `None`, then the reference shape does not get rescaled.

- **scales** *(float or tuple of float, optional)* – The scale value of each scale. They must provided in ascending order, i.e. from lowest to highest scale. If `float`, then a single scale is assumed.

- **patch_shape** *(int, int or list of (int, int), optional)* – The shape of the patches of the mask that is used to extract the appearance vectors. If a `list` is provided, then it defines a patch shape per scale.

- **max_shape_components** *(int, float, list of those or None, optional)* – The number of shape components to keep. If `int`, then it sets the exact number of components. If `float`, then it defines the variance percentage that will be kept. If `list`, then it should define a value per scale. If a single number, then this will be applied to all scales. If `None`, then all the components are kept. Note that the unused components will be permanently trimmed.

- **verbose** *(bool, optional)* – If `True`, then the progress of building the ATM will be printed.

- **batch_size** *(int or None, optional)* – If an `int` is provided, then the training is performed in an incremental fashion on image batches of size equal to the provided value. If `None`, then the training is performed directly on all the images.

### build_fitter_interfaces *(sampling)*
Method that builds the correct Lucas-Kanade fitting interface.

**Parameters**
- **sampling** *(list of int or ndarray or None)* – It defines a sampling mask per scale. If `int`, then it defines the sub-sampling step of the sampling mask. If `ndarray`, then it explicitly defines the sampling mask. If `None`, then no sub-sampling is applied.

**Returns**
- **fitter_interfaces** *(list)* – The list of Lucas-Kanade interface per scale.

### increment *(template, shapes, group=None, shape_forgetting_factor=1.0, verbose=False, batch_size=None)*
Method to increment the trained ATM with a new set of training shapes and a new template.

**Parameters**
- **template** *(menpo.image.Image)* – The template image.
- **shapes** *(list of menpo.shape.PointCloud)* – The list of training shapes.
- **group** *(str or None, optional)* – The landmark group of the template that will be used to train the ATM. If `None` and the template only has a single landmark group, then that is the one that will be used.
- **shape_forgetting_factor** *(float or list)* – Forgetting factor that weights the relative contribution of new samples vs old samples for the shape model. If 1.0, all samples are weighted equally and, hence, the result is the exact same as performing batch PCA on the concatenated list of old and new simples. If <1.0, more emphasis is put on the new samples.
- **verbose** *(bool, optional)* – If `True`, then the progress of building the ATM will be printed.
- **batch_size** *(int or None, optional)* – If an `int` is provided, then the training is performed in an incremental fashion on image batches of size equal to the provided value. If `None`, then the training is performed directly on all the images.

### instance *(shape_weights=None, scale_index=-1)*
Generates a novel ATM instance given a set of shape weights. If no weights are provided, the mean ATM instance is returned.

---

2.1. Deformable Models

143
Parameters

- `shape_weights ((n_weights,) ndarray or list or None, optional) – The weights of the shape model that will be used to create a novel shape instance. If None, the weights are assumed to be zero, thus the mean shape is used.

- `scale_index (int, optional) – The scale to be used.

Returns image (menpo.image.Image) – The ATM instance.

`random_instance (scale_index=-1)`
Generates a random instance of the ATM.

Parameters `scale_index (int, optional) – The scale to be used.

Returns image (menpo.image.Image) – The ATM instance.

`view_atm_widget (n_shape_parameters=5, parameters_bounds=(-3.0, 3.0), mode='multiple', figure_size=(7, 7))`
Visualizes the ATM using an interactive widget.

Parameters

- `n_shape_parameters (int or list of int or None, optional) – The number of shape principal components to be used for the parameters sliders. If int, then the number of sliders per scale is the minimum between `n_parameters` and the number of active components per scale. If list of int, then a number of sliders is defined per scale. If None, all the active components per scale will have a slider.

- `parameters_bounds ((float, float), optional) – The minimum and maximum bounds, in std units, for the sliders.

- `mode ({single, multiple}, optional) – If 'single', only a single slider is constructed along with a drop down menu. If 'multiple', a slider is constructed for each parameter.

- `figure_size ((int, int), optional) – The size of the rendered figure.

`view_shape_models_widget (n_parameters=5, parameters_bounds=(-3.0, 3.0), mode='multiple', figure_size=(7, 7))`
Visualizes the shape models of the ATM object using an interactive widget.

Parameters

- `n_parameters (int or list of int or None, optional) – The number of shape principal components to be used for the parameters sliders. If int, then the number of sliders per scale is the minimum between `n_parameters` and the number of active components per scale. If list of int, then a number of sliders is defined per scale. If None, all the active components per scale will have a slider.

- `parameters_bounds ((float, float), optional) – The minimum and maximum bounds, in std units, for the sliders.

- `mode ({single, multiple}, optional) – If 'single', only a single slider is constructed along with a drop down menu. If 'multiple', a slider is constructed for each parameter.

- `figure_size ((int, int), optional) – The size of the rendered figure.

property n_scales
Returns the number of scales.

Type `int`
class menpofit.atm.PatchATM

Class for training a multi-scale Patch-Based Active Template Model.

Parameters

- **template** (menpo.image.Image) – The template image.
- **shapes** (list of menpo.shape.PointCloud) – The list of training shapes.
- **group** (str or None, optional) – The landmark group of the template that will be used to train the ATM. If None and the template only has a single landmark group, then that is the one that will be used.
- **holistic_features** (closure or list of closure, optional) – The features that will be extracted from the training images. Note that the features are extracted before warping the images to the reference shape. If list, then it must define a feature function per scale. Please refer to menpo.feature for a list of potential features.
- **reference_shape** (menpo.shape.PointCloud or None, optional) – The reference shape that will be used for building the ATM. The purpose of the reference shape is to normalise the size of the training images. The normalization is performed by rescaling all the training images so that the scale of their ground truth shapes matches the scale of the reference shape. Note that the reference shape is rescaled with respect to the diagonal before performing the normalisation. If None, then the mean shape will be used.
- **diagonal** (int or None, optional) – This parameter is used to rescale the reference shape so that the diagonal of its bounding box matches the provided value. In other words, this parameter controls the size of the model at the highest scale. If None, then the reference shape does not get rescaled.
- **scales** (float or tuple of float, optional) – The scale value of each scale. They must provided in ascending order, i.e. from lowest to highest scale. If float, then a single scale is assumed.
- **patch_shape** ((int, int) or list of (int, int), optional) – The shape of the patches to be extracted. If a list is provided, then it defines a patch shape per scale.
- **max_shape_components** (int, float, list of those or None, optional) – The number of shape components to keep. If int, then it sets the exact number of components. If float, then it defines the variance percentage that will be kept. If list, then it should define a value per scale. If a single number, then this will be applied to all scales. If None, then all the components are kept. Note that the unused components will be permanently trimmed.
- **verbose** (bool, optional) – If True, then the progress of building the ATM will be printed.
- **batch_size** (int or None, optional) – If an int is provided, then the training is performed in an incremental fashion on image batches of size equal to the provided value. If None, then the training is performed directly on all the images.

build_fitter_interfaces (sampling)

Method that builds the correct Lucas-Kanade fitting interface.

Parameters sampling (list of int or ndarray or None) – It defines a sampling mask per scale. If int, then it defines the sub-sampling step of the sampling mask. If ndarray, then it explicitly
defines the sampling mask. If None, then no sub-sampling is applied.

Returns fitter_interfaces (list) – The list of Lucas-Kanade interface per scale.

increment (template, shapes, group=None, shape_forgetting_factor=1.0, verbose=False, batch_size=None)
Method to increment the trained ATM with a new set of training shapes and a new template.

Parameters

• template (menpo.image.Image) – The template image.

• shapes (list of menpo.shape.PointCloud) – The list of training shapes.

• group (str or None, optional) – The landmark group of the template that will be used to train the ATM. If None and the template only has a single landmark group, then that is the one that will be used.

• shape_forgetting_factor ([0.0, 1.0] float, optional) – Forgetting factor that weights the relative contribution of new samples vs old samples for the shape model. If 1.0, all samples are weighted equally and, hence, the result is the exact same as performing batch PCA on the concatenated list of old and new simples. If <1.0, more emphasis is put on the new samples.

• verbose (bool, optional) – If True, then the progress of building the ATM will be printed.

• batch_size (int or None, optional) – If an int is provided, then the training is performed in an incremental fashion on image batches of size equal to the provided value. If None, then the training is performed directly on all the images.

instance (shape_weights=None, scale_index=-1)
Generates a novel ATM instance given a set of shape weights. If no weights are provided, the mean ATM instance is returned.

Parameters

• shape_weights (n_weights,) ndarray or list or None, optional) – The weights of the shape model that will be used to create a novel shape instance. If None, the weights are assumed to be zero, thus the mean shape is used.

• scale_index (int, optional) – The scale to be used.

Returns image (menpo.image.Image) – The ATM instance.

random_instance (scale_index=-1)
Generates a random instance of the ATM.

Parameters scale_index (int, optional) – The scale to be used.

Returns image (menpo.image.Image) – The ATM instance.

view_atm_widget (n_shape_parameters=5, parameters_bounds=(-3.0, 3.0), mode='multiple', figure_size=(7, 7))
Visualizes the ATM using an interactive widget.

Parameters

• n_shape_parameters (int or list of int or None, optional) – The number of shape principal components to be used for the parameters sliders. If int, then the number of sliders per scale is the minimum between n_parameters and the number of active components per scale. If list of int, then a number of sliders is defined per scale. If None, all the active components per scale will have a slider.
• **parameters_bounds** ((float, float), optional) – The minimum and maximum bounds, in std units, for the sliders.

• **mode** ([single, multiple], optional) – If 'single', only a single slider is constructed along with a drop down menu. If 'multiple', a slider is constructed for each parameter.

• **figure_size** ((int, int), optional) – The size of the rendered figure.

`view_shape_models_widget` (n_parameters=5, parameters_bounds=(-3.0, 3.0), mode='multiple', figure_size=(7, 7))

Visualizes the shape models of the ATM object using an interactive widget.

### Parameters

- **n_parameters** (int or list of int or None, optional) – The number of shape principal components to be used for the parameters sliders. If int, then the number of sliders per scale is the minimum between n_parameters and the number of active components per scale. If list of int, then a number of sliders is defined per scale. If None, all the active components per scale will have a slider.

- **parameters_bounds** ((float, float), optional) – The minimum and maximum bounds, in std units, for the sliders.

- **mode** ([single, multiple], optional) – If 'single', only a single slider is constructed along with a drop down menu. If 'multiple', a slider is constructed for each parameter.

- **figure_size** ((int, int), optional) – The size of the rendered figure.

`property n_scales`  

Returns the number of scales.  

**Type** int

---

### Fitter

LucasKanadeATMFitter

`class menpofit.atm.LucasKanadeATMFitter(atm, lk_algorithm_cls=<class 'menpofit.atm.algorithm.InverseCompositional'>, n_shape=None, sampling=None)`

**Bases:** MultiScaleParametricFitter

Class for defining an ATM fitter using the Lucas-Kanade optimization.

**Parameters**

- **atm** (`ATM` or subclass) – The trained ATM model.

- **lk_algorithm_cls** (`class`, optional) – The Lukas-Kanade optimisation algorithm that will get applied. The possible algorithms are:

<table>
<thead>
<tr>
<th>Class</th>
<th>Warp Direction</th>
<th>Warp Update</th>
</tr>
</thead>
<tbody>
<tr>
<td>ForwardCompositional</td>
<td>Forward</td>
<td>Compositional</td>
</tr>
<tr>
<td>InverseCompositional</td>
<td>Inverse</td>
<td></td>
</tr>
</tbody>
</table>

- **n_shape** (int or float or list of those or None, optional) – The number of shape components that will be used. If int, then it defines the exact number of active components. If float, then it defines the percentage of variance to keep. If int or float, then the provided value will
be applied for all scales. If list, then it defines a value per scale. If None, then all the available components will be used. Note that this simply sets the active components without trimming the unused ones. Also, the available components may have already been trimmed to max_shape_components during training.

- **sampling** (list of int or ndarray or None) – It defines a sampling mask per scale. If int, then it defines the sub-sampling step of the sampling mask. If ndarray, then it explicitly defines the sampling mask. If None, then no sub-sampling is applied.

**fit_from_bb** *(image, bounding_box, max_iters=20, gt_shape=None, return_costs=False, **kwargs)*

Fits the multi-scale fitter to an image given an initial bounding box.

**Parameters**

- **image** *(menpo.image.Image or subclass)* – The image to be fitted.
- **bounding_box** *(menpo.shape.PointDirectedGraph)* – The initial bounding box from which the fitting procedure will start. Note that the bounding box is used in order to align the model’s reference shape.
- **max_iters** *(int or list of int, optional)* – The maximum number of iterations. If int, then it specifies the maximum number of iterations over all scales. If list of int, then specifies the maximum number of iterations per scale.
- **gt_shape** *(menpo.shape.PointCloud, optional)* – The ground truth shape associated to the image.
- **return_costs** *(bool, optional)* – If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.
- **kwargs** *(dict, optional)* – Additional keyword arguments that can be passed to specific implementations.

**Returns** **fitting_result** *(MultiScaleNonParametricIterativeResult or subclass)* – The multi-scale fitting result containing the result of the fitting procedure.

**fit_from_shape** *(image, initial_shape, max_iters=20, gt_shape=None, return_costs=False, **kwargs)*

Fits the multi-scale fitter to an image given an initial shape.

**Parameters**

- **image** *(menpo.image.Image or subclass)* – The image to be fitted.
- **initial_shape** *(menpo.shape.PointCloud)* – The initial shape estimate from which the fitting procedure will start.
- **max_iters** *(int or list of int, optional)* – The maximum number of iterations. If int, then it specifies the maximum number of iterations over all scales. If list of int, then specifies the maximum number of iterations per scale.
- **gt_shape** *(menpo.shape.PointCloud, optional)* – The ground truth shape associated to the image.
- **return_costs** *(bool, optional)* – If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.
• **kwargs** (*dict*, optional) – Additional keyword arguments that can be passed to specific implementations.

Returns **fitting_result** (*MultiScaleNonParametricIterativeResult* or subclass) – The multi-scale fitting result containing the result of the fitting procedure.

**warped_images** (*image*, *shapes*)

Given an input test image and a list of shapes, it warps the image into the shapes. This is useful for generating the warped images of a fitting procedure stored within a *MultiScaleParametricIterativeResult*.

Parameters

• **image** (*menpo.image.Image* or subclass) – The input image to be warped.

• **shapes** (*list* of *menpo.shape.PointCloud*) – The list of shapes in which the image will be warped. The shapes are obtained during the iterations of a fitting procedure.

Returns **warped_images** (*list* of *menpo.image.MaskedImage* or *ndarray*) – The warped images.

**property atm**

The trained ATM model.

Type: *ATM* or subclass

**property holistic_features**

The features that are extracted from the input image at each scale in ascending order, i.e. from lowest to highest scale.

Type: *list* of *closure*

**property n_scales**

Returns the number of scales.

Type: *int*

**property reference_shape**

The reference shape that is used to normalise the size of an input image so that the scale of its initial fitting shape matches the scale of this reference shape.

Type: *menpo.shape.PointCloud*

**property scales**

The scale value of each scale in ascending order, i.e. from lowest to highest scale.

Type: *list* of *int* or *float*

### Lucas-Kanade Optimisation Algorithms

#### ForwardCompositional

class *menpofit.atm.ForwardCompositional* (*atm_interface*, *eps=1e-05*)

Bases: *Compositional*

Forward Compositional (FC) Gauss-Newton algorithm.

**run** (*image*, *initial_shape*, *gt_shape=None*, *max_iters=20*, *return_costs=False*, *map_inference=False*)

Execute the optimization algorithm.

Parameters

• **image** (*menpo.image.Image*) – The input test image.
initial_shape (menpo.shape.PointCloud) – The initial shape from which the optimization will start.

gt_shape (menpo.shape.PointCloud or None, optional) – The ground truth shape of the image. It is only needed in order to get passed in the optimization result object, which has the ability to compute the fitting error.

max_iters (int, optional) – The maximum number of iterations. Note that the algorithm may converge, and thus stop, earlier.

return_costs (bool, optional) – If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.

map_inference (bool, optional) – If True, then the solution will be given after performing MAP inference.

Returns fitting_result (ParametricIterativeResult) – The parametric iterative fitting result.

property template
Returns the template of the ATM.

Type menpo.image.Image or subclass

property transform
Returns the model driven differential transform object of the AAM, e.g. DifferentiablePiecewiseAffine or DifferentiableThinPlateSplines.

Type subclass of DL and DX

InverseCompositional

class menpofit.atm.InverseCompositional (atm_interface, eps=1e-05)
Bases: Compositional

Inverse Compositional (IC) Gauss-Newton algorithm.

run (image, initial_shape, gt_shape=None, max_iters=20, return_costs=False, map_inference=False)
Execute the optimization algorithm.

Parameters

• image (menpo.image.Image) – The input test image.

• initial_shape (menpo.shape.PointCloud) – The initial shape from which the optimization will start.

• gt_shape (menpo.shape.PointCloud or None, optional) – The ground truth shape of the image. It is only needed in order to get passed in the optimization result object, which has the ability to compute the fitting error.

• max_iters (int, optional) – The maximum number of iterations. Note that the algorithm may converge, and thus stop, earlier.

• return_costs (bool, optional) – If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned
MenpoFit Documentation, Release 0.6.1

fitting_result. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.

- **map_inference** (bool, optional) – If True, then the solution will be given after performing MAP inference.

Returns *fitting_result* (*ParametricIterativeResult*) – The parametric iterative fitting result.

**property** *template*
Returns the template of the ATM.

Type *menpo.image.Image* or subclass

**property** *transform*
Returns the model driven differential transform object of the AAM, e.g. *DifferentiablePiecewiseAffine* or *DifferentiableThinPlateSplines*.

Type subclass of *DL* and *DX*

### 2.1.4 menpofit.clm

**Constrained Local Model**

Deformable model that consists of a generative parametric shape model and discriminatively trained experts per part.

**CLM**

**class** *menpofit.clm.CLM* (*images*, *group=None*, *holistic_features=<function no_op>*, *reference_shape=None*, *diagonal=None*, *scales=(0.5, 1)*, *patch_shape=(17, 17)*, *patch_normalisation=<function no_op>*, *context_shape=(34, 34)*, *cosine_mask=True*, *sample_offsets=None*, *shape_model_cls=<class 'menpofit.modelinstance.OrthoPDM'>*, *expert_ensemble_cls=<class 'menpofit.clm.expert.ensemble.CorrelationFilterExpertEnsemble'>*, *max_shape_components=None*, *verbose=False*, *batch_size=None*)

**Bases:** object

Class for training a multi-scale holistic Constrained Local Model. Please see the references for a basic list of relevant papers.

**Parameters**

- **images** (list of *menpo.image.Image*) – The list of training images.
- **group** (str or None, optional) – The landmark group that will be used to train the CLM. If None and the images only have a single landmark group, then that is the one that will be used. Note that all the training images need to have the specified landmark group.
- **holistic_features** (closure or list of closure, optional) – The features that will be extracted from the training images. If list, then it must define a feature function per scale. Please refer to *menpo.feature* for a list of potential features.
- **reference_shape** (*menpo.shape.PointCloud* or None, optional) – The reference shape that will be used for building the CLM. The purpose of the reference shape is to normalise the size of the training images. The normalization is performed by rescaling all the training images so that the scale of their ground truth shapes matches the scale of the reference shape.
Note that the reference shape is rescaled with respect to the diagonal before performing the normalisation. If None, then the mean shape will be used.

- **diagonal** *(int or None, optional)* – This parameter is used to rescale the reference shape so that the diagonal of its bounding box matches the provided value. In other words, this parameter controls the size of the model at the highest scale. If None, then the reference shape does not get rescaled.

- **scales** *(float or tuple of float, optional)* – The scale value of each scale. They must provided in ascending order, i.e. from lowest to highest scale. If float, then a single scale is assumed.

- **patch_shape** *((int, int) or list of (int, int), optional)* – The shape of the patches to be extracted. If a list is provided, then it defines a patch shape per scale.

- **patch_normalisation** *(callable, optional)* – The normalisation function to be applied on the extracted patches.

- **context_shape** *((int, int) or list of (int, int), optional)* – The context shape for the convolution. If a list is provided, then it defines a context shape per scale.

- **cosine_mask** *(bool, optional)* – If True, then a cosine mask (Hanning function) will be applied on the extracted patches.

- **sample_offsets** *((nOffsets, nDims) ndarray or None, optional)* – The offsets to sample from within a patch. So \((0, 0)\) is the centre of the patch (no offset) and \((1, 0)\) would be sampling the patch from 1 pixel up the first axis away from the centre. If None, then no offsets are applied.

- **shape_model_cls** *(subclass of PDM, optional)* – The class to be used for building the shape model. The most common choice is OrthoPDM.

- **expert_ensemble_cls** *(subclass of ExpertEnsemble, optional)* – The class to be used for training the ensemble of experts. The most common choice is CorrelationFilterExpertEnsemble.

- **max_shape_components** *(int, float, list of those or None, optional)* – The number of shape components to keep. If int, then it sets the exact number of components. If float, then it defines the variance percentage that will be kept. If list, then it should define a value per scale. If a single number, then this will be applied to all scales. If None, then all the components are kept. Note that the unused components will be permanently trimmed.

- **verbose** *(bool, optional)* – If True, then the progress of building the CLM will be printed.

- **batch_size** *(int or None, optional)* – If an int is provided, then the training is performed in an incremental fashion on image batches of size equal to the provided value. If None, then the training is performed directly on the all the images.

### References

**increment** *(images, group=None, shape_forgetting_factor=1.0, verbose=False, batch_size=None)*  
Method to increment the trained CLM with a new set of training images.

**Parameters**

- **images** *(list of menpo.image.Image)* – The list of training images.

- **group** *(str or None, optional)* – The landmark group that will be used to train the CLM. If None and the images only have a single landmark group, then that is the one that will be used. Note that all the training images need to have the specified landmark group.
• **shape_forgetting_factor** ([0.0, 1.0] float, optional) – Forgetting factor that weights the relative contribution of new samples vs old samples for the shape model. If 1.0, all samples are weighted equally and, hence, the result is the exact same as performing batch PCA on the concatenated list of old and new simples. If <1.0, more emphasis is put on the new samples.

• **verbose** (bool, optional) – If True, then the progress of building the CLM will be printed.

• **batch_size** (int or None, optional) – If an int is provided, then the training is performed in an incremental fashion on image batches of size equal to the provided value. If None, then the training is performed directly on all the images.

**shape_instance** *(shape_weights=None, scale_index=-1)*

Generates a novel shape instance given a set of shape weights. If no weights are provided, the mean shape is returned.

Parameters

• **shape_weights** ((n_weights,) ndarray or list or None, optional) – The weights of the shape model that will be used to create a novel shape instance. If None, the weights are assumed to be zero, thus the mean shape is used.

• **scale_index** (int, optional) – The scale to be used.

Returns

**instance** *(menpo.shape.PointCloud)* – The shape instance.

**view_clm_widget** *(n_shape_parameters=5, parameters_bounds=(-3.0, 3.0), mode='multiple', figure_size=(7, 7))*

Visualizes the CLM object using an interactive widget.

Parameters

• **n_shape_parameters** (int or list of int or None, optional) – The number of shape principal components to be used for the parameters sliders. If int, then the number of sliders per scale is the minimum between n_parameters and the number of active components per scale. If list of int, then a number of sliders is defined per scale. If None, all the active components per scale will have a slider.

• **parameters_bounds** ((float, float), optional) – The minimum and maximum bounds, in std units, for the sliders.

• **mode** ([single, multiple], optional) – If 'single', only a single slider is constructed along with a drop down menu. If 'multiple', a slider is constructed for each parameter.

• **figure_size** ((int, int), optional) – The size of the rendered figure.

Raises **ValueError** – Only convolution-based expert ensembles can be visualized.

**view_expert_ensemble_widget** *(figure_size=(7, 7))*

Visualizes the ensemble of experts of the CLM object using an interactive widget.

Parameters **figure_size** ((int, int), optional) – The size of the plotted figures.

Raises **ValueError** – Only convolution-based expert ensembles can be visualized.

**view_shape_models_widget** *(n_parameters=5, parameters_bounds=(-3.0, 3.0), mode='multiple', figure_size=(7, 7))*

Visualizes the shape models of the CLM object using an interactive widget.

Parameters
• **n_parameters** (*int* or *list* of *int* or *None*, optional) – The number of shape principal components to be used for the parameters sliders. If *int*, then the number of sliders per scale is the minimum between *n_parameters* and the number of active components per scale. If *list* of *int*, then a number of sliders is defined per scale. If *None*, all the active components per scale will have a slider.

• **parameters_bounds** (*tuple* of (*float*, *float*), optional) – The minimum and maximum bounds, in std units, for the sliders.

• **mode** (*{'single', 'multiple'}*, optional) – If 'single', only a single slider is constructed along with a drop down menu. If 'multiple', a slider is constructed for each parameter.

• **figure_size** (*tuple* of (*int*, *int*), optional) – The size of the rendered figure.

**property n_scales**

Returns the number of scales.

  Type  *int*

---

**Fitter**

**GradientDescentCLMFitter**

```python
class menpofit.clm.GradientDescentCLMFitter(clm, gd_algorithm_cls=<class 'menpofit.clm.algorithm.gd.RegularisedLandmarkMeanShift'>, n_shape=None)
```

Bases: CLMFitter

Class for defining an CLM fitter using gradient descent optimization.

**Note:** When using a method with a parametric shape model, the first step is to **reconstruct the initial shape** using the shape model. The generated reconstructed shape is then used as initialisation for the iterative optimisation. This step takes place at each scale and it is not considered as an iteration, thus it is not counted for the provided *max_iters*.

---

**Parameters**

• **clm** (*CLM* or *subclass*) – The trained CLM model.

• **gd_algorithm_cls** (*class*, optional) – The gradient descent optimisation algorithm that will get applied. The possible options are *RegularisedLandmarkMeanShift* and *ActiveShapeModel*.

• **n_shape** (*int* or *float* or *list* of those or *None*, optional) – The number of shape components that will be used. If *int*, then it defines the exact number of active components. If *float*, then it defines the percentage of variance to keep. If *int* or *float*, then the provided value will be applied for all scales. If *list*, then it defines a value per scale. If *None*, then all the available components will be used. Note that this simply sets the active components without trimming the unused ones. Also, the available components may have already been trimmed to *max_shape_components* during training.

  **fit_from_bb** (*image*, *bounding_box*, *max_iters=20*, *gt_shape=None*, *return_costs=False*, **kwargs)

  Fits the multi-scale fitter to an image given an initial bounding box.

  **Parameters**
• **image** ([menpo.image.Image](https://menpo.org/docs/api/menpo.image.html#menpo.image.Image) or subclass) – The image to be fitted.

• **bounding_box** ([menpo.shape.PointDirectedGraph](https://menpo.org/docs/api/menpo.shape.html#menpo.shape.PointDirectedGraph)) – The initial bounding box from which the fitting procedure will start. Note that the bounding box is used in order to align the model’s reference shape.

• **max_iters** (*int* or *list* of *int*, optional) – The maximum number of iterations. If *int*, then it specifies the maximum number of iterations over all scales. If *list* of *int*, then specifies the maximum number of iterations per scale.

• **gt_shape** ([menpo.shape.PointCloud](https://menpo.org/docs/api/menpo.shape.html#menpo.shape.PointCloud), optional) – The ground truth shape associated to the image.

• **return_costs** (*bool*, optional) – If *True*, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting result. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.

• **kwargs** (*dict*, optional) – Additional keyword arguments that can be passed to specific implementations.

**Returns** fitting_result ([MultiScaleNonParametricIterativeResult](https://menpo.org/docs/api/menpo.shape.html#menpo.shape.MultiScaleNonParametricIterativeResult) or subclass) – The multi-scale fitting result containing the result of the fitting procedure.

**fit_from_shape** (*image*, *initial_shape*, *max_iters*=20, *gt_shape*=None, *return_costs*=False, **kwargs)

Fits the multi-scale fitter to an image given an initial shape.

**Parameters**

• **image** ([menpo.image.Image](https://menpo.org/docs/api/menpo.image.html#menpo.image.Image) or subclass) – The image to be fitted.

• **initial_shape** ([menpo.shape.PointCloud](https://menpo.org/docs/api/menpo.shape.html#menpo.shape.PointCloud)) – The initial shape estimate from which the fitting procedure will start.

• **max_iters** (*int* or *list* of *int*, optional) – The maximum number of iterations. If *int*, then it specifies the maximum number of iterations over all scales. If *list* of *int*, then specifies the maximum number of iterations per scale.

• **gt_shape** ([menpo.shape.PointCloud](https://menpo.org/docs/api/menpo.shape.html#menpo.shape.PointCloud), optional) – The ground truth shape associated to the image.

• **return_costs** (*bool*, optional) – If *True*, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting result. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.

• **kwargs** (*dict*, optional) – Additional keyword arguments that can be passed to specific implementations.

**Returns** fitting_result ([MultiScaleNonParametricIterativeResult](https://menpo.org/docs/api/menpo.shape.html#menpo.shape.MultiScaleNonParametricIterativeResult) or subclass) – The multi-scale fitting result containing the result of the fitting procedure.

**property** clm

The trained CLM model.

**Type** CLM or subclass

**property** holistic_features

The features that are extracted from the input image at each scale in ascending order, i.e. from lowest to highest scale.
Type  list of closure

**property n_scales**
Returns the number of scales.

Type  int

**property reference_shape**
The reference shape that is used to normalise the size of an input image so that the scale of its initial fitting shape matches the scale of this reference shape.

Type  menpo.shape.PointCloud

**property scales**
The scale value of each scale in ascending order, i.e. from lowest to highest scale.

Type  list of int or float

### Gradient Descent Optimisation Algorithms

**ActiveShapeModel**

```python
class menpofit.clm.ActiveShapeModel(expert_ensemble, shape_model, gaussian_covariance=10, eps=1e-05)
```

Bases: GradientDescentCLMAlgorithm

Active Shape Model (ASM) algorithm.

**Parameters**

- **expert_ensemble** (subclass of ExpertEnsemble) – The ensemble of experts object, e.g. CorrelationFilterExpertEnsemble.
- **shape_model** (subclass of PDM, optional) – The shape model object, e.g. OrthoPDM.
- **gaussian_covariance** (int or float, optional) – The covariance of the Gaussian kernel.
- **eps** (float, optional) – Value for checking the convergence of the optimization.

**References**

```python
run(image, initial_shape, gt_shape=None, max_iters=20, return_costs=False, map_inference=False)
```

Execute the optimization algorithm.

**Parameters**

- **image** (menpo.image.Image) – The input test image.
- **initial_shape** (menpo.shape.PointCloud) – The initial shape from which the optimization will start.
- **gt_shape** (menpo.shape.PointCloud or None, optional) – The ground truth shape of the image. It is only needed in order to get passed in the optimization result object, which has the ability to compute the fitting error.
- **max_iters** (int, optional) – The maximum number of iterations. Note that the algorithm may converge, and thus stop, earlier.
- **return_costs** (bool, optional) – If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that this argument currently has no effect and will raise a warning.
if set to ‘"True"’. This is because it is not possible to evaluate the cost function of this algorithm.

- **map_inference** *(bool, optional)* – If True, then the solution will be given after performing MAP inference.

Returns **fitting_result** *(ParametricIterativeResult)* – The parametric iterative fitting result.

### RegularisedLandmarkMeanShift

```python
class menpofit.clm.RegularisedLandmarkMeanShift(expert_ensemble, shape_model, kernel_covariance=10, eps=1e-05)
```

Bases: GradientDescentCLMAlgorithm

Regularized Landmark Mean-Shift (RLMS) algorithm.

**Parameters**

- **expert_ensemble** *(subclass of ExpertEnsemble)* – The ensemble of experts object, e.g. CorrelationFilterExpertEnsemble.
- **shape_model** *(subclass of PDM, optional)* – The shape model object, e.g. OrthoPDM.
- **kernel_covariance** *(int or float, optional)* – The covariance of the kernel.
- **eps** *(float, optional)* – Value for checking the convergence of the optimization.

**run** *(image, initial_shape, gt_shape=None, max_iters=20, return_costs=False, map_inference=False)*

Execute the optimization algorithm.

**Parameters**

- **image** *(menpo.image.Image)* – The input test image.
- **initial_shape** *(menpo.shape.PointCloud)* – The initial shape from which the optimization will start.
- **gt_shape** *(menpo.shape.PointCloud or None, optional)* – The ground truth shape of the image. It is only needed in order to get passed in the optimization result object, which has the ability to compute the fitting error.
- **max_iters** *(int, optional)* – The maximum number of iterations. Note that the algorithm may converge, and thus stop, earlier.
- **return_costs** *(bool, optional)* – If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that this argument currently has no effect and will raise a warning if set to ‘"True"’. This is because it is not possible to evaluate the cost function of this algorithm.
- **map_inference** *(bool, optional)* – If True, then the solution will be given after performing MAP inference.

Returns **fitting_result** *(ParametricIterativeResult)* – The parametric iterative fitting result.
Experts Ensemble

Algorithms for learning an ensemble of discriminative experts.

CorrelationFilterExpertEnsemble

class menpofit.clm.CorrelationFilterExpertEnsemble(images, shapes, icf_cls=<class 'menpofit.clm.expert.base.IncrementalCorrelationFilterThinWrapper'>, patch_shape=(17, 17), context_shape=(34, 34), response_covariance=3, patch_normalisation=functools.partial(<function normalize_norm>, mode='per_channel', error_on_divide_by_zero=False), cosine_mask=True, sample_offsets=None, prefix='", verbose=False)

Bases: ConvolutionBasedExpertEnsemble

Class for defining an ensemble of correlation filter experts.

Parameters

- **images** (list of menpo.image.Image) – The list of training images.
- **shapes** (list of menpo.shape.PointCloud) – The list of training shapes that correspond to the images.
- **icf_cls** (class, optional) – The incremental correlation filter class. For example IncrementalCorrelationFilterThinWrapper.
- **patch_shape** ((int, int), optional) – The shape of the patches that will be extracted around the landmarks. Those patches are used to train the experts.
- **context_shape** ((int, int), optional) – The context shape for the convolution.
- **response_covariance** (int, optional) – The covariance of the generated Gaussian response.
- **patch_normalisation** (callable, optional) – A normalisation function that will be applied on the extracted patches.
- **cosine_mask** (bool, optional) – If True, then a cosine mask (Hanning function) will be applied on the extracted patches.
- **sample_offsets** ((n_offsets, n_dims) ndarray or None, optional) – The offsets to sample from within a patch. So (0, 0) is the centre of the patch (no offset) and (1, 0) would be sampling the patch from 1 pixel up the first axis away from the centre. If None, then no offsets are applied.
- **prefix** (str, optional) – The prefix of the printed progress information.
- **verbose** (bool, optional) – If True, then information will be printed regarding the training progress.

increment(images, shapes, prefix="", verbose=False)

Increments the learned ensemble of convolution-based experts given a new set of training data.

Parameters
• **images** (list of menpo.image.Image) – The list of training images.

• **shapes** (list of menpo.shape.PointCloud) – The list of training shapes that correspond to the images.

• **prefix** (str, optional) – The prefix of the printed training progress.

• **verbose** (bool, optional) – If True, then information about the training progress will be printed.

**predict_probability** (*image, shape*)

Method for predicting the probability map of the response experts on a given image. Note that the provided shape must have the same number of points as the number of experts.

Parameters

• **image** (menpo.image.Image or subclass) – The test image.

• **shape** (menpo.shape.PointCloud) – The shape that corresponds to the image from which the patches will be extracted.

Returns **probability_map** ((n_experts, 1, height, width) ndarray) – The probability map of the response of each expert.

**predict_response** (*image, shape*)

Method for predicting the response of the experts on a given image. Note that the provided shape must have the same number of points as the number of experts.

Parameters

• **image** (menpo.image.Image or subclass) – The test image.

• **shape** (menpo.shape.PointCloud) – The shape that corresponds to the image from which the patches will be extracted.

Returns **response** ((n_experts, 1, height, width) ndarray) – The response of each expert.

**view_frequency_filter_images_widget** (*figure_size=(7, 7), style='coloured', browser_style='buttons'*)

Visualizes the filters on the frequency domain using an interactive widget.

Parameters

• **figure_size** ((int, int), optional) – The initial size of the rendered figure.

• **style** ({'coloured', 'minimal'}, optional) – If 'coloured', then the style of the widget will be coloured. If minimal, then the style is simple using black and white colours.

• **browser_style** ({'buttons', 'slider'}, optional) – It defines whether the selector of the objects will have the form of plus/minus buttons or a slider.

**view_spatial_filter_images_widget** (*figure_size=(7, 7), style='coloured', browser_style='buttons'*)

Visualizes the filters on the spatial domain using an interactive widget.

Parameters

• **figure_size** ((int, int), optional) – The initial size of the rendered figure.

• **style** ({'coloured', 'minimal'}, optional) – If 'coloured', then the style of the widget will be coloured. If minimal, then the style is simple using black and white colours.
• **browser_style** (['buttons', 'slider'], optional) – It defines whether the selector of the objects will have the form of plus/minus buttons or a slider.

**property frequency_filter_images**
Returns a list of n_experts filter images on the frequency domain.

Type list of menpo.image.Image

**property n_experts**
Returns the number of experts.

Type int

**property n_sample_offsets**
Returns the number of offsets that are sampled within a patch.

Type int

**property padded_size**
Returns the convolution pad size, i.e. floor(1.5 * patch_shape - 1).

Type (int, int)

**property search_shape**
Returns the search shape (patch_shape).

Type (int, int)

**property spatial_filter_images**
Returns a list of n_experts filter images on the spatial domain.

Type list of menpo.image.Image

**Experts**

Discriminative experts

**IncrementalCorrelationFilterThinWrapper**

```
class menpofit.clm.IncrementalCorrelationFilterThinWrapper (cf_callable=<function mccf>,
icf_callable=<function imccf>)
```

Bases: object

Wrapper class for defining an Incremental Correlation Filter.

**Parameters**

• **cf_callable** (callable, optional) – The correlation filter function. Possible options are:

<table>
<thead>
<tr>
<th>Class</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>mccf</td>
<td>Multi-Channel Correlation Filter</td>
</tr>
<tr>
<td>mosse</td>
<td>Minimum Output Sum of Squared Errors Filter</td>
</tr>
</tbody>
</table>

• **icf_callable** (callable, optional) – The incremental correlation filter function. Possible options are:
<table>
<thead>
<tr>
<th>Class</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>imccf</td>
<td>Incremental Multi-Channel Correlation Filter</td>
</tr>
<tr>
<td>imosse</td>
<td>Incremental Minimum Output Sum of Squared Errors Filter</td>
</tr>
</tbody>
</table>

**increment** \((A, B, n_x, Z, t)\)

Method that trains the correlation filter.

**Parameters**

- **A** \((N,)\) ndarray – The current auto-correlation array, where \(N = (\text{patch}_h + \text{response}_h - 1) \times (\text{patch}_w + \text{response}_w - 1) \times n\_channels\)
- **B** \((N, N)\) ndarray – The current cross-correlation array, where \(N = (\text{patch}_h + \text{response}_h - 1) \times (\text{patch}_w + \text{response}_w - 1) \times n\_channels\)
- **n_x** (int) – The current number of images.
- **Z** (list or \((n\_images, n\_channels, \text{patch}_h, \text{patch}_w)\) ndarray) – The training images (patches). If list, then it consists of \(n\_images\) \((n\_channels, \text{patch}_h, \text{patch}_w)\) ndarray members.
- **t** \(((1, \text{response}_h, \text{response}_w)\) ndarray) – The desired response.

**Returns**

- **correlation Filter** \((n\_channels, \text{response}_h, \text{response}_w)\) ndarray – The learned correlation filter.
- **auto_correlation** \((N,)\) ndarray – The auto-correlation array, where \(N = (\text{patch}_h + \text{response}_h - 1) \times (\text{patch}_w + \text{response}_w - 1) \times n\_channels\)
- **cross_correlation** \((N, N)\) ndarray – The cross-correlation array, where \(N = (\text{patch}_h + \text{response}_h - 1) \times (\text{patch}_w + \text{response}_w - 1) \times n\_channels\)

**train** \((X, t)\)

Method that trains the correlation filter.

**Parameters**

- **X** (list or \((n\_images, n\_channels, \text{patch}_h, \text{patch}_w)\) ndarray) – The training images (patches). If list, then it consists of \(n\_images\) \((n\_channels, \text{patch}_h, \text{patch}_w)\) ndarray members.
- **t** \(((1, \text{response}_h, \text{response}_w)\) ndarray) – The desired response.

**Returns**

- **correlation Filter** \((n\_channels, \text{response}_h, \text{response}_w)\) ndarray – The learned correlation filter.
- **auto_correlation** \((N,)\) ndarray – The auto-correlation array, where \(N = (\text{patch}_h + \text{response}_h - 1) \times (\text{patch}_w + \text{response}_w - 1) \times n\_channels\)
- **cross_correlation** \((N, N)\) ndarray – The cross-correlation array, where \(N = (\text{patch}_h + \text{response}_h - 1) \times (\text{patch}_w + \text{response}_w - 1) \times n\_channels\)
2.1.5 menpofit.unified_aam_clm

Unified Active Appearance Model and Constrained Local Model

This method combines a holistic AAM with a part-based CLM under a unified optimisation problem.

UnifiedAAMCLM

class menpofit.unified_aam_clm.base.UnifiedAAMCLM(images, group=None, holistic_features=<function no_op>, reference_shape=None, diagonal=None, scales=(0.5, 1.0), expert_ensemble_cls=<class 'menpofit.clm.expert.ensemble.CorrelationFilterExpertEnsemble'>, patch_shape=(17, 17), context_shape=(34, 34), sample_offsets=None, transform=<class 'menpofit.transform.piecewiseaffine.DifferentiablePiecewiseAffine'>, shape_model_cls=<class 'menpofit.modelinstance.OrthoPDM'>, max_shape_components=None, max_appearance_components=None, sigma=None, boundary=3, response_covariance=2, patch_normalisation=<function no_op>, cosine_mask=True, verbose=False)

Bases: object

Class for training a multi-scale unified holistic AAM and CLM as presented in [1]. Please see the references for AAMs and CLMs in their respective base classes.

Parameters

- **images** (list of menpo.image.Image) – The list of training images.
- **group** (str or None, optional) – The landmark group that will be used to train the model. If None and the images only have a single landmark group, then that is the one that will be used. Note that all the training images need to have the specified landmark group.
- **holistic_features** (closure or list of closure, optional) – The features that will be extracted from the training images. Note that the features are extracted before warping the images to the reference shape. If list, then it must define a feature function per scale. Please refer to menpo.feature for a list of potential features.
- **reference_shape** (menpo.shape.PointCloud or None, optional) – The reference shape that will be used for building the AAM. The purpose of the reference shape is to normalise the size of the training images. The normalization is performed by rescaling all the training images so that the scale of their ground truth shapes matches the scale of the reference shape. Note that the reference shape is rescaled with respect to the diagonal before performing the normalization. If None, then the mean shape will be used.
- **diagonal** (int or None, optional) – This parameter is used to rescale the reference shape so that the diagonal of its bounding box matches the provided value. In other words, this parameter controls the size of the model at the highest scale. If None, then the reference shape does not get rescaled.
• **scales** *(float or tuple of float, optional)* – The scale value of each scale. They must provided in ascending order, i.e. from lowest to highest scale. If float, then a single scale is assumed.

• **expert_ensemble_cls** *(subclass of ExpertEnsemble, optional)* – The class to be used for training the ensemble of experts. The most common choice is CorrelationFilterExpertEnsemble.

• **patch_shape** *((int, int) or list of (int, int), optional)* – The shape of the patches to be extracted. If a list is provided, then it defines a patch shape per scale.

• **context_shape** *((int, int) or list of (int, int), optional)* – The context shape for the convolution. If a list is provided, then it defines a context shape per scale.

• **sample_offsets** *((n_offsets, n_dims) ndarray or None, optional)* – The sample_offsets to sample from within a patch. So (0, 0) is the centre of the patch (no offset) and (1, 0) would be sampling the patch from 1 pixel up the first axis away from the centre. If None, then no sample_offsets are applied.

• **transform** *(subclass of DL and DX, optional)* – A differential warp transform object, e.g. DifferentiablePiecewiseAffine or DifferentiableThinPlateSplines.

• **shape_model_cls** *(subclass of OrthoPDM, optional)* – The class to be used for building the shape model. The most common choice is OrthoPDM.

• **max_shape_components** *(int, float, list of those or None, optional)* – The number of shape components to keep. If int, then it sets the exact number of components. If float, then it defines the variance percentage that will be kept. If list, then it should define a value per scale. If a single number, then this will be applied to all scales. If None, then all the components are kept. Note that the unused components will be permanently trimmed.

• **max_appearance_components** *(int, float, list of those or None, optional)* – The number of appearance components to keep. If int, then it sets the exact number of components. If float, then it defines the variance percentage that will be kept. If list, then it should define a value per scale. If a single number, then this will be applied to all scales. If None, then all the components are kept. Note that the unused components will be permanently trimmed.

• **sigma** *(float or None, optional)* – If not None, the input images are smoothed with an isotropic Gaussian filter with the specified standard deviation.

• **boundary** *(int, optional)* – The number of pixels to be left as a safe margin on the boundaries of the reference frame (has potential effects on the gradient computation).

• **response_covariance** *(int, optional)* – The covariance of the generated Gaussian response.

• **patch_normalisation** *(callable, optional)* – The normalisation function to be applied on the extracted patches.

• **cosine_mask** *(bool, optional)* – If True, then a cosine mask (Hanning function) will be applied on the extracted patches.

• **verbose** *(bool, optional)* – If True, then the progress of building the model will be printed.

---

**References**

**appearance_reconstructions** *(appearance_parameters, n_iters_per_scale)*

Method that generates the appearance reconstructions given a set of appearance parameters. This is to be
combined with a `UnifiedAAMCLMResult` object, in order to generate the appearance reconstructions of a fitting procedure.

Parameters

- **appearance_parameters** (list of \((n\_params, n\_d)\) ndarray) – A set of appearance parameters per fitting iteration. It can be retrieved as a property of an `UnifiedAAMCLMResult` object.

- **n_iters_per_scale** (list of int) – The number of iterations per scale. This is necessary in order to figure out which appearance parameters correspond to the model of each scale. It can be retrieved as a property of a `UnifiedAAMCLMResult` object.

Returns **appearance_reconstructions** (list of `menpo.image.Image`) – List of the appearance reconstructions that correspond to the provided parameters.

**build_fitter_interfaces**( sampling)

Method that builds the correct fitting interface for a `UnifiedAAMCLMFitter`.

Parameters **sampling** (list of int or `ndarray` or None) – It defines a sampling mask per scale. If int, then it defines the sub-sampling step of the sampling mask. If `ndarray`, then it explicitly defines the sampling mask. If None, then no sub-sampling is applied.

Returns **fitter_interfaces** (list) – The list of fitting interfaces per scale.

**instance**( shape_weights=None, appearance_weights=None, scale_index=-1)

Generates a novel instance of the AAM part of the model given a set of shape and appearance weights. If no weights are provided, then the mean AAM instance is returned.

Parameters

- **shape_weights** ((n_weights,) `ndarray` or `list` or None, optional) – The weights of the shape model that will be used to create a novel shape instance. If None, the weights are assumed to be zero, thus the mean shape is used.

- **appearance_weights** ((n_weights,) `ndarray` or `list` or None, optional) – The weights of the appearance model that will be used to create a novel appearance instance. If None, the weights are assumed to be zero, thus the mean appearance is used.

- **scale_index** (int, optional) – The scale to be used.

Returns **image** (menpo.image.Image) – The AAM instance.

**random_instance**( scale_index=-1)

Generates a random instance of the AAM part of the model.

Parameters **scale_index** (int, optional) – The scale to be used.

Returns **image** (menpo.image.Image) – The AAM instance.

**shape_instance**( shape_weights=None, scale_index=-1)

Generates a novel shape instance given a set of shape weights. If no weights are provided, the mean shape is returned.

Parameters

- **shape_weights** ((n_weights,) `ndarray` or `list` or None, optional) – The weights of the shape model that will be used to create a novel shape instance. If None, the weights are assumed to be zero, thus the mean shape is used.

- **scale_index** (int, optional) – The scale to be used.

Returns **instance** (menpo.shape.PointCloud) – The shape instance.
**view_aam_widget** *(n_shape_parameters=5, n_appearance_parameters=5, parameters_bounds=(-3.0, 3.0), mode='multiple', figure_size=(10, 8))*

Visualizes the AAM part of the model using an interactive widget.

**Parameters**

- **n_shape_parameters** *(int or list of int or None, optional)* – The number of shape principal components to be used for the parameters sliders. If *int*, then the number of sliders per scale is the minimum between *n_parameters* and the number of active components per scale. If *list of int*, then a number of sliders is defined per scale. If *None*, all the active components per scale will have a slider.

- **n_appearance_parameters** *(int or list of int or None, optional)* – The number of appearance principal components to be used for the parameters sliders. If *int*, then the number of sliders per scale is the minimum between *n_parameters* and the number of active components per scale. If *list of int*, then a number of sliders is defined per scale. If *None*, all the active components per scale will have a slider.

- **parameters_bounds** *(float, float, optional)* – The minimum and maximum bounds, in std units, for the sliders.

- **mode** *(single, multiple, optional)* – If 'single', only a single slider is constructed along with a drop down menu. If 'multiple', a slider is constructed for each parameter.

- **figure_size** *(int, int, optional)* – The size of the rendered figure.

**view_appearance_models_widget** *(n_parameters=5, parameters_bounds=(-3.0, 3.0), mode='multiple', figure_size=(10, 8))*

Visualizes the appearance models of the Unified AAM CLM object using an interactive widget.

**Parameters**

- **n_parameters** *(int or list of int or None, optional)* – The number of appearance principal components to be used for the parameters sliders. If *int*, then the number of sliders per scale is the minimum between *n_parameters* and the number of active components per scale. If *list of int*, then a number of sliders is defined per scale. If *None*, all the active components per scale will have a slider.

- **parameters_bounds** *(float, float, optional)* – The minimum and maximum bounds, in std units, for the sliders.

- **mode** *(single, multiple, optional)* – If 'single', only a single slider is constructed along with a drop down menu. If 'multiple', a slider is constructed for each parameter.

- **figure_size** *(int, int, optional)* – The size of the rendered figure.

**view_clm_widget** *(n_shape_parameters=5, parameters_bounds=(-3.0, 3.0), mode='multiple', figure_size=(10, 8))*

Visualizes the CLM part of the model using an interactive widget.

**Parameters**

- **n_shape_parameters** *(int or list of int or None, optional)* – The number of shape principal components to be used for the parameters sliders. If *int*, then the number of sliders per scale is the minimum between *n_parameters* and the number of active components per scale. If *list of int*, then a number of sliders is defined per scale. If *None*, all the active components per scale will have a slider.

- **parameters_bounds** *(float, float, optional)* – The minimum and maximum bounds, in std units, for the sliders.

2.1. Deformable Models 165
• **mode** ([single, multiple], optional) – If 'single', only a single slider is constructed along with a drop down menu. If 'multiple', a slider is constructed for each parameter.

• **figure_size** ((int, int), optional) – The size of the rendered figure.

**Raises** `ValueError` – Only convolution-based expert ensembles can be visualized.

`view_expert_ensemble_widget` (figure_size=(10, 8))

Visualizes the ensemble of experts of the Unified AAM CLM object using an interactive widget.

**Parameters**

• **figure_size** ((int, int), optional) – The size of the plotted figures.

**Raises** `ValueError` – Only convolution-based expert ensembles can be visualized.

`view_shape_models_widget` (n_parameters=5, parameters_bounds=(-3.0, 3.0), mode='multiple', figure_size=(10, 8))

Visualizes the shape models of the Unified AAM CLM object using an interactive widget.

**Parameters**

• **n_parameters** (int or list of int or None, optional) – The number of shape principal components to be used for the parameters sliders. If int, then the number of sliders per scale is the minimum between `n_parameters` and the number of active components per scale. If list of int, then a number of sliders is defined per scale. If None, all the active components per scale will have a slider.

• **parameters_bounds** ((float, float), optional) – The minimum and maximum bounds, in std units, for the sliders.

• **mode** ([single, multiple], optional) – If 'single', only a single slider is constructed along with a drop down menu. If 'multiple', a slider is constructed for each parameter.

• **figure_size** ((int, int), optional) – The size of the rendered figure.

**property n_scales**

Returns the number of scales.

Type `int`

**Fitter**

The model optimised with a combination of Lucas-Kanade and Regularised Landmark Mean Shift algorithms.

**UnifiedAAMCLMFitter**

class `menpo.fit.unified_aam_clm.UnifiedAAMCLMFitter` (unified_aam_clm, algorithm_cls=<class 'menpo.fit.unified_aam_clm.algorithm.AlternatingRegularisedLandmarkMeanShift'>, n_shape=None, n_appearance=None, sampling=None)

Bases: `MultiScaleParametricFitter`

Class defining a Unified AAM - CLM fitter.

**Note:** When using a method with a parametric shape model, the first step is to **reconstruct the initial shape** using the shape model. The generated reconstructed shape is then used as initialisation for the iterative optimi-
This step takes place at each scale and it is not considered as an iteration, thus it is not counted for the provided max_iters.

### Parameters

- **unified_aam_clm** (*UnifiedAAMCLM or subclass*) – The trained unified AAM-CLM model.
- **algorithm_cls** (*class*, optional) – The unified optimisation algorithm that will get applied. The possible algorithms are:

<table>
<thead>
<tr>
<th>Class</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProjectOutRegularisedLandmarkMeanShift</td>
<td>Project-Out IC + RLMS</td>
</tr>
<tr>
<td>AlternatingRegularisedLandmarkMeanShift</td>
<td>Alternating IC + RLMS</td>
</tr>
</tbody>
</table>

- **n_shape** (*int or float or list of those or None, optional*) – The number of shape components that will be used. If *int*, then it defines the exact number of active components. If *float*, then it defines the percentage of variance to keep. If *int* or *float*, then the provided value will be applied for all scales. If *list*, then it defines a value per scale. If *None*, then all the available components will be used. Note that this simply sets the active components without trimming the unused ones. Also, the available components may have already been trimmed to max_shape_components during training.

- **n_appearance** (*int or float or list of those or None, optional*) – The number of appearance components that will be used. If *int*, then it defines the exact number of active components. If *float*, then it defines the percentage of variance to keep. If *int* or *float*, then the provided value will be applied for all scales. If *list*, then it defines a value per scale. If *None*, then all the available components will be used. Note that this simply sets the active components without trimming the unused ones. Also, the available components may have already been trimmed to max_appearance_components during training.

- **sampling** (*list of *int* or *ndarray* or *None*) – It defines a sampling mask per scale. If *int*, then it defines the sub-sampling step of the sampling mask. If *ndarray*, then it explicitly defines the sampling mask. If *None*, then no sub-sampling is applied.

### appearance_reconstructions

Method that generates the appearance reconstructions given a set of appearance parameters. This is to be combined with a *UnifiedAAMCLMResult* object, in order to generate the appearance reconstructions of a fitting procedure.

#### Parameters

- **appearance_parameters** (*list of *(n_params,)* *ndarray*) – A set of appearance parameters per fitting iteration. It can be retrieved as a property of an *UnifiedAAMCLMResult* object.

- **n_iters_per_scale** (*list of *int*) – The number of iterations per scale. This is necessary in order to figure out which appearance parameters correspond to the model of each scale. It can be retrieved as a property of a *UnifiedAAMCLMResult* object.

#### Returns

- **appearance_reconstructions** (*list of *menpo.image.Image*) – List of the appearance reconstructions that correspond to the provided parameters.

### fit_from_bb

Fits the multi-scale fitter to an image given an initial bounding box.

#### Parameters

- **image**
- **bounding_box**
- **max_iters=20**
- **gt_shape=None**
- **return_costs=False**
- ****kwargs
• **image** *(menpo.image.Image or subclass)* – The image to be fitted.

• **bounding_box** *(menpo.shape.PointDirectedGraph)* – The initial bounding box from which the fitting procedure will start. Note that the bounding box is used in order to align the model’s reference shape.

• **max_iters** *(int or list of int, optional)* – The maximum number of iterations. If *int*, then it specifies the maximum number of iterations over all scales. If *list of int*, then specifies the maximum number of iterations per scale.

• **gt_shape** *(menpo.shape.PointCloud, optional)* – The ground truth shape associated to the image.

• **return_costs** *(bool, optional)* – If *True*, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned *fitting_result*. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.

• **kwargs** *(dict, optional)* – Additional keyword arguments that can be passed to specific implementations.

Returns **fitting_result** *(MultiScaleNonParametricIterativeResult or subclass)* – The multi-scale fitting result containing the result of the fitting procedure.

The multi-scale fitting result containing the result of the fitting procedure.

---

**fit_from_shape** *(image, initial_shape, max_iters=20, gt_shape=None, return_costs=False, **kwargs)*

Fits the multi-scale fitter to an image given an initial shape.

Parameters

• **image** *(menpo.image.Image or subclass)* – The image to be fitted.

• **initial_shape** *(menpo.shape.PointCloud)* – The initial shape estimate from which the fitting procedure will start.

• **max_iters** *(int or list of int, optional)* – The maximum number of iterations. If *int*, then it specifies the maximum number of iterations over all scales. If *list of int*, then specifies the maximum number of iterations per scale.

• **gt_shape** *(menpo.shape.PointCloud, optional)* – The ground truth shape associated to the image.

• **return_costs** *(bool, optional)* – If *True*, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned *fitting_result*. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.

• **kwargs** *(dict, optional)* – Additional keyword arguments that can be passed to specific implementations.

Returns **fitting_result** *(MultiScaleNonParametricIterativeResult or subclass)* – The multi-scale fitting result containing the result of the fitting procedure.

---

**warped_images** *(image, shapes)*

Given an input test image and a list of shapes, it warps the image into the shapes. This is useful for generating the warped images of a fitting procedure stored within an *UnifiedAAMCLMResult*.

Parameters

• **image** *(menpo.image.Image or subclass)* – The input image to be warped.
• `shapes` *(list of menpo.shape.PointCloud)* – The list of shapes in which the image will be warped. The shapes are obtained during the iterations of a fitting procedure.

**Returns** `warped_images` *(list of menpo.image.MaskedImage or ndarray)* – The warped images.

**property** *holistic_features*

The features that are extracted from the input image at each scale in ascending order, i.e. from lowest to highest scale.

**Type** *list of closure*

**property** *n_scales*

Returns the number of scales.

**Type** *int*

**property** *reference_shape*

The reference shape that is used to normalise the size of an input image so that the scale of its initial fitting shape matches the scale of this reference shape.

**Type** *menpo.shape.PointCloud*

**property** *response_covariance*

Returns the covariance value of the desired Gaussian response used to train the ensemble of experts.

**Type** *int*

**property** *scales*

The scale value of each scale in ascending order, i.e. from lowest to highest scale.

**Type** *list of int or float*

**property** *unified_aam_clm*

The trained unified AAM-CLM model.

**Type** *UnifiedAAMCLM or subclass*

### Optimisation Algorithms

#### AlternatingRegularisedLandmarkMeanShift

class *menpofit.unified_aam_clm.AlternatingRegularisedLandmarkMeanShift* *(aam_interface, expert_ensemble, patch_shape, response_covariance, eps=1e-05, **kwargs)*

**Bases:** UnifiedAlgorithm

Alternating Inverse Compositional + Regularized Landmark Mean Shift

**run** *(image, initial_shape, gt_shape=None, max_iters=20, return_costs=False, prior=False, a=0.5)*

Execute the optimization algorithm.

**Parameters**

• *image* *(menpo.image.Image)* – The input test image.
• **initial_shape** (*menpo.shape.PointCloud*) – The initial shape from which the optimization will start.

• **gt_shape** (*menpo.shape.PointCloud* or None, optional) – The ground truth shape of the image. It is only needed in order to get passed in the optimization result object, which has the ability to compute the fitting error.

• **max_iters** (*int*, optional) – The maximum number of iterations. Note that the algorithm may converge, and thus stop, earlier.

• **return_costs** (*bool*, optional) – If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.

• **prior** (*bool*, optional) – If True, use a Gaussian priors over the latent shape and appearance spaces. see the reference [1] section 3.1.1 for details.

• **a** (*float*, optional) – Ratio of the image noise variance and the shape noise variance. See [1] section 5 equations (25) & (26) and footnote 6.

Returns **fitting_result** (*UnifiedAAMCLMAlgorithmResult*) – The parametric iterative fitting result.

### References

ProjectOutRegularisedLandmarkMeanShift

class menpofit.unified_aam_clm.ProjectOutRegularisedLandmarkMeanShift (*aam_interface*,  
expert_ensemble,  
patch_shape,  
response_covariance,  
eps=1e-05,  
**kwargs)

Bases: UnifiedAlgorithm

Project-Out Inverse Compositional + Regularized Landmark Mean Shift

run (*image*, *initial_shape*, *gt_shape=None*, *max_iters=20*, *return_costs=False*, *prior=False*, *a=0.5*)

Execute the optimization algorithm.

Parameters

• **image** (*menpo.image.Image*) – The input test image.

• **initial_shape** (*menpo.shape.PointCloud*) – The initial shape from which the optimization will start.

• **gt_shape** (*menpo.shape.PointCloud* or None, optional) – The ground truth shape of the image. It is only needed in order to get passed in the optimization result object, which has the ability to compute the fitting error.

• **max_iters** (*int*, optional) – The maximum number of iterations. Note that the algorithm may converge, and thus stop, earlier.
• **return_costs** *(bool, optional)* – If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned *fitting_result*. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.

• **prior** *(bool, optional)* – If True, use a Gaussian priors over the latent shape and appearance spaces. see the reference [1] section 3.1.1 for details.

• **a** *(float, optional)* – Ratio of the image noise variance and the shape noise variance. See [1] section 5 equations (25) & (26) and footnote 6.

**Returns**

- **fitting_result** *(UnifiedAAMCLMAlgorithmResult)* – The parametric iterative fitting result.

---

### References

#### Fitting Result

**UnifiedAAMCLMResult**

```python
class menpofit.unified_aam_clm.result.UnifiedAAMCLMResult(results, scales, affine_transforms, scale_transforms, image=None, gt_shape=None)
```

**Bases:** *MultiScaleParametricIterativeResult*

Class for storing the multi-scale iterative fitting result of a Unified AAM-CLM. It holds the shapes, shape parameters, appearance parameters and costs per iteration.

**Note:** When using a method with a parametric shape model, the first step is to **reconstruct the initial shape** using the shape model. The generated reconstructed shape is then used as initialisation for the iterative optimisation. This step is not counted in the number of iterations.

### Parameters

- **results** *(list of UnifiedAAMCLMAlgorithmResult)* – The list of optimization results per scale.
- **scales** *(list or tuple)* – The list of scale values per scale (low to high).
- **affine_transforms** *(list of menpo.transform.Affine)* – The list of affine transforms per scale that transform the shapes into the original image space.
- **scale_transforms** *(list of menpo.shape.Scale)* – The list of scaling transforms per scale.
- **image** *(menpo.image.Image or subclass or None, optional)* – The image on which the fitting process was applied. Note that a copy of the image will be assigned as an attribute. If None, then no image is assigned.
- **gt_shape** *(menpo.shape.PointCloud or None, optional)* – The ground truth shape associated with the image. If None, then no ground truth shape is assigned.

---

2.1. Deformable Models
displacements()

A list containing the displacement between the shape of each iteration and the shape of the previous one.

Type list of ndarray

displacements_stats(stat_type='mean')

A list containing a statistical metric on the displacements between the shape of each iteration and the shape of the previous one.

Parameters stat_type({'mean', 'median', 'min', 'max'}, optional) – Specifies a statistic metric to be extracted from the displacements.

Returns displacements_stat (list of float) – The statistical metric on the points displacements for each iteration.

Raises ValueError – type must be ‘mean’, ‘median’, ‘min’ or ‘max’

displacements_stats (stat_type='mean')

A list containing the displacement between the shape of each iteration and the shape of the previous one.

Parameters stat_type({'mean', 'median', 'min', 'max'}, optional) – Specifies a statistic metric to be extracted from the displacements.

Returns displacements_stat (list of float) – The statistical metric on the points displacements for each iteration.

Raises ValueError – type must be ‘mean’, ‘median’, ‘min’ or ‘max’

errors(compute_error=None)

Returns a list containing the error at each fitting iteration, if the ground truth shape exists.

Parameters compute_error (callable, optional) – Callable that computes the error between the shape at each iteration and the ground truth shape.

Returns errors (list of float) – The error at each iteration of the fitting process.

Raises ValueError – Ground truth shape has not been set, so the final error cannot be computed

final_error(compute_error=None)

Returns the final error of the fitting process, if the ground truth shape exists. This is the error computed based on the final_shape.

Parameters compute_error (callable, optional) – Callable that computes the error between the fitted and ground truth shapes.

Returns final_error (float) – The final error at the end of the fitting process.

Raises ValueError – Ground truth shape has not been set, so the final error cannot be computed

initial_error(compute_error=None)

Returns the initial error of the fitting process, if the ground truth shape and initial shape exist. This is the error computed based on the initial_shape.

Parameters compute_error (callable, optional) – Callable that computes the error between the initial and ground truth shapes.

Returns initial_error (float) – The initial error at the beginning of the fitting process.

Raises

• ValueError – Initial shape has not been set, so the initial error cannot be computed

• ValueError – Ground truth shape has not been set, so the initial error cannot be computed

plot_costs(figure_id=None, new_figure=False, render_lines=True, line_colour='b', line_style='-', line_width=2, render_markers=True, marker_style='o', marker_size=4, marker_face_colour='b', marker_edge_colour='k', marker_edge_width=1.0, render_axes=True, axes_font_name='sans-serif', axes_font_size=10, axes_font_style='normal', axes_font_weight='normal', axes_x_limits=0.0, axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None, figure_size=(10, 6), render_grid=True, grid_line_style='–', grid_line_width=0.5)

Plot of the cost function evolution at each fitting iteration.
Parameters

- **figure_id** *(object, optional)* – The id of the figure to be used.
- **new_figure** *(bool, optional)* – If True, a new figure is created.
- **render_lines** *(bool, optional)* – If True, the line will be rendered.
- **line_colour** *(colour or None, optional)* – The colour of the line. If None, the colour is sampled from the jet colormap. Example colour options are

  ```python
  {'r', 'g', 'b', 'c', 'm', 'k', 'w'}
  or
  (3, ) ndarray
  ```

- **line_style** *({-', '--', '-.', ':'}, optional)* – The style of the lines.
- **line_width** *(float, optional)* – The width of the lines.
- **render_markers** *(bool, optional)* – If True, the markers will be rendered.
- **marker_style** *(marker, optional)* – The style of the markers. Example marker options

  ```python
  {'.', ',', 'o', 'v', '^', '<', '>', '+', 'x', 'D', 'd', 's', 'p', '*', 'h', 'H', '1', '2', '3', '4', '8'}
  ```

- **marker_size** *(int, optional)* – The size of the markers in points.
- **marker_face_colour** *(colour or None, optional)* – The face (filling) colour of the markers. If None, the colour is sampled from the jet colormap. Example colour options are

  ```python
  {'r', 'g', 'b', 'c', 'm', 'k', 'w'}
  or
  (3, ) ndarray
  ```

- **marker_edge_colour** *(colour or None, optional)* – The edge colour of the markers. If None, the colour is sampled from the jet colormap. Example colour options are

  ```python
  {'r', 'g', 'b', 'c', 'm', 'k', 'w'}
  or
  (3, ) ndarray
  ```

- **marker_edge_width** *(float, optional)* – The width of the markers’ edge.
- **render_axes** *(bool, optional)* – If True, the axes will be rendered.
- **axes_font_name** *(See below, optional)* – The font of the axes. Example options

  ```python
  {'serif', 'sans-serif', 'cursive', 'fantasy', 'monospace'}
  ```

- **axes_font_size** *(int, optional)* – The font size of the axes.
- **axes_font_style** *({'normal', 'italic', 'oblique'})* – The font style of the axes.
- **axes_font_weight** *(See below, optional)* – The font weight of the axes. Example options


• **axes_x_limits** *(float or (float, float) or None, optional)* – The limits of the x axis. If *float*, then it sets padding on the right and left of the graph as a percentage of the curves’ width. If *tuple* or *list*, then it defines the axis limits. If *None*, then the limits are set automatically.

• **axes_y_limits** *(float or (float, float) or None, optional)* – The limits of the y axis. If *float*, then it sets padding on the top and bottom of the graph as a percentage of the curves’ height. If *tuple* or *list*, then it defines the axis limits. If *None*, then the limits are set automatically.

• **axes_x_ticks** *(list or tuple or None, optional)* – The ticks of the x axis.

• **axes_y_ticks** *(list or tuple or None, optional)* – The ticks of the y axis.

• **figure_size** *((float, float) or None, optional)* – The size of the figure in inches.

• **render_grid** *(bool, optional)* – If *True*, the grid will be rendered.

• **grid_line_style** *(\{'-\', ' --\', ' -.\', ':\'} , optional)* – The style of the grid lines.

• **grid_line_width** *(float, optional)* – The width of the grid lines.

Returns **renderer** *(menpo.visualize.GraphPlotter)* – The renderer object.

```python
plot_displacements(stat_type='mean', figure_id=None, new_figure=False, render_lines=True, line_colour='b', line_style='-', line_width=2, render_markers=True, marker_style='o', marker_size=4, marker_face_colour='b', marker_edge_colour='k', marker_edge_width=1.0, render_axes=True, axes_font_name='sans-serif', axes_font_size=10, axes_font_style='normal', axes_font_weight='normal', axes_x_limits=0.0, axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None, figure_size=(10, 6), render_grid=True, grid_line_style='-', grid_line_width=0.5)
```

Plot of a statistical metric of the displacement between the shape of each iteration and the shape of the previous one.

**Parameters**

• **stat_type** *(\{'mean, median, min, max\}', optional)* – Specifies a statistic metric to be extracted from the displacements (see also *displacements_stats()* method).

• **figure_id** *(object, optional)* – The id of the figure to be used.

• **new_figure** *(bool, optional)* – If *True*, a new figure is created.

• **render_lines** *(bool, optional)* – If *True*, the line will be rendered.

• **line_colour** *(colour or None (See below), optional)* – The colour of the line. If *None*, the colour is sampled from the jet colormap. Example *colour* options are

```python
{r, g, b, c, m, k, w}
or
(3, ) ndarry
```

• **line_style** *(str (See below), optional)* – The style of the lines. Example options:

```python
{\'-\', ' --\', '-', ':'}
```
• **line_width** *(float, optional)* – The width of the lines.

• **render_markers** *(bool, optional)* – If True, the markers will be rendered.

• **marker_style** *(str (See below), optional)* – The style of the markers. Example marker options

    \{., , , o, v, ^, <, >, +, x, D, d, s, p, *, h, H, l, 2, 3, 4, 8\}

• **marker_size** *(int, optional)* – The size of the markers in points.

• **marker_face_colour** *(colour or None, optional)* – The face (filling) colour of the markers. If None, the colour is sampled from the jet colormap. Example colour options are

    \{r, g, b, c, m, k, w\}

    or

    \{(3, ) ndarray\}

• **marker_edge_colour** *(colour or None, optional)* – The edge colour of the markers. If None, the colour is sampled from the jet colormap. Example colour options are

    \{r, g, b, c, m, k, w\}

    or

    \{(3, ) ndarray\}

• **marker_edge_width** *(float, optional)* – The width of the markers’ edge.

• **render_axes** *(bool, optional)* – If True, the axes will be rendered.

• **axes_font_name** *(str (See below), optional)* – The font of the axes. Example options

    \{serif, sans-serif, cursive, fantasy, monospace\}

• **axes_font_size** *(int, optional)* – The font size of the axes.

• **axes_font_style** *(str (See below), optional)* – The font style of the axes. Example options

    \{normal, italic, oblique\}

• **axes_font_weight** *(str (See below), optional)* – The font weight of the axes. Example options

    \{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black\}

• **axes_x_limits** *(float or (float, float) or None, optional)* – The limits of the x axis. If *float*, then it sets padding on the right and left of the graph as a percentage of the curves’ width. If *tuple* or *list*, then it defines the axis limits. If None, then the limits are set automatically.

• **axes_y_limits** *(float or (float, float) or None, optional)* – The limits of the y axis. If *float*, then it sets padding on the top and bottom of the graph as a percentage of the curves’ height. If *tuple* or *list*, then it defines the axis limits. If None, then the limits are set automatically.

• **axes_x_ticks** *(list or tuple or None, optional)* – The ticks of the x axis.

• **axes_y_ticks** *(list or tuple or None, optional)* – The ticks of the y axis.
• **figure_size** ((float, float) or None, optional) – The size of the figure in inches.

• **render_grid** (bool, optional) – If True, the grid will be rendered.

• **grid_line_style** (('-', ' --', '-.', ':'), optional) – The style of the grid lines.

• **grid_line_width** (float, optional) – The width of the grid lines.

Returns **renderer** (menpo.visualize.GraphPlotter) – The renderer object.

```python
plot_errors(compute_error=None, figure_id=None, new_figure=False, render_lines=True, line_colour='b', line_style='-', line_width=2, render_markers=True, marker_style='o', marker_size=4, marker_face_colour='b', marker_edge_colour='k', marker_edge_width=1.0, render_axes=True, axes_font_name='sans-serif', axes_font_size=10, axes_font_style='normal', axes_font_weight='normal', axes_x_limits=0.0, axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None, figure_size=(10, 6), render_grid=True, grid_line_style='--', grid_line_width=0.5)
```

Plot of the error evolution at each fitting iteration.

**Parameters**

• **compute_error** (callable, optional) – Callable that computes the error between the shape at each iteration and the ground truth shape.

• **figure_id** (object, optional) – The id of the figure to be used.

• **new_figure** (bool, optional) – If True, a new figure is created.

• **render_lines** (bool, optional) – If True, the line will be rendered.

• **line_colour** (colour or None (See below), optional) – The colour of the line. If None, the colour is sampled from the jet colormap. Example colour options are

```python
{r, g, b, c, m, k, w}
```

or

(3, ) ndarray

• **line_style** (str (See below), optional) – The style of the lines. Example options:

```python
(\{-, --, -.\}, )
```

• **line_width** (float, optional) – The width of the lines.

• **render_markers** (bool, optional) – If True, the markers will be rendered.

• **marker_style** (str (See below), optional) – The style of the markers. Example marker options

```python
{., ,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
```

• **marker_size** (int, optional) – The size of the markers in points.

• **marker_face_colour** (colour or None, optional) – The face (filling) colour of the markers. If None, the colour is sampled from the jet colormap. Example colour options are

```python
{r, g, b, c, m, k, w}
```

or

(3, ) ndarray

• **marker_edge_colour** (colour or None, optional) – The edge colour of the markers. If None, the colour is sampled from the jet colormap. Example colour options are
• **marker_edge_width** *(float, optional)* – The width of the markers’ edge.

• **render_axes** *(bool, optional)* – If True, the axes will be rendered.

• **axes_font_name** *(str (See below), optional)* – The font of the axes. Example options
  
  ```
  {serif, sans-serif, cursive, fantasy, monospace}
  ```

• **axes_font_size** *(int, optional)* – The font size of the axes.

• **axes_font_style** *(str (See below), optional)* – The font style of the axes. Example options
  
  ```
  {normal, italic, oblique}
  ```

• **axes_font_weight** *(str (See below), optional)* – The font weight of the axes. Example options
  
  ```
  {ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
  ```

• **axes_x_limits** *(float or (float, float) or None, optional)* – The limits of the x axis. If `float`, then it sets padding on the right and left of the graph as a percentage of the curves’ width. If `tuple` or `list`, then it defines the axis limits. If `None`, then the limits are set automatically.

• **axes_y_limits** *(float or (float, float) or None, optional)* – The limits of the y axis. If `float`, then it sets padding on the top and bottom of the graph as a percentage of the curves’ height. If `tuple` or `list`, then it defines the axis limits. If `None`, then the limits are set automatically.

• **axes_x_ticks** *(list or tuple or None, optional)* – The ticks of the x axis.

• **axes_y_ticks** *(list or tuple or None, optional)* – The ticks of the y axis.

• **figure_size** *((float, float) or None, optional)* – The size of the figure in inches.

• **render_grid** *(bool, optional)* – If True, the grid will be rendered.

• **grid_line_style** *(('-', '--', '-.', ':'), optional)* – The style of the grid lines.

• **grid_line_width** *(float, optional)* – The width of the grid lines.

**Returns** `renderer` *(menpo.visualize.GraphPlotter)* – The renderer object.

**reconstructed_initial_error** *(compute_error=None)*

Returns the error of the reconstructed initial shape of the fitting process, if the ground truth shape exists. This is the error computed based on the `reconstructed_initial_shapes[0]`.

**Parameters** `compute_error` *(callable, optional)* – Callable that computes the error between the reconstructed initial and ground truth shapes.

**Returns** `reconstructed_initial_error` *(float)* – The error that corresponds to the initial shape’s reconstruction.

**Raises** `ValueError` – Ground truth shape has not been set, so the reconstructed initial error cannot be computed
to_result (pass_image=True, pass_initial_shape=True, pass_gt_shape=True)

Returns a Result instance of the object, i.e. a fitting result object that does not store the iterations. This can be useful for reducing the size of saved fitting results.

Parameters

- **pass_image** (bool, optional) – If True, then the image will get passed (if it exists).
- **pass_initial_shape** (bool, optional) – If True, then the initial shape will get passed (if it exists).
- **pass_gt_shape** (bool, optional) – If True, then the ground truth shape will get passed (if it exists).

Returns `result` (Result) – The final “lightweight” fitting result.

view (figure_id=None, new_figure=False, render_image=True, render_final_shape=True, render_initial_shape=False, render_gt_shape=False, subplots_enabled=True, channels=None, interpolation='bilinear', cmap_name=None, alpha=1.0, masked=True, final_marker_face_colour='r', final_marker_edge_colour='k', final_line_colour='r', initial_marker_face_colour='b', initial_marker_edge_colour='k', initial_line_colour='b', gt_marker_face_colour='y', gt_marker_edge_colour='k', gt_line_colour='y', render_lines=True, line_style='-', line_width=2, render_markers=True, marker_style='o', marker_size=4, marker_edge_width=1.0, render_numbering=False, numbers_horizontal_align='center', numbers_vertical_align='bottom', numbers_font_name='sans-serif', numbers_font_style='normal', numbers_font_size=10, numbers_font_weight='normal', numbers_font_colour='k', render_legend=True, legend_title=' ', legend_font_name='sans-serif', legend_font_style='normal', legend_font_size=10, legend_font_weight='normal', legend_marker_scale=None, legend_location=2, legend_bbox_to_anchor=(1.05, 1.0), legend_border_axes_pad=None, legend_n_columns=1, legend_horizontal_spacing=None, legend_vertical_spacing=None, legend_border=True, legend_border_padding=None, legend_shadow=False, legend_rounded_corners=False, render_axes=True, axes_font_name='sans-serif', axes_font_style='normal', axes_font_weight='normal', axes_x_limits=None, axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None, figure_size=(7, 7))

Visualize the fitting result. The method renders the final fitted shape and optionally the initial shape, ground truth shape and the image, if they were provided.

Parameters

- **figure_id** (object, optional) – The id of the figure to be used.
- **new_figure** (bool, optional) – If True, a new figure is created.
- **render_image** (bool, optional) – If True and the image exists, then it gets rendered.
- **render_final_shape** (bool, optional) – If True, then the final fitting shape gets rendered.
- **render_initial_shape** (bool, optional) – If True and the initial fitting shape exists, then it gets rendered.
- **render_gt_shape** (bool, optional) – If True and the ground truth shape exists, then it gets rendered.
- **subplots_enabled** (bool, optional) – If True, then the requested final, initial and ground truth shapes get rendered on separate subplots.
- **channels** (int or list of int or all or None) – If int or list of int, the specified channel(s) will be rendered. If all, all the channels will be rendered in subplots. If None and the image is RGB, it will be rendered in RGB mode. If None and the image is not RGB, it is equivalent to all.
• **interpolation** *(See Below, optional)* – The interpolation used to render the image. For example, if `bilinear`, the image will be smooth and if `nearest`, the image will be pixelated. Example options

```
{none, nearest, bilinear, bicubic, spline16, spline36, hanning, hamming, hermite, kaiser, quadric, catrom, gaussian, bessel, mitchell, sinc, lanczos}
```

• **cmap_name** *(str, optional)* – If `None`, single channel and three channel images default to greyscale and rgb colormaps respectively.

• **alpha** *(float, optional)* – The alpha blending value, between 0 (transparent) and 1 (opaque).

• **masked** *(bool, optional)* – If `True`, then the image is rendered as masked.

• **final_marker_face_colour** *(See Below, optional)* – The face (filling) colour of the markers of the final fitting shape. Example options

```
{r, g, b, c, m, k, w}
```

• **final_marker_edge_colour** *(See Below, optional)* – The edge colour of the markers of the final fitting shape. Example options

```
{r, g, b, c, m, k, w}
```

• **final_line_colour** *(See Below, optional)* – The line colour of the final fitting shape. Example options

```
{r, g, b, c, m, k, w}
```

• **initial_marker_face_colour** *(See Below, optional)* – The face (filling) colour of the markers of the initial shape. Example options

```
{r, g, b, c, m, k, w}
```

• **initial_marker_edge_colour** *(See Below, optional)* – The edge colour of the markers of the initial shape. Example options

```
{r, g, b, c, m, k, w}
```

• **initial_line_colour** *(See Below, optional)* – The line colour of the initial shape. Example options

```
{r, g, b, c, m, k, w}
```
• **gt_marker_face_colour** *(See Below, optional)* – The face (filling) colour of the markers of the ground truth shape. Example options

```
[r, g, b, c, m, k, w]
```

or

```
(3, ) ndarray
```

• **gt_marker_edge_colour** *(See Below, optional)* – The edge colour of the markers of the ground truth shape. Example options

```
[r, g, b, c, m, k, w]
```

or

```
(3, ) ndarray
```

• **gt_line_colour** *(See Below, optional)* – The line colour of the ground truth shape. Example options

```
[r, g, b, c, m, k, w]
```

or

```
(3, ) ndarray
```

• **render_lines** *(bool or list of bool, optional)* – If True, the lines will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per shape in *(final, initial, groundtruth)* order.

• **line_style** *(str or list of str, optional)* – The style of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per shape in *(final, initial, groundtruth)* order. Example options:

```
{'-', '--', '-.', ':'}
```

• **line_width** *(float or list of float, optional)* – The width of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per shape in *(final, initial, groundtruth)* order.

• **render_markers** *(bool or list of bool, optional)* – If True, the markers will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per shape in *(final, initial, groundtruth)* order.

• **marker_style** *(str or list of str, optional)* – The style of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per shape in *(final, initial, groundtruth)* order. Example options:

```
{.,,,o,v,^,<,>,,+,x,D,d,s,p,*,,h,H,1,2,3,4,8}
```

• **marker_size** *(int or list of int, optional)* – The size of the markers in points. You can either provide a single value that will be used for all shapes or a list with a different value per shape in *(final, initial, groundtruth)* order.

• **marker_edge_width** *(float or list of float, optional)* – The width of the markers’ edge. You can either provide a single value that will be used for all shapes or a list with a different value per shape in *(final, initial, groundtruth)* order.

• **render_numbering** *(bool, optional)* – If True, the landmarks will be numbered.

• **numbers_horizontal_align** *(center, right, left)* – The horizontal alignment of the numbers’ texts.

• **numbers_vertical_align** *(center, top, bottom, baseline)* – The vertical alignment of the numbers’ texts.
- **numbers_font_name** *(See Below, optional)* – The font of the numbers. Example options

```
[serif, sans-serif, cursive, fantasy, monospace]
```

- **numbers_font_size** *(int, optional)* – The font size of the numbers.

- **numbers_font_style** *(normal, italic, oblique), optional)* – The font style of the numbers.

- **numbers_font_weight** *(See Below, optional)* – The font weight of the numbers. Example options

```
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```

- **numbers_font_colour** *(See Below, optional)* – The font colour of the numbers. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- **render_legend** *(bool, optional)* – If True, the legend will be rendered.

- **legend_title** *(str, optional)* – The title of the legend.

- **legend_font_name** *(See below, optional)* – The font of the legend. Example options

```
[serif, sans-serif, cursive, fantasy, monospace]
```

- **legend_font_style** *(normal, italic, oblique), optional)* – The font style of the legend.

- **legend_font_size** *(int, optional)* – The font size of the legend.

- **legend_font_weight** *(See Below, optional)* – The font weight of the legend. Example options

```
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```

- **legend_marker_scale** *(float, optional)* – The relative size of the legend markers with respect to the original

- **legend_location** *(int, optional)* – The location of the legend. The predefined values are:
| 'best'       | 0 |
| 'upper right'| 1 |
| 'upper left' | 2 |
| 'lower left' | 3 |
| 'lower right'| 4 |
| 'right'      | 5 |
| 'center left' | 6 |
| 'center right'| 7 |
| 'lower center'| 8 |
| 'upper center'| 9 |
| 'center'     | 10 |

- `legend_bbox_to_anchor ((float, float) tuple, optional)` – The bbox that the legend will be anchored.
- `legend_border_axes_pad (float, optional)` – The pad between the axes and legend border.
- `legend_n_columns (int, optional)` – The number of the legend’s columns.
- `legend_horizontal_spacing (float, optional)` – The spacing between the columns.
- `legend_vertical_spacing (float, optional)` – The vertical space between the legend entries.
- `legend_border (bool, optional)` – If True, a frame will be drawn around the legend.
- `legend_border_padding (float, optional)` – The fractional whitespace inside the legend border.
- `legend_shadow (bool, optional)` – If True, a shadow will be drawn behind legend.
- `legend_rounded_corners (bool, optional)` – If True, the frame’s corners will be rounded (fancybox).
- `render_axes (bool, optional)` – If True, the axes will be rendered.
- `axes_font_name (See Below, optional)` – The font of the axes. Example options

{serif, sans-serif, cursive, fantasy, monospace}

- `axes_font_size (int, optional)` – The font size of the axes.
- `axes_font_style ((normal, italic, oblique), optional)` – The font style of the axes.
- `axes_font_weight (See Below, optional)` – The font weight of the axes. Example options

{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}

- `axes_x_limits (float or (float, float) or None, optional)` – The limits of the x axis. If float, then it sets padding on the right and left of the Image as a percentage of the Image’s width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- `axes_y_limits ((float, float) tuple or None, optional)` – The limits of the y axis. If float, then it sets padding on the top and bottom of the Image as a percentage of the
Image’s height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.

- **axes_x_ticks** *(list or tuple or None, optional)* – The ticks of the x axis.
- **axes_y_ticks** *(list or tuple or None, optional)* – The ticks of the y axis.
- **figure_size** *(float, float) tuple or None optional)* – The size of the figure in inches.

Returns *renderer* *(class)* – The renderer object.

`view_iterations` *(figure_id=None, new_figure=False, iters=None, render_image=True, subplots_enabled=False, cmap_name=None, alpha=1.0, masked=True, render_lines=True, line_style='-', line_width=2, line_color=None, render_markers=True, marker_edge_color=None, marker_face_color=None, marker_style='o', marker_size=4, marker_edge_width=1.0, render_numbering=False, numbers_horizontal_align='center', numbers_vertical_align='bottom', numbers_font_name='sans-serif', numbers_font_size=10, numbers_font_style='normal', numbers_font_weight='normal', numbers_font_colour='k', render_legend=True, legend_title='', legend_font_name='sans-serif', legend_font_style='normal', legend_font_size=10, legend_font_weight='normal', legend_location=2, legend_bbox_to_anchor=(1.05, 1.0), legend_border_axes_pad=None, legend_n_columns=1, legend_horizontal_spacing=None, legend_vertical_spacing=None, legend border=True, legend_border_padding=None, legend_shadow=False, legend_rounded_corners=False, axes_pad=None, axes_font_name='sans-serif', axes_font_size=10, axes_font_style='normal', axes_font_weight='normal', axes_x_limits=None, axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None, figure_size=(7, 7))

Visualize the iterations of the fitting process.

**Parameters**

- **figure_id** *(object, optional)* – The id of the figure to be used.
- **new_figure** *(bool, optional)* – If True, a new figure is created.
- **iters** *(int or list of int or None, optional)* – The iterations to be visualized. If None, then all the iterations are rendered.
- **render_image** *(bool, optional)* – If True and the image exists, then it gets rendered.
- **subplots_enabled** *(bool, optional)* – If True, then the requested final, initial and ground truth shapes get rendered on separate subplots.
- **channels** *(int or list of int or all or None)* – If int or list of int, the specified channel(s) will be rendered. If all, all the channels will be rendered in subplots. If None and the image is RGB, it will be rendered in RGB mode. If None and the image is not RGB, it is equivalent to all.

<table>
<thead>
<tr>
<th>No.</th>
<th>Visualised shape</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>self.initial_shape</td>
<td>Initial shape</td>
</tr>
<tr>
<td>1</td>
<td>self.reconstructed_initial_shape</td>
<td>Reconstructed initial</td>
</tr>
<tr>
<td>2</td>
<td>self.shapes[2]</td>
<td>Iteration 1</td>
</tr>
<tr>
<td>i</td>
<td>self.shapes[i]</td>
<td>Iteration i-1</td>
</tr>
<tr>
<td>n_iters+1</td>
<td>self.final_shape</td>
<td>Final shape</td>
</tr>
</tbody>
</table>

2.1. Deformable Models
- **interpolation** (*str* (See Below), optional) – The interpolation used to render the image. For example, if *bilinear*, the image will be smooth and if *nearest*, the image will be pixelated. Example options

```plaintext
{none, nearest, bilinear, bicubic, spline16, spline36, hanning, hamming, hermite, kaiser, quadric, catrom, gaussian, bessel, mitchell, sinc, lanczos}
```

- **cmap_name** (*str*, optional) – If *None*, single channel and three channel images default to greyscale and rgb colormaps respectively.

- **alpha** (*float*, optional) – The alpha blending value, between 0 (transparent) and 1 (opaque).

- **masked** (*bool*, optional) – If *True*, then the image is rendered as masked.

- **render_lines** (*bool or list of bool*, optional) – If *True*, the lines will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.

- **line_style** (*str or list of str* (See below), optional) – The style of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options:

```plaintext
{-, --, -., :}
```

- **line_width** (*float or list of float*, optional) – The width of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.

- **line_colour** (*colour or list of colour* (See Below), optional) – The colour of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```plaintext
{r, g, b, c, m, k, w}
```

or

```plaintext
(3, ) ndarray
```

- **render_markers** (*bool or list of bool*, optional) – If *True*, the markers will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.

- **marker_style** (*str or list of str* (See below), optional) – The style of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options:

```plaintext
{., ,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
```

- **marker_size** (*int or list of int*, optional) – The size of the markers in points. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.

- **marker_edge_colour** (*colour or list of colour* (See Below), optional) – The edge colour of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```plaintext
{r, g, b, c, m, k, w}
```

or

```plaintext
(3, ) ndarray
```
- **marker_face_colour** *(colour or list of colour)* (See Below, optional) – The face (filling) colour of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```
[r, g, b, c, m, k, w]
or
(3, ) ndarray
```

- **marker_edge_width** *(float or list of float)*, optional) – The width of the markers’ edge. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.

- **render_numbering** *(bool)*, optional) – If True, the landmarks will be numbered.

- **numbers_horizontal_align** *(str)* (See below), optional) – The horizontal alignment of the numbers’ texts. Example options

```
{center, right, left}
```

- **numbers_vertical_align** *(str)* (See below), optional) – The vertical alignment of the numbers’ texts. Example options

```
{center, top, bottom, baseline}
```

- **numbers_font_name** *(str)* (See below), optional) – The font of the numbers. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- **numbers_font_size** *(int)*, optional) – The font size of the numbers.

- **numbers_font_style** *(normal, italic, oblique)*, optional) – The font style of the numbers.

- **numbers_font_weight** *(str)* (See below), optional) – The font weight of the numbers. Example options

```
{ultralight, light, normal, regular, book, medium, roman,
semibold, demibold, demi, bold, heavy, extra bold, black}
```

- **numbers_font_colour** *(See Below, optional)* – The font colour of the numbers. Example options

```
[r, g, b, c, m, k, w]
or
(3, ) ndarray
```

- **render_legend** *(bool)*, optional) – If True, the legend will be rendered.

- **legend_title** *(str)*, optional) – The title of the legend.

- **legend_font_name** *(See below, optional)* – The font of the legend. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- **legend_font_style** *(str)* (See below), optional) – The font style of the legend. Example options
MenpoFit Documentation, Release 0.6.1

- **legend_font_size** *(int, optional)* – The font size of the legend.
- **legend_font_weight** *(str (See below), optional)* – The font weight of the legend. Example options

```plaintext
(normal, italic, oblique)
```

- **legend_marker_scale** *(float, optional)* – The relative size of the legend markers with respect to the original
- **legend_location** *(int, optional)* – The location of the legend. The predefined values are:

```plaintext
\[
\begin{array}{c|c}
\text{value} & \text{location} \\
0 & \text{best} \\
1 & \text{upper right} \\
2 & \text{upper left} \\
3 & \text{lower left} \\
4 & \text{lower right} \\
5 & \text{right} \\
6 & \text{center left} \\
7 & \text{center right} \\
8 & \text{lower center} \\
9 & \text{upper center} \\
10 & \text{center} \\
\end{array}
\]
```
- **legend_bbox_to_anchor** *((float, float) tuple, optional)* – The bbox that the legend will be anchored.
- **legend_border_axes_pad** *(float, optional)* – The pad between the axes and legend border.
- **legend_n_columns** *(int, optional)* – The number of the legend’s columns.
- **legend_horizontal_spacing** *(float, optional)* – The spacing between the columns.
- **legend_vertical_spacing** *(float, optional)* – The vertical space between the legend entries.
- **legend_border** *(bool, optional)* – If True, a frame will be drawn around the legend.
- **legend_border_padding** *(float, optional)* – The fractional whitespace inside the legend border.
- **legend_shadow** *(bool, optional)* – If True, a shadow will be drawn behind legend.
- **legend_rounded_corners** *(bool, optional)* – If True, the frame’s corners will be rounded (fancybox).
- **render_axes** *(bool, optional)* – If True, the axes will be rendered.
- **axes_font_name** *(str (See below), optional)* – The font of the axes. Example options

```plaintext
(serif, sans-serif, cursive, fantasy, monospace)
```
- **axes_font_size** *(int, optional)* – The font size of the axes.
• **axes_font_style** ((normal, italic, oblique), optional) – The font style of the axes.

• **axes_font_weight** (str (See below), optional) – The font weight of the axes. Example options

   {ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}

• **axes_x_limits** (float or (float, float) or None, optional) – The limits of the x axis. If float, then it sets padding on the right and left of the Image as a percentage of the Image’s width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.

• **axes_y_limits** ((float, float) tuple or None, optional) – The limits of the y axis. If float, then it sets padding on the top and bottom of the Image as a percentage of the Image’s height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.

• **axes_x_ticks** (list or tuple or None, optional) – The ticks of the x axis.

• **axes_y_ticks** (list or tuple or None, optional) – The ticks of the y axis.

• **figure_size** ((float, float) tuple or None optional) – The size of the figure in inches.

   Returns `renderer` (class) – The renderer object.

   `view_widget` (figure_size=(7, 7))

   Visualizes the result object using an interactive widget.

   Parameters `figure_size` ((int, int), optional) – The initial size of the rendered figure.

property **appearance_parameters**

Returns the list of appearance parameters obtained at each iteration of the fitting process. The list includes the parameters of the initial_shape (if it exists) and final_shape.

   Type list of (n_params,) ndarray

property **costs**

Returns a list with the cost per iteration. It returns None if the costs are not computed.

   Type list of float or None

property **final_shape**

Returns the final shape of the fitting process.

   Type menpo.shape.PointCloud

property **gt_shape**

Returns the ground truth shape associated with the image. In case there is not an attached ground truth shape, then None is returned.

   Type menpo.shape.PointCloud or None

property **image**

Returns the image that the fitting was applied on, if it was provided. Otherwise, it returns None.

   Type menpo.shape.Image or subclass or None

property **initial_shape**

Returns the initial shape that was provided to the fitting method to initialise the fitting process. In case the initial shape does not exist, then None is returned.

   Type menpo.shape.PointCloud or None
**property is_iterative**
Flag whether the object is an iterative fitting result.

Type: `bool`

**property n_iters**
Returns the total number of iterations of the fitting process.

Type: `int`

**property n_iters_per_scale**
Returns the number of iterations per scale of the fitting process.

Type: `list of int`

**property n_scales**
Returns the number of scales used during the fitting process.

Type: `int`

**property reconstructed_initial_shapes**
Returns the result of the reconstruction step that takes place at each scale before applying the iterative optimisation.

Type: `list of menpo.shape.PointCloud`

**property shape_parameters**
Returns the list of shape parameters obtained at each iteration of the fitting process. The list includes the parameters of the `initial_shape` (if it exists) and `final_shape`.

Type: `list of (n_params,) ndarray`

**property shapes**
Returns the list of shapes obtained at each iteration of the fitting process. The list includes the `initial_shape` (if it exists) and `final_shape`.

Type: `list of menpo.shape.PointCloud`

---

**UnifiedAAMCLMAlgorithmResult**

```python
class menpofit.unified_aam_clm.result.UnifiedAAMCLMAlgorithmResult(shapes, shape_parameters, appearance_parameters, initial_shape=None, image=None, gt_shape=None, costs=None)
```

Bases: `ParametricIterativeResult`

Class for storing the iterative result of a Unified AAM-CLM optimisation algorithm.

**Note:** When using a method with a parametric shape model, the first step is to **reconstruct the initial shape** using the shape model. The generated reconstructed shape is then used as initialisation for the iterative optimisation. This step is not counted in the number of iterations.
MenpoFit Documentation, Release 0.6.1

- **shapes** (list of `menpo.shape.PointCloud`) – The list of shapes per iteration. The first and last members correspond to the initial and final shapes, respectively.

- **shape_parameters** (list of `(n_shape_parameters,) ndarray`) – The list of shape parameters per iteration. The first and last members correspond to the initial and final shapes, respectively.

- **appearance_parameters** (list of `(n_appearance_parameters,) ndarray`) – The list of appearance parameters per iteration. The first and last members correspond to the initial and final shapes, respectively.

- **initial_shape** (`menpo.shape.PointCloud` or `None`, optional) – The initial shape from which the fitting process started. If `None`, then no initial shape is assigned.

- **image** (`menpo.image.Image` or subclass or `None`, optional) – The image on which the fitting process was applied. Note that a copy of the image will be assigned as an attribute. If `None`, then no image is assigned.

- **gt_shape** (`menpo.shape.PointCloud` or `None`, optional) – The ground truth shape associated with the image. If `None`, then no ground truth shape is assigned.

- **costs** (list of `float` or `None`, optional) – The list of cost per iteration. If `None`, then it is assumed that the cost function cannot be computed for the specific algorithm.

**displacements()**

A list containing the displacement between the shape of each iteration and the shape of the previous one.

Type list of `ndarray`

**displacements_stats**(stat_type='mean')

A list containing a statistical metric on the displacements between the shape of each iteration and the shape of the previous one.

Parameters stat_type({'mean', 'median', 'min', 'max'}, optional) – Specifies a statistic metric to be extracted from the displacements.

Returns displacements_stat (list of float) – The statistical metric on the points displacements for each iteration.

Raises ValueError – type must be ‘mean’, ‘median’, ‘min’ or ‘max’

**errors**(compute_error=None)

Returns a list containing the error at each fitting iteration, if the ground truth shape exists.

Parameters compute_error (callable, optional) – Callable that computes the error between the shape at each iteration and the ground truth shape.

Returns errors (list of float) – The error at each iteration of the fitting process.

Raises ValueError – Ground truth shape has not been set, so the final error cannot be computed

**final_error**(compute_error=None)

Returns the final error of the fitting process, if the ground truth shape exists. This is the error computed based on the final shape.

Parameters compute_error (callable, optional) – Callable that computes the error between the fitted and ground truth shapes.

Returns final_error (float) – The final error at the end of the fitting process.

Raises ValueError – Ground truth shape has not been set, so the final error cannot be computed

2.1. Deformable Models
**initial_error** *(compute_error=None)*

Returns the initial error of the fitting process, if the ground truth shape and initial shape exist. This is the error computed based on the `initial_shape`.

**Parameters**

- `compute_error` *(callable, optional)* – Callable that computes the error between the initial and ground truth shapes.

**Returns**

- `initial_error` *(float)* – The initial error at the beginning of the fitting process.

**Raises**

- `ValueError` – Initial shape has not been set, so the initial error cannot be computed
- `ValueError` – Ground truth shape has not been set, so the initial error cannot be computed

**plot_costs** *(figure_id=None, new_figure=False, render_lines=True, line_colour='b', line_style='-', line_width=2, render_markers=True, marker_style='o', marker_size=4, marker_face_colour='b', marker_edge_colour='k', marker_edge_width=1.0, render_axes=True, axes_font_name='sans-serif', axes_font_size=10, axes_font_style='normal', axes_font_weight='normal', axes_x_limits=0.0, axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None, figure_size=(10, 6), render_grid=True, grid_line_style='–', grid_line_width=0.5)*

Plot of the cost function evolution at each fitting iteration.

**Parameters**

- `figure_id` *(object, optional)* – The id of the figure to be used.
- `new_figure` *(bool, optional)* – If True, a new figure is created.
- `render_lines` *(bool, optional)* – If True, the line will be rendered.
- `line_colour` *(colour or None, optional)* – The colour of the line. If None, the colour is sampled from the jet colormap. Example colour options are

```plaintext
('r', 'g', 'b', 'c', 'm', 'k', 'w')
or
(3, ) ndarray
```

- `line_style` *(({'-', '--', '-.', ':'}, optional)* – The style of the lines.
- `line_width` *(float, optional)* – The width of the lines.
- `render_markers` *(bool, optional)* – If True, the markers will be rendered.
- `marker_style` *(marker, optional)* – The style of the markers. Example marker options

```plaintext
('.', ',', 'o', 'v', '^', '<', '>', '+', 'x', 'D', 'd', 's', 'p', '*', 'h', 'H', '1', '2', '3', '4', '8')
```

- `marker_size` *(int, optional)* – The size of the markers in points.
- `marker_face_colour` *(colour or None, optional)* – The face (filling) colour of the markers. If None, the colour is sampled from the jet colormap. Example colour options are

```plaintext
('r', 'g', 'b', 'c', 'm', 'k', 'w')
or
(3, ) ndarray
```

- `marker_edge_colour` *(colour or None, optional)* – The edge colour of the markers. If None, the colour is sampled from the jet colormap. Example colour options are
**MenpoFit Documentation, Release 0.6.1**

```python
('r', 'g', 'b', 'c', 'm', 'k', 'w')
or
(3, ) ndarray
```

- **marker_edge_width** *(float, optional)* – The width of the markers’ edge.
- **render_axes** *(bool, optional)* – If True, the axes will be rendered.
- **axes_font_name** *(See below, optional)* – The font of the axes. Example options
  ```python
  ['serif', 'sans-serif', 'cursive', 'fantasy', 'monospace']
  ```
- **axes_font_size** *(int, optional)* – The font size of the axes.
- **axes_font_style** *(['normal', 'italic', 'oblique'], optional)* – The font style of the axes.
- **axes_font_weight** *(See below, optional)* – The font weight of the axes. Example options
  ```python
  ['ultralight', 'light', 'normal', 'regular', 'book', 'medium',
  'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy',
  'extra bold', 'black']
  ```
- **axes_x_limits** *(float or (float, float) or None, optional)* – The limits of the x axis. If float, then it sets padding on the right and left of the graph as a percentage of the curves’ width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- **axes_y_limits** *(float or (float, float) or None, optional)* – The limits of the y axis. If float, then it sets padding on the top and bottom of the graph as a percentage of the curves’ height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- **axes_x_ticks** *(list or tuple or None, optional)* – The ticks of the x axis.
- **axes_y_ticks** *(list or tuple or None, optional)* – The ticks of the y axis.
- **figure_size** *((float, float) or None, optional)* – The size of the figure in inches.
- **render_grid** *(bool, optional)* – If True, the grid will be rendered.
- **grid_line_style** *(('-', '--', '-.', ':'), optional)* – The style of the grid lines.
- **grid_line_width** *(float, optional)* – The width of the grid lines.

**Returns** renderer *(menpo.visualize.GraphPlotter)* – The renderer object.

```python
plot_displacements(stat_type='mean', figure_id=None, new_figure=False, render_lines=True,
line_colour='b', line_style='-', line_width=2, render_markers=True,
marker_style='o', marker_size=4, marker_face_colour='b',
marker_edge_colour='k', marker_edge_width=1.0, render_axes=True,
axes_font_name='sans-serif', axes_font_size=10, axes_font_style='normal',
axes_font_weight='normal', axes_x_limits=0.0, axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None, figure_size=(10, 6),
render_grid=True, grid_line_style='-', grid_line_width=0.5)
```

Plot of a statistical metric of the displacement between the shape of each iteration and the shape of the previous one.

**Parameters**
• **stat_type**({mean, median, min, max}, optional) – Specifies a statistic metric to be extracted from the displacements (see also `displacements_stats()` method).

• **figure_id** (object, optional) – The id of the figure to be used.

• **new_figure** (bool, optional) – If True, a new figure is created.

• **render_lines** (bool, optional) – If True, the line will be rendered.

• **line_colour** (colour or None (See below), optional) – The colour of the line. If None, the colour is sampled from the jet colormap. Example `colour` options are

  ```
  {r, g, b, c, m, k, w}
  or
  (3, ) ndarray
  ```

• **line_style** (str (See below), optional) – The style of the lines. Example options:

  ```
  {-, --, -, ., :}
  ```

• **line_width** (float, optional) – The width of the lines.

• **render_markers** (bool, optional) – If True, the markers will be rendered.

• **marker_style** (str (See below), optional) – The style of the markers. Example `marker` options

  ```
  {., ,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
  ```

• **marker_size** (int, optional) – The size of the markers in points.

• **marker_face_colour** (colour or None, optional) – The face (filling) colour of the markers. If None, the colour is sampled from the jet colormap. Example `colour` options are

  ```
  {r, g, b, c, m, k, w}
  or
  (3, ) ndarray
  ```

• **marker_edge_colour** (colour or None, optional) – The edge colour of the markers. If None, the colour is sampled from the jet colormap. Example `colour` options are

  ```
  {r, g, b, c, m, k, w}
  or
  (3, ) ndarray
  ```

• **marker_edge_width** (float, optional) – The width of the markers’ edge.

• **render_axes** (bool, optional) – If True, the axes will be rendered.

• **axes_font_name** (str (See below), optional) – The font of the axes. Example options

  ```
  {serif, sans-serif, cursive, fantasy, monospace}
  ```

• **axes_font_size** (int, optional) – The font size of the axes.

• **axes_font_style** (str (See below), optional) – The font style of the axes. Example options

  ```
  {normal, italic, oblique}
  ```
• **axes_font_weight** *(str)* (See below), optional – The font weight of the axes. Example options:

```plaintext
(ultralight, light, normal, regular, book, medium, roman,
 semibold, demibold, demi, bold, heavy, extra bold, black)
```

• **axes_x_limits** *(float)* (See below), optional – The limits of the x axis. If `float`, then it sets padding on the right and left of the graph as a percentage of the curves’ width. If `tuple` or `list`, then it defines the axis limits. If `None`, then the limits are set automatically.

• **axes_y_limits** *(float)* (See below), optional – The limits of the y axis. If `float`, then it sets padding on the top and bottom of the graph as a percentage of the curves’ height. If `tuple` or `list`, then it defines the axis limits. If `None`, then the limits are set automatically.

• **axes_x_ticks** *(list)* (See below), optional – The ticks of the x axis.

• **axes_y_ticks** *(list)* (See below), optional – The ticks of the y axis.

• **figure_size** *(float)* (See below), optional – The size of the figure in inches.

• **render_grid** *(bool)*, optional – If `True`, the grid will be rendered.

• **grid_line_style** *(str)* (See below), optional – The style of the grid lines.

• **grid_line_width** *(float)*, optional – The width of the grid lines.

Returns **renderer** *(menpo.visualize.GraphPlotter)* – The renderer object.

**plot_errors** *(compute_error=None, figure_id=None, new_figure=False, render_lines=True, line_colour='b', line_style='-', line_width=2, render_markers=True, marker_style='o', marker_size=4, marker_face_colour='b', marker_edge_colour='k', marker_edge_width=1.0, render_axes=True, axes_font_name='sans-serif', axes_font_size=10, axes_font_style='normal', axes_font_weight='normal', axes_x_limits=0.0, axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None, figure_size=(10, 6), render_grid=True, grid_line_style='--', grid_line_width=0.5)*

Plot of the error evolution at each fitting iteration.

Parameters

• **compute_error** *(callable)*, optional – Callable that computes the error between the shape at each iteration and the ground truth shape.

• **figure_id** *(object)*, optional – The id of the figure to be used.

• **new_figure** *(bool)*, optional – If `True`, a new figure is created.

• **render_lines** *(bool)*, optional – If `True`, the line will be rendered.

• **line_colour** *(colour)* or `None` (See below), optional – The colour of the line. If `None`, the colour is sampled from the jet colormap. Example `colour` options are:

```plaintext
{r, g, b, c, m, k, w}
```
or

```plaintext
(3, ) ndarray
```

• **line_style** *(str)* (See below), optional – The style of the lines. Example options:

```plaintext
(\{-, --, -, :\})
```

• **line_width** *(float)*, optional – The width of the lines.
• **render_markers** (*bool*, optional) – If True, the markers will be rendered.

• **marker_style** (*str* (See below), optional) – The style of the markers. Example *marker* options

\[
\{., \,, \, o, \, v, \, ^, \, <, \, >, \, +, \, *, \, h, \, H, \, 1, \, 2, \, 3, \, 4, \, 8\}
\]

• **marker_size** (*int*, optional) – The size of the markers in points.

• **marker_face_colour** (*colour* or *None*, optional) – The face (filling) colour of the markers. If *None*, the colour is sampled from the jet colormap. Example *colour* options are

\[
\{r, \, g, \, b, \, c, \, m, \, k, \, w\}
\]

or

\[
(3, \, ) \text{ ndarray}
\]

• **marker_edge_colour** (*colour* or *None*, optional) – The edge colour of the markers. If *None*, the colour is sampled from the jet colormap. Example *colour* options are

\[
\{r, \, g, \, b, \, c, \, m, \, k, \, w\}
\]

or

\[
(3, \, ) \text{ ndarray}
\]

• **marker_edge_width** (*float*, optional) – The width of the markers’ edge.

• **render_axes** (*bool*, optional) – If True, the axes will be rendered.

• **axes_font_name** (*str* (See below), optional) – The font of the axes. Example options

\[
\{\text{serif, sans-serif, cursive, fantasy, monospace}\}
\]

• **axes_font_size** (*int*, optional) – The font size of the axes.

• **axes_font_style** (*str* (See below), optional) – The font style of the axes. Example options

\[
\{\text{normal, italic, oblique}\}
\]

• **axes_font_weight** (*str* (See below), optional) – The font weight of the axes. Example options

\[
\{\text{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}\}
\]

• **axes_x_limits** (*float* or (*float*, *float*) or *None*, optional) – The limits of the x axis. If *float*, then it sets padding on the right and left of the graph as a percentage of the curves’ width. If *tuple* or *list*, then it defines the axis limits. If *None*, then the limits are set automatically.

• **axes_y_limits** (*float* or (*float*, *float*) or *None*, optional) – The limits of the y axis. If *float*, then it sets padding on the top and bottom of the graph as a percentage of the curves’ height. If *tuple* or *list*, then it defines the axis limits. If *None*, then the limits are set automatically.

• **axes_x_ticks** (*list* or *tuple* or *None*, optional) – The ticks of the x axis.

• **axes_y_ticks** (*list* or *tuple* or *None*, optional) – The ticks of the y axis.

• **figure_size** (*(*float*, *float*)* or *None*, optional) – The size of the figure in inches.
MenpoFit Documentation, Release 0.6.1

- **render_grid** *(bool, optional)* – If True, the grid will be rendered.
- **grid_line_style** *({'-', '--', '-.', ':'}, optional)* – The style of the grid lines.
- **grid_line_width** *(float, optional)* – The width of the grid lines.

Returns **renderer** *(menpo.visualize.GraphPlotter)* – The renderer object.

**reconstructed_initial_error** *(compute_error=None)*

Returns the error of the reconstructed initial shape of the fitting process, if the ground truth shape exists. This is the error computed based on the **reconstructed_initial_shape**.

Parameters **compute_error** *(callable, optional)* – Callable that computes the error between the reconstructed initial and ground truth shapes.

Returns **reconstructed_initial_error** *(float)* – The error that corresponds to the initial shape’s reconstruction.

Raises **ValueError** – Ground truth shape has not been set, so the reconstructed initial error cannot be computed.

**to_result** *(pass_image=True, pass_initial_shape=True, pass_gt_shape=True)*

Returns a **Result** instance of the object, i.e. a fitting result object that does not store the iterations. This can be useful for reducing the size of saved fitting results.

Parameters

- **pass_image** *(bool, optional)* – If True, then the image will get passed (if it exists).
- **pass_initial_shape** *(bool, optional)* – If True, then the initial shape will get passed (if it exists).
- **pass_gt_shape** *(bool, optional)* – If True, then the ground truth shape will get passed (if it exists).

Returns **result** *(Result)* – The final “lightweight” fitting result.

**view** *(figure_id=None, new_figure=False, render_image=True, render_final_shape=True, render_initial_shape=False, render_gt_shape=False, subplots_enabled=True, channels=None, interpolation='bilinear', cmap_name=None, alpha=1.0, masked=True, final_marker_face_colour='r', final_marker_edge_colour='k', final_line_colour='r', initial_marker_face_colour='b', initial_marker_edge_colour='k', initial_line_colour='b', gt_marker_face_colour='y', gt_marker_edge_colour='k', gt_line_colour='y', render_lines=True, line_style='-', line_width=2, render_markers=True, marker_style='o', marker_size=4, marker_edge_width=1.0, render_numbering=False, numbers_horizontal_align='center', numbers_vertical_align='bottom', numbers_font_name='sans-serif', numbers_font_size=10, numbers_font_style='normal', numbers_font_weight='normal', renderLegend=True, legend_title='', legend_font_name='sans-serif', legend_font_style='normal', legend_font_size=10, legend_font_weight='normal', legend_marker_scale=None, legend_location=1, legend_bbox_to_anchor=(1.05, 1.0), legend_border_axes_pad=None, legend_n_columns=1, legend_horizontal_spacing=None, legend_vertical_spacing=None, legend_border=True, legend_border_padding=None, legend_shadow=False, legend_rounded_corners=False, render_axes=True, axes_font_name='sans-serif', axes_font_size=10, axes_font_style='normal', axes_font_weight='normal', axes_x_limits=None, axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None, figure_size=(7, 7))

Visualize the fitting result. The method renders the final fitted shape and optionally the initial shape, ground truth shape and the image, id they were provided.

Parameters

2.1. Deformable Models

195
• **figure_id** (*object*, optional) – The id of the figure to be used.

• **new_figure** (*bool*, optional) – If True, a new figure is created.

• **render_image** (*bool*, optional) – If True and the image exists, then it gets rendered.

• **render_final_shape** (*bool*, optional) – If True, then the final fitting shape gets rendered.

• **render_initial_shape** (*bool*, optional) – If True and the initial fitting shape exists, then it gets rendered.

• **render_gt_shape** (*bool*, optional) – If True and the ground truth shape exists, then it gets rendered.

• **subplots_enabled** (*bool*, optional) – If True, then the requested final, initial and ground truth shapes get rendered on separate subplots.

• **channels** (*int* or *list of int* or *all* or *None*) – If *int* or *list of int*, the specified channel(s) will be rendered. If all, all the channels will be rendered in subplots. If None and the image is RGB, it will be rendered in RGB mode. If None and the image is not RGB, it is equivalent to all.

• **interpolation** (*See Below*, optional) – The interpolation used to render the image. For example, if bilinear, the image will be smooth and if nearest, the image will be pixelated. Example options

```python
{none, nearest, bilinear, bicubic, spline16, spline36, hanning, hamming, hermite, kaiser, quadric, catrom, gaussian, bessel, mitchell, sinc, lanczos}
```

• **cmap_name** (*str*, optional.) – If None, single channel and three channel images default to greyscale and rgb colormaps respectively.

• **alpha** (*float*, optional) – The alpha blending value, between 0 (transparent) and 1 (opaque).

• **masked** (*bool*, optional) – If True, then the image is rendered as masked.

• **final_marker_face_colour** (*See Below*, optional) – The face (filling) colour of the markers of the final fitting shape. Example options

```python
[r, g, b, c, m, k, w]  
or  
(3, ) ndarray
```

• **final_marker_edge_colour** (*See Below*, optional) – The edge colour of the markers of the final fitting shape. Example options

```python
[r, g, b, c, m, k, w]  
or  
(3, ) ndarray
```

• **final_line_colour** (*See Below*, optional) – The line colour of the final fitting shape. Example options

```python
[r, g, b, c, m, k, w]  
or  
(3, ) ndarray
```
• **initial_marker_face_colour** *(See Below, optional) – The face (filling) colour of the markers of the initial shape. Example options*

```python
[r, g, b, c, m, k, w]
or
(3, ) ndarray
```

• **initial_marker_edge_colour** *(See Below, optional) – The edge colour of the markers of the initial shape. Example options*

```python
[r, g, b, c, m, k, w]
or
(3, ) ndarray
```

• **initial_line_colour** *(See Below, optional) – The line colour of the initial shape. Example options*

```python
[r, g, b, c, m, k, w]
or
(3, ) ndarray
```

• **gt_marker_face_colour** *(See Below, optional) – The face (filling) colour of the markers of the ground truth shape. Example options*

```python
[r, g, b, c, m, k, w]
or
(3, ) ndarray
```

• **gt_marker_edge_colour** *(See Below, optional) – The edge colour of the markers of the ground truth shape. Example options*

```python
[r, g, b, c, m, k, w]
or
(3, ) ndarray
```

• **gt_line_colour** *(See Below, optional) – The line colour of the ground truth shape. Example options*

```python
[r, g, b, c, m, k, w]
or
(3, ) ndarray
```

• **render_lines** *(bool or list of bool, optional) – If True, the lines will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (final, initial, groundtruth) order.*

• **line_style** *(str or list of str, optional) – The style of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (final, initial, groundtruth) order. Example options:*

```python
['-', '--', '-.', ':']
```

• **line_width** *(float or list of float, optional) – The width of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (final, initial, groundtruth) order.*

• **render_markers** *(bool or list of bool, optional) – If True, the markers will be rendered. You can either provide a single value that will be used for all shapes or a list with a
different value per shape in \((\text{final}, \text{initial}, \text{groundtruth})\) order.

- **marker_style** *(str or list of str, optional)* – The style of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per shape in \((\text{final}, \text{initial}, \text{groundtruth})\) order. Example options:

  \[
  \{.,,,\circ,v,>,<,\times,D,d,s,p,*;h,H,1,2,3,4,8\}
  \]

- **marker_size** *(int or list of int, optional)* – The size of the markers in points. You can either provide a single value that will be used for all shapes or a list with a different value per shape in \((\text{final}, \text{initial}, \text{groundtruth})\) order.

- **marker_edge_width** *(float or list of float, optional)* – The width of the markers’ edge. You can either provide a single value that will be used for all shapes or a list with a different value per shape in \((\text{final}, \text{initial}, \text{groundtruth})\) order.

- **render_numbering** *(bool, optional)* – If True, the landmarks will be numbered.

- **numbers_horizontal_align** *(center, right, left, optional)* – The horizontal alignment of the numbers’ texts.

- **numbers_vertical_align** *(center, top, bottom, baseline, optional)* – The vertical alignment of the numbers’ texts.

- **numbers_font_name** *(See Below, optional)* – The font of the numbers. Example options

  \[
  \{\text{serif, sans-serif, cursive, fantasy, monospace}\}
  \]

- **numbers_font_size** *(int, optional)* – The font size of the numbers.

- **numbers_font_style** *(normal, italic, oblique, optional)* – The font style of the numbers.

- **numbers_font_weight** *(See Below, optional)* – The font weight of the numbers. Example options

  \[
  \{\text{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}\}
  \]

- **numbers_font_colour** *(See Below, optional)* – The font colour of the numbers. Example options

  \[
  \{r, g, b, c, m, k, w\}
  \text{ or }
  (3, ) \text{ ndarray}
  \]

- **render_legend** *(bool, optional)* – If True, the legend will be rendered.

- **legend_title** *(str, optional)* – The title of the legend.

- **legend_font_name** *(See below, optional)* – The font of the legend. Example options

  \[
  \{\text{serif, sans-serif, cursive, fantasy, monospace}\}
  \]

- **legend_font_style** *(normal, italic, oblique, optional)* – The font style of the legend.

- **legend_font_size** *(int, optional)* – The font size of the legend.
• **legend_font_weight** *(See Below, optional)* – The font weight of the legend.

Example options

<table>
<thead>
<tr>
<th>Font Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>ultralight</td>
</tr>
<tr>
<td>light</td>
</tr>
<tr>
<td>normal</td>
</tr>
<tr>
<td>regular</td>
</tr>
<tr>
<td>book</td>
</tr>
<tr>
<td>medium</td>
</tr>
<tr>
<td>roman</td>
</tr>
<tr>
<td>semibold</td>
</tr>
<tr>
<td>demibold</td>
</tr>
<tr>
<td>demi</td>
</tr>
<tr>
<td>bold</td>
</tr>
<tr>
<td>heavy</td>
</tr>
<tr>
<td>extra bold</td>
</tr>
<tr>
<td>black</td>
</tr>
</tbody>
</table>

• **legend_marker_scale** *(float, optional)* – The relative size of the legend markers with respect to the original

• **legend_location** *(int, optional)* – The location of the legend. The predefined values are:

<table>
<thead>
<tr>
<th>Location</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>best</td>
<td>0</td>
</tr>
<tr>
<td>upper right</td>
<td>1</td>
</tr>
<tr>
<td>upper left</td>
<td>2</td>
</tr>
<tr>
<td>lower left</td>
<td>3</td>
</tr>
<tr>
<td>lower right</td>
<td>4</td>
</tr>
<tr>
<td>right</td>
<td>5</td>
</tr>
<tr>
<td>center left</td>
<td>6</td>
</tr>
<tr>
<td>center right</td>
<td>7</td>
</tr>
<tr>
<td>lower center</td>
<td>8</td>
</tr>
<tr>
<td>upper center</td>
<td>9</td>
</tr>
<tr>
<td>center</td>
<td>10</td>
</tr>
</tbody>
</table>

• **legend_bbox_to_anchor** *((float, float) tuple, optional)* – The bbox that the legend will be anchored.

• **legend_border_axes_pad** *(float, optional)* – The pad between the axes and legend border.

• **legend_n_columns** *(int, optional)* – The number of the legend’s columns.

• **legend_horizontal_spacing** *(float, optional)* – The spacing between the columns.

• **legend_vertical_spacing** *(float, optional)* – The vertical space between the legend entries.

• **legend_border** *(bool, optional)* – If True, a frame will be drawn around the legend.

• **legend_border_padding** *(float, optional)* – The fractional whitespace inside the legend border.

• **legend_shadow** *(bool, optional)* – If True, a shadow will be drawn behind legend.

• **legend_rounded_corners** *(bool, optional)* – If True, the frame’s corners will be rounded (fancybox).

• **render_axes** *(bool, optional)* – If True, the axes will be rendered.

• **axes_font_name** *(See Below, optional)* – The font of the axes. Example options

<table>
<thead>
<tr>
<th>Font Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>serif</td>
</tr>
<tr>
<td>sans-serif</td>
</tr>
<tr>
<td>cursive</td>
</tr>
<tr>
<td>fantasy</td>
</tr>
<tr>
<td>monospace</td>
</tr>
</tbody>
</table>

• **axes_font_size** *(int, optional)* – The font size of the axes.

• **axes_font_style** *(normal, italic, oblique), optional)* – The font style of the axes.
• **axes_font_weight** *(See Below, optional)* – The font weight of the axes. Example options

```python
(uextralight, light, normal, regular, book, medium, roman,
semibold, demibold, demi, bold, heavy, extra bold, black)
```

• **axes_x_limits** *(float or (float, float) or None, optional)* – The limits of the x axis. If `float`, then it sets padding on the right and left of the Image as a percentage of the Image’s width. If `tuple` or `list`, then it defines the axis limits. If `None`, then the limits are set automatically.

• **axes_y_limits** *(float, float) tuple or None, optional)* – The limits of the y axis. If `float`, then it sets padding on the top and bottom of the Image as a percentage of the Image’s height. If `tuple` or `list`, then it defines the axis limits. If `None`, then the limits are set automatically.

• **axes_x_ticks** *(list or tuple or None, optional)* – The ticks of the x axis.

• **axes_y_ticks** *(list or tuple or None, optional)* – The ticks of the y axis.

• **figure_size** *(float, float) tuple or None optional)* – The size of the figure in inches.

`view_iterations` *(renderer)* – The renderer object.

Visualize the iterations of the fitting process.

**Parameters**

• **figure_id** *(object, optional)* – The id of the figure to be used.

• **new_figure** *(bool, optional)* – If `True`, a new figure is created.

• **iters** *(int or list of int or None, optional)* – The iterations to be visualized. If `None`, then all the iterations are rendered.
### MenpoFit Documentation, Release 0.6.1

#### 2.1. Deformable Models

<table>
<thead>
<tr>
<th>No.</th>
<th>Visualised shape</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>self.initial_shape</td>
<td>Initial shape</td>
</tr>
<tr>
<td>1</td>
<td>self.reconstructed_initial_shape</td>
<td>Reconstructed initial</td>
</tr>
<tr>
<td>2</td>
<td>self.shapes[2]</td>
<td>Iteration 1</td>
</tr>
<tr>
<td>i</td>
<td>self.shapes[i]</td>
<td>Iteration i-1</td>
</tr>
<tr>
<td>n_iters+1</td>
<td>self.final_shape</td>
<td>Final shape</td>
</tr>
</tbody>
</table>

- **render_image** (*bool*, optional) – If True and the image exists, then it gets rendered.
- **subplots_enabled** (*bool*, optional) – If True, then the requested final, initial and ground truth shapes get rendered on separate subplots.
- **channels** (*int* or *list* of *int* or *all* or *None*) – If *int* or *list* of *int*, the specified channel(s) will be rendered. If *all*, all the channels will be rendered in subplots. If *None* and the image is RGB, it will be rendered in RGB mode. If *None* and the image is not RGB, it is equivalent to *all*.
- **interpolation** (*str* (See Below), optional) – The interpolation used to render the image. For example, if *bilinear*, the image will be smooth and if *nearest*, the image will be pixelated. Example options

```python
{none, nearest, bilinear, bicubic, spline16, spline36, hanning, hamming, hermite, quadric, catrom, gaussian, bessel, mitchell, sinc, lanczos}
```

- **cmap_name** (*str*, optional) – If *None*, single channel and three channel images default to greyscale and rgb colormaps respectively.
- **alpha** (*float*, optional) – The alpha blending value, between 0 (transparent) and 1 (opaque).
- **masked** (*bool*, optional) – If True, then the image is rendered as masked.
- **render_lines** (*bool* or *list* of *bool*, optional) – If True, the lines will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.
- **line_style** (*str* or *list* of *str* (See below), optional) – The style of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options:

```python
{-,-,--,--,-,} 

or

{3, } ndarray
```

- **line_width** (*float* or *list* of *float*, optional) – The width of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.
- **line_colour** (*colour* or *list* of *colour* (See Below), optional) – The colour of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```python
{r, g, b, c, m, k, w}
```

- **render_markers** (*bool* or *list* of *bool*, optional) – If True, the markers will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.
• **marker_style** (*str or `list` of *str*) (See below), optional) – The style of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```python
{., ., o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
```

• **marker_size** (*int or list of *int*), optional) – The size of the markers in points. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.

• **marker_edge_colour** (*colour or list of *colour*) (See Below), optional) – The edge colour of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```python
[r, g, b, c, m, k, w]

or

(3, ) ndarray
```

• **marker_face_colour** (*colour or list of *colour*) (See Below), optional) – The face (filling) colour of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```python
[r, g, b, c, m, k, w]

or

(3, ) ndarray
```

• **marker_edge_width** (*float or list of *float*), optional) – The width of the markers’ edge. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.

• **render_numbering** (*bool*, optional) – If True, the landmarks will be numbered.

• **numbers_horizontal_align** (*str*, optional) – The horizontal alignment of the numbers’ texts. Example options

```python
{center, right, left}
```

• **numbers_vertical_align** (*str*, optional) – The vertical alignment of the numbers’ texts. Example options

```python
{center, top, bottom, baseline}
```

• **numbers_font_name** (*str* (See below), optional) – The font of the numbers. Example options

```python
{serif, sans-serif, cursive, fantasy, monospace}
```

• **numbers_font_size** (*int*, optional) – The font size of the numbers.

• **numbers_font_style** (*{normal, italic, oblique}*), optional) – The font style of the numbers.

• **numbers_font_weight** (*str*, optional) – The font weight of the numbers. Example options

```python
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```
- **numbers_font_colour** *(See Below, optional)* – The font colour of the numbers. Example options

\[
\{r, g, b, c, m, k, w\}
\]

or

\[
(3, ) \text{ ndarray}
\]

- **render_legend** *(bool, optional)* – If True, the legend will be rendered.

- **legend_title** *(str, optional)* – The title of the legend.

- **legend_font_name** *(See below, optional)* – The font of the legend. Example options

\[
\{\text{serif, sans-serif, cursive, fantasy, monospace}\}
\]

- **legend_font_style** *(str (See below), optional)* – The font style of the legend. Example options

\[
\{\text{normal, italic, oblique}\}
\]

- **legend_font_size** *(int, optional)* – The font size of the legend.

- **legend_font_weight** *(str (See below), optional)* – The font weight of the legend. Example options

\[
\{\text{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}\}
\]

- **legend_marker_scale** *(float, optional)* – The relative size of the legend markers with respect to the original

- **legend_location** *(int, optional)* – The location of the legend. The predefined values are:

```
<table>
<thead>
<tr>
<th>Location</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>'best'</td>
<td>0</td>
</tr>
<tr>
<td>'upper right'</td>
<td>1</td>
</tr>
<tr>
<td>'upper left'</td>
<td>2</td>
</tr>
<tr>
<td>'lower left'</td>
<td>3</td>
</tr>
<tr>
<td>'lower right'</td>
<td>4</td>
</tr>
<tr>
<td>'right'</td>
<td>5</td>
</tr>
<tr>
<td>'center left'</td>
<td>6</td>
</tr>
<tr>
<td>'center right'</td>
<td>7</td>
</tr>
<tr>
<td>'lower center'</td>
<td>8</td>
</tr>
<tr>
<td>'upper center'</td>
<td>9</td>
</tr>
<tr>
<td>'center'</td>
<td>10</td>
</tr>
</tbody>
</table>
```

- **legend_bbox_to_anchor** *(((float, float) tuple, optional)* – The bbox that the legend will be anchored.

- **legend_border_axes_pad** *(float, optional)* – The pad between the axes and legend border.

- **legend_n_columns** *(int, optional)* – The number of the legend’s columns.

- **legend_horizontal_spacing** *(float, optional)* – The spacing between the columns.

- **legend_vertical_spacing** *(float, optional)* – The vertical space between the legend entries.
- **legend_border** (*bool, optional*) – If True, a frame will be drawn around the legend.
- **legend_border_padding** (*float, optional*) – The fractional whitespace inside the legend border.
- **legend_shadow** (*bool, optional*) – If True, a shadow will be drawn behind legend.
- **legend_rounded_corners** (*bool, optional*) – If True, the frame’s corners will be rounded (fancybox).
- **render_axes** (*bool, optional*) – If True, the axes will be rendered.
- **axes_font_name** (*str (See below), optional*) – The font of the axes. Example options
  
  ```
  {serif, sans-serif, cursive, fantasy, monospace}
  ```

- **axes_font_size** (*int, optional*) – The font size of the axes.
- **axes_font_style** (*{normal, italic, oblique}, optional*) – The font style of the axes.
- **axes_font_weight** (*str (See below), optional*) – The font weight of the axes. Example options
  
  ```
  {ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
  ```

- **axes_x_limits** (*float or (float, float) or None, optional*) – The limits of the x axis. If float, then it sets padding on the right and left of the Image as a percentage of the Image’s width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.

- **axes_y_limits** (*((float, float) tuple or None, optional*) – The limits of the y axis. If float, then it sets padding on the top and bottom of the Image as a percentage of the Image’s height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.

- **axes_x_ticks** (*list or tuple or None, optional*) – The ticks of the x axis.

- **axes_y_ticks** (*list or tuple or None, optional*) – The ticks of the y axis.

- **figure_size** (*((float, float) tuple or None optional*) – The size of the figure in inches.

  Returns **renderer** (*class*) – The renderer object.

  **view_widget** (*figure_size=(7, 7))*)

  Visualizes the result object using an interactive widget.

  Parameters **figure_size** (*((int, int), optional*) – The initial size of the rendered figure.

  **property appearance_parameters**

  Returns the **list** of appearance parameters obtained at each iteration of the fitting process. The list includes the parameters of the initial_shape (if it exists) and final_shape.

  Type **list of (n_params,) ndarray**

  **property costs**

  Returns a **list** with the cost per iteration. It returns None if the costs are not computed.

  Type **list of float or None**

  **property final_shape**

  Returns the final shape of the fitting process.

  Type **menpo.shape.PointCloud**
```python

property gt_shape
Returns the ground truth shape associated with the image. In case there is not an attached ground truth shape, then None is returned.

Type menpo.shape.PointCloud or None

property image
Returns the image that the fitting was applied on, if it was provided. Otherwise, it returns None.

Type menpo.shape.Image or subclass or None

property initial_shape
Returns the initial shape that was provided to the fitting method to initialise the fitting process. In case the initial shape does not exist, then None is returned.

Type menpo.shape.PointCloud or None

property is_iterative
Flag whether the object is an iterative fitting result.

Type bool

property n_iters
Returns the total number of iterations of the fitting process.

Type int

property reconstructed_initial_shape
Returns the initial shape’s reconstruction with the shape model that was used to initialise the iterative optimisation process.

Type menpo.shape.PointCloud

property shape_parameters
Returns the list of shape parameters obtained at each iteration of the fitting process. The list includes the parameters of the reconstructed_initial_shape and final_shape.

Type list of (n_params,) ndarray

property shapes
Returns the list of shapes obtained at each iteration of the fitting process. The list includes the initial_shape (if it exists), reconstructed_initial_shape and final_shape.

Type list of menpo.shape.PointCloud
```

### 2.1.6 menpofit.dlib

**Ensemble of Regression Trees (provided by DLib)**

Method that employs gradient boosting for learning an ensemble of regression trees to estimate the landmark positions directly from a sparse subset of pixel intensities.
DlibERT

```python
class menpofit.dlib.DlibERT(images, group=None, bounding_box_group_glob=None,
reference_shape=None, diagonal=None, scales=(0.5, 1.0),
n_perturbations=30, n_dlib_perturbations=1,
perturb_from_gt_bounding_box=<function noisy_shape_from_bounding_box>,
n_iterations=10, feature_padding=0, n_pixel_pairs=400, distance_prior_weighting=0.1,
regularisation_weight=0.1, n_split_tests=20, n_trees=500,
n_tree_levels=5, verbose=False)
```

Bases: `MultiScaleNonParametricFitter`

Class for training a multi-scale Ensemble of Regression Trees model. This class uses the implementation provided by the official DLib package (http://dlib.net/) and makes it multi-scale.

Parameters

- **images** *(list of menpo.image.Image)* – The list of training images.
- **group** *(str or None, optional)* – The landmark group that corresponds to the ground truth shape of each image. If `None` and the images only have a single landmark group, then that is the one that will be used. Note that all the training images need to have the specified landmark group.
- **bounding_box_group_glob** *(glob or None, optional)* – Glob that defines the bounding boxes to be used for training. If `None`, then the bounding boxes of the ground truth shapes are used.
- **reference_shape** *(menpo.shapePointCloud or None, optional)* – The reference shape that will be used for normalising the size of the training images. The normalization is performed by rescaling all the training images so that the scale of their ground truth shapes matches the scale of the reference shape. Note that the reference shape is rescaled with respect to the `diagonal` before performing the normalisation. If `None`, then the mean shape will be used.
- **diagonal** *(int or None, optional)* – This parameter is used to rescale the reference shape so that the diagonal of its bounding box matches the provided value. In other words, this parameter controls the size of the model at the highest scale. If `None`, then the reference shape does not get rescaled.
- **scales** *(float or tuple of float, optional)* – The scale value of each scale. They must be provided in ascending order, i.e. from lowest to highest scale. If `float`, then a single scale is assumed.
- **n_perturbations** *(int or None, optional)* – The number of perturbations to be generated from each of the bounding boxes using `perturb_from_gt_bounding_box`. Note that the total number of perturbations is `n_perturbations * n_dlib_perturbations`.
- **perturb_from_gt_bounding_box** *(function, optional)* – The function that will be used to generate the perturbations.
- **n_dlib_perturbations** *(int or None or list of those, optional)* – The number of perturbations to be generated from the part of DLib. DLib calls this “oversampling amount”. If `list`, it must specify a value per scale. Note that the total number of perturbations is `n_perturbations * n_dlib_perturbations`.
- **n_iterations** *(int or list of int, optional)* – The number of iterations (cascades) of each level. If `list`, it must specify a value per scale. If `int`, then it defines the total number of iterations (cascades) over all scales.
• **feature_padding** *(float or list of float, optional)* – When we randomly sample the pixels for the feature pool we do so in a box fit around the provided training landmarks. By default, this box is the tightest box that contains the landmarks. However, you can expand or shrink the size of the pixel sampling region by setting a different value of padding. To explain this precisely, for a padding of 0 we say that the pixels are sampled from a box of size 1x1. The padding value is added to each side of the box. So a padding of 0.5 would cause the algorithm to sample pixels from a box that was 2x2, effectively multiplying the area pixels are sampled from by 4. Similarly, setting the padding to -0.2 would cause it to sample from a box 0.6x0.6 in size. If list, it must specify a value per scale.

• **n_pixel_pairs** *(int or list of int, optional)* – P parameter from [1]. At each level of the cascade we randomly sample pixels from the image. These pixels are used to generate features for the random trees. So in general larger settings of this parameter give better accuracy but make the algorithm run slower. If list, it must specify a value per scale.

• **distance_prior_weighting** *(float or list of float, optional)* – To decide how to split nodes in the regression trees the algorithm looks at pairs of pixels in the image. These pixel pairs are sampled randomly but with a preference for selecting pixels that are near each other. This parameter controls this “nearness” preference. In particular, smaller values will make the algorithm prefer to select pixels close together and larger values will make it care less about picking nearby pixel pairs. Note that this is the inverse of how it is defined in [1]. For this object, you should think of distance_prior_weighting as “the fraction of the bounding box will we traverse to find a neighboring pixel”. Nominally, this is normalized between 0 and 1. So reasonable settings are values in the range (0, 1). If list, it must specify a value per scale.

• **regularisation_weight** *(float or list of float, optional)* – Boosting regularization parameter - nu from [1]. Larger values may cause overfitting but improve performance on training data. If list, it must specify a value per scale.

• **n_split_tests** *(int or list of int, optional)* – When generating the random trees we randomly sample n_split_tests possible split features at each node and pick the one that gives the best split. Larger values of this parameter will usually give more accurate outputs but take longer to train. It is equivalent of S from [1]. If list, it must specify a value per scale.

• **n_trees** *(int or list of int, optional)* – Number of trees created for each cascade. The total number of trees in the learned model is equal n_trees * n_tree_levels. Equivalent to K from [1]. If list, it must specify a value per scale.

• **n_tree_levels** *(int or list of int, optional)* – The number of levels in the tree (depth of tree). In particular, there are pow(2, n_tree_levels) leaves in each tree. Equivalent to F from [1]. If list, it must specify a value per scale.

• **verbose** *(bool, optional)* – If True, then the progress of building ERT will be printed.

References

fit_from_bb *(image, bounding_box, gt_shape=None)*

Fits the model to an image given an initial bounding box.

Parameters

• **image** *(menpo.image.Image or subclass)* – The image to be fitted.

• **bounding_box** *(menpo.shape.PointDirectedGraph)* – The initial bounding box from which the fitting procedure will start.
• **gt_shape** (*menpo.shape.PointCloud*, optional) – The ground truth shape associated to the image.

Returns **fitting_result** (*MultiScaleNonParametricIterativeResult*) – The result of the fitting procedure.

**fit_from_shape** (*image, initial_shape, gt_shape=None*)

Fits the model to an image. Note that it is not possible to initialise the fitting process from a shape. Thus, this method raises a warning and calls `fit_from_bb` with the bounding box of the provided `initial_shape`.

Parameters

• **image** (*menpo.image.Image* or subclass) – The image to be fitted.

• **initial_shape** (*menpo.shape.PointCloud*) – The initial shape estimate from which the fitting procedure will start. Note that the shape won’t actually be used, only its bounding box.

• **gt_shape** (*menpo.shape.PointCloud*, optional) – The ground truth shape associated to the image.

Returns **fitting_result** (*MultiScaleNonParametricIterativeResult*) – The result of the fitting procedure.

**property holistic_features**

The features that are extracted from the input image at each scale in ascending order, i.e. from lowest to highest scale.

Type *list* of *closure*

**property n_scales**

Returns the number of scales.

Type *int*

**property reference_shape**

The reference shape that is used to normalise the size of an input image so that the scale of its initial fitting shape matches the scale of this reference shape.

Type *menpo.shape.PointCloud*

**property scales**

The scale value of each scale in ascending order, i.e. from lowest to highest scale.

Type *list* of *int* or *float*

### DlibWrapper

**class menpofit.dlib.DlibWrapper**(model)

Bases: *object*

Wrapper class for fitting a pre-trained ERT model. Pre-trained models are provided by the official DLib package ([http://dlib.net/](http://dlib.net/)).

Parameters **model** (*Path* or *str*) – Path to the pre-trained model.

**fit_from_bb** (*image, bounding_box, gt_shape=None*)

Fits the model to an image given an initial bounding box.

Parameters

• **image** (*menpo.image.Image* or subclass) – The image to be fitted.
• **bounding_box** (*menpo.shape.PointDirectedGraph*) – The initial bounding box.

• **gt_shape** (*menpo.shape.PointCloud*) – The ground truth shape associated to the image.

Returns **fitting_result** (*Result*) – The result of the fitting procedure.

*fit_from_shape* (*image*, *initial_shape*, *gt_shape=None*)
Fits the model to an image. Note that it is not possible to initialise the fitting process from a shape. Thus, this method raises a warning and calls *fit_from_bb* with the bounding box of the provided *initial_shape*.

Parameters

• **image** (*menpo.image.Image* or subclass) – The image to be fitted.

• **initial_shape** (*menpo.shape.PointCloud*) – The initial shape estimate from which the fitting procedure will start. Note that the shape won’t actually be used, only its bounding box.

• **gt_shape** (*menpo.shape.PointCloud*) – The ground truth shape associated to the image.

Returns **fitting_result** (*Result*) – The result of the fitting procedure.

### 2.1.7 menpofit.lk

**Lucas-Kanade Alignment**

LK performs alignment (or optical flow estimation) between a template image and a test image with respect to an affine transformation.

**LucasKanadeFitter**

```python
class menpofit.lk.LucasKanadeFitter(template, group=None, holistic_features=<function no_op>, diagonal=None, transform=<class 'menpofit.transform.homogeneous.DifferentiableAlignmentAffine'>, scales=(0.5, 1.0), algorithm_cls=<class 'menpofit.lk.algorithm.InverseCompositional'>, residual_cls=<class 'menpofit.lk.residual.SSD'>)
```

**Bases:** *MultiScaleNonParametricFitter*

Class for defining a multi-scale Lucas-Kanade fitter that performs alignment with respect to a homogeneous transform. Please see the references for a basic list of relevant papers.

Parameters

• **template** (*menpo.image.Image*) – The template image.

• **group** (*str* or *None*, optional) – The landmark group of the *template* that will be used as reference shape. If *None* and the *template* only has a single landmark group, then that is the one that will be used.

• **holistic_features** (*closure* or *list* of *closure*, optional) – The features that will be extracted from the training images. Note that the features are extracted before warping the images to the reference shape. If *list*, then it must define a feature function per scale. Please refer to *menpo.feature* for a list of potential features.

• **diagonal** (*int* or *None*, optional) – This parameter is used to rescale the reference shape (specified by *group*) so that the diagonal of its bounding box matches the provided value. In other words, this parameter controls the size of the model at the highest scale. If *None*, then the reference shape does not get rescaled.
• **scales** *(tuple of float, optional)* – The scale value of each scale. They must provided in ascending order, i.e. from lowest to highest scale.

• **transform** *(subclass of DP and DX, optional)* – A differential homogeneous transform object, e.g. DifferentiableAlignmentAffine.

• **algorithm_cls** *(class, optional)* – The Lukas-Kanade optimisation algorithm that will get applied. The possible algorithms in menpofit.lk.algorithm are:

<table>
<thead>
<tr>
<th>Class</th>
<th>Warp Direction</th>
<th>Warp Update</th>
</tr>
</thead>
<tbody>
<tr>
<td>ForwardAdditive</td>
<td>Forward</td>
<td>Additive</td>
</tr>
<tr>
<td>ForwardCompositional</td>
<td>Forward</td>
<td>Compositional</td>
</tr>
<tr>
<td>InverseCompositional</td>
<td>Inverse</td>
<td></td>
</tr>
</tbody>
</table>

• **residual_cls** *(class subclass, optional)* – The residual that will get applied. All possible residuals are:

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSD</td>
<td>Sum of Squared Differences</td>
</tr>
<tr>
<td>FourierSSD</td>
<td>Sum of Squared Differences on Fourier domain</td>
</tr>
<tr>
<td>ECC</td>
<td>Enhanced Correlation Coefficient</td>
</tr>
<tr>
<td>GradientImages</td>
<td>Image Gradient</td>
</tr>
<tr>
<td>GradientCorrelation</td>
<td>Gradient Correlation</td>
</tr>
</tbody>
</table>

References

**fit_from_bb** *(image, bounding_box, max_iters=20, gt_shape=None, return_costs=False, **kwargs)*

Fits the multi-scale fitter to an image given an initial bounding box.

**Parameters**

• **image** *(menpo.image.Image or subclass)* – The image to be fitted.

• **bounding_box** *(menpo.shape.PointDirectedGraph)* – The initial bounding box from which the fitting procedure will start. Note that the bounding box is used in order to align the model’s reference shape.

• **max_iters** *(int or list of int, optional)* – The maximum number of iterations. If int, then it specifies the maximum number of iterations over all scales. If list of int, then specifies the maximum number of iterations per scale.

• **gt_shape** *(menpo.shape.PointCloud, optional)* – The ground truth shape associated to the image.

• **return_costs** *(bool, optional)* – If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.

• **kwargs** *(dict, optional)* – Additional keyword arguments that can be passed to specific implementations.

**Returns** fitting_result *(MultiScaleNonParametricIterativeResult or subclass)* – The multi-scale fitting result containing the result of the fitting procedure.
fit_from_shape(image, initial_shape, max_iters=20, gt_shape=None, return_costs=False, **kwargs)

Fits the multi-scale fitter to an image given an initial shape.

Parameters

- **image** (menpo.image.Image or subclass) – The image to be fitted.
- **initial_shape** (menpo.shape.PointCloud) – The initial shape estimate from which the fitting procedure will start.
- **max_iters** (int or list of int, optional) – The maximum number of iterations. If int, then it specifies the maximum number of iterations over all scales. If list of int, then specifies the maximum number of iterations per scale.
- **gt_shape** (menpo.shape.PointCloud, optional) – The ground truth shape associated to the image.
- **return_costs** (bool, optional) – If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.
- **kwargs** (dict, optional) – Additional keyword arguments that can be passed to specific implementations.

Returns **fitting_result** (MultiScaleNonParametricIterativeResult or subclass) – The multi-scale fitting result containing the result of the fitting procedure.

warped_images(image, shapes)

Given an input test image and a list of shapes, it warps the image into the shapes. This is useful for generating the warped images of a fitting procedure stored within a LucasKanadeResult.

Parameters

- **image** (menpo.image.Image or subclass) – The input image to be warped.
- **shapes** (list of menpo.shape.PointCloud) – The list of shapes in which the image will be warped. The shapes are obtained during the iterations of a fitting procedure.

Returns **warped_images** (list of menpo.image.MaskedImage or ndarray) – The warped images.

property holistic_features

The features that are extracted from the input image at each scale in ascending order, i.e. from lowest to highest scale.

Type list of closure

property n_scales

Returns the number of scales.

Type int

property reference_shape

The reference shape that is used to normalise the size of an input image so that the scale of its initial fitting shape matches the scale of this reference shape.

Type menpo.shape.PointCloud

property scales

The scale value of each scale in ascending order, i.e. from lowest to highest scale.

Type list of int or float
Optimisation Algorithms

Forward Additive

class menpofit.lk.ForwardAdditive(template, transform, residual, eps=1e-10)
Bases: LucasKanade

Forward Additive (FA) Lucas-Kanade algorithm.

run(image, initial_shape, gt_shape=None, max_iters=20, return_costs=False)
Execute the optimization algorithm.

Parameters

- **image** (menpo.image.Image) – The input test image.
- **initial_shape** (menpo.shape.PointCloud) – The initial shape from which the optimization will start.
- **gt_shape** (menpo.shape.PointCloud or None, optional) – The ground truth shape of the image. It is only needed in order to get passed in the optimization result object, which has the ability to compute the fitting error.
- **max_iters** (int, optional) – The maximum number of iterations. Note that the algorithm may converge, and thus stop, earlier.
- **return_costs** (bool, optional) – If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.

Returns fitting_result (LucasKanadeAlgorithmResult) – The parametric iterative fitting result.

warped_images(image, shapes)
Given an input test image and a list of shapes, it warps the image into the shapes. This is useful for generating the warped images of a fitting procedure stored within a LucasKanadeResult.

Parameters

- **image** (menpo.image.Image or subclass) – The input image to be warped.
- **shapes** (list of menpo.shape.PointCloud) – The list of shapes in which the image will be warped. The shapes are obtained during the iterations of a fitting procedure.

Returns warped_images (list of menpo.image.MaskedImage or ndarray) – The warped images.

Forward Compositional

class menpofit.lk.ForwardCompositional(template, transform, residual, eps=1e-10)
Bases: LucasKanade

Forward Compositional (FC) Lucas-Kanade algorithm

Parameters

- **template** (menpo.image.Image or subclass) – The image template.
- **transform** (subclass of DP and DX, optional) – A differential affine transform object, e.g. DifferentiableAlignmentAffine.
• **residual** *(class subclass, optional)* – The residual that will get applied. All possible residuals are:

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSD</td>
<td>Sum of Squared Differences</td>
</tr>
<tr>
<td>FourierSSD</td>
<td>Sum of Squared Differences on Fourier domain</td>
</tr>
<tr>
<td>ECC</td>
<td>Enhanced Correlation Coefficient</td>
</tr>
<tr>
<td>GradientImages</td>
<td>Image Gradient</td>
</tr>
<tr>
<td>GradientCorrelation</td>
<td>Gradient Correlation</td>
</tr>
</tbody>
</table>

• **eps** *(float, optional)* – Value for checking the convergence of the optimization.

**run**(image, initial_shape, gt_shape=None, max_iters=20, return_costs=False)

Execute the optimization algorithm.

**Parameters**

- **image** *(menpo.image.Image)* – The input test image.
- **initial_shape** *(menpo.shape.PointCloud)* – The initial shape from which the optimization will start.
- **gt_shape** *(menpo.shape.PointCloud or None, optional)* – The ground truth shape of the image. It is only needed in order to get passed in the optimization result object, which has the ability to compute the fitting error.
- **max_iters** *(int, optional)* – The maximum number of iterations. Note that the algorithm may converge, and thus stop, earlier.
- **return_costs** *(bool, optional)* – If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.

Returns fitting_result *(LucasKanadeAlgorithmResult)* – The parametric iterative fitting result.

**warped_images**(image, shapes)

Given an input test image and a list of shapes, it warps the image into the shapes. This is useful for generating the warped images of a fitting procedure stored within a LucasKanadeResult.

**Parameters**

- **image** *(menpo.image.Image or subclass)* – The input image to be warped.
- **shapes** *(list of menpo.shape.PointCloud)* – The list of shapes in which the image will be warped. The shapes are obtained during the iterations of a fitting procedure.

Returns warped_images *(list of menpo.image.MaskedImage or ndarray)* – The warped images.

**InverseCompositional**

```python
class menpofit.lk.InverseCompositional(template, transform, residual, eps=1e-10)
```

**Bases**: LucasKanade

Inverse Compositional (IC) Lucas-Kanade algorithm

**Parameters**

2.1. Deformable Models
• **template** (*menpo.image.Image or subclass*) – The image template.

• **transform** (*subclass of DP and DX, optional*) – A differential affine transform object, e.g. DifferentiableAlignmentAffine.

• **residual** (*class subclass, optional*) – The residual that will get applied. All possible residuals are:

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSD</td>
<td>Sum of Squared Differences</td>
</tr>
<tr>
<td>FourierSSD</td>
<td>Sum of Squared Differences on Fourier domain</td>
</tr>
<tr>
<td>ECC</td>
<td>Enhanced Correlation Coefficient</td>
</tr>
<tr>
<td>GradientImages</td>
<td>Image Gradient</td>
</tr>
<tr>
<td>GradientCorrelation</td>
<td>Gradient Correlation</td>
</tr>
</tbody>
</table>

• **eps** (*float, optional*) – Value for checking the convergence of the optimization.

**run** (*image, initial_shape, gt_shape=None, max_iters=20, return_costs=False*)

Execute the optimization algorithm.

**Parameters**

• **image** (*menpo.image.Image*) – The input test image.

• **initial_shape** (*menpo.shape.PointCloud*) – The initial shape from which the optimization will start.

• **gt_shape** (*menpo.shape.PointCloud or None, optional*) – The ground truth shape of the image. It is only needed in order to get passed in the optimization result object, which has the ability to compute the fitting error.

• **max_iters** (*int, optional*) – The maximum number of iterations. Note that the algorithm may converge, and thus stop, earlier.

• **return_costs** (*bool, optional*) – If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.

**Returns** fitting_result (*LucasKanadeAlgorithmResult*) – The parametric iterative fitting result.

**warped_images** (*image, shapes*)

Given an input test image and a list of shapes, it warps the image into the shapes. This is useful for generating the warped images of a fitting procedure stored within a *LucasKanadeResult*.

**Parameters**

• **image** (*menpo.image.Image or subclass*) – The input image to be warped.

• **shapes** (*list of menpo.shape.PointCloud*) – The list of shapes in which the image will be warped. The shapes are obtained during the iterations of a fitting procedure.

**Returns** warped_images (*list of menpo.image.MaskedImage or ndarray*) – The warped images.

**Residuals**
SSD

class menpofit.lk.SSD(kernel=None)
Bases: Residual

Class for Sum of Squared Differences residual.

References

cost_closure()
   Method to compute the optimization cost.

   Returns cost (float) – The cost value.

classmethod gradient(image, forward=None)
   Calculates the gradients of the given method.

   If forward is provided, then the gradients are warped (as required in the forward additive algorithm)

   Parameters

   • image (menpo.image.Image) – The image to calculate the gradients for

   • forward (tuple or None, optional) – A tuple containing the extra weights required for the function warp (which should be passed as a function handle), i.e. ("menpo.image.Image", "menpo.transform.AlignableTransform"). If None, then the optimization algorithm is assumed to be inverse.

hessian(sdi, sdi2=None)
   Calculates the Gauss-Newton approximation to the Hessian.

   This is abstracted because some residuals expect the Hessian to be pre-processed. The Gauss-Newton approximation to the Hessian is defined as:

   \[ JJ^T \]

   Parameters

   • sdi ((N, n_params) ndarray) – The steepest descent images.

   • sdi2 ((N, n_params) ndarray or None, optional) – The steepest descent images.

   Returns H ((n_params, n_params) ndarray) – The approximation to the Hessian

steepest_descent_images(image, dW_dp, forward=None)
   Calculates the standard steepest descent images.

   Within the forward additive framework this is defined as

   \[ \nabla I \frac{\partial W}{\partial p} \]

   The input image is vectorised (N-pixels) so that masked images can be handled.

   Parameters

   • image (menpo.image.Image) – The image to calculate the steepest descent images from, could be either the template or input image depending on which framework is used.

   • dW_dp (ndarray) – The Jacobian of the warp.
• **forward** (tuple or None, optional) – A tuple containing the extra weights required for the function *warp* (which should be passed as a function handle), i.e. (*menpo.image.Image*, *menpo.transform.AlignableTransform*>). If None, then the optimization algorithm is assumed to be inverse.

**Returns** VT_dW_dp ((N, n_params) ndarray) – The steepest descent images

```python
steepest_descent_update(sdi, image, template)
```
Calculates the steepest descent parameter updates.

These are defined, for the forward additive algorithm, as:

\[
\sum_x |\nabla I \frac{\partial W}{\partial p}|^T [T(x) - I(W(x;p))]
\]

**Parameters**

- **sdi** ((N, n_params) ndarray) – The steepest descent images.
- **image** (menpo.image.Image) – Either the warped image or the template (depending on the framework)
- **template** (menpo.image.Image) – Either the warped image or the template (depending on the framework)

**Returns** sd_delta_p ((n_params,) ndarray) – The steepest descent parameter updates.

---

**FourierSSD**

```python
class menpo.fit.lk.FourierSSD(kernel=None)
```
Bases: Residual

Class for Sum of Squared Differences on the Fourier domain residual.

**References**

- **cost_closure** ()
  Method to compute the optimization cost.

  **Returns** cost (float) – The cost value.

- **classmethod gradient(image, forward=None)**
  Calculates the gradients of the given method.

  If *forward* is provided, then the gradients are warped (as required in the forward additive algorithm)

  **Parameters**

  - **image** (menpo.image.Image) – The image to calculate the gradients for
  - **forward** (tuple or None, optional) – A tuple containing the extra weights required for the function *warp* (which should be passed as a function handle), i.e. (*menpo.image.Image*, *menpo.transform.AlignableTransform*>). If None, then the optimization algorithm is assumed to be inverse.

- **hessian(sdi, sdi2=None)**
  Calculates the Gauss-Newton approximation to the Hessian.
This is abstracted because some residuals expect the Hessian to be pre-processed. The Gauss-Newton approximation to the Hessian is defined as:

$$JJ^T$$

**Parameters**

- `sdi` ((N, n_params) ndarray) – The steepest descent images.
- `sdi2` ((N, n_params) ndarray or None, optional) – The steepest descent images.

**Returns**

- `H` ((n_params, n_params) ndarray) – The approximation to the Hessian

**steepest_descent_images** *(image, dW_dp, forward=None)*

Calculates the standard steepest descent images.

Within the forward additive framework this is defined as

$$\nabla I \frac{\partial W}{\partial p}$$

The input image is vectorised (N-pixels) so that masked images can be handled.

**Parameters**

- `image` (menpo.image.Image) – The image to calculate the steepest descent images from, could be either the template or input image depending on which framework is used.
- `dW_dp` (ndarray) – The Jacobian of the warp.
- `forward` (tuple or None, optional) – A tuple containing the extra weights required for the function `warp` (which should be passed as a function handle), i.e. (`menpo.image.Image`, `menpo.transform.AlignableTransform`). If None, then the optimization algorithm is assumed to be inverse.

**Returns**

- `VT_dW_dp` ((N, n_params) ndarray) – The steepest descent images

**steepest_descent_update** *(sdi, image, template)*

Calculates the steepest descent parameter updates.

These are defined, for the forward additive algorithm, as:

$$\sum_x [\nabla I \frac{\partial W}{\partial p}]^T [T(x) - I(W(x;p))]$$

**Parameters**

- `sdi` ((N, n_params) ndarray) – The steepest descent images.
- `image` (menpo.image.Image) – Either the warped image or the template (depending on the framework)
- `template` (menpo.image.Image) – Either the warped image or the template (depending on the framework)

**Returns**

- `sd_delta_p` ((n_params,) ndarray) – The steepest descent parameter updates.

**ECC**

class menpofit.lk.ECC

Bases: Residual

Class for Enhanced Correlation Coefficient residual.

### 2.1. Deformable Models
References

**cost_closure()**
Method to compute the optimization cost.

**Returns**
`cost (float) – The cost value.`

**classmethod gradient(image, forward=None)**
Calculates the gradients of the given method.

If `forward` is provided, then the gradients are warped (as required in the forward additive algorithm)

**Parameters**

- **image** (`menpo.image.Image`) – The image to calculate the gradients for
- **forward** (tuple or None, optional) – A tuple containing the extra weights required for the function `warp` (which should be passed as a function handle), i.e. (`menpo.image.Image`, `menpo.transform.AlignableTransform`). If None, then the optimization algorithm is assumed to be inverse.

**hessian(sdi, sdi2=None)**
Calculates the Gauss-Newton approximation to the Hessian.

This is abstracted because some residuals expect the Hessian to be pre-processed. The Gauss-Newton approximation to the Hessian is defined as:

\[ JJ^T \]

**Parameters**

- **sdi** ((N, n_params) ndarray) – The steepest descent images.
- **sdi2** ((N, n_params) ndarray or None, optional) – The steepest descent images.

**Returns**
`H ((n_params, n_params) ndarray) – The approximation to the Hessian`

**steepest_descent_images(image, dW_dp, forward=None)**
Calculates the standard steepest descent images.

Within the forward additive framework this is defined as

\[ \nabla I \frac{\partial W}{\partial p} \]

The input image is vectorised (N-pixels) so that masked images can be handled.

**Parameters**

- **image** (`menpo.image.Image`) – The image to calculate the steepest descent images from, could be either the template or input image depending on which framework is used.
- **dW_dp** (ndarray) – The Jacobian of the warp.
- **forward** (tuple or None, optional) – A tuple containing the extra weights required for the function `warp` (which should be passed as a function handle), i.e. (`menpo.image.Image`, `menpo.transform.AlignableTransform`). If None, then the optimization algorithm is assumed to be inverse.

**Returns**
`VT_dW_dp ((N, n_params) ndarray) – The steepest descent images`
steepest_descent_update (sdi, image, template)
Calculates the steepest descent parameter updates.

These are defined, for the forward additive algorithm, as:

\[
\sum_x \left| \nabla I \frac{\partial W}{\partial p} \right|^T [T(x) - I(W(x;p))]
\]

Parameters
- **sdi** ((N, n_params) ndarray) – The steepest descent images.
- **image** (menpo.image.Image) – Either the warped image or the template (depending on the framework)
- **template** (menpo.image.Image) – Either the warped image or the template (depending on the framework)

Returns **sd_delta_p** ((n_params,) ndarray) – The steepest descent parameter updates.

GradientImages

class menpofit.lk.GradientImages
Bases: Residual

Class for Gradient Images residual.

References

cost_closure ()
Method to compute the optimization cost.

Returns **cost** (float) – The cost value.

classmethod gradient (image, forward=None)
Calculates the gradients of the given method.

If **forward** is provided, then the gradients are warped (as required in the forward additive algorithm)

Parameters
- **image** (menpo.image.Image) – The image to calculate the gradients for
- **forward** (tuple or None, optional) – A tuple containing the extra weights required for the function **warp** (which should be passed as a function handle), i.e. (`menpo.image.Image`, `menpo.transform.AlignableTransform`). If None, then the optimization algorithm is assumed to be inverse.

hessian (sdi, sdi2=None)
Calculates the Gauss-Newton approximation to the Hessian.

This is abstracted because some residuals expect the Hessian to be pre-processed. The Gauss-Newton approximation to the Hessian is defined as:

\[
JJ^T
\]

Parameters
- **sdi** ((N, n_params) ndarray) – The steepest descent images.
- **sdi2** ((N, n_params) ndarray or None, optional) – The steepest descent images.
Returns $H((n_{\text{params}}, n_{\text{params}}) \text{ndarray})$ – The approximation to the Hessian

**steepest_descent_images** (image, $dW_{dp}$, forward=None)
Calculates the standard steepest descent images.

Within the forward additive framework this is defined as

$$\nabla I \frac{\partial W}{\partial p}$$

The input image is vectorised ($N$-pixels) so that masked images can be handled.

**Parameters**
- image (menpo.image.Image) – The image to calculate the steepest descent images from, could be either the template or input image depending on which framework is used.
- dW_dp (ndarray) – The Jacobian of the warp.
- forward (tuple or None, optional) – A tuple containing the extra weights required for the function `warp` (which should be passed as a function handle), i.e. (`menpo.image.Image`, `menpo.transform.AlignableTransform`). If None, then the optimization algorithm is assumed to be inverse.

Returns $VT_dW_dp((N, n_{\text{params}}) \text{ndarray})$ – The steepest descent images

**steepest_descent_update** (sdi, image, template)
Calculates the steepest descent parameter updates.

These are defined, for the forward additive algorithm, as:

$$\sum_x |\nabla I \frac{\partial W}{\partial p}|^T [T(x) - I(W(x;p))]$$

**Parameters**
- sdi ((N, n_{\text{params}}) \text{ndarray}) – The steepest descent images.
- image (menpo.image.Image) – Either the warped image or the template (depending on the framework)
- template (menpo.image.Image) – Either the warped image or the template (depending on the framework)

Returns sd_delta_p ((n_{\text{params}},) \text{ndarray}) – The steepest descent parameter updates.

**GradientCorrelation**

```python
class menpofit.lk.GradientCorrelation
    Bases: Residual

    Class for Gradient Correlation residual.
```

**References**

```python
cost_closure()
Method to compute the optimization cost.

    Returns cost (float) – The cost value.
```
classmethod gradient(image, forward=None)
Calculates the gradients of the given method.

If forward is provided, then the gradients are warped (as required in the forward additive algorithm)

Parameters
• image (menpo.image.Image) – The image to calculate the gradients for
• forward (tuple or None, optional) – A tuple containing the extra weights required for the function warp (which should be passed as a function handle), i.e. ("menpo.image.Image", "menpo.transform.AlignableTransform>”). If None, then the optimization algorithm is assumed to be inverse.

hessian(sdi, sdi2=None)
Calculates the Gauss-Newton approximation to the Hessian.

This is abstracted because some residuals expect the Hessian to be pre-processed. The Gauss-Newton approximation to the Hessian is defined as:

$$J^TJ$$

Parameters
• sdi ((N, n_params) ndarray) – The steepest descent images.
• sdi2 ((N, n_params) ndarray or None, optional) – The steepest descent images.

Returns H ((n_params, n_params) ndarray) – The approximation to the Hessian

steepest_descent_images(image, dW_dp, forward=None)
Calculates the standard steepest descent images.

Within the forward additive framework this is defined as

$$\nabla I \frac{\partial W}{\partial p}$$

The input image is vectorised (N-pixels) so that masked images can be handled.

Parameters
• image (menpo.image.Image) – The image to calculate the steepest descent images from, could be either the template or input image depending on which framework is used.
• dW_dp (ndarray) – The Jacobian of the warp.
• forward (tuple or None, optional) – A tuple containing the extra weights required for the function warp (which should be passed as a function handle), i.e. ("menpo.image.Image", "menpo.transform.AlignableTransform>”). If None, then the optimization algorithm is assumed to be inverse.

Returns VT_dW_dp ((N, n_params) ndarray) – The steepest descent images

steepest_descent_update(sdi, image, template)
Calculates the steepest descent parameter updates.

These are defined, for the forward additive algorithm, as:

$$\sum_x |\nabla I \frac{\partial W}{\partial p}|^T[T(x) - I(W(x; p))]$$

Parameters
• sdi ((N, n_params) ndarray) – The steepest descent images.
• **image** (*menpo.image.Image*) – Either the warped image or the template (depending on the framework)

• **template** (*menpo.image.Image*) – Either the warped image or the template (depending on the framework)

Returns `sd_delta_p(n_params,) ndarray` – The steepest descent parameter updates.

## Fitting Result

**LucasKanadeResult**

```python
class menpofit.lk.result.LucasKanadeResult(results, scales, affine_transforms, scale_transforms, image=None, gt_shape=None):
    ...
```

Bases: `MultiScaleParametricIterativeResult`

Class for storing the multi-scale iterative fitting result of an ATM. It holds the shapes, shape parameters and costs per iteration.

**Parameters**

- **results** ([list of ATMAlgorithmResult]) – The list of optimization results per scale.
- **scales** ([list or tuple]) – The list of scale values per scale (low to high).
- **affine_transforms** ([list of menpo.transform.Affine]) – The list of affine transforms per scale that transform the shapes into the original image space.
- **scale_transforms** ([list of menpo.shape.Scale]) – The list of scaling transforms per scale.
- **image** (*menpo.image.Image or subclass or None*, optional) – The image on which the fitting process was applied. Note that a copy of the image will be assigned as an attribute. If `None`, then no image is assigned.
- **gt_shape** (*menpo.shape.PointCloud or None*, optional) – The ground truth shape associated with the image. If `None`, then no ground truth shape is assigned.

**displacements()**

A list containing the displacement between the shape of each iteration and the shape of the previous one.

Type `list of ndarray`

**displacements_stats(stat_type=’mean’)**

A list containing a statistical metric on the displacements between the shape of each iteration and the shape of the previous one.

Parameters `stat_type({’mean’, ’median’, ’min’, ’max’}, optional)` – Specifies a statistic metric to be extracted from the displacements.

Returns `displacements_stat(list of float)` – The statistical metric on the points displacements for each iteration.

Raises `ValueError` – type must be ‘mean’, ‘median’, ‘min’ or ‘max’

**errors(compute_error=None)**

Returns a list containing the error at each fitting iteration, if the ground truth shape exists.

Parameters `compute_error(callable, optional)` – Callable that computes the error between the shape at each iteration and the ground truth shape.

Returns `errors(list of float)` – The error at each iteration of the fitting process.
Raises **ValueError** – Ground truth shape has not been set, so the final error cannot be computed.

**final_error** *(compute_error=None)*

Returns the final error of the fitting process, if the ground truth shape exists. This is the error computed based on the `final_shape`.

**Parameters**

- **compute_error** *(callable, optional)* – Callable that computes the error between the fitted and ground truth shapes.

**Returns**

- **final_error** *(float)* – The final error at the end of the fitting process.

Raises **ValueError** – Ground truth shape has not been set, so the final error cannot be computed.

**initial_error** *(compute_error=None)*

Returns the initial error of the fitting process, if the ground truth shape and initial shape exist. This is the error computed based on the `initial_shape`.

**Parameters**

- **compute_error** *(callable, optional)* – Callable that computes the error between the initial and ground truth shapes.

**Returns**

- **initial_error** *(float)* – The initial error at the beginning of the fitting process.

Raises

- **ValueError** – Initial shape has not been set, so the initial error cannot be computed.
- **ValueError** – Ground truth shape has not been set, so the initial error cannot be computed.

**plot_costs** *(figure_id=None, new_figure=False, render_lines=True, line_colour='b', line_style='-', line_width=2, render_markers=True, marker_style='o', marker_size=4, marker_face_colour='b', marker_edge_colour='k', marker_edge_width=1.0, render_axes=True, axes_font_name='sans-serif', axes_font_size=10, axes_font_style='normal', axes_font_weight='normal', axes_x_limits=0.0, axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None, figure_size=(10, 6), render_grid=True, grid_line_style='-', grid_line_width=0.5)*

Plot of the cost function evolution at each fitting iteration.

**Parameters**

- **figure_id** *(object, optional)* – The id of the figure to be used.
- **new_figure** *(bool, optional)* – If True, a new figure is created.
- **render_lines** *(bool, optional)* – If True, the line will be rendered.
- **line_colour** *(colour or None, optional)* – The colour of the line. If None, the colour is sampled from the jet colormap. Example colour options are

```python
{'r', 'g', 'b', 'c', 'm', 'k', 'w'}
```

or

```python
(3, ) ndarray
```

- **line_style** *(str, optional)* – The style of the lines.
- **line_width** *(float, optional)* – The width of the lines.
- **render_markers** *(bool, optional)* – If True, the markers will be rendered.
- **marker_style** *(marker, optional)* – The style of the markers. Example marker options
marker_size (int, optional) – The size of the markers in points.

marker_face_colour (colour or None, optional) – The face (filling) colour of the markers. If None, the colour is sampled from the jet colormap. Example colour options are

- \{‘r’, ‘g’, ‘b’, ‘c’, ‘m’, ‘k’, ‘w’\}
- or
- (3, ) ndarray

marker_edge_colour (colour or None, optional) – The edge colour of the markers. If None, the colour is sampled from the jet colormap. Example colour options are

- \{‘r’, ‘g’, ‘b’, ‘c’, ‘m’, ‘k’, ‘w’\}
- or
- (3, ) ndarray

marker_edge_width (float, optional) – The width of the markers’ edge.

render_axes (bool, optional) – If True, the axes will be rendered.

axes_font_name (See below, optional) – The font of the axes. Example options


axes_font_size (int, optional) – The font size of the axes.


axes_font_weight (See below, optional) – The font weight of the axes. Example options


axes_x_limits (float or (float, float) or None, optional) – The limits of the x axis. If float, then it sets padding on the right and left of the graph as a percentage of the curves’ width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.

axes_y_limits (float or (float, float) or None, optional) – The limits of the y axis. If float, then it sets padding on the top and bottom of the graph as a percentage of the curves’ height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.

axes_x_ticks (list or tuple or None, optional) – The ticks of the x axis.

axes_y_ticks (list or tuple or None, optional) – The ticks of the y axis.

figure_size ((float, float) or None, optional) – The size of the figure in inches.

render_grid (bool, optional) – If True, the grid will be rendered.

grid_line_style (\{‘-’, ‘--’, ‘-.’, ':'\}, optional) – The style of the grid lines.
• **grid_line_width** *(float, optional)* – The width of the grid lines.

**Returns** renderer *(menpo.visualize.GraphPlotter)* – The renderer object.

**plot_displacements** *(stat_type='mean', figure_id=None, new_figure=False, render_lines=True, line_colour='b', line_style='-', line_width=2, render_markers=True, marker_style='o', marker_size=4, marker_face_colour='b', marker_edge_colour='k', marker_edge_width=1.0, render_axes=True, axes_font_name='sans-serif', axes_font_size=10, axes_font_style='normal', axes_font_weight='normal', axes_x_limits=0.0, axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None, figure_size=(10, 6), render_grid=True, grid_line_style='-', grid_line_width=0.5)*

Plot of a statistical metric of the displacement between the shape of each iteration and the shape of the previous one.

**Parameters**

• **stat_type** *(mean, median, min, max, optional)* – Specifies a statistic metric to be extracted from the displacements (see also `displacements_stats()` method).

• **figure_id** *(object, optional)* – The id of the figure to be used.

• **new_figure** *(bool, optional)* – If True, a new figure is created.

• **render_lines** *(bool, optional)* – If True, the line will be rendered.

• **line_colour** *(colour or None, optional)* – The colour of the line. If None, the colour is sampled from the jet colormap. Example colour options are

```
[r, g, b, c, m, k, w]
```

or

```
(3, ) ndarray
```

• **line_style** *(str, optional)* – The style of the lines. Example options:

```
(\-, \--, \-, ., :)
```

• **line_width** *(float, optional)* – The width of the lines.

• **render_markers** *(bool, optional)* – If True, the markers will be rendered.

• **marker_style** *(str, optional)* – The style of the markers. Example marker options

```
{., ., o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
```

• **marker_size** *(int, optional)* – The size of the markers in points.

• **marker_face_colour** *(colour or None, optional)* – The face (filling) colour of the markers. If None, the colour is sampled from the jet colormap. Example colour options are

```
[r, g, b, c, m, k, w]
```

or

```
(3, ) ndarray
```

• **marker_edge_colour** *(colour or None, optional)* – The edge colour of the markers. If None, the colour is sampled from the jet colormap. Example colour options are
or

3, ) ndarray

• **marker_edge_width** *(float, optional)* – The width of the markers’ edge.

• **render_axes** *(bool, optional)* – If True, the axes will be rendered.

• **axes_font_name** *(str (See below), optional)* – The font of the axes. Example options

| {serif, sans-serif, cursive, fantasy, monospace} |

• **axes_font_size** *(int, optional)* – The font size of the axes.

• **axes_font_style** *(str (See below), optional)* – The font style of the axes. Example options

| {normal, italic, oblique} |

• **axes_font_weight** *(str (See below), optional)* – The font weight of the axes. Example options

| {ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black} |

• **axes_x_limits** *(float or (float, float) or None, optional)* – The limits of the x axis. If float, then it sets padding on the right and left of the graph as a percentage of the curves’ width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.

• **axes_y_limits** *(float or (float, float) or None, optional)* – The limits of the y axis. If float, then it sets padding on the top and bottom of the graph as a percentage of the curves’ height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.

• **axes_x_ticks** *(list or tuple or None, optional)* – The ticks of the x axis.

• **axes_y_ticks** *(list or tuple or None, optional)* – The ticks of the y axis.

• **figure_size** *((float, float) or None, optional)* – The size of the figure in inches.

• **render_grid** *(bool, optional)* – If True, the grid will be rendered.

• **grid_line_style** *(('-', ' --', ' -. ', ': '), optional)* – The style of the grid lines.

• **grid_line_width** *(float, optional)* – The width of the grid lines.

**Returns** renderer *(menpo.visualize.GraphPlotter)* – The renderer object.

**plot_errors** *(compute_error=None, figure_id=None, new_figure=False, render_lines=True, line_colour='b', line_style='-', line_width=2, render_markers=True, marker_style='o', marker_size=4, marker_face_colour='b', marker_edge_colour='k', marker_edge_width=1.0, render_axes=True, axes_font_name='sans-serif', axes_font_size=10, axes_font_style='normal', axes_font_weight='normal', axes_x_limits=0.0, axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None, figure_size=(10, 6), render_grid=True, grid_line_style='-', grid_line_width=0.5) *

Plot of the error evolution at each fitting iteration.

**Parameters**
• **compute_error** (*callable*, optional) – Callable that computes the error between the shape at each iteration and the ground truth shape.

• **figure_id** (*object*, optional) – The id of the figure to be used.

• **new_figure** (*bool*, optional) – If True, a new figure is created.

• **render_lines** (*bool*, optional) – If True, the line will be rendered.

• **line_colour** (*colour* or *None* (See below), optional) – The colour of the line. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{r, g, b, c, m, k, w}
```

or

```
(3, ) ndarray
```

• **line_style** (*str* (See below), optional) – The style of the lines. Example options:

```
{-, --, -., :}
```

• **line_width** (*float*, optional) – The width of the lines.

• **render_markers** (*bool*, optional) – If True, the markers will be rendered.

• **marker_style** (*str* (See below), optional) – The style of the markers. Example *marker* options

```
{., ,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
```

• **marker_size** (*int*, optional) – The size of the markers in points.

• **marker_face_colour** (*colour* or *None*, optional) – The face (filling) colour of the markers. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{r, g, b, c, m, k, w}
```

or

```
(3, ) ndarray
```

• **marker_edge_colour** (*colour* or *None*, optional) – The edge colour of the markers. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{r, g, b, c, m, k, w}
```

or

```
(3, ) ndarray
```

• **marker_edge_width** (*float*, optional) – The width of the markers’ edge.

• **render_axes** (*bool*, optional) – If True, the axes will be rendered.

• **axes_font_name** (*str* (See below), optional) – The font of the axes. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

• **axes_font_size** (*int*, optional) – The font size of the axes.

• **axes_font_style** (*str* (See below), optional) – The font style of the axes. Example options

```
{normal, italic, oblique}
```
• **axes_font_weight** (*str* (See below), optional) – The font weight of the axes. Example options

```plaintext
(ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black)
```

• **axes_x_limits** (*float* or (*float*, *float*) or *None*, optional) – The limits of the x axis. If *float*, then it sets padding on the right and left of the graph as a percentage of the curves’ width. If *tuple* or *list*, then it defines the axis limits. If *None*, then the limits are set automatically.

• **axes_y_limits** (*float* or (*float*, *float*) or *None*, optional) – The limits of the y axis. If *float*, then it sets padding on the top and bottom of the graph as a percentage of the curves’ height. If *tuple* or *list*, then it defines the axis limits. If *None*, then the limits are set automatically.

• **axes_x_ticks** (*list* or *tuple* or *None*, optional) – The ticks of the x axis.

• **axes_y_ticks** (*list* or *tuple* or *None*, optional) – The ticks of the y axis.

• **figure_size** (*(*float*, *float*)* or *None*, optional) – The size of the figure in inches.

• **render_grid** (*bool*, optional) – If *True*, the grid will be rendered.

• **grid_line_style** (*{'-', '--', '-.', ':'}*), optional) – The style of the grid lines.

• **grid_line_width** (*float*, optional) – The width of the grid lines.

Returns **renderer** (*menpo.visualize.GraphPlotter*) – The renderer object.

**reconstructed_initial_error** (*compute_error=None*)

Returns the error of the reconstructed initial shape of the fitting process, if the ground truth shape exists. This is the error computed based on the `reconstructed_initial_shapes[0]`.

Parameters **compute_error** (*callable*, optional) – Callable that computes the error between the reconstructed initial and ground truth shapes.

Returns **reconstructed_initial_error** (*float*) – The error that corresponds to the initial shape’s reconstruction.

Raises **ValueError** – Ground truth shape has not been set, so the reconstructed initial error cannot be computed

**to_result** (*pass_image=True, pass_initial_shape=True, pass_gt_shape=True*)

Returns a **Result** instance of the object, i.e. a fitting result object that does not store the iterations. This can be useful for reducing the size of saved fitting results.

Parameters

• **pass_image** (*bool*, optional) – If *True*, then the image will get passed (if it exists).

• **pass_initial_shape** (*bool*, optional) – If *True*, then the initial shape will get passed (if it exists).

• **pass_gt_shape** (*bool*, optional) – If *True*, then the ground truth shape will get passed (if it exists).

Returns **result** (*Result*) – The final “lightweight” fitting result.
Visualize the fitting result. The method renders the final fitted shape and optionally the initial shape, ground truth shape and the image, if they were provided.

**Parameters**

- `figure_id` *(object, optional)* – The id of the figure to be used.
- `new_figure` *(bool, optional)* – If True, a new figure is created.
- `render_image` *(bool, optional)* – If True and the image exists, then it gets rendered.
- `render_final_shape` *(bool, optional)* – If True, then the final fitting shape gets rendered.
- `render_initial_shape` *(bool, optional)* – If True and the initial fitting shape exists, then it gets rendered.
- `render_gt_shape` *(bool, optional)* – If True and the ground truth shape exists, then it gets rendered.
- `subplots_enabled` *(bool, optional)* – If True, then the requested final, initial and ground truth shapes get rendered on separate subplots.
- `channels` *(int or list of int or all or None)* – If int or list of int, the specified channel(s) will be rendered. If all, all the channels will be rendered in subplots. If None and the image is RGB, it will be rendered in RGB mode. If None and the image is not RGB, it is equivalent to all.
- `interpolation` *(See Below, optional)* – The interpolation used to render the image. For example, if bilinear, the image will be smooth and if nearest, the image will be pixelated. Example options

```
{none, nearest, bilinear, bicubic, spline16, spline36, hanning, hamming, hermite, kaiser, quadric, catrom, gaussian, bessel, mitchell, sinc, lanczos}
```

- `cmap_name` *(str, optional)* – If None, single channel and three channel images default to greyscale and rgb colormaps respectively.
- `alpha` *(float, optional)* – The alpha blending value, between 0 (transparent) and 1 (opaque).
• **masked** *(bool, optional)* – If True, then the image is rendered as masked.

• **final_marker_face_colour** *(See Below, optional)* – The face (filling) colour of the markers of the final fitting shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• **final_marker_edge_colour** *(See Below, optional)* – The edge colour of the markers of the final fitting shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• **final_line_colour** *(See Below, optional)* – The line colour of the final fitting shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• **initial_marker_face_colour** *(See Below, optional)* – The face (filling) colour of the markers of the initial shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• **initial_marker_edge_colour** *(See Below, optional)* – The edge colour of the markers of the initial shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• **initial_line_colour** *(See Below, optional)* – The line colour of the initial shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• **gt_marker_face_colour** *(See Below, optional)* – The face (filling) colour of the markers of the ground truth shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• **gt_marker_edge_colour** *(See Below, optional)* – The edge colour of the markers of the ground truth shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```
MenpoFit Documentation, Release 0.6.1

- **gt_line_colour** *(See Below, optional)* – The line colour of the ground truth shape. Example options

  ```markdown
  {r, g, b, c, m, k, w}
  or
  (3,) ndarray
  ```

- **render_lines** *(bool or list of bool, optional)* – If True, the lines will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per shape in *(final, initial, groundtruth)* order.

- **line_style** *(str or list of str, optional)* – The style of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per shape in *(final, initial, groundtruth)* order. Example options:

  ```markdown
  {'-', '--', '-.', ':'}
  ```

- **line_width** *(float or list of float, optional)* – The width of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per shape in *(final, initial, groundtruth)* order.

- **render_markers** *(bool or list of bool, optional)* – If True, the markers will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per shape in *(final, initial, groundtruth)* order.

- **marker_style** *(str or list of str, optional)* – The style of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per shape in *(final, initial, groundtruth)* order. Example options:

  ```markdown
  {., ,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
  ```

- **marker_size** *(int or list of int, optional)* – The size of the markers in points. You can either provide a single value that will be used for all shapes or a list with a different value per shape in *(final, initial, groundtruth)* order.

- **marker_edge_width** *(float or list of float, optional)* – The width of the markers’ edge. You can either provide a single value that will be used for all shapes or a list with a different value per shape in *(final, initial, groundtruth)* order.

- **render_numbering** *(bool, optional)* – If True, the landmarks will be numbered.

- **numbers_horizontal_align** *(center, right, left, optional)* – The horizontal alignment of the numbers’ texts.

- **numbers_vertical_align** *(center, top, bottom, baseline, optional)* – The vertical alignment of the numbers’ texts.

- **numbers_font_name** *(See Below, optional)* – The font of the numbers. Example options

  ```markdown
  {serif, sans-serif, cursive, fantasy, monospace}
  ```

- **numbers_font_size** *(int, optional)* – The font size of the numbers.

- **numbers_font_style** *(normal, italic, oblique, optional)* – The font style of the numbers.

- **numbers_font_weight** *(See Below, optional)* – The font weight of the numbers. Example options
- **numbers_font_colour** *(See Below, optional)* – The font colour of the numbers. Example options

  \{r, g, b, c, m, k, w\}
  \textbf{or}
  \textbf{(3, ) ndarray}

- **render_legend** *(bool, optional)* – If True, the legend will be rendered.
- **legend_title** *(str, optional)* – The title of the legend.
- **legend_font_name** *(See below, optional)* – The font of the legend. Example options

  \{serif, sans-serif, cursive, fantasy, monospace\}

- **legend_font_style** *(\{normal, italic, oblique\}, optional)* – The font style of the legend.
- **legend_font_size** *(int, optional)* – The font size of the legend.
- **legend_font_weight** *(See Below, optional)* – The font weight of the legend. Example options

  \{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black\}

- **legend_marker_scale** *(float, optional)* – The relative size of the legend markers with respect to the original
- **legend_location** *(int, optional)* – The location of the legend. The predefined values are:

<table>
<thead>
<tr>
<th>Location</th>
<th>Integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>'best'</td>
<td>0</td>
</tr>
<tr>
<td>'upper right'</td>
<td>1</td>
</tr>
<tr>
<td>'upper left'</td>
<td>2</td>
</tr>
<tr>
<td>'lower left'</td>
<td>3</td>
</tr>
<tr>
<td>'lower right'</td>
<td>4</td>
</tr>
<tr>
<td>'right'</td>
<td>5</td>
</tr>
<tr>
<td>'center left'</td>
<td>6</td>
</tr>
<tr>
<td>'center right'</td>
<td>7</td>
</tr>
<tr>
<td>'lower center'</td>
<td>8</td>
</tr>
<tr>
<td>'upper center'</td>
<td>9</td>
</tr>
<tr>
<td>'center'</td>
<td>10</td>
</tr>
</tbody>
</table>

- **legend_bbox_to_anchor** *(\(float, float\) tuple, optional)* – The bbox that the legend will be anchored.
- **legend_border_axes_pad** *(float, optional)* – The pad between the axes and legend border.
- **legend_n_columns** *(int, optional)* – The number of the legend’s columns.
- **legend_horizontal_spacing** *(float, optional)* – The spacing between the columns.
• **legend_vertical_spacing** *(float, optional)* – The vertical space between the legend entries.

• **legend_border** *(bool, optional)* – If True, a frame will be drawn around the legend.

• **legend_border_padding** *(float, optional)* – The fractional whitespace inside the legend border.

• **legend_shadow** *(bool, optional)* – If True, a shadow will be drawn behind legend.

• **legend_rounded_corners** *(bool, optional)* – If True, the frame’s corners will be rounded (fancybox).

• **render_axes** *(bool, optional)* – If True, the axes will be rendered.

• **axes_font_name** *(See Below, optional)* – The font of the axes. Example options:

```
[serif, sans-serif, cursive, fantasy, monospace]
```

• **axes_font_size** *(int, optional)* – The font size of the axes.

• **axes_font_style** *(normal, italic, oblique, optional)* – The font style of the axes.

• **axes_font_weight** *(See Below, optional)* – The font weight of the axes. Example options:

```
[ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black]
```

• **axes_x_limits** *(float or (float, float) or None, optional)* – The limits of the x axis. If float, then it sets padding on the right and left of the Image as a percentage of the Image’s width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.

• **axes_y_limits** *((float, float) tuple or None, optional)* – The limits of the y axis. If float, then it sets padding on the top and bottom of the Image as a percentage of the Image’s height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.

• **axes_x_ticks** *(list or tuple or None, optional)* – The ticks of the x axis.

• **axes_y_ticks** *(list or tuple or None, optional)* – The ticks of the y axis.

• **figure_size** *((float, float) tuple or None, optional)* – The size of the figure in inches.

**Returns** *renderer* *(class)* – The renderer object.

---

2.1. Deformable Models
view_iterations (figure_id=None, new_figure=False, iters=None, render_image=True, subplots_enabled=False, channels=None, interpolation='bilinear', cmap_name=None, alpha=1.0, masked=True, line_style='-', line_width=2, line_color=None, render_markers=True, marker_edge_color=None, marker_face_color=None, marker_style='o', marker_size=4, marker_edge_width=2, marker_face_width=2, render_numbering=False, render_lines=True, line_color=None, line_style='-', line_width=2, line_marker='o', marker_face_color='k', marker_edge_color='k', render_numbering=False, render_grid=False, grid_color='grey', grid_linestyle='-.', grid_width=0.5, grid_alpha=0.5, render_gridlabels=True, gridlabel_font_name='sans-serif', gridlabel_font_size=8, gridlabel_font_color='k', render_gridlines=False, gridline_color='k', gridline_width=0.5, gridline_alpha=0.5, render_grid_lines=False, grid_lines_style='-', grid_lines_width=2, grid_lines_face_color='k', grid_lines_edge_color='k', render_numbering=False, numbers_horizontal_align='center', numbers_vertical_align='bottom', numbers_font_name='sans-serif', numbers_font_size=10, numbers_font_style='normal', numbers_font_weight='normal', numbers_font_color='k', render_legend=True, legend_title=' ', legend_font_name='sans-serif', legend_font_style='normal', legend_font_size=10, legend_font_weight='normal', legend_marker_scale=None, legend_location=2, legend_bbox_to_anchor=(1.05, 1.0), legend_border_axes_pad=None, legend_n_columns=1, legend_horizontal_spacing=None, legend_vertical_spacing=None, legend_border=True, legend_border_padding=None, legend_shadow=False, legend_rounded_corners=False, render_axes=True, axes_font_name='sans-serif', axes_font_size=10, axes_font_style='normal', axes_font_weight='normal', axes_x_limits=None, axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None, figure_size=(7, 7))

Visualize the iterations of the fitting process.

Parameters

- **figure_id** (*object*, optional) – The id of the figure to be used.

- **new_figure** (*bool*, optional) – If True, a new figure is created.

- **iters** (*int* or *list* of *int* or *None*, optional) – The iterations to be visualized. If None, then all the iterations are rendered.

<table>
<thead>
<tr>
<th>No.</th>
<th>Visualised shape</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>self.initial_shape</td>
<td>Initial shape</td>
</tr>
<tr>
<td>1</td>
<td>self.reconstructed_initial_shape</td>
<td>Reconstructed initial</td>
</tr>
<tr>
<td>2+i</td>
<td>self.shapes[2]</td>
<td>Iteration i-1</td>
</tr>
<tr>
<td>n_iters+1</td>
<td>self.final_shape</td>
<td>Final shape</td>
</tr>
</tbody>
</table>

- **render_image** (*bool*, optional) – If True and the image exists, then it gets rendered.

- **subplots_enabled** (*bool*, optional) – If True, then the requested final, initial and ground truth shapes get rendered on separate subplots.

- **channels** (*int* or *list* of *int* or *all* or *None*, optional) – If int or list of int, the specified channel(s) will be rendered. If all, all the channels will be rendered in subplots. If None and the image is RGB, it will be rendered in RGB mode. If None and the image is not RGB, it is equivalent to all.

- **interpolation** (*str* (See Below), optional) – The interpolation used to render the image. For example, if bilinear, the image will be smooth and if nearest, the image will be pixelated. Example options

```python
{none, nearest, bilinear, bicubic, spline16, spline36, hanning, hamming, hermite, kaiser, quadric, catrom, gaussian, bessel, mitchell, sinc, lanczos}
```

- **cmap_name** (*str*, optional.) – If None, single channel and three channel images default to greyscale and rgb colormaps respectively.
• **alpha** *(float, optional)* – The alpha blending value, between 0 (transparent) and 1 (opaque).

• **masked** *(bool, optional)* – If True, then the image is rendered as masked.

• **render_lines** *(bool or list of bool, optional)* – If True, the lines will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.

• **line_style** *(str or list of str)* (See below), optional – The style of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.

```plaintext
(-, --, -. , :)
```

• **line_width** *(float or list of float, optional)* – The width of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.

• **line_colour** *(colour or list of colour)* (See Below), optional – The colour of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.

```plaintext
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• **render_markers** *(bool or list of bool, optional)* – If True, the markers will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.

• **marker_style** *(str or list of str)* (See below), optional – The style of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.

```plaintext
{., ,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
```

• **marker_size** *(int or list of int, optional)* – The size of the markers in points. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.

• **marker_edge_colour** *(colour or list of colour)* (See Below), optional – The edge colour of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.

```plaintext
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• **marker_face_colour** *(colour or list of colour)* (See Below), optional – The face (filling) colour of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.

```plaintext
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• **marker_edge_width** *(float or list of float, optional)* – The width of the markers’ edge. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.
value per iteration shape.

- **render_numbering** *(bool, optional)* – If True, the landmarks will be numbered.
- **numbers_horizontal_align** *(str (See below), optional)* – The horizontal alignment of the numbers’ texts. Example options
  
  `{center, right, left}`

- **numbers_vertical_align** *(str (See below), optional)* – The vertical alignment of the numbers’ texts. Example options
  
  `{center, top, bottom, baseline}`

- **numbers_font_name** *(str (See below), optional)* – The font of the numbers. Example options
  
  `{serif, sans-serif, cursive, fantasy, monospace}`

- **numbers_font_size** *(int, optional)* – The font size of the numbers.

- **numbers_font_style** *(str (See below), optional)* – The font style of the numbers. Example options
  
  `{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}`

- **numbers_font_colour** *(See Below, optional)* – The font colour of the numbers. Example options
  
  `{r, g, b, c, m, k, w}`

  or
  
  `{(3, ) ndarray}`

- **render_legend** *(bool, optional)* – If True, the legend will be rendered.
- **legend_title** *(str, optional)* – The title of the legend.
- **legend_font_name** *(See below, optional)* – The font of the legend. Example options
  
  `{serif, sans-serif, cursive, fantasy, monospace}`

- **legend_font_style** *(str (See below), optional)* – The font style of the legend. Example options
  
  `{normal, italic, oblique}`

- **legend_font_size** *(int, optional)* – The font size of the legend.

- **legend_font_weight** *(str (See below), optional)* – The font weight of the legend. Example options
  
  `{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}`
• `legend_marker_scale` *(float, optional)* – The relative size of the legend markers with respect to the original

• `legend_location` *(int, optional)* – The location of the legend. The predefined values are:

<table>
<thead>
<tr>
<th>Location</th>
<th>Integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>'best'</td>
<td>0</td>
</tr>
<tr>
<td>'upper right'</td>
<td>1</td>
</tr>
<tr>
<td>'upper left'</td>
<td>2</td>
</tr>
<tr>
<td>'lower left'</td>
<td>3</td>
</tr>
<tr>
<td>'lower right'</td>
<td>4</td>
</tr>
<tr>
<td>'right'</td>
<td>5</td>
</tr>
<tr>
<td>'center left'</td>
<td>6</td>
</tr>
<tr>
<td>'center right'</td>
<td>7</td>
</tr>
<tr>
<td>'lower center'</td>
<td>8</td>
</tr>
<tr>
<td>'upper center'</td>
<td>9</td>
</tr>
<tr>
<td>'center'</td>
<td>10</td>
</tr>
</tbody>
</table>

• `legend_bbox_to_anchor` *((float, float) tuple, optional)* – The bbox that the legend will be anchored.

• `legend_border_axes_pad` *(float, optional)* – The pad between the axes and legend border.

• `legend_n_columns` *(int, optional)* – The number of the legend’s columns.

• `legend_horizontal_spacing` *(float, optional)* – The spacing between the columns.

• `legend_vertical_spacing` *(float, optional)* – The vertical space between the legend entries.

• `legend_border` *(bool, optional)* – If True, a frame will be drawn around the legend.

• `legend_border_padding` *(float, optional)* – The fractional whitespace inside the legend border.

• `legend_shadow` *(bool, optional)* – If True, a shadow will be drawn behind legend.

• `legend_rounded_corners` *(bool, optional)* – If True, the frame’s corners will be rounded (fancybox).

• `render_axes` *(bool, optional)* – If True, the axes will be rendered.

• `axes_font_name` *(str (See below), optional)* – The font of the axes. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

• `axes_font_size` *(int, optional)* – The font size of the axes.

• `axes_font_style` *((normal, italic, oblique), optional)* – The font style of the axes.

• `axes_font_weight` *(str (See below), optional)* – The font weight of the axes. Example options

```
{ultralight, light, normal, regular, book, medium, roman,
 semibold, demibold, demi, bold, heavy, extra bold, black}
```

• `axes_x_limits` *(float or (float, float) or None, optional)* – The limits of the x axis. If float, then it sets padding on the right and left of the Image as a percentage of the Image’s
width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.

- **axes_y_limits** ((float, float) tuple or None, optional) – The limits of the y axis. If float, then it sets padding on the top and bottom of the Image as a percentage of the Image’s height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.

- **axes_x_ticks** (list or tuple or None, optional) – The ticks of the x axis.

- **axes_y_ticks** (list or tuple or None, optional) – The ticks of the y axis.

- **figure_size** ((float, float) tuple or None optional) – The size of the figure in inches.

**Returns renderer (class)** – The renderer object.

**view_widget** (figure_size=(7, 7))

Visualizes the result object using an interactive widget.

**Parameters** figure_size (int, int, optional) – The initial size of the rendered figure.

**property costs**

- Returns a list with the cost per iteration. It returns None if the costs are not computed.

**Type** list of float or None

**property final_shape**

- Returns the final shape of the fitting process.

**Type** menpo.shape.PointCloud

**property gt_shape**

- Returns the ground truth shape associated with the image. In case there is not an attached ground truth shape, then None is returned.

**Type** menpo.shape.PointCloud or None

**property homogeneous_parameters**

- Returns the list of parameters of the homogeneous transform obtained at each iteration of the fitting process. The list includes the parameters of the initial_shape (if it exists) and final_shape.

**Type** list of (n_params,) ndarray

**property image**

- Returns the image that the fitting was applied on, if it was provided. Otherwise, it returns None.

**Type** menpo.shape.Image or subclass or None

**property initial_shape**

- Returns the initial shape that was provided to the fitting method to initialise the fitting process. In case the initial shape does not exist, then None is returned.

**Type** menpo.shape.PointCloud or None

**property is_iterative**

- Flag whether the object is an iterative fitting result.

**Type** bool

**property n_iters**

- Returns the total number of iterations of the fitting process.

**Type** int

**property n_iters_per_scale**

- Returns the number of iterations per scale of the fitting process.
Type list of int

**property n_scales**
Returns the number of scales used during the fitting process.

Type int

**property reconstructed_initial_shapes**
Returns the result of the reconstruction step that takes place at each scale before applying the iterative optimisation.

Type list of menpo.shape<PointCloud>

**property shape_parameters**
Returns the list of shape parameters obtained at each iteration of the fitting process. The list includes the parameters of the initial_shape (if it exists) and final_shape.

Type list of (n_params,) ndarray

**property shapes**
Returns the list of shapes obtained at each iteration of the fitting process. The list includes the initial_shape (if it exists) and final_shape.

Type list of menpo.shape<PointCloud>

**LucasKanadeAlgorithmResult**

class menpo.fit.lk.result.LucasKanadeAlgorithmResult(shapes, homogeneous_parameters, initial_shape=None, image=None, gt_shape=None, costs=None)

Bases: ParametricIterativeResult

Class for storing the iterative result of a Lucas-Kanade Image Alignment optimization algorithm.

**Parameters**

- **shapes** (list of menpo.shape<PointCloud>) – The list of shapes per iteration. The first and last members correspond to the initial and final shapes, respectively.

- **homogeneous_parameters** (list of (n_parameters,) ndarray) – The list of parameters of the homogeneous transform per iteration. The first and last members correspond to the initial and final shapes, respectively.

- **initial_shape** (menpo.shape<PointCloud> or None, optional) – The initial shape from which the fitting process started. If None, then no initial shape is assigned.

- **image** (menpo.image.Image or subclass or None, optional) – The image on which the fitting process was applied. Note that a copy of the image will be assigned as an attribute. If None, then no image is assigned.

- **gt_shape** (menpo.shape<PointCloud> or None, optional) – The ground truth shape associated with the image. If None, then no ground truth shape is assigned.

- **costs** (list of float or None, optional) – The list of cost per iteration. If None, then it is assumed that the cost function cannot be computed for the specific algorithm.

**displacements()**
A list containing the displacement between the shape of each iteration and the shape of the previous one.

Type list of ndarray

2.1. Deformable Models
displacements_stats (stat_type='mean')
  A list containing a statistical metric on the displacements between the shape of each iteration and the shape of the previous one.

  Parameters stat_type ({'mean', 'median', 'min', 'max'}, optional) – Specifies a statistic metric to be extracted from the displacements.

  Returns displacements_stat (list of float) – The statistical metric on the points displacements for each iteration.

  Raises ValueError – type must be ‘mean’, ‘median’, ‘min’ or ‘max’

errors (compute_error=None)
  Returns a list containing the error at each fitting iteration, if the ground truth shape exists.

  Parameters compute_error (callable, optional) – Callable that computes the error between the shape at each iteration and the ground truth shape.

  Returns errors (list of float) – The error at each iteration of the fitting process.

  Raises ValueError – Ground truth shape has not been set, so the final error cannot be computed

final_error (compute_error=None)
  Returns the final error of the fitting process, if the ground truth shape exists. This is the error computed based on the final_shape.

  Parameters compute_error (callable, optional) – Callable that computes the error between the fitted and ground truth shapes.

  Returns final_error (float) – The final error at the end of the fitting process.

  Raises ValueError – Ground truth shape has not been set, so the final error cannot be computed

initial_error (compute_error=None)
  Returns the initial error of the fitting process, if the ground truth shape and initial shape exist. This is the error computed based on the initial_shape.

  Parameters compute_error (callable, optional) – Callable that computes the error between the initial and ground truth shapes.

  Returns initial_error (float) – The initial error at the beginning of the fitting process.

  Raises
    • ValueError – Initial shape has not been set, so the initial error cannot be computed
    • ValueError – Ground truth shape has not been set, so the initial error cannot be computed

plot_costs (figure_id=None, new_figure=False, render_lines=True, line_colour='b', line_style='-', line_width=2, render_markers=True, marker_style='o', marker_size=4, marker_face_colour='b', marker_edge_colour='k', marker_edge_width=1.0, render_axes=True, axes_font_name='sans-serif', axes_font_size=10, axes_font_style='normal', axes_font_weight='normal', axes_x_limits=0.0, axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None, figure_size=(10, 6), render_grid=True, grid_line_style='–', grid_line_width=0.5)
  Plot of the cost function evolution at each fitting iteration.

  Parameters
    • figure_id (object, optional) – The id of the figure to be used.
    • new_figure (bool, optional) – If True, a new figure is created.
• **render_lines** (*bool*, optional) – If True, the line will be rendered.

• **line_colour** (*colour* or *None*, optional) – The colour of the line. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{'r', 'g', 'b', 'c', 'm', 'k', 'w'}
or
(3, ) ndarray
```

• **line_style** (*({'-', '--', '-.', ':'}, optional) – The style of the lines.

• **line_width** (*float*, optional) – The width of the lines.

• **render_markers** (*bool*, optional) – If True, the markers will be rendered.

• **marker_style** (*marker*, optional) – The style of the markers. Example *marker* options

```
{'.', ',', 'o', 'v', '^', '<', '>', '+', 'x', 'D', 'd', 's', 'p', '*', 'h', 'H', '1', '2', '3', '4', '8'}
```

• **marker_size** (*int*, optional) – The size of the markers in points.

• **marker_face_colour** (*colour* or *None*, optional) – The face (filling) colour of the markers. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{'r', 'g', 'b', 'c', 'm', 'k', 'w'}
or
(3, ) ndarray
```

• **marker_edge_colour** (*colour* or *None*, optional) – The edge colour of the markers. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{'r', 'g', 'b', 'c', 'm', 'k', 'w'}
or
(3, ) ndarray
```

• **marker_edge_width** (*float*, optional) – The width of the markers’ edge.

• **render_axes** (*bool*, optional) – If True, the axes will be rendered.

• **axes_font_name** (*See below, optional*) – The font of the axes. Example options

```
{'serif', 'sans-serif', 'cursive', 'fantasy', 'monospace'}
```

• **axes_font_size** (*int*, optional) – The font size of the axes.

• **axes_font_style** (*{'normal', 'italic', 'oblique'}*, optional) – The font style of the axes.

• **axes_font_weight** (*See below, optional*) – The font weight of the axes. Example options

```
{'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'}
```

• **axes_x_limits** (*float* or (*float*, *float*) or *None*, optional) – The limits of the x axis. If *float*, then it sets padding on the right and left of the graph as a percentage of the curves’
width. If \textit{tuple} or \textit{list}, then it defines the axis limits. If \textit{None}, then the limits are set automatically.

- \textbf{axes_y_limits} (\textit{float} or \textit{float, float} or \textit{None}, optional) – The limits of the y axis. If \textit{float}, then it sets padding on the top and bottom of the graph as a percentage of the curves’ height. If \textit{tuple} or \textit{list}, then it defines the axis limits. If \textit{None}, then the limits are set automatically.

- \textbf{axes_x_ticks} (\textit{list} or \textit{tuple} or \textit{None}, optional) – The ticks of the x axis.

- \textbf{axes_y_ticks} (\textit{list} or \textit{tuple} or \textit{None}, optional) – The ticks of the y axis.

- \textbf{figure_size} ((\textit{float, float}) or \textit{None}, optional) – The size of the figure in inches.

- \textbf{render_grid} (\textit{bool}, optional) – If \textit{True}, the grid will be rendered.

- \textbf{grid_line_style} (\textit{\{'-', '--', '-.', ':'\}}, optional) – The style of the grid lines.

- \textbf{grid_line_width} (\textit{float}, optional) – The width of the grid lines.

\textbf{Returns} \textit{renderer} (menpo.visualize.GraphPlotter) – The renderer object.

\textbf{plot_displacements} (\textit{stat_type}='mean', \textit{figure_id}=None, \textit{new_figure}=False, \textit{render_lines}=True, \\
\textit{line_colour}='b', \textit{line_style}=':', \textit{line_width}=2, \textit{render_markers}=True, \\
\textit{marker_style}='o', \textit{marker_size}=4, \textit{marker_face_colour}='b', \\
\textit{marker_edge_colour}='k', \textit{marker_edge_width}=1.0, \textit{render_axes}=True, \\
\textit{axes_font_name}='sans-serif', \textit{axes_font_size}=10, \textit{axes_font_style}='normal', \\
\textit{axes_font_weight}='normal', \textit{axes_x_limits}=0.0, \textit{axes_y_limits}=None, \\
\textit{axes_x_ticks}=None, \textit{axes_y_ticks}=None, \textit{figure_size}=(10, 6), \\
\textit{render_grid}=True, \textit{grid_line_style}='--', \textit{grid_line_width}=0.5)

Plot of a statistical metric of the displacement between the shape of each iteration and the shape of the previous one.

\textbf{Parameters}

- \textbf{stat_type} ({\textit{mean, median, min, max}}, optional) – Specifies a statistic metric to be extracted from the displacements (see also \textit{displacements_stats()} method).

- \textbf{figure_id} (\textit{object}, optional) – The id of the figure to be used.

- \textbf{new_figure} (\textit{bool}, optional) – If \textit{True}, a new figure is created.

- \textbf{render_lines} (\textit{bool}, optional) – If \textit{True}, the line will be rendered.

- \textbf{line_colour} (\textit{\textit{colour}}, \textit{\textit{None}} (See below), optional) – The colour of the line. If \textit{None}, the colour is sampled from the jet colormap. Example \textit{colour} options are

\begin{verbatim}
{r, g, b, c, m, k, w}

or

{3, } ndarray
\end{verbatim}

- \textbf{line_style} (\textit{str} (See below), optional) – The style of the lines. Example options:

\begin{verbatim}
{-, --, -, :}
\end{verbatim}

- \textbf{line_width} (\textit{float}, optional) – The width of the lines.

- \textbf{render_markers} (\textit{bool}, optional) – If \textit{True}, the markers will be rendered.

- \textbf{marker_style} (\textit{str} (See below), optional) – The style of the markers. Example \textit{marker} options
• `marker_size` *(int, optional)* – The size of the markers in points.

• `marker_face_colour` *(colour or None, optional)* – The face (filling) colour of the markers. If `None`, the colour is sampled from the jet colormap. Example `colour` options are

```python
[r, g, b, c, m, k, w]
or
(3, ) ndarray
```

• `marker_edge_colour` *(colour or None, optional)* – The edge colour of the markers. If `None`, the colour is sampled from the jet colormap. Example `colour` options are

```python
[r, g, b, c, m, k, w]
or
(3, ) ndarray
```

• `marker_edge_width` *(float, optional)* – The width of the markers’ edge.

• `render_axes` *(bool, optional)* – If `True`, the axes will be rendered.

• `axes_font_name` *(str (See below), optional)* – The font of the axes. Example options

```python
{serif, sans-serif, cursive, fantasy, monospace}
```

• `axes_font_size` *(int, optional)* – The font size of the axes.

• `axes_font_style` *(str (See below), optional)* – The font style of the axes. Example options

```python
{normal, italic, oblique}
```

• `axes_font_weight` *(str (See below), optional)* – The font weight of the axes. Example options

```python
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```

• `axes_x_limits` *(float or (float, float) or None, optional)* – The limits of the x axis. If `float`, then it sets padding on the right and left of the graph as a percentage of the curves’ width. If `tuple` or `list`, then it defines the axis limits. If `None`, then the limits are set automatically.

• `axes_y_limits` *(float or (float, float) or None, optional)* – The limits of the y axis. If `float`, then it sets padding on the top and bottom of the graph as a percentage of the curves’ height. If `tuple` or `list`, then it defines the axis limits. If `None`, then the limits are set automatically.

• `axes_x_ticks` *(list or tuple or None, optional)* – The ticks of the x axis.

• `axes_y_ticks` *(list or tuple or None, optional)* – The ticks of the y axis.

• `figure_size` *(float, float) or None, optional)* – The size of the figure in inches.

• `render_grid` *(bool, optional)* – If `True`, the grid will be rendered.

• `grid_line_style` *(('-','--','-.',':'), optional)* – The style of the grid lines.
• **grid_line_width** *(float, optional)* – The width of the grid lines.

Returns **renderer** *(menpo.visualize.GraphPlotter)* – The renderer object.

**plot_errors** *(compute_error=None, figure_id=None, new_figure=False, render_lines=True, line_colour='b', line_style='-', line_width=2, render_markers=True, marker_style='o', marker_size=4, marker_face_colour='b', marker_edge_colour='k', marker_edge_width=1.0, render_axes=True, axes_font_name='sans-serif', axes_font_size=10, axes_font_style='normal', axes_font_weight='normal', axes_x_limits=0.0, axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None, figure_size=(10, 6), render_grid=True, grid_line_style='–', grid_line_width=0.5)*

Plot of the error evolution at each fitting iteration.

Parameters

• **compute_error** *(callable, optional)* – Callable that computes the error between the shape at each iteration and the ground truth shape.

• **figure_id** *(object, optional)* – The id of the figure to be used.

• **new_figure** *(bool, optional)* – If True, a new figure is created.

• **render_lines** *(bool, optional)* – If True, the line will be rendered.

• **line_colour** *(colour or None (See below), optional)* – The colour of the line. If None, the colour is sampled from the jet colormap. Example *colour options are*

```python
{r, g, b, c, m, k, w}
or
(3,) ndarray
```

• **line_style** *(str (See below), optional)* – The style of the lines. Example options:

```python
{-, --, -., :}
```

• **line_width** *(float, optional)* – The width of the lines.

• **render_markers** *(bool, optional)* – If True, the markers will be rendered.

• **marker_style** *(str (See below), optional)* – The style of the markers. Example *marker options are*

```python
{., ,, o, v, ^, <, >, *, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
```

• **marker_size** *(int, optional)* – The size of the markers in points.

• **marker_face_colour** *(colour or None, optional)* – The face (filling) colour of the markers. If None, the colour is sampled from the jet colormap. Example *colour options are*

```python
{r, g, b, c, m, k, w}
or
(3,) ndarray
```

• **marker_edge_colour** *(colour or None, optional)* – The edge colour of the markers. If None, the colour is sampled from the jet colormap. Example *colour options are*

```python
{r, g, b, c, m, k, w}
or
(3,) ndarray
```

• **marker_edge_width** *(float, optional)* – The width of the markers’ edge.
• **render_axes** (*bool*, optional) – If *True*, the axes will be rendered.

• **axes_font_name** (*str* (See below), optional) – The font of the axes. Example options

```python
{serif, sans-serif, cursive, fantasy, monospace}
```

• **axes_font_size** (*int*, optional) – The font size of the axes.

• **axes_font_style** (*str* (See below), optional) – The font style of the axes. Example options

```python
{normal, italic, oblique}
```

• **axes_font_weight** (*str* (See below), optional) – The font weight of the axes. Example options

```python
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```

• **axes_x_limits** (*float* or (*float*, *float*), optional) – The limits of the x axis. If *float*, then it sets padding on the right and left of the graph as a percentage of the curves’ width. If *tuple* or *list*, then it defines the axis limits. If *None*, then the limits are set automatically.

• **axes_y_limits** (*float* or (*float*, *float*), optional) – The limits of the y axis. If *float*, then it sets padding on the top and bottom of the graph as a percentage of the curves’ height. If *tuple* or *list*, then it defines the axis limits. If *None*, then the limits are set automatically.

• **axes_x_ticks** (*list* or *tuple* or *None*, optional) – The ticks of the x axis.

• **axes_y_ticks** (*list* or *tuple* or *None*, optional) – The ticks of the y axis.

• **figure_size** (*float*, optional) – The size of the figure in inches.

• **render_grid** (*bool*, optional) – If *True*, the grid will be rendered.

• **grid_line_style** (*{'-', '--', '-.', ':'}, optional*) – The style of the grid lines.

• **grid_line_width** (*float*, optional) – The width of the grid lines.

Returns **renderer** *(menpo.visualize.GraphPlotter)* – The renderer object.

**reconstructed_initial_error** *(compute_error=None)*

Returns the error of the reconstructed initial shape of the fitting process, if the ground truth shape exists. This is the error computed based on the `reconstructed_initial_shape`.

Parameters **compute_error** (*callable*, optional) – Callable that computes the error between the reconstructed initial and ground truth shapes.

Returns **reconstructed_initial_error** (*float*) – The error that corresponds to the initial shape’s reconstruction.

Raises **ValueError** – Ground truth shape has not been set, so the reconstructed initial error cannot be computed

**to_result** *(pass_image=True, pass_initial_shape=True, pass_gt_shape=True)*

Returns a **Result** instance of the object, i.e. a fitting result object that does not store the iterations. This can be useful for reducing the size of saved fitting results.

Parameters

2.1. Deformable Models
• **pass_image** (*bool*, optional) – If *True*, then the image will get passed (if it exists).

• **pass_initial_shape** (*bool*, optional) – If *True*, then the initial shape will get passed (if it exists).

• **pass_gt_shape** (*bool*, optional) – If *True*, then the ground truth shape will get passed (if it exists).

**Returns**

*result* (*Result*) – The final “lightweight” fitting result.

**view**

- *figure_id* (object, optional) – The id of the figure to be used.
- *new_figure* (*bool*, optional) – If *True*, a new figure is created.
- *render_image* (*bool*, optional) – If *True* and the image exists, then it gets rendered.
- *render_final_shape* (*bool*, optional) – If *True*, then the final fitting shape gets rendered.
- *render_initial_shape* (*bool*, optional) – If *True* and the initial fitting shape exists, then it gets rendered.
- *render_gt_shape* (*bool*, optional) – If *True* and the ground truth shape exists, then it gets rendered.
- *subplots_enabled* (*bool*, optional) – If *True*, then the requested final, initial and ground truth shapes get rendered on separate subplots.
- *channels* (*int or list of int or all or None*) – If *int or list of int*, the specified channel(s) will be rendered. If *all*, all the channels will be rendered in subplots. If *None* and the image is RGB, it will be rendered in RGB mode. If *None* and the image is not RGB, it is equivalent to *all*.
- *interpolation* (*See Below, optional*) – The interpolation used to render the image. For example, if *bilinear*, the image will be smooth and if *nearest*, the image will be pixelated. Example options
- **cmap_name** *(str, optional)* – If None, single channel and three channel images default to greyscale and rgb colormaps respectively.

- **alpha** *(float, optional)* – The alpha blending value, between 0 (transparent) and 1 (opaque).

- **masked** *(bool, optional)* – If True, then the image is rendered as masked.

- **final_marker_face_colour** *(See Below, optional)* – The face (filling) colour of the markers of the final fitting shape. Example options

  ```python
  [r, g, b, c, m, k, w]
  or
  (3, ) ndarray
  ```

- **final_marker_edge_colour** *(See Below, optional)* – The edge colour of the markers of the final fitting shape. Example options

  ```python
  [r, g, b, c, m, k, w]
  or
  (3, ) ndarray
  ```

- **final_line_colour** *(See Below, optional)* – The line colour of the final fitting shape. Example options

  ```python
  [r, g, b, c, m, k, w]
  or
  (3, ) ndarray
  ```

- **initial_marker_face_colour** *(See Below, optional)* – The face (filling) colour of the markers of the initial shape. Example options

  ```python
  [r, g, b, c, m, k, w]
  or
  (3, ) ndarray
  ```

- **initial_marker_edge_colour** *(See Below, optional)* – The edge colour of the markers of the initial shape. Example options

  ```python
  [r, g, b, c, m, k, w]
  or
  (3, ) ndarray
  ```

- **initial_line_colour** *(See Below, optional)* – The line colour of the initial shape. Example options

  ```python
  [r, g, b, c, m, k, w]
  or
  (3, ) ndarray
  ```

- **gt_marker_face_colour** *(See Below, optional)* – The face (filling) colour of the markers of the ground truth shape. Example options

  ```python
  [r, g, b, c, m, k, w]
  or
  (3, ) ndarray
  ```
- **gt_marker_edge_colour** *(See Below, optional)* – The edge colour of the markers of the ground truth shape. Example options

\[
\{r, g, b, c, m, k, w\}
\]

or

\[(3, ) \text{ndarray}\]

- **gt_line_colour** *(See Below, optional)* – The line colour of the ground truth shape. Example options

\[
\{r, g, b, c, m, k, w\}
\]

or

\[(3, ) \text{ndarray}\]

- **render_lines** *(bool or list of bool, optional)* – If True, the lines will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per shape in *(final, initial, groundtruth)* order.

- **line_style** *(str or list of str, optional)* – The style of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per shape in *(final, initial, groundtruth)* order. Example options:

\[
\{-', '--', '-.', ':'\}
\]

- **line_width** *(float or list of float, optional)* – The width of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per shape in *(final, initial, groundtruth)* order.

- **render_markers** *(bool or list of bool, optional)* – If True, the markers will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per shape in *(final, initial, groundtruth)* order.

- **marker_style** *(str or list of str, optional)* – The style of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per shape in *(final, initial, groundtruth)* order. Example options:

\[
\{., ,, o, v, ^, <, >, \ldots, x, D, d, s, p, \ldots, h, H, 1, 2, 3, 4, 8\}
\]

- **marker_size** *(int or list of int, optional)* – The size of the markers in points. You can either provide a single value that will be used for all shapes or a list with a different value per shape in *(final, initial, groundtruth)* order.

- **marker_edge_width** *(float or list of float, optional)* – The width of the markers’ edge. You can either provide a single value that will be used for all shapes or a list with a different value per shape in *(final, initial, groundtruth)* order.

- **render_numbering** *(bool, optional)* – If True, the landmarks will be numbered.

- **numbers_horizontal_align** *(center, right, left), optional)* – The horizontal alignment of the numbers’ texts.

- **numbers_vertical_align** *(center, top, bottom, baseline), optional)* – The vertical alignment of the numbers’ texts.
• **numbers_font_name** *(See Below, optional)* – The font of the numbers. Example options

{serif, sans-serif, cursive, fantasy, monospace}

• **numbers_font_size** *(int, optional)* – The font size of the numbers.

• **numbers_font_style** *(normal, italic, oblique), optional)* – The font style of the numbers.

• **numbers_font_weight** *(See Below, optional)* – The font weight of the numbers. Example options

{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}

• **numbers_font_colour** *(See Below, optional)* – The font colour of the numbers. Example options

{r, g, b, c, m, k, w}

or

(3, ) ndarray

• **render_legend** *(bool, optional)* – If True, the legend will be rendered.

• **legend_title** *(str, optional)* – The title of the legend.

• **legend_font_name** *(See below, optional)* – The font of the legend. Example options

{serif, sans-serif, cursive, fantasy, monospace}

• **legend_font_style** *(normal, italic, oblique), optional)* – The font style of the legend.

• **legend_font_size** *(int, optional)* – The font size of the legend.

• **legend_font_weight** *(See Below, optional)* – The font weight of the legend. Example options

{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}

• **legend_marker_scale** *(float, optional)* – The relative size of the legend markers with respect to the original

• **legend_location** *(int, optional)* – The location of the legend. The predefined values are:
• `legend_bbox_to_anchor` ((float, float) tuple, optional) – The bbox that the legend will be anchored.

• `legend_border_axes_pad` (float, optional) – The pad between the axes and legend border.

• `legend_n_columns` (int, optional) – The number of the legend’s columns.

• `legend_horizontal_spacing` (float, optional) – The spacing between the columns.

• `legend_vertical_spacing` (float, optional) – The vertical space between the legend entries.

• `legend_border` (bool, optional) – If True, a frame will be drawn around the legend.

• `legend_border_padding` (float, optional) – The fractional whitespace inside the legend border.

• `legend_shadow` (bool, optional) – If True, a shadow will be drawn behind legend.

• `legend_rounded_corners` (bool, optional) – If True, the frame’s corners will be rounded (fancybox).

• `render_axes` (bool, optional) – If True, the axes will be rendered.

• `axes_font_name` (See Below, optional) – The font of the axes. Example options

{serif, sans-serif, cursive, fantasy, monospace}

• `axes_font_size` (int, optional) – The font size of the axes.

• `axes_font_style` ((normal, italic, oblique), optional) – The font style of the axes.

• `axes_font_weight` (See Below, optional) – The font weight of the axes. Example options

{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}

• `axes_x_limits` (float or (float, float) or None, optional) – The limits of the x axis. If float, then it sets padding on the right and left of the Image as a percentage of the Image’s width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.

• `axes_y_limits` ((float, float) tuple or None, optional) – The limits of the y axis. If float, then it sets padding on the top and bottom of the Image as a percentage of the
Image’s height. If *tuple* or *list*, then it defines the axis limits. If *None*, then the limits are set automatically.

- **axes_x_ticks** (*list* or *tuple* or *None*, optional) – The ticks of the x axis.
- **axes_y_ticks** (*list* or *tuple* or *None*, optional) – The ticks of the y axis.
- **figure_size** (*float*, *float* *tuple* or *None* optional) – The size of the figure in inches.

Returns **renderer** (*class*) – The renderer object.

```
view_iterations (figure_id=None, new_figure=False, iters=None, render_image=True, subplots_enabled=False, channels=None, cmap_name=None, alpha=1.0, masked=True, render_lines=True, line_style='-', line_width=2, line_colour=None, render_markers=True, marker_edge_colour=None, marker_face_colour=None, marker_style='o', marker_size=4, marker_edge_width=1.0, render_numbering=False, numbers_horizontal_align='center', numbers_vertical_align='bottom', numbers_font_name='sans-serif', numbers_font_size=10, numbers_font_style='normal', numbers_font_weight='normal', numbers_font_colour='k', render_legend=True, legend_title='', legend_font_name='sans-serif', legend_font_style='normal', legend_font_size=10, legend_font_weight='normal', legend_location=2, legend_bbox_to_anchor=(1.05, 1.0), legend_border_axes_pad=None, legend_n_columns=1, legend_horizontal_spacing=None, legend_vertical_spacing=None, legend_border=True, legend_border_padding=None, legend_shadow=False, legend_rounded_corners=False, axes_color='k', axes_font_name='sans-serif', axes_font_size=10, axes_font_style='normal', axes_font_weight='normal', axes_x_limits=None, axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None, figure_size=(7, 7))
```

Visualize the iterations of the fitting process.

**Parameters**

- **figure_id** (*object*, optional) – The id of the figure to be used.
- **new_figure** (*bool*, optional) – If *True*, a new figure is created.
- **iters** (*int* or *list* of *int* or *None*, optional) – The iterations to be visualized. If *None*, then all the iterations are rendered.

<table>
<thead>
<tr>
<th>No.</th>
<th>Visualised shape</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>self.initial_shape</td>
<td>Initial shape</td>
</tr>
<tr>
<td>1</td>
<td>self.reconstructed_initial_shape</td>
<td>Reconstructed initial</td>
</tr>
<tr>
<td>2</td>
<td>self.shapes[2]</td>
<td>Iteration 1</td>
</tr>
<tr>
<td>1</td>
<td>self.shapes[1]</td>
<td>Iteration i-1</td>
</tr>
<tr>
<td>n_iters+1</td>
<td>self.final_shape</td>
<td>Final shape</td>
</tr>
</tbody>
</table>

- **render_image** (*bool*, optional) – If *True* and the image exists, then it gets rendered.
- **subplots_enabled** (*bool*, optional) – If *True*, then the requested final, initial and ground truth shapes get rendered on separate subplots.
- **channels** (*int* or *list* of *int* or *all* or *None*) – If *int* or *list* of *int*, the specified channel(s) will be rendered. If *all*, all the channels will be rendered in subplots. If *None* and the image is RGB, it will be rendered in RGB mode. If *None* and the image is not RGB, it is equivalent to *all*.

2.1. Deformable Models 251
• **interpolation** *(str, optional)* – The interpolation used to render the image. For example, if *bilinear*, the image will be smooth and if *nearest*, the image will be pixelated. Example options

```plaintext
{none, nearest, bilinear, bicubic, spline16, spline36, hanning, hamming, hermite, kaiser, quadric, catrom, gaussian, bessel, mitchell, sinc, lanczos}
```

• **cmap_name** *(str, optional)* – If *None*, single channel and three channel images default to greyscale and rgb colormaps respectively.

• **alpha** *(float, optional)* – The alpha blending value, between 0 (transparent) and 1 (opaque).

• **masked** *(bool, optional)* – If *True*, then the image is rendered as masked.

• **render_lines** *(bool or list of bool, optional)* – If *True*, the lines will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.

• **line_style** *(str or list of str, optional)* – The style of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options:

```plaintext
{-, --, -, :}
```

• **line_width** *(float or list of float, optional)* – The width of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.

• **line_colour** *(colour or list of colour, optional)* – The colour of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```plaintext
{r, g, b, c, m, k, w}
```

• **render_markers** *(bool or list of bool, optional)* – If *True*, the markers will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.

• **marker_style** *(str or list of str, optional)* – The style of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```plaintext
{., ,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
```

• **marker_size** *(int or list of int, optional)* – The size of the markers in points. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.

• **marker_edge_colour** *(colour or list of colour, optional)* – The edge colour of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```plaintext
{r, g, b, c, m, k, w}
```

or

```plaintext
(3, ) ndarray
```
• **marker_face_colour** (*colour* or *list* of *colour* (See Below), optional) – The face (filling) colour of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```python
[r, g, b, c, m, k, w]
or
(3, ) ndarray
```

• **marker_edge_width** (*float* or *list* of *float*, optional) – The width of the markers’ edge. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.

• **render_numbering** (*bool*, optional) – If True, the landmarks will be numbered.

• **numbers_horizontal_align** (*str* (See below), optional) – The horizontal alignment of the numbers’ texts. Example options

```python
{center, right, left}
```

• **numbers_vertical_align** (*str* (See below), optional) – The vertical alignment of the numbers’ texts. Example options

```python
{center, top, bottom, baseline}
```

• **numbers_font_name** (*str* (See below), optional) – The font of the numbers. Example options

```python
{serif, sans-serif, cursive, fantasy, monospace}
```

• **numbers_font_size** (*int*, optional) – The font size of the numbers.

• **numbers_font_style** (*str* (See above, optional) – The font style of the numbers.

• **numbers_font_weight** (*str* (See above, optional) – The font weight of the numbers. Example options

```python
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```

• **numbers_font_colour** (See Below, optional) – The font colour of the numbers. Example options

```python
[r, g, b, c, m, k, w]
or
(3, ) ndarray
```

• **render_legend** (*bool*, optional) – If True, the legend will be rendered.

• **legend_title** (*str*, optional) – The title of the legend.

• **legend_font_name** (See below, optional) – The font of the legend. Example options

```python
{serif, sans-serif, cursive, fantasy, monospace}
```

• **legend_font_style** (*str* (See below), optional) – The font style of the legend. Example options
• **legend_font_size** (*int*, optional) – The font size of the legend.

• **legend_font_weight** (*str* (See below), optional) – The font weight of the legend. Example options

  ```
  {ultralight, light, normal, regular, book, medium, roman,
   semibold, demibold, demi, bold, heavy, extra bold, black}
  ```

• **legend_marker_scale** (*float*, optional) – The relative size of the legend markers with respect to the original

• **legend_location** (*int*, optional) – The location of the legend. The predefined values are:

<table>
<thead>
<tr>
<th>Legend Location</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>best</td>
<td>0</td>
</tr>
<tr>
<td>upper right</td>
<td>1</td>
</tr>
<tr>
<td>upper left</td>
<td>2</td>
</tr>
<tr>
<td>lower left</td>
<td>3</td>
</tr>
<tr>
<td>lower right</td>
<td>4</td>
</tr>
<tr>
<td>right</td>
<td>5</td>
</tr>
<tr>
<td>center left</td>
<td>6</td>
</tr>
<tr>
<td>center right</td>
<td>7</td>
</tr>
<tr>
<td>lower center</td>
<td>8</td>
</tr>
<tr>
<td>upper center</td>
<td>9</td>
</tr>
<tr>
<td>center</td>
<td>10</td>
</tr>
</tbody>
</table>

• **legend_bbox_to_anchor** (*(*float*, *float*)* tuple, optional) – The bbox that the legend will be anchored.

• **legend_border_axes_pad** (*float*, optional) – The pad between the axes and legend border.

• **legend_n_columns** (*int*, optional) – The number of the legend’s columns.

• **legend_horizontal_spacing** (*float*, optional) – The spacing between the columns.

• **legend_vertical_spacing** (*float*, optional) – The vertical space between the legend entries.

• **legend_border** (*bool*, optional) – If `True`, a frame will be drawn around the legend.

• **legend_border_padding** (*float*, optional) – The fractional whitespace inside the legend border.

• **legend_shadow** (*bool*, optional) – If `True`, a shadow will be drawn behind legend.

• **legend_rounded_corners** (*bool*, optional) – If `True`, the frame’s corners will be rounded (fancybox).

• **render_axes** (*bool*, optional) – If `True`, the axes will be rendered.

• **axes_font_name** (*str* (See below), optional) – The font of the axes. Example options

  ```
  {serif, sans-serif, cursive, fantasy, monospace}
  ```

• **axes_font_size** (*int*, optional) – The font size of the axes.
- **axes_font_style** ((normal, italic, oblique), optional) – The font style of the axes.

- **axes_font_weight** (str (See below), optional) – The font weight of the axes. Example options
  
  `{ultralight, light, normal, regular, book, medium, roman, semibold, demi, bold, heavy, extra bold, black}`

- **axes_x_limits** (float or (float, float) or None, optional) – The limits of the x axis. If float, then it sets padding on the right and left of the Image as a percentage of the Image’s width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.

- **axes_y_limits** ((float, float) tuple or None, optional) – The limits of the y axis. If float, then it sets padding on the top and bottom of the Image as a percentage of the Image’s height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.

- **axes_x_ticks** (list or tuple or None, optional) – The ticks of the x axis.

- **axes_y_ticks** (list or tuple or None, optional) – The ticks of the y axis.

- **figure_size** ((float, float) tuple or None optional) – The size of the figure in inches.

  Returns **renderer** (class) – The renderer object.

  **view_widget** (figure_size=(7, 7))

  Visualizes the result object using an interactive widget.

  Parameters **figure_size** ((int, int), optional) – The initial size of the rendered figure.

  **property costs**

  Returns a list with the cost per iteration. It returns None if the costs are not computed.

  Type **list** of float or None

  **property final_shape**

  Returns the final shape of the fitting process.

  Type **menpo.shape.PointCloud**

  **property gt_shape**

  Returns the ground truth shape associated with the image. In case there is not an attached ground truth shape, then None is returned.

  Type **menpo.shape.PointCloud** or None

  **property homogeneous_parameters**

  Returns the list of parameters of the homogeneous transform obtained at each iteration of the fitting process. The list includes the parameters of the initial_shape (if it exists) and final_shape.

  Type **list** of (n_params,) ndarray

  **property image**

  Returns the image that the fitting was applied on, if it was provided. Otherwise, it returns None.

  Type **menpo.shape.Image** or subclass or None

  **property initial_shape**

  Returns the initial shape that was provided to the fitting method to initialise the fitting process. In case the initial shape does not exist, then None is returned.

  Type **menpo.shape.PointCloud** or None
**property is_iterative**
Flag whether the object is an iterative fitting result.

Type *bool*

**property n_iters**
Returns the total number of iterations of the fitting process.

Type *int*

**property reconstructed_initial_shape**
Returns the initial shape’s reconstruction with the shape model that was used to initialise the iterative optimisation process.

Type *menpo.shape.PointCloud*

**property shape_parameters**
Returns the list of shape parameters obtained at each iteration of the fitting process. The list includes the parameters of the reconstructed_initial_shape and final_shape.

Type *list of (n_params,) ndarray*

**property shapes**
Returns the list of shapes obtained at each iteration of the fitting process. The list includes the initial_shape (if it exists), reconstructed_initial_shape and final_shape.

Type *list of menpo.shape.PointCloud*

### 2.1.8 menpofit.sdm

**Supervised Descent Method**

SDM is a cascaded-regression deformable model that learns average descent directions that minimise a given cost function.

**SupervisedDescentFitter**

class menpofit.sdm.SupervisedDescentFitter(images, group=None, bounding_box_group_glob=None, sd_algorithm_cls=None, reference_shape=None, diagonal=None, holistic_features=<function no_op>, patch_features=<function no_op>, patch_shape=(17, 17), scales=(0.5, 1.0), n_iterations=3, n_perturbations=30, perturb_from_gt_bounding_box=<function noisy_shape_from_bounding_box>, noisy_shape_from_bounding_box=None, batch_size=None, verbose=False)

Bases: *MultiScaleNonParametricFitter*

Class for training a multi-scale Supervised Descent model.

**Parameters**

- **images** (list of menpo.image.Image) – The list of training images.
- **group** (str or None, optional) – The landmark group that corresponds to the ground truth shape of each image. If None and the images only have a single landmark group, then that
is the one that will be used. Note that all the training images need to have the specified landmark group.

- **bounding_box_group_glob** *(glob or None, optional)* – Glob that defines the bounding boxes to be used for training. If None, then the bounding boxes of the ground truth shapes are used.

- **sd_algorithm_cls** *(class, optional)* – The Supervised Descent algorithm to be used. The possible algorithms are are separated in the following four categories:

  **Non-parametric:**

<table>
<thead>
<tr>
<th>Class</th>
<th>Regression</th>
</tr>
</thead>
<tbody>
<tr>
<td>NonParametricNewton</td>
<td>IRLRegression</td>
</tr>
<tr>
<td>NonParametricGaussNewton</td>
<td>IIRLRegression</td>
</tr>
<tr>
<td>NonParametricPCRRegression</td>
<td>PCRRegression</td>
</tr>
<tr>
<td>NonParametricOptimalRegression</td>
<td>OptimalLinearRegression</td>
</tr>
<tr>
<td>NonParametricOPPRegression</td>
<td>OPPRegression</td>
</tr>
</tbody>
</table>

  **Parametric shape:**

<table>
<thead>
<tr>
<th>Class</th>
<th>Regression</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParametricShapeNewton</td>
<td>IRLRegression</td>
</tr>
<tr>
<td>ParametricShapeGaussNewton</td>
<td>IIRLRegression</td>
</tr>
<tr>
<td>ParametricShapePCRRegression</td>
<td>PCRRegression</td>
</tr>
<tr>
<td>ParametricShapeOptimalRegression</td>
<td>OptimalLinearRegression</td>
</tr>
<tr>
<td>ParametricShapeOPPRegression</td>
<td>ParametricShapeOPPRegression</td>
</tr>
</tbody>
</table>

  **Parametric appearance:**

<table>
<thead>
<tr>
<th>Class</th>
<th>Regression</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParametricAppearanceProjectOutNewton</td>
<td>IRLRegression</td>
</tr>
<tr>
<td>ParametricAppearanceProjectOutGaussNewton</td>
<td>IIRLRegression</td>
</tr>
<tr>
<td>ParametricAppearanceMeanTemplateNewton</td>
<td>IRLRegression</td>
</tr>
<tr>
<td>ParametricAppearanceMeanTemplateGaussNewton</td>
<td>IIRLRegression</td>
</tr>
<tr>
<td>ParametricAppearanceWeightsNewton</td>
<td>IRLRegression</td>
</tr>
<tr>
<td>ParametricAppearanceWeightsGaussNewton</td>
<td>IIRLRegression</td>
</tr>
</tbody>
</table>

  **Parametric shape and appearance:**

<table>
<thead>
<tr>
<th>Class</th>
<th>Regression</th>
</tr>
</thead>
<tbody>
<tr>
<td>FullyParametricProjectOutNewton</td>
<td>IRLRegression</td>
</tr>
<tr>
<td>FullyParametricProjectOutGaussNewton</td>
<td>IIRLRegression</td>
</tr>
<tr>
<td>FullyParametricMeanTemplateNewton</td>
<td>IRLRegression</td>
</tr>
<tr>
<td>FullyParametricWeightsNewton</td>
<td>IRLRegression</td>
</tr>
<tr>
<td>FullyParametricProjectOutOPP</td>
<td>OPPRegression</td>
</tr>
</tbody>
</table>

  - **reference_shape** *(menpo.shape.PointCloud or None, optional)* – The reference shape that will be used for normalising the size of the training images. The normalization is performed by rescaling all the training images so that the scale of their ground truth shapes matches the scale of the reference shape. Note that the reference shape is rescaled with respect to the **diagonal** before performing the normalisation. If None, then the mean shape will be used.

  2.1. Deformable Models
• **diagonal** (*int* or *None*, optional) – This parameter is used to rescale the reference shape so that the diagonal of its bounding box matches the provided value. In other words, this parameter controls the size of the model at the highest scale. If *None*, then the reference shape does not get rescaled.

• **holistic_features** (*closure* or *list* of *closure*, optional) – The features that will be extracted from the training images. Note that the features are extracted before warping the images to the reference shape. If *list*, then it must define a feature function per scale. Please refer to *menpo.feature* for a list of potential features.

• **patch_features** (*closure* or *list* of *closure*, optional) – The features that will be extracted from the patches of the training images. Note that, as opposed to *holistic_features*, these features are extracted after extracting the patches. If *list*, then it must define a feature function per scale. Please refer to *menpo.feature* and *menpofit.feature* for a list of potential features.

• **patch_shape** (*int*, *int*) or *list* of (*int*, *int*), optional) – The shape of the patches to be extracted. If a *list* is provided, then it defines a patch shape per scale.

• **scales** (*float* or *tuple* of *float*, optional) – The scale value of each scale. They must be provided in ascending order, i.e. from lowest to highest scale. If *float*, then a single scale is assumed.

• **n_iterations** (*int* or *list* of *int*, optional) – The number of iterations (cascades) of each level. If *list*, it must specify a value per scale. If *int*, then it defines the total number of iterations (cascades) over all scales.

• **n_perturbations** (*int*, optional) – The number of perturbations to be generated from each of the bounding boxes using *perturb_from_gt_bounding_box*.

• **perturb_from_gt_bounding_box** (*callable*, optional) – The function that will be used to generate the perturbations from each of the bounding boxes.

• **batch_size** (*int* or *None*, optional) – If an *int* is provided, then the training is performed in an incremental fashion on image batches of size equal to the provided value. If *None*, then the training is performed directly on the all the images.

• **verbose** (*bool*, optional) – If *True*, then the progress of the training will be printed.

---

**References**

**fit_from_bb** (*image*, *bounding_box*, *max_iters=20*, *gt_shape=None*, *return_costs=False*, **kwargs)

Fits the multi-scale fitter to an image given an initial bounding box.

**Parameters**

• **image** (*menpo.image.Image* or subclass) – The image to be fitted.

• **bounding_box** (*menpo.shape.PointDirectedGraph*) – The initial bounding box from which the fitting procedure will start. Note that the bounding box is used in order to align the model’s reference shape.

• **max_iters** (*int* or *list* of *int*, optional) – The maximum number of iterations. If *int*, then it specifies the maximum number of iterations over all scales. If *list* of *int*, then specifies the maximum number of iterations per scale.

• **gt_shape** (*menpo.shape.PointCloud*, optional) – The ground truth shape associated to the image.
• `return_costs` (bool, optional) – If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned `fitting_result`. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.

• `kwargs` (dict, optional) – Additional keyword arguments that can be passed to specific implementations.

Returns `fitting_result` (`MultiScaleNonParametricIterativeResult` or subclass) – The multi-scale fitting result containing the result of the fitting procedure.

`fit_from_shape(image, initial_shape, max_iters=20, gt_shape=None, return_costs=False, **kwargs)`

Fits the multi-scale fitter to an image given an initial shape.

Parameters

• `image` (menpo.image.Image or subclass) – The image to be fitted.

• `initial_shape` (menpo.shape.PointCloud) – The initial shape estimate from which the fitting procedure will start.

• `max_iters` (int or list of int, optional) – The maximum number of iterations. If int, then it specifies the maximum number of iterations over all scales. If list of int, then specifies the maximum number of iterations per scale.

• `gt_shape` (menpo.shape.PointCloud, optional) – The ground truth shape associated to the image.

• `return_costs` (bool, optional) – If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned `fitting_result`. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.

• `kwargs` (dict, optional) – Additional keyword arguments that can be passed to specific implementations.

Returns `fitting_result` (`MultiScaleNonParametricIterativeResult` or subclass) – The multi-scale fitting result containing the result of the fitting procedure.

`increment(images, group=None, bounding_box_group_glob=None, verbose=False, batch_size=None)`

Method to increment the trained SDM with a new set of training images.

Parameters

• `images` (list of menpo.image.Image) – The list of training images.

• `group` (str or None, optional) – The landmark group that corresponds to the ground truth shape of each image. If None and the images only have a single landmark group, then that is the one that will be used. Note that all the training images need to have the specified landmark group.

• `bounding_box_group_glob` (glob or None, optional) – Glob that defines the bounding boxes to be used for training. If None, then the bounding boxes of the ground truth shapes are used.

• `verbose` (bool, optional) – If True, then the progress of training will be printed.
• **batch_size** *(int or None, optional)* – If an *int* is provided, then the training is performed in an incremental fashion on image batches of size equal to the provided value. If *None*, then the training is performed directly on all the images.

**property holistic_features**
The features that are extracted from the input image at each scale in ascending order, i.e. from lowest to highest scale.

*Type* list of closure

**property n_scales**
Returns the number of scales.

*Type* int

**property reference_shape**
The reference shape that is used to normalise the size of an input image so that the scale of its initial fitting shape matches the scale of this reference shape.

*Type* menpo.shape.PointCloud

**property scales**
The scale value of each scale in ascending order, i.e. from lowest to highest scale.

*Type* list of int or float

**Pre-defined Models**
Models with pre-defined algorithms that are commonly-used in literature.

**SDM**

menpofit.sdm.SDM*(images, group=None, bounding_box_group_glob=None, reference_shape=None, diagonal=None, holistic_features=<function no_op>, patch_features=<function no_op>, patch_shape=(17, 17), scales=(0.5, 1.0), n_iterations=3, n_perturbations=30, perturb_from_gt_bounding_box=<function noisy_shape_from_bounding_box>, batch_size=None, verbose=False)*

Class for training a non-parametric multi-scale Supervised Descent model using *NonParametricNewton*.

**Parameters**

• **images** *(list of menpo.image.Image)* – The list of training images.

• **group** *(str or None, optional)* – The landmark group that corresponds to the ground truth shape of each image. If *None* and the images only have a single landmark group, then that is the one that will be used. Note that all the training images need to have the specified landmark group.

• **bounding_box_group_glob** *(glob or None, optional)* – Glob that defines the bounding boxes to be used for training. If *None*, then the bounding boxes of the ground truth shapes are used.

• **reference_shape** *(menpo.shape.PointCloud or None, optional)* – The reference shape that will be used for normalising the size of the training images. The normalization is performed by rescaling all the training images so that the scale of their ground truth shapes matches the scale of the reference shape. Note that the reference shape is rescaled with respect to the *diagonal* before performing the normalisation. If *None*, then the mean shape will be used.
• **diagonal** *(int or None, optional)* – This parameter is used to rescale the reference shape so that the diagonal of its bounding box matches the provided value. In other words, this parameter controls the size of the model at the highest scale. If *None*, then the reference shape does not get rescaled.

• **holistic_features** *(closure or list of closure, optional)* – The features that will be extracted from the training images. Note that the features are extracted before warping the images to the reference shape. If *list*, then it must define a feature function per scale. Please refer to *menpo.feature* for a list of potential features.

• **patch_features** *(closure or list of closure, optional)* – The features that will be extracted from the patches of the training images. Note that, as opposed to *holistic_features*, these features are extracted after extracting the patches. If *list*, then it must define a feature function per scale. Please refer to *menpo.feature* and *menpofit.feature* for a list of potential features.

• **patch_shape** *(int, int) or list of (int, int), optional)* – The shape of the patches to be extracted. If a *list* is provided, then it defines a patch shape per scale.

• **scales** *(float or tuple of float, optional)* – The scale value of each scale. They must provided in ascending order, i.e. from lowest to highest scale. If *float*, then a single scale is assumed.

• **n_iterations** *(int or list of int, optional)* – The number of iterations (cascades) of each level. If *list*, it must specify a value per scale. If *int*, then it defines the total number of iterations (cascades) over all scales.

• **n_perturbations** *(int, optional)* – The number of perturbations to be generated from each of the bounding boxes using *perturb_from_gt_bounding_box*.

• **perturb_from_gt_bounding_box** *(callable, optional)* – The function that will be used to generate the perturbations from each of the bounding boxes.

• **batch_size** *(int or None, optional)* – If an *int* is provided, then the training is performed in an incremental fashion on image batches of size equal to the provided value. If *None*, then the training is performed directly on all the images.

• **verbose** *(bool, optional)* – If *True*, then the progress of the training will be printed.

---

**References**

**RegularizedSDM**

```python
class menpofit.sdm.RegularizedSDM(images, group=None, bounding_box_group_glob=None, alpha=0.0001, reference_shape=None, diagonal=None, holistic_features=<function no_op>, patch_features=<function no_op>, patch_shape=(17, 17), scales=(0.5, 1.0), n_iterations=6, n_perturbations=30, perturb_from_gt_bounding_box=<function noisy_shape_from_bounding_box>, batch_size=None, verbose=False)
```

Bases: SupervisedDescentFitter

Class for training a non-parametric multi-scale Supervised Descent model using *NonParametricNewton* with regularization.

Parameters
• **images** (*list of menpo.image.Image*) – The list of training images.

• **group** (*str or None, optional*) – The landmark group that corresponds to the ground truth shape of each image. If *None* and the images only have a single landmark group, then that is the one that will be used. Note that all the training images need to have the specified landmark group.

• **bounding_box_group_glob** (*glob or None, optional*) – Glob that defines the bounding boxes to be used for training. If *None*, then the bounding boxes of the ground truth shapes are used.

• **alpha** (*float, optional*) – The regression regularization parameter.

• **reference_shape** (*menpo.shape.PointCloud or None, optional*) – The reference shape that will be used for normalising the size of the training images. The normalization is performed by rescaling all the training images so that the scale of their ground truth shapes matches the scale of the reference shape. Note that the reference shape is rescaled with respect to the *diagonal* before performing the normalisation. If *None*, then the mean shape will be used.

• **diagonal** (*int or None, optional*) – This parameter is used to rescale the reference shape so that the diagonal of its bounding box matches the provided value. In other words, this parameter controls the size of the model at the highest scale. If *None*, then the reference shape does not get rescaled.

• **holistic_features** (*closure or list of closure, optional*) – The features that will be extracted from the training images. Note that the features are extracted before warping the images to the reference shape. If *list*, then it must define a feature function per scale. Please refer to *menpo.feature* for a list of potential features.

• **patch_features** (*closure or list of closure, optional*) – The features that will be extracted from the patches of the training images. Note that, as opposed to *holistic_features*, these features are extracted after extracting the patches. If *list*, then it must define a feature function per scale. Please refer to *menpo.feature* and *menpofit.feature* for a list of potential features.

• **patch_shape** (*(*int, int*) or list of (*int, int*), optional*) – The shape of the patches to be extracted. If a *list* is provided, then it defines a patch shape per scale.

• **scales** (*float or tuple of float, optional*) – The scale value of each scale. They must provided in ascending order, i.e. from lowest to highest scale. If *float*, then a single scale is assumed.

• **n_iterations** (*int or list of int, optional*) – The number of iterations (cascades) of each level. If *list*, it must specify a value per scale. If *int*, then it defines the total number of iterations (cascades) over all scales.

• **n_perturbations** (*int, optional*) – The number of perturbations to be generated from each of the bounding boxes using *perturb_from_gt_bounding_box*.

• **perturb_from_gt_bounding_box** (*callable, optional*) – The function that will be used to generate the perturbations from each of the bounding boxes.

• **batch_size** (*int or None, optional*) – If an *int* is provided, then the training is performed in an incremental fashion on image batches of size equal to the provided value. If *None*, then the training is performed directly on all the images.

• **verbose** (*bool, optional*) – If true, then the progress of the training will be printed.
**References**

**fit_from_bb** (*image, bounding_box, max_iters=20, gt_shape=None, return_costs=False, **kwargs*)
Fits the multi-scale fitter to an image given an initial bounding box.

**Parameters**

- **image** (*menpo.image.Image* or subclass) – The image to be fitted.
- **bounding_box** (*menpo.shape.PointDirectedGraph*) – The initial bounding box from which the fitting procedure will start. Note that the bounding box is used in order to align the model’s reference shape.
- **max_iters** (*int* or *list of int*, optional) – The maximum number of iterations. If *int*, then it specifies the maximum number of iterations over all scales. If *list* of *int*, then specifies the maximum number of iterations per scale.
- **gt_shape** (*menpo.shape.PointCloud*, optional) – The ground truth shape associated to the image.
- **return_costs** (*bool*, optional) – If *True*, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.
- **kwargs** (*dict*, optional) – Additional keyword arguments that can be passed to specific implementations.

**Returns**
fitting_result (*MultiScaleNonParametricIterativeResult* or subclass) – The multi-scale fitting result containing the result of the fitting procedure.

**fit_from_shape** (*image, initial_shape, max_iters=20, gt_shape=None, return_costs=False, **kwargs*)
Fits the multi-scale fitter to an image given an initial shape.

**Parameters**

- **image** (*menpo.image.Image* or subclass) – The image to be fitted.
- **initial_shape** (*menpo.shape.PointCloud*) – The initial shape estimate from which the fitting procedure will start.
- **max_iters** (*int* or *list of int*, optional) – The maximum number of iterations. If *int*, then it specifies the maximum number of iterations over all scales. If *list* of *int*, then specifies the maximum number of iterations per scale.
- **gt_shape** (*menpo.shape.PointCloud*, optional) – The ground truth shape associated to the image.
- **return_costs** (*bool*, optional) – If *True*, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.
- **kwargs** (*dict*, optional) – Additional keyword arguments that can be passed to specific implementations.

**Returns**
fitting_result (*MultiScaleNonParametricIterativeResult* or subclass) – The multi-scale fitting result containing the result of the fitting procedure.
increment(images, group=None, bounding_box_group_glob=None, verbose=False, batch_size=None)

Method to increment the trained SDM with a new set of training images.

**Parameters**

- **images** *(list of menpo.image.Image)* – The list of training images.
- **group** *(str or None, optional)* – The landmark group that corresponds to the ground truth shape of each image. If None and the images only have a single landmark group, then that is the one that will be used. Note that all the training images need to have the specified landmark group.
- **bounding_box_group_glob** *(glob or None, optional)* – Globs that defines the bounding boxes to be used for training. If None, then the bounding boxes of the ground truth shapes are used.
- **verbose** *(bool, optional)* – If True, then the progress of training will be printed.
- **batch_size** *(int or None, optional)* – If an int is provided, then the training is performed in an incremental fashion on image batches of size equal to the provided value. If None, then the training is performed directly on the all the images.

**property** holistic_features

The features that are extracted from the input image at each scale in ascending order, i.e. from lowest to highest scale.

**Type** *list of closure*

**property** n_scales

Returns the number of scales.

**Type** *int*

**property** reference_shape

The reference shape that is used to normalise the size of an input image so that the scale of its initial fitting shape matches the scale of this reference shape.

**Type** *menpo.shape.PointCloud*

**property** scales

The scale value of each scale in ascending order, i.e. from lowest to highest scale.

**Type** *list of int or float*

**Non-Parametric Algorithms**

The cascaded regression of these algorithms is performed between landmark coordinates and image-based features.

**NonParametricNewton**

class menpo.fit.sdm.NonParametricNewton (patch_features=<function no_op>, patch_shape=(17, 17), n_iterations=3, compute_error=<function euclidean_bb_normalised_error>, alpha=0, bias=True)

**Bases**: NonParametricSDAlgorithm

Class for training a non-parametric cascaded-regression algorithm using Incremental Regularized Linear Regression (IRLRegression).
Parameters

- **patch_features** *(callable, optional)* – The features to be extracted from the patches of an image.
- **patch_shape** *((int, int), optional)* – The shape of the extracted patches.
- **n_iterations** *(int, optional)* – The number of iterations (cascades).
- **compute_error** *(callable, optional)* – The function to be used for computing the fitting error when training each cascade.
- **alpha** *(float, optional)* – The regularization parameter.
- **bias** *(bool, optional)* – Flag that controls whether to use a bias term.

**increment** *(images, gt_shapes, current_shapes, prefix=", verbose=False)*
Method to increment the model with the set of current shapes.

Parameters

- **images** *(list of menpo.image.Image)* – The list of training images.
- **gt_shapes** *(list of menpo.shape.PointCloud)* – The list of ground truth shapes that correspond to the images.
- **current_shapes** *(list of menpo.shape.PointCloud)* – The list of current shapes that correspond to the images.
- **prefix** *(str, optional)* – The prefix to use when printing information.
- **verbose** *(bool, optional)* – If True, then information is printed during training.

Returns **current_shapes** *(list of menpo.shape.PointCloud)* – The list of current shapes that correspond to the images.

**run** *(image, initial_shape, gt_shape=None, return_costs=False, **kwargs)*
Run the algorithm to an image given an initial shape.

Parameters

- **image** *(menpo.image.Image or subclass)* – The image to be fitted.
- **initial_shape** *(menpo.shape.PointCloud)* – The initial shape from which the fitting procedure will start.
- **gt_shape** *(class : menpo.shape.PointCloud or None, optional)* – The ground truth shape associated to the image.
- **return_costs** *(bool, optional)* – If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that this argument currently has no effect and will raise a warning if set to ‘‘True’’. This is because it is not possible to evaluate the cost function of this algorithm.

Returns **fitting_result** *(NonParametricIterativeResult)* – The result of the fitting procedure.

**train** *(images, gt_shapes, current_shapes, prefix=", verbose=False)*
Method to train the model given a set of initial shapes.

Parameters

- **images** *(list of menpo.image.Image)* – The list of training images.
• **gt_shapes** (list of menpo.shape.PointCloud) – The list of ground truth shapes that correspond to the images.

• **current_shapes** (list of menpo.shape.PointCloud) – The list of current shapes that correspond to the images, which will be used as initial shapes.

• **prefix** (str, optional) – The prefix to use when printing information.

• **verbose** (bool, optional) – If True, then information is printed during training.

**Returns** current_shapes (list of menpo.shape.PointCloud) – The list of current shapes that correspond to the images.

---

### NonParametricGaussNewton

```python
class menpofit.sdm.NonParametricGaussNewton(patch_features=<function no_op>,
patch_shape=(17, 17), n_iterations=3,
compute_error=<function euclidean_bb_normalised_error>,
alpha=0, bias=True, alpha2=0)
```

**Bases:** NonParametricSDAlgorithm

Class for training a non-parametric cascaded-regression algorithm using Indirect Incremental Regularized Linear Regression (IIRLRegression).

**Parameters**

• **patch_features** (callable, optional) – The features to be extracted from the patches of an image.

• **patch_shape** ((int, int), optional) – The shape of the extracted patches.

• **n_iterations** (int, optional) – The number of iterations (cascades).

• **compute_error** (callable, optional) – The function to be used for computing the fitting error when training each cascade.

• **alpha** (float, optional) – The regularization parameter.

• **bias** (bool, optional) – Flag that controls whether to use a bias term.

• **alpha2** (float, optional) – The regularization parameter of the Hessian matrix.

**increment** (images, gt_shapes, current_shapes, prefix=", verbose=False)

Method to increment the model with the set of current shapes.

**Parameters**

• **images** (list of menpo.image.Image) – The list of training images.

• **gt_shapes** (list of menpo.shape.PointCloud) – The list of ground truth shapes that correspond to the images.

• **current_shapes** (list of menpo.shape.PointCloud) – The list of current shapes that correspond to the images.

• **prefix** (str, optional) – The prefix to use when printing information.

• **verbose** (bool, optional) – If True, then information is printed during training.

**Returns** current_shapes (list of menpo.shape.PointCloud) – The list of current shapes that correspond to the images.
run\((image, \text{initial\_shape}, \text{gt\_shape}=None, \text{return\_costs}=False, **\text{kwargs})\)

Run the algorithm to an image given an initial shape.

**Parameters**

- **image** (\text{menpo.\text{image}.\text{Image} or subclass}) – The image to be fitted.
- **initial\_shape** (\text{menpo.\text{shape}.\text{PointCloud}}) – The initial shape from which the fitting procedure will start.
- **gt\_shape** (class : \text{menpo.\text{shape}.\text{PointCloud}} or None, optional) – The ground truth shape associated to the image.
- **return\_costs** (bool, optional) – If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting\_result. Note that this argument currently has no effect and will raise a warning if set to ‘True’. This is because it is not possible to evaluate the cost function of this algorithm.

**Returns** fitting\_result (\text{NonParametricIterativeResult}) – The result of the fitting procedure.

train\((\text{images}, \text{gt\_shapes}, \text{current\_shapes}, \text{prefix}='', \text{verbose}=False)\)

Method to train the model given a set of initial shapes.

**Parameters**

- **images** (list of \text{menpo.\text{image}.\text{Image}}) – The list of training images.
- **gt\_shapes** (list of \text{menpo.\text{shape}.\text{PointCloud}}) – The list of ground truth shapes that correspond to the images.
- **current\_shapes** (list of \text{menpo.\text{shape}.\text{PointCloud}}) – The list of current shapes that correspond to the images, which will be used as initial shapes.
- **prefix** (str, optional) – The prefix to use when printing information.
- **verbose** (bool, optional) – If True, then information is printed during training.

**Returns** current\_shapes (list of \text{menpo.\text{shape}.\text{PointCloud}}) – The list of current shapes that correspond to the images.

**NonParametricPCRRegression**

class menpofit.sdm.NonParametricPCRRegression\((\text{patch\_features}=<\text{function no\_op}>, \text{patch\_shape}=(17, 17), \text{n\_iterations}=3, \text{compute\_error}=<\text{function euclidean\_bb\_normalised\_error}>, \text{variance}=None, \text{bias}=True)\)

Bases: NonParametricSDAlgorithm

Class for training a non-parametric cascaded-regression algorithm using Principal Component Regression (PCRRegression).

**Parameters**

- **patch\_features** (callable, optional) – The features to be extracted from the patches of an image.
- **patch\_shape** (\(\text{(int, int)}\), optional) – The shape of the extracted patches.
- **n\_iterations** (\text{int}, optional) – The number of iterations (cascades).
• **compute_error** (*callable*, optional) – The function to be used for computing the fitting error when training each cascade.

• **variance** (*float* or *None*, optional) – The SVD variance.

• **bias** (*bool*, optional) – Flag that controls whether to use a bias term.

**Method**

**increment** (*images, gt_shapes, current_shapes, prefix=", verbose=False*)

Method to increment the model with the set of current shapes.

**Parameters**

- **images** (*list of menpo.image.Image*) – The list of training images.

- **gt_shapes** (*list of menpo.shape.PointCloud*) – The list of ground truth shapes that correspond to the images.

- **current_shapes** (*list of menpo.shape.PointCloud*) – The list of current shapes that correspond to the images.

- **prefix** (*str*, optional) – The prefix to use when printing information.

- **verbose** (*bool*, optional) – If True, then information is printed during training.

**Returns**

**current_shapes** (*list of menpo.shape.PointCloud*) – The list of current shapes that correspond to the images.

**Method**

**run** (*image, initial_shape=None, return_costs=False, **kwargs*)

Run the algorithm to an image given an initial shape.

**Parameters**

- **image** (*menpo.image.Image* or subclass) – The image to be fitted.

- **initial_shape** (*menpo.shape.PointCloud*) – The initial shape from which the fitting procedure will start.

- **gt_shape** (*class : menpo.shape.PointCloud* or *None*, optional) – The ground truth shape associated to the image.

- **return_costs** (*bool*, optional) – If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned *fitting_result*. Note that this argument currently has no effect and will raise a warning if set to "True". This is because it is not possible to evaluate the cost function of this algorithm.

**Returns**

**fitting_result** (*NonParametricIterativeResult*) – The result of the fitting procedure.

**Method**

**train** (*images, gt_shapes, current_shapes, prefix=", verbose=False*)

Method to train the model given a set of initial shapes.

**Parameters**

- **images** (*list of menpo.image.Image*) – The list of training images.

- **gt_shapes** (*list of menpo.shape.PointCloud*) – The list of ground truth shapes that correspond to the images.

- **current_shapes** (*list of menpo.shape.PointCloud*) – The list of current shapes that correspond to the images, which will be used as initial shapes.

- **prefix** (*str*, optional) – The prefix to use when printing information.

- **verbose** (*bool*, optional) – If True, then information is printed during training.
Returns current_shapes (list of menpo.shape.PointCloud) – The list of current shapes that correspond to the images.

NonParametricOptimalRegression

class menpo.fit.sdm.NonParametricOptimalRegression (patch_features=<function no_op>, patch_shape=(17, 17), n_iterations=3, compute_error=<function euclidean_bb_normalised_error>, variance=None, bias=True)

Bases: NonParametricSDAlgorithm

Class for training a non-parametric cascaded-regression algorithm using Multivariate Linear Regression with optimal reconstructions (OptimalLinearRegression).

Parameters

• patch_features (callable, optional) – The features to be extracted from the patches of an image.
• patch_shape ((int, int), optional) – The shape of the extracted patches.
• n_iterations (int, optional) – The number of iterations (cascades).
• compute_error (callable, optional) – The function to be used for computing the fitting error when training each cascade.
• variance (float or None, optional) – The SVD variance.
• bias (bool, optional) – Flag that controls whether to use a bias term.

increment (images, gt_shapes, current_shapes, prefix=", verbose=False)

Method to increment the model with the set of current shapes.

Parameters

• images (list of menpo.image.Image) – The list of training images.
• gt_shapes (list of menpo.shape.PointCloud) – The list of ground truth shapes that correspond to the images.
• current_shapes (list of menpo.shape.PointCloud) – The list of current shapes that correspond to the images.
• prefix (str, optional) – The prefix to use when printing information.
• verbose (bool, optional) – If True, then information is printed during training.

Returns current_shapes (list of menpo.shape.PointCloud) – The list of current shapes that correspond to the images.

run (image, initial_shape, gt_shape=None, return_costs=False, **kwargs)

Run the algorithm to an image given an initial shape.

Parameters

• image (menpo.image.Image or subclass) – The image to be fitted.
• initial_shape (menpo.shape.PointCloud) – The initial shape from which the fitting procedure will start.
• gt_shape (class : menpo.shape.PointCloud or None, optional) – The ground truth shape associated to the image.

2.1. Deformable Models
• **return_costs** *(bool, optional)* – If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that this argument currently has no effect and will raise a warning if set to ‘‘True‘. This is because it is not possible to evaluate the cost function of this algorithm.

**Returns**

 fitting_result *(NonParametricIterativeResult)* – The result of the fitting procedure.

**train** *(images, gt_shapes, current_shapes, prefix=”, verbose=False)*

 Method to train the model given a set of initial shapes.

**Parameters**

• **images** *(list of menpo.image.Image)* – The list of training images.

• **gt_shapes** *(list of menpo.shape.PointCloud)* – The list of ground truth shapes that correspond to the images.

• **current_shapes** *(list of menpo.shape.PointCloud)* – The list of current shapes that correspond to the images, which will be used as initial shapes.

• **prefix** *(str, optional)* – The prefix to use when printing information.

• **verbose** *(bool, optional)* – If True, then information is printed during training.

**Returns**

 current_shapes *(list of menpo.shape.PointCloud)* – The list of current shapes that correspond to the images.

### NonParametricOPPRegression

**class** menpofit.sdm.NonParametricOPPRegression *(patch_features=<function no_op>, patch_shape=(17, 17), n_iterations=3, compute_error=<function euclidean_bb_normalised_error>, bias=True)*

**Bases:** NonParametricSDAlgorithm

Class for training a non-parametric cascaded-regression algorithm using Multivariate Linear Regression with Orthogonal Procrustes Problem reconstructions (**OPPRegression**).

**Parameters**

• **patch_features** *(callable, optional)* – The features to be extracted from the patches of an image.

• **patch_shape** *(int, int, optional)* – The shape of the extracted patches.

• **n_iterations** *(int, optional)* – The number of iterations (cascades).

• **compute_error** *(callable, optional)* – The function to be used for computing the fitting error when training each cascade.

• **bias** *(bool, optional)* – Flag that controls whether to use a bias term.

**increment** *(images, gt_shapes, current_shapes, prefix=”, verbose=False)*

 Method to increment the model with the set of current shapes.

**Parameters**

• **images** *(list of menpo.image.Image)* – The list of training images.
- **gt_shapes** (list of menpo.shape.PointCloud) – The list of ground truth shapes that correspond to the images.
- **current_shapes** (list of menpo.shape.PointCloud) – The list of current shapes that correspond to the images.
- **prefix** (str, optional) – The prefix to use when printing information.
- **verbose** (bool, optional) – If True, then information is printed during training.

Returns **current_shapes** (list of menpo.shape.PointCloud) – The list of current shapes that correspond to the images.

### run(image, initial_shape, gt_shape=None, return_costs=False, **kwargs)

Run the algorithm to an image given an initial shape.

**Parameters**

- **image** (menpo.image.Image or subclass) – The image to be fitted.
- **initial_shape** (menpo.shape.PointCloud) – The initial shape from which the fitting procedure will start.
- **gt_shape** (class : menpo.shape.PointCloud or None, optional) – The ground truth shape associated to the image.
- **return_costs** (bool, optional) – If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that this argument currently has no effect and will raise a warning if set to ‘True’. This is because it is not possible to evaluate the cost function of this algorithm.

Returns **fitting_result** (NonParametricIterativeResult) – The result of the fitting procedure.

### train(images, gt_shapes, current_shapes, prefix='', verbose=False)

Method to train the model given a set of initial shapes.

**Parameters**

- **images** (list of menpo.image.Image) – The list of training images.
- **gt_shapes** (list of menpo.shape.PointCloud) – The list of ground truth shapes that correspond to the images.
- **current_shapes** (list of menpo.shape.PointCloud) – The list of current shapes that correspond to the images, which will be used as initial shapes.
- **prefix** (str, optional) – The prefix to use when printing information.
- **verbose** (bool, optional) – If True, then information is printed during training.

Returns **current_shapes** (list of menpo.shape.PointCloud) – The list of current shapes that correspond to the images.

### Parametric Shape Algorithms

The cascaded regression of these algorithms is performed between the parameters of a statistical shape model and image-based features.
**ParametricShapeNewton**

```python
class menpo.sdm.ParametricShapeNewton(
    patch_features=<function no_op>,
    patch_shape=(17, 17),
    n_iterations=3,
    shape_model_cls=<class 'menpo.fit.modelinstance.OrthoPDM'>,
    compute_error=<function euclidean_bb_normalised_error>,
    alpha=0,
    bias=True)
```

**Bases:** ParametricShapeSDAlgorithm

Class for training a cascaded-regression algorithm that employs a parametric shape model using Incremental Regularized Linear Regression (IRLRegression).

**Parameters**

- `patch_features` *(callable, optional)* – The features to be extracted from the patches of an image.
- `patch_shape` *(int, int, optional)* – The shape of the extracted patches.
- `n_iterations` *(int, optional)* – The number of iterations (cascades).
- `shape_model_cls` *(subclass of PDM, optional)* – The class to be used for building the shape model. The most common choice is OrthoPDM.
- `compute_error` *(callable, optional)* – The function to be used for computing the fitting error when training each cascade.
- `alpha` *(float, optional)* – The regularization parameter.
- `bias` *(bool, optional)* – Flag that controls whether to use a bias term.

**increment** *(images, gt_shapes, current_shapes, prefix=",", verbose=False)*

Method to increment the model with the set of current shapes.

**Parameters**

- `images` *(list of menpo.image.Image)* – The list of training images.
- `gt_shapes` *(list of menpo.shape.PointCloud)* – The list of ground truth shapes that correspond to the images.
- `current_shapes` *(list of menpo.shape.PointCloud)* – The list of current shapes that correspond to the images.
- `prefix` *(str, optional)* – The prefix to use when printing information.
- `verbose` *(bool, optional)* – If True, then information is printed during training.

**Returns** current_shapes *(list of menpo.shape.PointCloud)* – The list of current shapes that correspond to the images.

**run** *(image, initial_shape, gt_shape=None, return_costs=False, **kwargs)*

Run the algorithm to an image given an initial shape.

**Parameters**

- `image` *(menpo.image.Image or subclass)* – The image to be fitted.
- `initial_shape` *(menpo.shape.PointCloud)* – The initial shape from which the fitting procedure will start.
- `gt_shape` *(menpo.shape.PointCloud or None, optional)* – The ground truth shape associated to the image.
• **return_costs** *(bool, optional)* – If `True`, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned `fitting_result`. Note that this argument currently has no effect and will raise a warning if set to `‘True’`. This is because it is not possible to evaluate the cost function of this algorithm.

**Returns fitting_result** *(ParametricIterativeResult)* – The result of the fitting procedure.

**train** *(images, gt_shapes, current_shapes, prefix='', verbose=False)*

Method to train the model given a set of initial shapes.

**Parameters**

• **images** *(list of menpo.image.Image)* – The list of training images.

• **gt_shapes** *(list of menpo.shape.PointCloud)* – The list of ground truth shapes that correspond to the images.

• **current_shapes** *(list of menpo.shape.PointCloud)* – The list of current shapes that correspond to the images, which will be used as initial shapes.

• **prefix** *(str, optional)* – The prefix to use when printing information.

• **verbose** *(bool, optional)* – If `True`, then information is printed during training.

**Returns current_shapes** *(list of menpo.shape.PointCloud)* – The list of current shapes that correspond to the images.

---

**ParametricShapeGaussNewton**

class menpo-fit.sdm.ParametricShapeGaussNewton *(patch_features=<function no_op>, patch_shape=(17, 17), n_iterations=3, shape_model_cls=<class 'menpo-fit.modelinstance.OrthoPDM'>, compute_error=<function euclidean_bb_normalised_error>, alpha=0, bias=True, alpha2=0)

**Bases:** ParametricShapeSDAlgorithm

Class for training a cascaded-regression algorithm that employs a parametric shape model using Indirect Incremental Regularized Linear Regression (*IIRLRegression*).

**Parameters**

• **patch_features** *(callable, optional)* – The features to be extracted from the patches of an image.

• **patch_shape** *((int, int), optional)* – The shape of the extracted patches.

• **n_iterations** *(int, optional)* – The number of iterations (cascades).

• **shape_model_cls** *(subclass of PDM, optional)* – The class to be used for building the shape model. The most common choice is `OrthoPDM`.

• **compute_error** *(callable, optional)* – The function to be used for computing the fitting error when training each cascade.

• **alpha** *(float, optional)* – The regularization parameter.

• **bias** *(bool, optional)* – Flag that controls whether to use a bias term.

• **alpha2** *(float, optional)* – The regularization parameter of the Hessian matrix.
**increment** *(images, gt_shapes, current_shapes, prefix='', verbose=False)*

Method to increment the model with the set of current shapes.

**Parameters**

- **images** *(list of menpo.image.Image)* – The list of training images.
- **gt_shapes** *(list of menpo.shape.PointCloud)* – The list of ground truth shapes that correspond to the images.
- **current_shapes** *(list of menpo.shape.PointCloud)* – The list of current shapes that correspond to the images.
- **prefix** *(str, optional)* – The prefix to use when printing information.
- **verbose** *(bool, optional)* – If True, then information is printed during training.

**Returns** **current_shapes** *(list of menpo.shape.PointCloud)* – The list of current shapes that correspond to the images.

**run** *(image, initial_shape, gt_shape=None, return_costs=False, **kwargs)*

Run the algorithm to an image given an initial shape.

**Parameters**

- **image** *(menpo.image.Image or subclass)* – The image to be fitted.
- **initial_shape** *(menpo.shape.PointCloud)* – The initial shape from which the fitting procedure will start.
- **gt_shape** *(menpo.shape.PointCloud or None, optional)* – The ground truth shape associated to the image.
- **return_costs** *(bool, optional)* – If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that this argument currently has no effect and will raise a warning if set to ‘‘True’’. This is because it is not possible to evaluate the cost function of this algorithm.

**Returns** **fitting_result** *(ParametricIterativeResult)* – The result of the fitting procedure.

**train** *(images, gt_shapes, current_shapes, prefix='', verbose=False)*

Method to train the model given a set of initial shapes.

**Parameters**

- **images** *(list of menpo.image.Image)* – The list of training images.
- **gt_shapes** *(list of menpo.shape.PointCloud)* – The list of ground truth shapes that correspond to the images.
- **current_shapes** *(list of menpo.shape.PointCloud)* – The list of current shapes that correspond to the images, which will be used as initial shapes.
- **prefix** *(str, optional)* – The prefix to use when printing information.
- **verbose** *(bool, optional)* – If True, then information is printed during training.

**Returns** **current_shapes** *(list of menpo.shape.PointCloud)* – The list of current shapes that correspond to the images.
**ParametricShapePCRRegression**

```python
class menpofit.sdm.ParametricShapePCRRegression(
    patch_features=<function no_op>,
    patch_shape=(17, 17), n_iterations=3,
    shape_model_cls=<class 'menpofit.modelinstance.OrthoPDM'>,
    compute_error=<function euclidean_bb_normalised_error>,
    variance=None, bias=True)
```

Bases: ParametricShapeSDAlgorithm

Class for training a cascaded-regression algorithm that employs a parametric shape model using Principal Component Regression (PCRRegression).

**Parameters**

- `patch_features` (*callable, optional*) – The features to be extracted from the patches of an image.
- `patch_shape` (*tuple (int, int), optional*) – The shape of the extracted patches.
- `n_iterations` (*int, optional*) – The number of iterations (cascades).
- `shape_model_cls` (*subclass of PDM, optional*) – The class to be used for building the shape model. The most common choice is OrthoPDM.
- `compute_error` (*callable, optional*) – The function to be used for computing the fitting error when training each cascade.
- `variance` (*float or None, optional*) – The SVD variance.
- `bias` (*bool, optional*) – Flag that controls whether to use a bias term.

**Raises** `ValueError` – variance must be set to a number between 0 and 1

**increment** *(images, gt_shapes, current_shapes, prefix=", verbose=False)*

Method to increment the model with the set of current shapes.

**Parameters**

- `images` (*list of menpo.image.Image*) – The list of training images.
- `gt_shapes` (*list of menpo.shape.PointCloud*) – The list of ground truth shapes that correspond to the images.
- `current_shapes` (*list of menpo.shape.PointCloud*) – The list of current shapes that correspond to the images.
- `prefix` (*str, optional*) – The prefix to use when printing information.
- `verbose` (*bool, optional*) – If True, then information is printed during training.

**Returns** `current_shapes` (*list of menpo.shape.PointCloud*) – The list of current shapes that correspond to the images.

**run** *(image, initial_shape, gt_shape=None, return_costs=False, **kwargs)*

Run the algorithm to an image given an initial shape.

**Parameters**

- `image` (*menpo.image.Image or subclass*) – The image to be fitted.
- `initial_shape` (*menpo.shape.PointCloud*) – The initial shape from which the fitting procedure will start.
MenpoFit Documentation, Release 0.6.1

- **gt_shape** (*menpo.shape.PointCloud* or *None*, optional) – The ground truth shape associated to the image.

- **return_costs** (*bool*, optional) – If *True*, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned *fitting_result*. Note that this argument currently has no effect and will raise a warning if set to ‘‘True’’. This is because it is not possible to evaluate the cost function of this algorithm.

**Returns** *fitting_result* (*ParametricIterativeResult*) – The result of the fitting procedure.

**train** (*images, gt_shapes, current_shapes, prefix=”, verbose=False*)

Method to train the model given a set of initial shapes.

**Parameters**

- **images** (*list of menpo.image.Image*) – The list of training images.

- **gt_shapes** (*list of menpo.shape.PointCloud*) – The list of ground truth shapes that correspond to the images.

- **current_shapes** (*list of menpo.shape.PointCloud*) – The list of current shapes that correspond to the images, which will be used as initial shapes.

- **prefix** (*str*, optional) – The prefix to use when printing information.

- **verbose** (*bool*, optional) – If *True*, then information is printed during training.

**Returns** *current_shapes* (*list of menpo.shape.PointCloud*) – The list of current shapes that correspond to the images.

**ParametricShapeOptimalRegression**

```python
class menpofit.sdm.ParametricShapeOptimalRegression (patch_features=<function no_op>,  
patch_shape=(17, 17),  
n_iterations=3,  
shape_model_cls=<class 'menpo.fit.modelinstance.OrthoPDM'>,  
compute_error=<function euclidean_bb_normalised_error>,  
variance=None, bias=True)
```

**Bases:** *ParametricShapeSDAlgorithm*

Class for training a cascaded-regression algorithm that employs a parametric shape model using Multivariate Linear Regression with optimal reconstructions (*OptimalLinearRegression*).

**Parameters**

- **patch_features** (*callable*, optional) – The features to be extracted from the patches of an image.

- **patch_shape** (*int, int*, optional) – The shape of the extracted patches.

- **n_iterations** (*int*, optional) – The number of iterations (cascades).

- **shape_model_cls** (*subclass of *PDM*, optional) – The class to be used for building the shape model. The most common choice is *OrthoPDM*.

- **compute_error** (*callable*, optional) – The function to be used for computing the fitting error when training each cascade.
MenpoFit Documentation, Release 0.6.1

- **variance** (`float` or `None`, optional) – The SVD variance.
- **bias** (`bool`, optional) – Flag that controls whether to use a bias term.

**increment** (`images`, `gt_shapes`, `current_shapes`, `prefix="", verbose=False`)
Method to increment the model with the set of current shapes.

**Parameters**
- **images** (`list of menpo.image.Image`) – The list of training images.
- **gt_shapes** (`list of menpo.shape.PointCloud`) – The list of ground truth shapes that correspond to the images.
- **current_shapes** (`list of menpo.shape.PointCloud`) – The list of current shapes that correspond to the images.
- **prefix** (`str`, optional) – The prefix to use when printing information.
- **verbose** (`bool`, optional) – If True, then information is printed during training.

**Returns** `current_shapes` (`list of menpo.shape.PointCloud`) – The list of current shapes that correspond to the images.

**run** (`image`, `initial_shape`, `gt_shape=None`, `return_costs=False`, `**kwargs`)
Run the algorithm to an image given an initial shape.

**Parameters**
- **image** (`menpo.image.Image` or subclass) – The image to be fitted.
- **initial_shape** (`menpo.shape.PointCloud`) – The initial shape from which the fitting procedure will start.
- **gt_shape** (`menpo.shape.PointCloud` or `None`, optional) – The ground truth shape associated to the image.
- **return_costs** (`bool`, optional) – If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that this argument currently has no effect and will raise a warning if set to ‘‘True‘‘. This is because it is not possible to evaluate the cost function of this algorithm.

**Returns** `fitting_result` (`ParametricIterativeResult`) – The result of the fitting procedure.

**train** (`images`, `gt_shapes`, `current_shapes`, `prefix="", verbose=False`)
Method to train the model given a set of initial shapes.

**Parameters**
- **images** (`list of menpo.image.Image`) – The list of training images.
- **gt_shapes** (`list of menpo.shape.PointCloud`) – The list of ground truth shapes that correspond to the images.
- **current_shapes** (`list of menpo.shape.PointCloud`) – The list of current shapes that correspond to the images, which will be used as initial shapes.
- **prefix** (`str`, optional) – The prefix to use when printing information.
- **verbose** (`bool`, optional) – If True, then information is printed during training.

**Returns** `current_shapes` (`list of menpo.shape.PointCloud`) – The list of current shapes that correspond to the images.
**ParametricShapeOPPRegression**

```python
class menpo.fit.sdm.ParametricShapeOPPRegression(
    patch_features=<function no_op>,
    patch_shape=(17, 17), n_iterations=3,
    shape_model_cls=<class 'menpo.fit.modelinstance.OrthoPDM'>,
    compute_error=<function euclidean_bb_normalised_error>,
    whiten=False, bias=True)
```

**Bases:** ParametricShapeSDAlgorithm

Class for training a cascaded-regression algorithm that employs a parametric shape model using Multivariate Linear Regression with Orthogonal Procrustes Problem reconstructions (OPPRegression).

**Parameters**

- `patch_features` *(callable, optional)* – The features to be extracted from the patches of an image.
- `patch_shape` *(int, int, optional)* – The shape of the extracted patches.
- `n_iterations` *(int, optional)* – The number of iterations (cascades).
- `shape_model_cls` *(subclass of PDM, optional)* – The class to be used for building the shape model. The most common choice is OrthoPDM.
- `compute_error` *(callable, optional)* – The function to be used for computing the fitting error when training each cascade.
- `whiten` *(bool, optional)* – Whether to use a whitened PCA model.
- `bias` *(bool, optional)* – Flag that controls whether to use a bias term.

**increment** *(images, gt_shapes, current_shapes, prefix=",", verbose=False)*

Method to increment the model with the set of current shapes.

**Parameters**

- `images` *(list of menpo.image.Image)* – The list of training images.
- `gt_shapes` *(list of menpo.shape.PointCloud)* – The list of ground truth shapes that correspond to the images.
- `current_shapes` *(list of menpo.shape.PointCloud)* – The list of current shapes that correspond to the images.
- `prefix` *(str, optional)* – The prefix to use when printing information.
- `verbose` *(bool, optional)* – If True, then information is printed during training.

**Returns** `current_shapes` *(list of menpo.shape.PointCloud)* – The list of current shapes that correspond to the images.

**run** *(image, initial_shape, gt_shape=None, return_costs=False, **kwargs)*

Run the algorithm to an image given an initial shape.

**Parameters**

- `image` *(menpo.image.Image or subclass)* – The image to be fitted.
- `initial_shape` *(menpo.shape.PointCloud)* – The initial shape from which the fitting procedure will start.
- `gt_shape` *(menpo.shape.PointCloud or None, optional)* – The ground truth shape associated to the image.
• **return_costs** *(bool, optional)* – If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that this argument currently has no effect and will raise a warning if set to ‘True’. This is because it is not possible to evaluate the cost function of this algorithm.

    **Returns fitting_result** *(ParametricIterativeResult)* – The result of the fitting procedure.

    **train** *(images, gt_shapes, current_shapes, prefix=”, verbose=False)*

    Method to train the model given a set of initial shapes.

    **Parameters**

    • **images** *(list of menpo.image.Image)* – The list of training images.

    • **gt_shapes** *(list of menpo.shape.PointCloud)* – The list of ground truth shapes that correspond to the images.

    • **current_shapes** *(list of menpo.shape.PointCloud)* – The list of current shapes that correspond to the images, which will be used as initial shapes.

    • **prefix** *(str, optional)* – The prefix to use when printing information.

    • **verbose** *(bool, optional)* – If True, then information is printed during training.

    **Returns current_shapes** *(list of menpo.shape.PointCloud)* – The list of current shapes that correspond to the images.

**Parametric Appearance Algorithms**

The cascaded regression of these algorithms is performed between landmark coordinates and features that are based on a statistical parametric appearance model.

**ParametricAppearanceProjectOutNewton**

    **class** menpofit.sdm.ParametricAppearanceProjectOutNewton *(patch_features=<function no_op>, patch_shape=(17, 17), n_iterations=3, appearance_model_cls=<class ‘menpo.model.pca.PCAVectorModel’>, compute_error=<function euclidean_bb_normalised_error>, alpha=0, bias=True)*

    **Bases:** ParametricAppearanceNewton

    Class for training a cascaded-regression Newton algorithm that employs a parametric appearance model using Incremental Regularized Linear Regression *(IRLRegression)*. The algorithm uses the projected-out appearance vectors as features in the regression.

    **increment** *(images, gt_shapes, current_shapes, prefix=”, verbose=False)*

    Method to increment the model with the set of current shapes.

    **Parameters**

    • **images** *(list of menpo.image.Image)* – The list of training images.
MenpoFit Documentation, Release 0.6.1

- **gt_shapes** (list of menpo.shape.PointCloud) – The list of ground truth shapes that correspond to the images.

- **current_shapes** (list of menpo.shape.PointCloud) – The list of current shapes that correspond to the images.

- **prefix** (str, optional) – The prefix to use when printing information.

- **verbose** (bool, optional) – If True, then information is printed during training.

**Returns** current_shapes (list of menpo.shape.PointCloud) – The list of current shapes that correspond to the images.

**run** (image, initial_shape, gt_shape=None, return_costs=False, **kwargs)

Run the algorithm to an image given an initial shape.

**Parameters**

- **image** (menpo.image.Image or subclass) – The image to be fitted.

- **initial_shape** (menpo.shape.PointCloud) – The initial shape from which the fitting procedure will start.

- **gt_shape** (menpo.shape.PointCloud or None, optional) – The ground truth shape associated to the image.

- **return_costs** (bool, optional) – If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that this argument currently has no effect and will raise a warning if set to ‘True’. This is because it is not possible to evaluate the cost function of this algorithm.

**Returns** fitting_result (NonParametricIterativeResult) – The result of the fitting procedure.

**train** (images, gt_shapes, current_shapes, prefix=", verbose=False)

Method to train the model given a set of initial shapes.

**Parameters**

- **images** (list of menpo.image.Image) – The list of training images.

- **gt_shapes** (list of menpo.shape.PointCloud) – The list of ground truth shapes that correspond to the images.

- **current_shapes** (list of menpo.shape.PointCloud) – The list of current shapes that correspond to the images, which will be used as initial shapes.

- **prefix** (str, optional) – The prefix to use when printing information.

- **verbose** (bool, optional) – If True, then information is printed during training.

**Returns** current_shapes (list of menpo.shape.PointCloud) – The list of current shapes that correspond to the images.
**ParametricAppearanceProjectOutGuassNewton**

```python
class menpo.fit.sdm.ParametricAppearanceProjectOutGuassNewton(
    patch_features=<function no_op>,
    patch_shape=(17, 17),
    n_iterations=3,
    appearance_model_cls=<class 'menpo.model.pca.PCAVectorModel'>,
    compute_error=<function euclidean_bb_normalised_error>,
    alpha=0,
    bias=True,
    alpha2=0)
```

**Bases:** `ParametricAppearanceGaussNewton`

Class for training a cascaded-regression Gauss-Newton algorithm that employs a parametric appearance model using Indirect Incremental Regularized Linear Regression (IIRLRegression). The algorithm uses the projected-out appearance vectors as features in the regression.

**increment** *(images, gt_shapes, current_shapes, prefix=", verbose=False)*

Method to increment the model with the set of current shapes.

**Parameters**

- **images** *(list of menpo.image.Image)* – The list of training images.
- **gt_shapes** *(list of menpo.shape.PointCloud)* – The list of ground truth shapes that correspond to the images.
- **current_shapes** *(list of menpo.shape.PointCloud)* – The list of current shapes that correspond to the images.
- **prefix** *(str, optional)* – The prefix to use when printing information.
- **verbose** *(bool, optional)* – If True, then information is printed during training.

Returns current_shapes *(list of menpo.shape.PointCloud)* – The list of current shapes that correspond to the images.

**run** *(image, initial_shape, gt_shape=None, return_costs=False, **kwargs)*

Run the algorithm to an image given an initial shape.

**Parameters**

- **image** *(menpo.image.Image or subclass)* – The image to be fitted.
- **initial_shape** *(menpo.shape.PointCloud)* – The initial shape from which the fitting procedure will start.
- **gt_shape** *(menpo.shape.PointCloud or None, optional)* – The ground truth shape associated to the image.
- **return_costs** *(bool, optional)* – If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that this argument currently has no effect and will raise a warning if set to `'True'`. This is because it is not possible to evaluate the cost function of this algorithm.
Returns fitting_result (NonParametricIterativeResult) – The result of the fitting procedure.

train(images, gt_shapes, current_shapes, prefix=",", verbose=False)
Method to train the model given a set of initial shapes.

Parameters

- **images** (list of menpo.image.Image) – The list of training images.
- **gt_shapes** (list of menpo.shape.PointCloud) – The list of ground truth shapes that correspond to the images.
- **current_shapes** (list of menpo.shape.PointCloud) – The list of current shapes that correspond to the images.
- **prefix** (str, optional) – The prefix to use when printing information.
- **verbose** (bool, optional) – If True, then information is printed during training.

Returns current_shapes (list of menpo.shape.PointCloud) – The list of current shapes that correspond to the images.

ParametricAppearanceMeanTemplateNewton

class menpofit.sdm.ParametricAppearanceMeanTemplateNewton (patch_features=<function no_op>,
patch_shape=(17, 17), n_iterations=3, appearance_model_cls=<class 'menpo.model.pca.PCAVectorModel'>,
compute_error=<function euclidean_bb_normalised_error>,
alpha=0, bias=True)

Bases: ParametricAppearanceNewton

Class for training a cascaded-regression Newton algorithm that employs a parametric appearance model using Incremental Regularized Linear Regression (IRLRegression). The algorithm uses the centered appearance vectors as features in the regression.

increment(images, gt_shapes, current_shapes, prefix=",", verbose=False)
Method to increment the model with the set of current shapes.

Parameters

- **images** (list of menpo.image.Image) – The list of training images.
- **gt_shapes** (list of menpo.shape.PointCloud) – The list of ground truth shapes that correspond to the images.
- **current_shapes** (list of menpo.shape.PointCloud) – The list of current shapes that correspond to the images.
- **prefix** (str, optional) – The prefix to use when printing information.
- **verbose** (bool, optional) – If True, then information is printed during training.

Returns current_shapes (list of menpo.shape.PointCloud) – The list of current shapes that correspond to the images.
run(image, initial_shape, gt_shape=None, return_costs=False, **kwargs)

Run the algorithm to an image given an initial shape.

Parameters

- **image** (menpo.image.Image or subclass) – The image to be fitted.
- **initial_shape** (menpo.shape.PointCloud) – The initial shape from which the fitting procedure will start.
- **gt_shape** (menpo.shape.PointCloud or None, optional) – The ground truth shape associated to the image.
- **return_costs** (bool, optional) – If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that this argument currently has no effect and will raise a warning if set to ‘True’. This is because it is not possible to evaluate the cost function of this algorithm.

Returns **fitting_result** (NonParametricIterativeResult) – The result of the fitting procedure.

train(images, gt_shapes, current_shapes, prefix="", verbose=False)

Method to train the model given a set of initial shapes.

Parameters

- **images** (list of menpo.image.Image) – The list of training images.
- **gt_shapes** (list of menpo.shape.PointCloud) – The list of ground truth shapes that correspond to the images.
- **current_shapes** (list of menpo.shape.PointCloud) – The list of current shapes that correspond to the images, which will be used as initial shapes.
- **prefix** (str, optional) – The prefix to use when printing information.
- **verbose** (bool, optional) – If True, then information is printed during training.

Returns **current_shapes** (list of menpo.shape.PointCloud) – The list of current shapes that correspond to the images.

ParametricAppearanceMeanTemplateGuassNewton

class menpoit.sdm.ParametricAppearanceMeanTemplateGuassNewton (patch_features=<function no_op>, patch_shape=(17, 17), n_iterations=3, appearance_model_cls=<class 'menpo.model.pca.PCAVectorModel'>, compute_error=<function euclidean_bb_normalised_error>, alpha=0, bias=True, alpha2=0)

Bases: ParametricAppearanceGaussNewton

2.1. Deformable Models
Class for training a cascaded-regression Gauss-Newton algorithm that employs a parametric appearance model using Indirect Incremental Regularized Linear Regression (IIRLRegression). The algorithm uses the centered appearance vectors as features in the regression.

**increment**(images, gt_shapes, current_shapes, prefix=",”, verbose=False)
Method to increment the model with the set of current shapes.

**Parameters**

- **images** (list of menpo.image.Image) – The list of training images.
- **gt_shapes** (list of menpo.shape.PointCloud) – The list of ground truth shapes that correspond to the images.
- **current_shapes** (list of menpo.shape.PointCloud) – The list of current shapes that correspond to the images.
- **prefix** (str, optional) – The prefix to use when printing information.
- **verbose** (bool, optional) – If True, then information is printed during training.

**Returns** current_shapes (list of menpo.shape.PointCloud) – The list of current shapes that correspond to the images.

**run**(image, initial_shape, gt_shape=None, return_costs=False, **kwargs)
Run the algorithm to an image given an initial shape.

**Parameters**

- **image** (menpo.image.Image or subclass) – The image to be fitted.
- **initial_shape** (menpo.shape.PointCloud) – The initial shape from which the fitting procedure will start.
- **gt_shape** (menpo.shape.PointCloud or None, optional) – The ground truth shape associated to the image.
- **return_costs** (bool, optional) – If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that this argument currently has no effect and will raise a warning if set to ‘‘True’’. This is because it is not possible to evaluate the cost function of this algorithm.

**Returns** fitting_result (NonParametricIterativeResult) – The result of the fitting procedure.

**train**(images, gt_shapes, current_shapes, prefix=",”, verbose=False)
Method to train the model given a set of initial shapes.

**Parameters**

- **images** (list of menpo.image.Image) – The list of training images.
- **gt_shapes** (list of menpo.shape.PointCloud) – The list of ground truth shapes that correspond to the images.
- **current_shapes** (list of menpo.shape.PointCloud) – The list of current shapes that correspond to the images, which will be used as initial shapes.
- **prefix** (str, optional) – The prefix to use when printing information.
- **verbose** (bool, optional) – If True, then information is printed during training.

**Returns** current_shapes (list of menpo.shape.PointCloud) – The list of current shapes that correspond to the images.
ParametricAppearanceWeightsNewton

class menpofit.sdm.ParametricAppearanceWeightsNewton(patch_features=<function no_op>, patch_shape=(17, 17), n_iterations=3, appearance_model_cls=<class 'menpo.model.pca.PCAVectorModel'>, compute_error=<function euclidean_bb_normalised_error>, alpha=0, bias=True)

Bases: ParametricAppearanceNewton

Class for training a cascaded-regression Newton algorithm that employs a parametric appearance model using Incremental Regularized Linear Regression (IRLRegression). The algorithm uses the projection weights of the appearance vectors as features in the regression.

increment (images, gt_shapes, current_shapes, prefix=",", verbose=False)

Method to increment the model with the set of current shapes.

Parameters

- **images** (list of menpo.image.Image) – The list of training images.
- **gt_shapes** (list of menpo.shape.PointCloud) – The list of ground truth shapes that correspond to the images.
- **current_shapes** (list of menpo.shape.PointCloud) – The list of current shapes that correspond to the images.
- **prefix** (str, optional) – The prefix to use when printing information.
- **verbose** (bool, optional) – If True, then information is printed during training.

Returns **current_shapes** (list of menpo.shape.PointCloud) – The list of current shapes that correspond to the images.

run (image, initial_shape, gt_shape=None, return_costs=False, **kwargs)

Run the algorithm to an image given an initial shape.

Parameters

- **image** (menpo.image.Image or subclass) – The image to be fitted.
- **initial_shape** (menpo.shape.PointCloud) – The initial shape from which the fitting procedure will start.
- **gt_shape** (menpo.shape.PointCloud or None, optional) – The ground truth shape associated to the image.
- **return_costs** (bool, optional) – If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that this argument currently has no effect and will raise a warning if set to "True". This is because it is not possible to evaluate the cost function of this algorithm.

Returns **fitting_result** (NonParametricIterativeResult) – The result of the fitting procedure.

train (images, gt_shapes, current_shapes, prefix=",", verbose=False)

Method to train the model given a set of initial shapes.

Parameters
- **images** *(list of menpo.image.Image)* – The list of training images.

- **gt_shapes** *(list of menpo.shape.PointCloud)* – The list of ground truth shapes that correspond to the images.

- **current_shapes** *(list of menpo.shape.PointCloud)* – The list of current shapes that correspond to the images, which will be used as initial shapes.

- **prefix** *(str, optional)* – The prefix to use when printing information.

- **verbose** *(bool, optional)* – If True, then information is printed during training.

Returns **current_shapes** *(list of menpo.shape.PointCloud)* – The list of current shapes that correspond to the images.

### ParametricAppearanceWeightsGaussNewton

```python
class menpofit.sdm.ParametricAppearanceWeightsGaussNewton(patch_features=<function no_op>,
    patch_shape=(17, 17),
    n_iterations=3,
    appearance_model_cls=<class 'menpo.model.pca.PCAVectorModel'>,
    compute_error=<function euclidean_bb_normalised_error>,
    alpha=0, bias=True, alpha2=0)
```

Bases: `ParametricAppearanceGaussNewton`

Class for training a cascaded-regression Gauss-Newton algorithm that employs a parametric appearance model using Indirect Incremental Regularized Linear Regression (*IIRLRegression*). The algorithm uses the projection weights of the appearance vectors as features in the regression.

- **increment** *(images, gt_shapes, current_shapes, prefix='', verbose=False)*

  Method to increment the model with the set of current shapes.

  Parameters

  - **images** *(list of menpo.image.Image)* – The list of training images.

  - **gt_shapes** *(list of menpo.shape.PointCloud)* – The list of ground truth shapes that correspond to the images.

  - **current_shapes** *(list of menpo.shape.PointCloud)* – The list of current shapes that correspond to the images.

  - **prefix** *(str, optional)* – The prefix to use when printing information.

  - **verbose** *(bool, optional)* – If True, then information is printed during training.

  Returns **current_shapes** *(list of menpo.shape.PointCloud)* – The list of current shapes that correspond to the images.

- **run** *(image, initial_shape=None, return_costs=False, **kwargs)*

  Run the algorithm to an image given an initial shape.

  Parameters

  - **image** *(menpo.image.Image or subclass)* – The image to be fitted.
- **initial_shape** ([menpo.shape.PointCloud](https://menpo.org/docs/api/menpo.shape.PointCloud.html)) – The initial shape from which the fitting procedure will start.

- **gt_shape** ([menpo.shape.PointCloud](https://menpo.org/docs/api/menpo.shape.PointCloud.html) or None, optional) – The ground truth shape associated to the image.

- **return_costs** (bool, optional) – If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that this argument currently has no effect and will raise a warning if set to ‘True’. This is because it is not possible to evaluate the cost function of this algorithm.

Returns **fitting_result** ([NonParametricIterativeResult](https://menpo.org/docs/api/menpo.fit.NonlinearityIterativeResult.html)) – The result of the fitting procedure.

**train** (images, gt_shapes, current_shapes, prefix=", verbose=False)

Method to train the model given a set of initial shapes.

Parameters

- **images** (list of menpo.image.Image) – The list of training images.

- **gt_shapes** (list of menpo.shape.PointCloud) – The list of ground truth shapes that correspond to the images.

- **current_shapes** (list of menpo.shape.PointCloud) – The list of current shapes that correspond to the images, which will be used as initial shapes.

- **prefix** (str, optional) – The prefix to use when printing information.

- **verbose** (bool, optional) – If True, then information is printed during training.

Returns **current_shapes** (list of menpo.shape.PointCloud) – The list of current shapes that correspond to the images.

### Fully Parametric Algorithms

The cascaded regression is performed between the parameters of a statistical shape model and features that are based on a statistical parametric appearance model.

#### FullyParametricProjectOutNewton

```python
class menpo.fit.FullyParametricProjectOutNewton(patch_features=<function no_op>, patch_shape=(17, 17), n_iterations=3, shape_model_cls=<class 'menpo.fit.modelinstance.OrthoPDM'>, appearance_model_cls=<class 'menpo.model.pca.PCAVectorModel'>, compute_error=<function euclidean_bb_normalised_error>, alpha=0, bias=True)
```

Bases: ParametricAppearanceProjectOut

Class for training a cascaded-regression algorithm that employs parametric shape and appearance models using Incremental Regularized Linear Regression ([IRLRegression](https://scikit-learn.org/stable/modules/linear_model.html#irls)). The algorithm uses the projected-out appearance vectors as features in the regression.

Parameters
- **patch_features** *(callable, optional)* – The features to be extracted from the patches of an image.
- **patch_shape** *(int, int, optional)* – The shape of the extracted patches.
- **n_iterations** *(int, optional)* – The number of iterations (cascades).
- **shape_model_cls** *(subclass of PDM, optional)* – The class to be used for building the shape model. The most common choice is OrthoPDM.
- **appearance_model_cls** *(menpo.model.PCAVectorModel or subclass)* – The class to be used for building the appearance model.
- **compute_error** *(callable, optional)* – The function to be used for computing the fitting error when training each cascade.
- **alpha** *(float, optional)* – The regularization parameter.
- **bias** *(bool, optional)* – Flag that controls whether to use a bias term.

**increment** *(images, gt_shapes, current_shapes, prefix=", verbose=False)*

Method to increment the model with the set of current shapes.

### Parameters

- **images** *(list of menpo.image.Image)* – The list of training images.
- **gt_shapes** *(list of menpo.shape.PointCloud)* – The list of ground truth shapes that correspond to the images.
- **current_shapes** *(list of menpo.shape.PointCloud)* – The list of current shapes that correspond to the images.
- **prefix** *(str, optional)* – The prefix to use when printing information.
- **verbose** *(bool, optional)* – If True, then information is printed during training.

**Returns** **current_shapes** *(list of menpo.shape.PointCloud)* – The list of current shapes that correspond to the images.

**run** *(image, initial_shape, gt_shape=None, return_costs=False, **kwargs)*

Run the algorithm to an image given an initial shape.

### Parameters

- **image** *(menpo.image.Image or subclass)* – The image to be fitted.
- **initial_shape** *(menpo.shape.PointCloud)* – The initial shape from which the fitting procedure will start.
- **gt_shape** *(menpo.shape.PointCloud or None, optional)* – The ground truth shape associated to the image.
- **return_costs** *(bool, optional)* – If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that this argument currently has no effect and will raise a warning if set to ‘True’. This is because it is not possible to evaluate the cost function of this algorithm.

**Returns** **fitting_result** *(ParametricIterativeResult)* – The result of the fitting procedure.

**train** *(images, gt_shapes, current_shapes, prefix=", verbose=False)*

Method to train the model given a set of initial shapes.

### Parameters
- **images** (list of `menpo.image.Image`) – The list of training images.
- **gt_shapes** (list of `menpo.shape.PointCloud`) – The list of ground truth shapes that correspond to the images.
- **current_shapes** (list of `menpo.shape.PointCloud`) – The list of current shapes that correspond to the images, which will be used as initial shapes.
- **prefix** (str, optional) – The prefix to use when printing information.
- **verbose** (bool, optional) – If True, then information is printed during training.

**Returns**

- **current_shapes** (list of `menpo.shape.PointCloud`) – The list of current shapes that correspond to the images.

---

**FullyParametricProjectOutGaussNewton**

class menpofit.sdm.FullyParametricProjectOutGaussNewton (patch_features=<function no_op>, patch_shape=(17, 17), n_iterations=3, shape_model_cls=<class 'menpo.fit.modelinstance.OrthoPDM'>, appearance_model_cls=<class 'menpo.model.pca.PCAVectorModel'>, compute_error=<function euclidean_bb_normalised_error>, alpha=0, bias=True, alpha2=0)

Bases: ParametricAppearanceProjectOut

Class for training a cascaded-regression algorithm that employs parametric shape and appearance models using Indirect Incremental Regularized Linear Regression (**IIRLRegression**). The algorithm uses the projected-out appearance vectors as features in the regression.

**Parameters**

- **patch_features** (callable, optional) – The features to be extracted from the patches of an image.
- **patch_shape** ((int, int), optional) – The shape of the extracted patches.
- **n_iterations** (int, optional) – The number of iterations (cascades).
- **shape_model_cls** (subclass of `PDM`, optional) – The class to be used for building the shape model. The most common choice is `OrthoPDM`.
- **appearance_model_cls** (menpo.model.PCAVectorModel or subclass) – The class to be used for building the appearance model.
- **compute_error** (callable, optional) – The function to be used for computing the fitting error when training each cascade.
- **alpha** (float, optional) – The regularization parameter.
- **bias** (bool, optional) – Flag that controls whether to use a bias term.
- **alpha2** (float, optional) – The regularization parameter of the Hessian matrix.

---

2.1. Deformable Models
**increment** *(images, gt_shapes, current_shapes, prefix=", verbose=False)*
Method to increment the model with the set of current shapes.

- **images** *(list of menpo.image.Image) – The list of training images.*
- **gt_shapes** *(list of menpo.shape.PointCloud) – The list of ground truth shapes that correspond to the images.*
- **current_shapes** *(list of menpo.shape.PointCloud) – The list of current shapes that correspond to the images.*
- **prefix** *(str, optional) – The prefix to use when printing information.*
- **verbose** *(bool, optional) – If True, then information is printed during training.*

**Returns** **current_shapes** *(list of menpo.shape.PointCloud) – The list of current shapes that correspond to the images.*

**run** *(image, initial_shape, gt_shape=None, return_costs=False, **kwargs)*
Run the algorithm to an image given an initial shape.

- **image** *(menpo.image.Image or subclass) – The image to be fitted.*
- **initial_shape** *(menpo.shape.PointCloud) – The initial shape from which the fitting procedure will start.*
- **gt_shape** *(menpo.shape.PointCloud or None, optional) – The ground truth shape associated to the image.*
- **return_costs** *(bool, optional) – If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that this argument currently has no effect and will raise a warning if set to ‘‘True’’. This is because it is not possible to evaluate the cost function of this algorithm.*

**Returns** **fitting_result** *(ParametricIterativeResult) – The result of the fitting procedure.*

**train** *(images, gt_shapes, current_shapes, prefix=", verbose=False)*
Method to train the model given a set of initial shapes.

- **images** *(list of menpo.image.Image) – The list of training images.*
- **gt_shapes** *(list of menpo.shape.PointCloud) – The list of ground truth shapes that correspond to the images.*
- **current_shapes** *(list of menpo.shape.PointCloud) – The list of current shapes that correspond to the images, which will be used as initial shapes.*
- **prefix** *(str, optional) – The prefix to use when printing information.*
- **verbose** *(bool, optional) – If True, then information is printed during training.*

**Returns** **current_shapes** *(list of menpo.shape.PointCloud) – The list of current shapes that correspond to the images.*
**FullyParametricMeanTemplateNewton**

```python
class menpofit.sdm.FullyParametricMeanTemplateNewton (patch_features=<function no_op>, patch_shape=(17, 17), n_iterations=3, shape_model_cls=<class 'menpo.fit.modelinstance.OrthoPDM'>, appearance_model_cls=<class 'menpo.model.pca.PCAVectorModel'>, compute_error=<function euclidean_bb_normalised_error>, alpha=0, bias=True)
```

**Bases:** `ParametricAppearanceMeanTemplate`

Class for training a cascaded-regression algorithm that employs parametric shape and appearance models using Incremental Regularized Linear Regression (`IRLRegression`). The algorithm uses the centered appearance vectors as features in the regression.

**Parameters**

- **patch_features** (callable, optional) – The features to be extracted from the patches of an image.
- **patch_shape** (int, int), optional) – The shape of the extracted patches.
- **n_iterations** (int, optional) – The number of iterations (cascades).
- **shape_model_cls** (subclass of `PDM`, optional) – The class to be used for building the shape model. The most common choice is `OrthoPDM`.
- **appearance_model_cls** (subclass of `menpo.model.PCAVectorModel`) – The class to be used for building the appearance model.
- **compute_error** (callable, optional) – The function to be used for computing the fitting error when training each cascade.
- **alpha** (float, optional) – The regularization parameter.
- **bias** (bool, optional) – Flag that controls whether to use a bias term.

**increment** *(images, gt_shapes, current_shapes, prefix=”, verbose=False)*

Method to increment the model with the set of current shapes.

**Parameters**

- **images** (list of `menpo.image.Image`) – The list of training images.
- **gt_shapes** (list of `menpo.shape.PointCloud`) – The list of ground truth shapes that correspond to the images.
- **current_shapes** (list of `menpo.shape.PointCloud`) – The list of current shapes that correspond to the images.
- **prefix** (str, optional) – The prefix to use when printing information.
- **verbose** (bool, optional) – If True, then information is printed during training.

**Returns** `current_shapes` (list of `menpo.shape.PointCloud`) – The list of current shapes that correspond to the images.

**run** *(image, initial_shape, gt_shape=None, return_costs=False, **kwargs)*

Run the algorithm to an image given an initial shape.
Parameters

- **image** (*menpo.image.Image* or subclass) – The image to be fitted.
- **initial_shape** (*menpo.shape.PointCloud*) – The initial shape from which the fitting procedure will start.
- **gt_shape** (*menpo.shape.PointCloud* or *None*, optional) – The ground truth shape associated to the image.
- **return_costs** (*bool*, optional) – If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that this argument currently has no effect and will raise a warning if set to ‘True’. This is because it is not possible to evaluate the cost function of this algorithm.

Returns **fitting_result** (*ParametricIterativeResult*) – The result of the fitting procedure.

**train** (*images, gt_shapes, current_shapes, prefix=", verbose=False*)

Method to train the model given a set of initial shapes.

Parameters

- **images** (*list* of *menpo.image.Image*) – The list of training images.
- **gt_shapes** (*list* of *menpo.shape.PointCloud*) – The list of ground truth shapes that correspond to the images.
- **current_shapes** (*list* of *menpo.shape.PointCloud*) – The list of current shapes that correspond to the images, which will be used as initial shapes.
- **prefix** (*str*, optional) – The prefix to use when printing information.
- **verbose** (*bool*, optional) – If True, then information is printed during training.

Returns **current_shapes** (*list* of *menpo.shape.PointCloud*) – The list of current shapes that correspond to the images.

**FullyParametricWeightsNewton**

class *menpofit.sdm.FullyParametricWeightsNewton* (*patch_features=<function no_op>, patch_shape=(17, 17), n_iterations=3, shape_model_cls=<class 'menpofit.modelinstance.OrthoPDM'>, appearance_model_cls=<class 'menpo.model.pca.PCAVectorModel'>, compute_error=<function euclidean_bb_normalised_error>, alpha=0, bias=True)

Bases: *ParametricAppearanceWeights*

Class for training a cascaded-regression algorithm that employs parametric shape and appearance models using Incremental Regularized Linear Regression (*IRLRegression*). The algorithm uses the projection weights of the appearance vectors as features in the regression.

Parameters

- **patch_features** (*callable*, optional) – The features to be extracted from the patches of an image.
- **patch_shape** (*(*int*, *int*), optional) – The shape of the extracted patches.
• **n_iterations** (*int*, optional) – The number of iterations (cascades).

• **shape_model_cls** (*subclass* of *PDM*, optional) – The class to be used for building the shape model. The most common choice is *OrthoPDM*.

• **appearance_model_cls** (*menpo.model.PCAVectorModel* or *subclass*) – The class to be used for building the appearance model.

• **compute_error** (*callable*, optional) – The function to be used for computing the fitting error when training each cascade.

• **alpha** (*float*, optional) – The regularization parameter.

• **bias** (*bool*, optional) – Flag that controls whether to use a bias term.

**increment** (*images*, *gt_shapes*, *current_shapes*, *prefix=", verbose=False*)

Method to increment the model with the set of current shapes.

Parameters

• **images** (*list* of *menpo.image.Image*) – The list of training images.

• **gt_shapes** (*list* of *menpo.shape.PontCloud*) – The list of ground truth shapes that correspond to the images.

• **current_shapes** (*list* of *menpo.shape.PointCloud*) – The list of current shapes that correspond to the images.

• **prefix** (*str*, optional) – The prefix to use when printing information.

• **verbose** (*bool*, optional) – If True, then information is printed during training.

Returns **current_shapes** (*list* of *menpo.shape.PointCloud*) – The list of current shapes that correspond to the images.

**run** (*image*, *initial_shape*, *gt_shape=None, return_costs=False, **kwargs*)

Run the algorithm to an image given an initial shape.

Parameters

• **image** (*menpo.image.Image* or *subclass*) – The image to be fitted.

• **initial_shape** (*menpo.shape.PontCloud*) – The initial shape from which the fitting procedure will start.

• **gt_shape** (*menpo.shape.PointCloud* or *None*, optional) – The ground truth shape associated to the image.

• **return_costs** (*bool*, optional) – If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that this argument currently has no effect and will raise a warning if set to ‘True’. This is because it is not possible to evaluate the cost function of this algorithm.

Returns **fitting_result** (*ParametricIterativeResult*) – The result of the fitting procedure.

**train** (*images*, *gt_shapes*, *current_shapes*, *prefix=", verbose=False*)

Method to train the model given a set of initial shapes.

Parameters

• **images** (*list* of *menpo.image.Image*) – The list of training images.

• **gt_shapes** (*list* of *menpo.shape.PointCloud*) – The list of ground truth shapes that correspond to the images.
• **current_shapes** *(list of menpo.shape.PointCloud)* – The list of current shapes that correspond to the images, which will be used as initial shapes.

• **prefix** *(str, optional)* – The prefix to use when printing information.

• **verbose** *(bool, optional)* – If True, then information is printed during training.

**Returns** current_shapes *(list of menpo.shape.PointCloud)* – The list of current shapes that correspond to the images.

---

### FullyParametricProjectOutOPP

class menpofit.sdm.FullyParametricProjectOutOPP

```
patch_features=<function no_op>,
patch_shape=(17, 17), n_iterations=3,
shape_model_cls=<class 'menpofit.modelinstance.OrthoPDM'>,
appearance_model_cls=<class 'menpo.model.pca.PCAVectorModel'>,
compute_error=<function euclidean_bb_normalised_error>,
bias=True)
```

**Bases:** ParametricAppearanceProjectOut

Class for training a cascaded-regression algorithm that employs parametric shape and appearance models using Multivariate Linear Regression with Orthogonal Procrustes Problem reconstructions (OPPRegression).

**Parameters**

• **patch_features** *(callable, optional)* – The features to be extracted from the patches of an image.

• **patch_shape** *(int, int, optional)* – The shape of the extracted patches.

• **n_iterations** *(int, optional)* – The number of iterations (cascades).

• **shape_model_cls** *(subclass of PDM, optional)* – The class to be used for building the shape model. The most common choice is OrthoPDM.

• **appearance_model_cls** *(menpo.model.PCAVectorModel or subclass)* – The class to be used for building the appearance model.

• **compute_error** *(callable, optional)* – The function to be used for computing the fitting error when training each cascade.

• **bias** *(bool, optional)* – Flag that controls whether to use a bias term.

**increment** *(images, gt_shapes, current_shapes, prefix="", verbose=False)*

Method to increment the model with the set of current shapes.

**Parameters**

• **images** *(list of menpo.image.Image)* – The list of training images.

• **gt_shapes** *(list of menpo.shape.PointCloud)* – The list of ground truth shapes that correspond to the images.

• **current_shapes** *(list of menpo.shape.PointCloud)* – The list of current shapes that correspond to the images.

• **prefix** *(str, optional)* – The prefix to use when printing information.

• **verbose** *(bool, optional)* – If True, then information is printed during training.
Returns current_shapes (list of menpo.shape.PointCloud) – The list of current shapes that correspond to the images.

run (image, initial_shape, gt_shape=None, return_costs=False, **kwargs)
Run the algorithm to an image given an initial shape.

Parameters
- image (menpo.image.Image or subclass) – The image to be fitted.
- initial_shape (menpo.shape.PointCloud) – The initial shape from which the fitting procedure will start.
- gt_shape (menpo.shape.PointCloud or None, optional) – The ground truth shape associated to the image.
- return_costs (bool, optional) – If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that this argument currently has no effect and will raise a warning if set to ‘‘True’’. This is because it is not possible to evaluate the cost function of this algorithm.

Returns fitting_result (ParametricIterativeResult) – The result of the fitting procedure.

train (images, gt_shapes, current_shapes, prefix=", verbose=False)
Method to train the model given a set of initial shapes.

Parameters
- images (list of menpo.image.Image) – The list of training images.
- gt_shapes (list of menpo.shape.PointCloud) – The list of ground truth shapes that correspond to the images.
- current_shapes (list of menpo.shape.PointCloud) – The list of current shapes that correspond to the images, which will be used as initial shapes.
- prefix (str, optional) – The prefix to use when printing information.
- verbose (bool, optional) – If True, then information is printed during training.

Returns current_shapes (list of menpo.shape.PointCloud) – The list of current shapes that correspond to the images.

2.2 Internal API

2.2.1 menpofit.builder

Building Functions
Collection of functions that are commonly-used by most deformable model builders.

align_shapes

menpofit.builder.align_shapes (shapes)
Function that aligns a set of shapes by applying Generalized Procrustes Analysis.

Parameters shapes (list of menpo.shape.PointCloud) – The input shapes.
MenpoFit Documentation, Release 0.6.1

Returns aligned_shapes (list of menpo.shape.PointCloud) – The list of aligned shapes.

build_patch_reference_frame

menpofit.builder.build_patch_reference_frame (landmarks, boundary=3, group='source', patch_shape=(17, 17))

Builds a patch-based reference frame from a particular set of landmarks.

Parameters

• landmarks (menpo.shape.PointCloud) – The landmarks that will be used to build the reference frame.
• boundary (int, optional) – The number of pixels to be left as a safe margin on the boundaries of the reference frame (has potential effects on the gradient computation).
• group (str, optional) – Group that will be assigned to the provided set of landmarks on the reference frame.
• patch_shape ((int, int), optional) – The shape of the patches.


build_reference_frame

menpofit.builder.build_reference_frame (landmarks, boundary=3, group='source')

Builds a reference frame from a particular set of landmarks.

Parameters

• landmarks (menpo.shape.PointCloud) – The landmarks that will be used to build the reference frame.
• boundary (int, optional) – The number of pixels to be left as a safe margin on the boundaries of the reference frame (has potential effects on the gradient computation).
• group (str, optional) – Group that will be assigned to the provided set of landmarks on the reference frame.


compute_features

menpofit.builder.compute_features (images, features, prefix='', verbose=False)

Function that extracts features from a list of images.

Parameters

• images (list of menpo.image.Image) – The set of images.
• features (callable) – The features extraction function. Please refer to menpo.feature and menpofit.feature.
• prefix (str) – The prefix of the printed information.
• verbose (bool, Optional) – Flag that controls information and progress printing.

Returns feature_images (list of menpo.image.Image) – The list of feature images.
compute_reference_shape

menpofit.builder.compute_reference_shape(shapes, diagonal, verbose=False)

Function that computes the reference shape as the mean shape of the provided shapes.

Parameters

- shapes (list of menpo.shape.PointCloud) – The set of shapes from which to build the reference shape.
- diagonal (int or None) – If int, it ensures that the mean shape is scaled so that the diagonal of the bounding box containing it matches the provided value. If None, then the mean shape is not rescaled.
- verbose (bool, optional) – If True, then progress information is printed.


densify_shapes

menpofit.builder.densify_shapes(shapes, reference_frame, transform)

Function that densifies a set of sparse shapes given a reference frame.

Parameters

- shapes (list of menpo.shape.PointCloud) – The input shapes.
- reference_frame (menpo.image.BooleanImage) – The reference frame, the mask of which will be used.
- transform (menpo.transform.Transform) – The transform to use for mapping the dense points.

Returns dense_shapes (list of menpo.shape.PointCloud) – The list of dense shapes.

extract_patches

menpofit.builder.extract_patches(images, shapes, patch_shape, normalise_function=<function no_op>, prefix=' ', verbose=False)

Function that extracts patches around the landmarks of the provided images.

Parameters

- images (list of menpo.image.Image) – The set of images to warp.
- shapes (list of menpo.shape.PointCloud) – The set of shapes that correspond to the images.
- patch_shape ((int, int)) – The shape of the patches.
- normalise_function (callable) – A normalisation function to apply on the values of the patches.
- prefix (str) – The prefix of the printed information.
- verbose (bool, Optional) – Flag that controls information and progress printing.

Returns patch_images (list of menpo.image.Image) – The list of images with the patches per image. Each output image has shape (n_center, n_offset, n_channels, patch_shape).
normalization_wrt_reference_shape

`menpofit.builder.normalization_wrt_reference_shape(images, group, diagonal, verbose=False)`

Function that normalizes the images’ sizes with respect to the size of the mean shape. This step is essential before building a deformable model.

The normalization includes: 1) Computation of the reference shape as the mean shape of the images’ landmarks. 2) Scaling of the reference shape using the diagonal. 3) Rescaling of all the images so that their shape’s scale is in correspondence with the reference shape’s scale.

Parameters

- **images** (list of `menpo.image.Image`) – The set of images to normalize.
- **group** (str) – If `str`, then it specifies the group of the images’s shapes. If `None`, then the images must have only one landmark group.
- **diagonal** (int or `None`) – If `int`, it ensures that the mean shape is scaled so that the diagonal of the bounding box containing it matches the provided value. If `None`, then the mean shape is not rescaled.
- **verbose** (bool, Optional) – Flag that controls information and progress printing.

Returns

- **reference_shape** (`menpo.shape.PointCloud`) – The reference shape that was used to resize all training images to a consistent object size.
- **normalized_images** (list of `menpo.image.Image`) – The images with normalized size.

rescale_images_to_reference_shape

`menpofit.builder.rescale_images_to_reference_shape(images, group, reference_shape, verbose=False)`

Function that normalizes the images’ sizes with respect to the size of the provided reference shape. In other words, the function rescales the provided images so that the size of the bounding box of their attached shape is the same as the size of the bounding box of the provided reference shape.

Parameters

- **images** (list of `menpo.image.Image`) – The set of images that will be rescaled.
- **group** (str or `None`) – If `str`, then it specifies the group of the images’s shapes. If `None`, then the images must have only one landmark group.
- **reference_shape** (`menpo.shape.PointCloud`) – The reference shape.
- **verbose** (bool, optional) – If `True`, then progress information is printed.

Returns **normalized_images** (list of `menpo.image.Image`) – The rescaled images.

scale_images

`menpofit.builder.scale_images(images, scale, prefix=",", return_transforms=False, verbose=False)`

Function that rescales a list of images and optionally returns the scale transforms.

Parameters

- **images** (list of `menpo.image.Image`) – The set of images to scale.
• **scale** *(float or tuple of floats)* – The scale factor. If a tuple, the scale to apply to each dimension. If a single float, the scale will be applied uniformly across each dimension.

• **prefix** *(str, optional)* – The prefix of the printed information.

• **return_transforms** *(bool, optional)* – If True, then a list with the menpo.transform.Scale objects that were used to perform the rescale for each image is also returned.

• **verbose** *(bool, optional)* – Flag that controls information and progress printing.

**Returns**

• **scaled_images** *(list of menpo.image.Image)* – The list of rescaled images.

• **scale_transforms** *(list of menpo.transform.Scale)* – The list of scale transforms that were used. It is returned only if return_transforms is True.

**warp_images**

menpofit.builder.warp_images(images, shapes, reference_frame, transform, prefix=", verbose=None)

Function that warps a list of images into the provided reference frame.

**Parameters**

• **images** *(list of menpo.image.Image)* – The set of images to warp.

• **shapes** *(list of menpo.shape.PointCloud)* – The set of shapes that correspond to the images.

• **reference_frame** *(menpo.image.BooleanImage)* – The reference frame to warp to.

• **transform** *(menpo.transform.Transform)* – Transform from the reference frame back to the image. Defines, for each pixel location on the reference frame, which pixel location should be sampled from on the image.

• **prefix** *(str)* – The prefix of the printed information.

• **verbose** *(bool, Optional)* – Flag that controls information and progress printing.

**Returns** **warped_images** *(list of menpo.image.MaskedImage)* – The list of warped images.

**Warnings**

MenpoFitBuilderWarning

class menpofit.builder.MenpoFitBuilderWarning

Bases: Warning

A warning that some part of building the model may cause issues.

    with_traceback()
        Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

MenpoFitModelBuilderWarning

class menpofit.builder.MenpoFitModelBuilderWarning

Bases: Warning
A warning that the parameters chosen to build a given model may cause unexpected behaviour.

```python
with_traceback()
   Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.
```

### 2.2.2 `menpofit.checks`

Functions for checking the parameters’ values that are passed in MenpoFit’s classes.

#### Parameters Check Functions

##### `check_diagonal`

```python
menpofit.checks.check_diagonal(diagonal)
```
Checks that the diagonal length used to normalize the images’ size is $\geq 20$.

- **Parameters**
  - `diagonal` *(int)* – The value to check.
  - **Returns**
    - `diagonal` *(int)* – The value if it’s correct.
  - **Raises**
    - `ValueError` – diagonal must be $\geq 20$ or None

##### `check_landmark_trilist`

```python
menpofit.checks.check_landmark_trilist(image, transform, group=None)
```
Checks that the provided image has a triangulated shape (thus an instance of `menpo.shape.TriMesh`) and the transform is `menpo.transform.PiecewiseAffine`

- **Parameters**
  - `image` *(menpo.image.Image or subclass)* – The input image.
  - `group` *(str or None, optional)* – The group of the shape to check.
  - **Raises**
    - `Warning` – The given images do not have an explicit triangulation applied. A Delaunay Triangulation will be computed and used for warping. This may be suboptimal and cause warping artifacts.

##### `check_trilist`

```python
menpofit.checks.check_trilist(shape, transform)
```
Checks that the provided shape is triangulated (thus an instance of `menpo.shape.TriMesh`) and the transform is `menpo.transform.PiecewiseAffine`

- **Parameters**
  - `shape` *(menpo.shape.TriMesh)* – The input shape (usually the reference/mean shape of a model).
  - **Raises**
    - `Warning` – The given images do not have an explicit triangulation applied. A Delaunay Triangulation will be computed and used for warping. This may be suboptimal and cause warping artifacts.
**check_model**

`menpofit.checks.check_model(model, cls)`

Function that checks whether the provided class object is a subclass of the provided base class. 

**Parameters**

- `model (class)` – The object.
- `cls (class)` – The required base class.

**Raises** `ValueError` – Model must be a `{cls}` instance.

**Multi-Scale Parameters Check Functions**

**check_scales**

`menpofit.checks.check_scales(scales)`

Checks that the provided scales argument is either int or float or an iterable of those. It makes sure that it returns a list of scales.

**Parameters**

- `scales (int or float or list/tuple of those)` – The value to check.

**Returns**

- `scales (list of int or float)` – The scales in a list.

**Raises** `ValueError` – scales must be an int/float or a list/tuple of int/float

**check_multi_scale_param**

`menpofit.checks.check_multi_scale_param(n_scales, types, param_name, param)`

General function for checking a parameter defined for multiple scales. It raises an error if the parameter is not an iterable with the correct size and correct types.

**Parameters**

- `n_scales (int)` – The number of scales.
- `types (tuple)` – The tuple of variable types that the parameter is allowed to have.
- `param_name (str)` – The name of the parameter.
- `param (types)` – The parameter value.

**Returns**

- `param (list of types)` – The list of values per scale.

**Raises** `ValueError` – `{param_name} must be in {types} or a list/tuple of {types} with the same length as the number of scales

**check_callable**

`menpofit.checks.check_callable(callables, n_scales)`

Checks the callable type per level.

**Parameters**

- `callables (callable or list of callables)` – The callable to be used per scale.
- `n_scales (int)` – The number of scales.

**Returns**

- `callable_list (list)` – A list of callables.
Raises \texttt{ValueError} – callables must be a callable or a list/tuple of callables with the same length as the number of scales

**check\_patch\_shape**

\texttt{menpofit.checks.check\_patch\_shape(patch\_shape, n\_scales)}

Function for checking a multi-scale \texttt{patch\_shape} parameter value.

\textbf{Parameters}

- \texttt{patch\_shape (list/tuple of int/float or list of those)} – The patch shape per scale
- \texttt{n\_scales (int)} – The number of scales.

\textbf{Returns} \texttt{patch\_shape (list of list/tuple of int/float)} – The list of patch shape per scale.

\textbf{Raises} \texttt{ValueError} – \texttt{patch\_shape} must be a list/tuple of int or a list/tuple of list/tuple of int/float with the same length as the number of scales

**check\_max\_iters**

\texttt{menpofit.checks.check\_max\_iters(max\_iters, n\_scales)}

Function that checks the value of a \texttt{max\_iters} parameter defined for multiple scales. It must be \texttt{int} or \texttt{list of int}.

\textbf{Parameters}

- \texttt{max\_iters (int or list of int)} – The value to check.
- \texttt{n\_scales (int)} – The number of scales.

\textbf{Returns} \texttt{max\_iters (list of int)} – The list of values per scale.

\textbf{Raises} \texttt{ValueError} – \texttt{max\_iters} can be integer, integer list containing 1 or \{n\_scales\} elements or \texttt{None}

**check\_max\_components**

\texttt{menpofit.checks.check\_max\_components(max\_components, n\_scales, var\_name)}

Checks the maximum number of components per scale. It must be \texttt{None} or \texttt{int} or \texttt{float} or a \texttt{list} of those containing 1 or \{n\_scales\} elements.

\textbf{Parameters}

- \texttt{max\_components (None or int or float or a list of those)} – The value to check.
- \texttt{n\_scales (int)} – The number of scales.
- \texttt{var\_name (str)} – The name of the variable.

\textbf{Returns} \texttt{max\_components (list of None or int or float)} – The list of max components per scale.

\textbf{Raises} \texttt{ValueError} – \{var\_name\} must be \texttt{None} or an \texttt{int > 0} or a \texttt{0 <= float <= 1} or a list of those containing 1 or \{n\_scales\} elements
set_models_components

menpofit.checks.set_models_components(models, n_components)
Function that sets the number of active components to a list of models.

Parameters
- models (list or class) – The list of models per scale.
- n_components (int or float or None or list of those) – The number of components per model.

Raises ValueError – n_components can be an integer or a float or None or a list containing 1 or \{n_scales\} of those

check_algorithm_cls

menpofit.checks.check_algorithm_cls(algorithm_cls, n_scales, base_algorithm_cls)
Function that checks whether the list of class objects defined per scale are subclasses of the provided base class.

Parameters
- algorithm_cls (class or list of class) – The list of objects per scale.
- n_scales (int) – The number of scales.
- base_algorithm_cls (class) – The required base class.

Raises ValueError – algorithm_cls must be a subclass of \{base_algorithm_cls\} or a list/tuple of \{base_algorithm_cls\} subclasses with the same length as the number of scales \{n_scales\}

check_sampling

menpofit.checks.check_sampling(sampling, n_scales)
Function that checks the value of a sampling parameter defined for multiple scales. It must be int or ndarray or list of those.

Parameters
- sampling (int or ndarray or list of those) – The value to check.
- n_scales (int) – The number of scales.

Returns sampling (list of int or ndarray) – The list of values per scale.

Raises
- ValueError – A sampling list can only contain 1 element or \{n_scales\} elements
- ValueError – sampling can be an integer or ndarray, a integer or ndarray list containing 1 or \{n_scales\} elements or None

check_graph

menpofit.checks.check_graph(graph, graph_types, param_name, n_scales)
Checks the provided graph per pyramidal level. The graph must be a subclass of graph_types or a list of those.

Parameters
- graph (graph or list of graph types) – The graph argument to check.
• **graph_types** (*graph or tuple of graphs*) – The *tuple* of allowed graph types.

• **param_name** (*str*) – The name of the graph parameter.

• **n_scales** (*int*) – The number of pyramidal levels.

Returns **graph** (*list of graph types*) – The graph per scale in a *list*.

Raises

• **ValueError** – `{param_name} must be a list of length equal to the number of scales.

• **ValueError** – `{param_name} must be a list of `{graph_types_str}`. `{}` given instead.

### 2.2.3 menpofit.differentiable

**Differentiable Abstract Classes**

Objects that are able to compute their own derivatives.

**DL**

```python
class menpofit.differentiable.DL
    Bases: object
    Object that is able to take its own derivative with respect to landmark changes.

    abstract d_dl(points)
    The derivative of this spatial object with respect to spatial changes in anchor landmark points or centres, evaluated at points.

    Parameters

    points ((n_points, n_dims) ndarray) – The spatial points at which the derivative should be evaluated.

    Returns

    d_dl ((n_points, n_centres, n_dims) ndarray) – The Jacobian wrt landmark changes.

    d_dl[i, k, m] is the scalar differential change that the any dimension of the i’th point experiences due to a first order change in the m’th dimension of the k’th landmark point.

    Note that at present this assumes that the change in every dimension is equal.
```

**DP**

```python
class menpofit.differentiable.DP
    Bases: object
    Object that is able to take its own derivative with respect to the parametrisation.

    The parametrisation of objects is typically defined by the *menpo.base.Vectorizable* interface. As a result, *DP* is a mix-in that should be inherited along with *menpo.base.Vectorizable*.

    abstract d_dp(points)
    The derivative of this spatial object with respect to the parametrisation changes evaluated at points.

    Parameters

    points ((n_points, n_dims) ndarray) – The spatial points at which the derivative should be evaluated.
```
Returns

\texttt{d_dp \((n\_points, n\_parameters, n\_dims) ndarray\)} – The Jacobian with respect to the parametrisation.

\(d\_dp[i, j, k]\) is the scalar differential change that the \(k\)’th dimension of the \(i\)’th point experiences due to a first order change in the \(j\)’th scalar in the parametrisation vector.

\textbf{DX}

\texttt{class menpofit.differentiable.DX}

\texttt{Bases: object}

Object that is able to take its own derivative with respect to spatial changes.

\texttt{abstract d_dx(points)}

The first order derivative of this spatial object with respect to spatial changes evaluated at points.

\texttt{Parameters points \((n\_points, n\_dims) ndarray\)} – The spatial points at which the derivative should be evaluated.

\texttt{Returns}

\texttt{d_dx \((n\_points, n\_dims, n\_dims) ndarray\)} – The Jacobian wrt spatial changes.

\(d\_dx[i, j, k]\) is the scalar differential change that the \(j\)’th dimension of the \(i\)’th point experiences due to a first order change in the \(k\)’th dimension.

It may be the case that the Jacobian is constant across space - in this case axis zero may have length 1 to allow for broadcasting.

\textbf{2.2.4 menpofit.error}

\textbf{Normalisers}

Functions that compute a metric which can be used to normalise the error between two shapes.

\textbf{Bounding Box Normalisers}

\texttt{bb_area}

\texttt{menpofit.error.bb_area(shape)}

Computes the area of the bounding box of the provided shape, i.e.

\[ h \times w \]

where \(h\) and \(w\) are the height and width of the bounding box.

\texttt{Parameters shape (menpo.shape.PointCloud or subclass)} – The input shape.

\texttt{Returns bb_area (float)} – The area of the bounding box.
bb_perimeter

\texttt{menpofit.error.bb\_perimeter(shape)}

Computes the perimeter of the bounding box of the provided shape, i.e.

\[ 2(h + w) \]

where \( h \) and \( w \) are the height and width of the bounding box.

\textbf{Parameters} \( \text{shape} \) (\texttt{menpo.shape.PointCloud or subclass}) – The input shape.

\textbf{Returns} \( \text{bb\_perimeter} \) (float) – The perimeter of the bounding box.

bb_avg_edge_length

\texttt{menpofit.error.bb\_avg\_edge\_length(shape)}

Computes the average edge length of the bounding box of the provided shape, i.e.

\[ \frac{h + w}{2} = \frac{2h + 2w}{4} \]

where \( h \) and \( w \) are the height and width of the bounding box.

\textbf{Parameters} \( \text{shape} \) (\texttt{menpo.shape.PointCloud or subclass}) – The input shape.

\textbf{Returns} \( \text{bb\_avg\_edge\_length} \) (float) – The average edge length of the bounding box.

bb_diagonal

\texttt{menpofit.error.bb\_diagonal(shape)}

Computes the diagonal of the bounding box of the provided shape, i.e.

\[ \sqrt{h^2 + w^2} \]

where \( h \) and \( w \) are the height and width of the bounding box.

\textbf{Parameters} \( \text{shape} \) (\texttt{menpo.shape.PointCloud or subclass}) – The input shape.

\textbf{Returns} \( \text{bb\_diagonal} \) (float) – The diagonal of the bounding box.

Distance Normalisers

distance_two_indices

\texttt{menpofit.error.distance\_two\_indices(index1, index2, shape)}

Computes the Euclidean distance between two points of a shape, i.e.

\[ \sqrt{(s_{i,x} - s_{j,x})^2 + (s_{i,y} - s_{j,y})^2} \]

where \( s_{i,x}, s_{i,y} \) are the \( x \) and \( y \) coordinates of the \( i \)'th point (\texttt{index1}) and \( s_{j,x}, s_{j,y} \) are the \( x \) and \( y \) coordinates of the \( j \)'th point (\texttt{index2}).

\textbf{Parameters}

\begin{itemize}
  \item \texttt{index1} (int) – The index of the first point.
\end{itemize}
MenpoFit Documentation, Release 0.6.1

- **index2** *(int)* – The index of the second point.
- **shape** *(menpo.shape_PointCloud)* – The input shape.

**Returns** **distance_two_indices** *(float)* – The Euclidean distance between the points.

**Errors**

Functions that compute the error between two shapes.

**Root Mean Square Error**

**root_mean_square_error**

*menpofit.error.root_mean_square_error*(shape, gt_shape)

Computes the root mean square error between two shapes, i.e.

$$\sqrt{\frac{1}{N} \sum_{i=1}^{N} (s_i - s_{*i})^2}$$

where $s_i$ and $s_{*i}$ are the coordinates of the $i$’th point of the final and ground truth shapes, and $N$ is the total number of points.

**Parameters**

- **shape** *(menpo.shape_PointCloud)* – The input shape (e.g. the final shape of a fitting procedure).
- **gt_shape** *(menpo.shape_PointCloud)* – The ground truth shape.

**Returns** **root_mean_square_error** *(float)* – The root mean square error.

**root_mean_square_bb_normalised_error**

*menpofit.error.root_mean_square_bb_normalised_error*(shape, gt_shape, norm_shape=None, norm_type='avg_edge_length')

Computes the root mean square error between two shapes normalised by a measure based on the ground truth shape’s bounding box, i.e.

$$\frac{\mathcal{F}(s, s^*)}{\mathcal{N}(s^*)}$$

where

$$\mathcal{F}(s, s^*) = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (s_i - s_{*i})^2}$$

where $s$ and $s^*$ are the final and ground truth shapes, respectively. $s_i$ and $s_{*i}$ are the coordinates of the $i$’th point of the final and ground truth shapes, and $N$ is the total number of points. Finally, $\mathcal{N}(s^*)$ is a normalising function that returns a measure based on the ground truth shape’s bounding box.

**Parameters**
• **shape** (*menpo.shapePointCloud*) – The input shape (e.g. the final shape of a fitting procedure).

• **gt_shape** (*menpo.shapePointCloud*) – The ground truth shape.

• **norm_shape** (*menpo.shapePointCloud* or *None*, optional) – The shape to be used to compute the normaliser. If *None*, then the ground truth shape is used.

• **norm_type** ([:set-off]{'area', 'perimeter', 'avg_edge_length', 'diagonal'}), optional) – The type of the normaliser. Possible options are:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bb_area</td>
<td>Area of norm_shape’s bounding box</td>
</tr>
<tr>
<td>bb_perimeter</td>
<td>Perimeter of norm_shape’s bounding box</td>
</tr>
<tr>
<td>bb_avg_edge_length</td>
<td>Average edge length of norm_shape’s bbox</td>
</tr>
<tr>
<td>bb_diagonal</td>
<td>Diagonal of norm_shape’s bounding box</td>
</tr>
</tbody>
</table>

**Returns** error (*float*) – The computed root mean square normalised error.

**root_mean_square_distance_normalised_error**

*menpoFit.error.root_mean_square_distance_normalised_error*(shape, gt_shape, distance_norm_f)

Computes the root mean square error between two shapes normalised by a distance measure between two shapes, i.e.

\[
\frac{\mathcal{F}(s, s^*)}{\mathcal{N}(s, s^*)}
\]

where

\[
\mathcal{F}(s, s^*) = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (s_i - s_i^*)^2}
\]

where *s* and *s* * are the final and ground truth shapes, respectively. *s* * and *s* * are the coordinates of the *i*’th point of the final and ground truth shapes, and *N* is the total number of points. Finally, \(\mathcal{N}(s, s^*)\) is a normalising function based on a distance metric between the two shapes.

**Parameters**

• **shape** (*menpo.shapePointCloud*) – The input shape (e.g. the final shape of a fitting procedure).

• **gt_shape** (*menpo.shapePointCloud*) – The ground truth shape.

• **distance_norm_f** (*callable*) – The function to be used for computing the normalisation distance metric.

**Returns** error (*float*) – The computed root mean square normalised error.

**root_mean_square_distance_indexed_normalised_error**

*menpoFit.error.root_mean_square_distance_indexed_normalised_error*(shape, gt_shape, index1, index2)

Computes the root mean square error between two shapes normalised by the distance measure between two
points of the ground truth shape, i.e.

\[ \mathcal{F}(s, s^*) \]

where

\[ \mathcal{F}(s, s^*) = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (s_i - s^*_i)^2} \]

where \( s \) and \( s^* \) are the final and ground truth shapes, respectively. \( s_i \) and \( s^*_i \) are the coordinates of the \( i \)'th point of the final and ground truth shapes, and \( N \) is the total number of points. Finally, \( N(s^*) \) is a normalising function that returns the distance between two points of the ground truth shape.

**Parameters**

- **shape** (*menpo.shape.PointCloud*) – The input shape (e.g. the final shape of a fitting procedure).
- **gt_shape** (*menpo.shape.PointCloud*) – The ground truth shape.
- **index1** (*int*) – The index of the first point.
- **index2** (*int*) – The index of the second point.

**Returns**

- **error** (*float*) – The computed root mean square normalised error.

---

**Euclidean Distance Error**

**euclidean_error**

*menpofit.error.euclidean_error*(shape, gt_shape)

Computes the Euclidean error between two shapes, i.e.

\[ \frac{1}{N} \sum_{i=1}^{N} \sqrt{(s_{i,x} - s^*_{i,x})^2 + (s_{i,y} - s^*_{i,y})^2} \]

where \((s_{i,x}, s_{i,y})\) are the x and y coordinates of the \( i \)'th point of the final shape, \((s^*_{i,x}, s^*_{i,y})\) are the x and y coordinates of the \( i \)'th point of the ground truth shape and \( N \) is the total number of points.

**Parameters**

- **shape** (*menpo.shape.PointCloud*) – The input shape (e.g. the final shape of a fitting procedure).
- **gt_shape** (*menpo.shape.PointCloud*) – The ground truth shape.

**Returns**

- **root_mean_square_error** (*float*) – The Euclidean error.

**euclidean_bb_normalised_error**

*menpofit.error.euclidean_bb_normalised_error*(shape, gt_shape, norm_shape=None, norm_type='avg_edge_length')

Computes the Euclidean error between two shapes normalised by a measure based on the ground truth shape’s bounding box, i.e.

\[ \frac{\mathcal{F}(s, s^*)}{N(s^*)} \]
where

\[
F(s, s^*) = \frac{1}{N} \sum_{i=1}^{N} \sqrt{(s_{i,x} - s_{i,x}^*)^2 + (s_{i,y} - s_{i,y}^*)^2}
\]

where \(s\) and \(s^*\) are the final and ground truth shapes, respectively. \((s_{i,x}, s_{i,y})\) are the x and y coordinates of the \(i\)’th point of the final shape, \((s_{i,x}^*, s_{i,y}^*)\) are the x and y coordinates of the \(i\)’th point of the ground truth shape and \(N\) is the total number of points. Finally, \(N(s^*)\) is a normalising function that returns a measure based on the ground truth shape’s bounding box.

**Parameters**

- **shape** (`menpo.shape.PointCloud`) – The input shape (e.g. the final shape of a fitting procedure).
- **gt_shape** (`menpo.shape.PointCloud`) – The ground truth shape.
- **norm_shape** (`menpo.shape.PointCloud` or `None`, optional) – The shape to be used to compute the normaliser. If `None`, then the ground truth shape is used.
- **norm_type** (`{'area', 'perimeter', 'avg_edge_length', 'diagonal'}`, optional) – The type of the normaliser. Possible options are:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bb_area</td>
<td>Area of norm_shape’s bounding box</td>
</tr>
<tr>
<td>bb_perimeter</td>
<td>Perimeter of norm_shape’s bounding box</td>
</tr>
<tr>
<td>bb_avg_edge_length</td>
<td>Average edge length of norm_shape’s bbox</td>
</tr>
<tr>
<td>bb_diagonal</td>
<td>Diagonal of norm_shape’s bounding box</td>
</tr>
</tbody>
</table>

**Returns** `error (float)` – The computed Euclidean normalised error.

**euclidean_distance_normalised_error**

`menpo.fit.error.euclidean_distance_normalised_error(shape, gt_shape, distance_norm_f)`

Computes the Euclidean error between two shapes normalised by a distance measure between two shapes, i.e.

\[
\frac{F(s, s^*)}{N(s, s^*)}
\]

where

\[
F(s, s^*) = \frac{1}{N} \sum_{i=1}^{N} \sqrt{(s_{i,x} - s_{i,x}^*)^2 + (s_{i,y} - s_{i,y}^*)^2}
\]

where \(s\) and \(s^*\) are the final and ground truth shapes, respectively. \((s_{i,x}, s_{i,y})\) are the x and y coordinates of the \(i\)’th point of the final shape, \((s_{i,x}^*, s_{i,y}^*)\) are the x and y coordinates of the \(i\)’th point of the ground truth shape and \(N\) is the total number of points. Finally, \(N(s, s^*)\) is a normalising function based on a distance metric between the two shapes.

**Parameters**

- **shape** (`menpo.shape.PointCloud`) – The input shape (e.g. the final shape of a fitting procedure).
- **gt_shape** (`menpo.shape.PointCloud`) – The ground truth shape.
• **distance_norm_f** *(callable)* – The function to be used for computing the normalisation distance metric.

**Returns** error *(float)* – The computed Euclidean normalised error.

**euclidean_distance_indexed_normalised_error**

`menpofit.error.euclidean_distance_indexed_normalised_error(shape, gt_shape, index1, index2)`

Computes the Euclidean error between two shapes normalised by the distance measure between two points of the ground truth shape, i.e.

\[
\mathcal{F}(s, s^*) \frac{N(s^*)}{N(s)}
\]

where

\[
\mathcal{F}(s, s^*) = \frac{1}{N} \sum_{i=1}^{N} \sqrt{(s_{i,x} - s_{i,x}^*)^2 + (s_{i,y} - s_{i,y}^*)^2}
\]

where \(s\) and \(s^*\) are the final and ground truth shapes, respectively. \((s_{i,x}, s_{i,y})\) are the \(x\) and \(y\) coordinates of the \(i\)’th point of the final shape, \((s_{i,x}^*, s_{i,y}^*)\) are the \(x\) and \(y\) coordinates of the \(i\)’th point of the ground truth shape and \(N\) is the total number of points. Finally, \(N(s^*)\) is a normalising function that returns the distance between two points of the ground truth shape.

**Parameters**

- **shape** *(menpo.shape.PointCloud)* – The input shape (e.g. the final shape of a fitting procedure).
- **gt_shape** *(menpo.shape.PointCloud)* – The ground truth shape.
- **index1** *(int)* – The index of the first point.
- **index2** *(int)* – The index of the second point.

**Returns** error *(float)* – The computed Euclidean normalised error.

**Statistical Measures**

Functions that compute statistical measures given a set of errors for multiple images.

**compute_cumulative_error**

`menpofit.error.compute_cumulative_error(errors, bins)`

Computes the values of the Cumulative Error Distribution (CED).

**Parameters**

- **errors** *(list of float)* – The list of errors per image.
- **bins** *(list of float)* – The values of the error bins centers at which the CED is evaluated.

**Returns** ced *(list of float)* – The computed CED.
area_under_curve_and_failure_rate

menpofit.error.area_under_curve_and_failure_rate(errors, step_error, max_error, min_error=0.0)

Computes the Area Under the Curve (AUC) and Failure Rate (FR) of a given Cumulative Distribution Error (CED).

Parameters

- **errors** (list of float) – The list of errors per image.
- **step_error** (float) – The sampling step of the error bins of the CED.
- **max_error** (float) – The maximum error value of the CED.
- **min_error** (float) – The minimum error value of the CED.

Returns

- **auc** (float) – The Area Under the Curve value.
- **fr** (float) – The Failure Rate value.

mad

menpofit.error.mad(errors)

Computes the Median Absolute Deviation of a set of errors.

Parameters **errors** (list of float) – The list of errors per image.

Returns **mad** (float) – The median absolute deviation value.

compute_statistical_measures

menpofit.error.compute_statistical_measures(errors, step_error, max_error, min_error=0.0)

Computes various statistics given a set of errors that correspond to multiple images. It can also deal with multiple sets of errors that correspond to different methods.

Parameters

- **errors** (list of float or list of list of float) – The list of errors per image. You can provide a list of lists for the errors of multiple methods.
- **step_error** (float) – The sampling step of the error bins of the CED for computing the Area Under the Curve and the Failure Rate.
- **max_error** (float) – The maximum error value of the CED for computing the Area Under the Curve and the Failure Rate.
- **min_error** (float) – The minimum error value of the CED for computing the Area Under the Curve and the Failure Rate.

Returns

- **mean** (float or list of float) – The mean value.
- **mean** (float or list of float) – The standard deviation.
- **median** (float or list of float) – The median value.
- **mad** (float or list of float) – The mean absolute deviation value.
• **max** *(float or list of float)* – The maximum value.
• **auc** *(float or list of float)* – The area under the curve value.
• **fr** *(float or list of float)* – The failure rate value.

### Object-Specific Errors

Error functions for specific objects.

**Face**

**bb_avg_edge_length_68_euclidean_error**

```python
menpofit.error.bb_avg_edge_length_68_euclidean_error(shape, gt_shape)
```

Computes the Euclidean error based on 68 points normalised by the average edge length of the 68-point ground truth shape's bounding box, i.e.

\[
\frac{\mathcal{F}(s, s^*)}{\mathcal{N}(s^*)}
\]

where

\[
\mathcal{F}(s, s^*) = \frac{1}{68} \sum_{i=1}^{68} \sqrt{(s_{i,x} - s^*_{i,x})^2 + (s_{i,y} - s^*_{i,y})^2}
\]

where \(s\) and \(s^*\) are the final and ground truth shapes, respectively. \((s_{i,x}, s_{i,y})\) are the \(x\) and \(y\) coordinates of the \(i\)’th point of the final shape, \((s^*_{i,x}, s^*_{i,y})\) are the \(x\) and \(y\) coordinates of the \(i\)’th point of the ground truth shape. Finally, \(\mathcal{N}(s^*)\) is a normalising function that returns the average edge length of the bounding box of the 68-point ground truth shape (**bb_avg_edge_length**).

**Parameters**

- **shape** (*menpo.shape.PointCloud*) – The input shape (e.g. the final shape of a fitting procedure). It must have 68 points.
- **gt_shape** (*menpo.shape.PointCloud*) – The ground truth shape. It must have 68 points.

**Returns** **normalised_error** *(float)* – The computed Euclidean normalised error.

**Raises**

- **ValueError** – Final shape must have 68 points
- **ValueError** – Ground truth shape must have 68 points

**bb_avg_edge_length_49_euclidean_error**

```python
menpofit.error.bb_avg_edge_length_49_euclidean_error(shape, gt_shape)
```

Computes the Euclidean error based on 49 points normalised by the average edge length of the 68-point ground truth shape's bounding box, i.e.

\[
\frac{\mathcal{F}(s, s^*)}{\mathcal{N}(s^*)}
\]
where

\[ F(s, s^*) = \frac{1}{49} \sum_{i=1}^{49} \sqrt{(s_i, x - s^*_i, x)^2 + (s_i, y - s^*_i, y)^2} \]

where \( s \) and \( s^* \) are the final and ground truth shapes, respectively. \((s_i, x, s_i, y)\) are the \( x \) and \( y \) coordinates of the \( i \)’th point of the final shape, \((s^*_i, x, s^*_i, y)\) are the \( x \) and \( y \) coordinates of the \( i \)’th point of the ground truth shape. Finally, \( N(s) \) is a normalising function that returns the average edge length of the bounding box of the 68-point ground truth shape (\( bb\_avg\_edge\_length \)).

**Parameters**

- **shape** (*menpo.shape.PointCloud*) – The input shape (e.g. the final shape of a fitting procedure). It must have 68 or 66 or 51 or 49 points.
- **gt_shape** (*menpo.shape.PointCloud*) – The ground truth shape. It must have 68 points.

**Returns**

- **normalised_error** (*float*) – The computed Euclidean normalised error.

**Raises**

- **ValueError** – Final shape must have 68 or 51 or 49 points
- **ValueError** – Ground truth shape must have 68 points

### mean_pupil_68_error

**menpofit.error.mean_pupil_68_error(shape, gt_shape)**

Computes the Euclidean error based on 68 points normalised with the distance between the mean eye points (pupils), i.e.

\[ \frac{F(s, s^*)}{N(s)} \]

where

\[ F(s, s^*) = \frac{1}{68} \sum_{i=1}^{68} \sqrt{(s_i, x - s^*_i, x)^2 + (s_i, y - s^*_i, y)^2} \]

where \( s \) and \( s^* \) are the final and ground truth shapes, respectively. \((s_i, x, s_i, y)\) are the \( x \) and \( y \) coordinates of the \( i \)’th point of the final shape, \((s^*_i, x, s^*_i, y)\) are the \( x \) and \( y \) coordinates of the \( i \)’th point of the ground truth shape. Finally, \( N(s) \) is the distance between the mean eye points (pupils).

**Parameters**

- **shape** (*menpo.shape.PointCloud*) – The input shape (e.g. the final shape of a fitting procedure). It must have 68 points.
- **gt_shape** (*menpo.shape.PointCloud*) – The ground truth shape. It must have 68 points.

**Returns**

- **normalised_error** (*float*) – The computed normalised Euclidean error.

**Raises**

- **ValueError** – Final shape must have 68 points
- **ValueError** – Ground truth shape must have 68 points
mean_pupil_49_error

`menpofit.error.mean_pupil_49_error(shape, gt_shape)`

Computes the euclidean error based on 49 points normalised with the distance between the mean eye points (pupils), i.e.

\[
\frac{\mathcal{F}(s, s^*)}{\mathcal{N}(s)}
\]

where

\[
\mathcal{F}(s, s^*) = \frac{1}{49} \sum_{i=1}^{49} \sqrt{(s_{i,x} - s_{i,x}^*)^2 + (s_{i,y} - s_{i,y}^*)^2}
\]

where \( s \) and \( s^* \) are the final and ground truth shapes, respectively. \((s_{i,x}, s_{i,y})\) are the \(i\)'th point of the final shape, \((s_{i,x}^*, s_{i,y}^*)\) are the \(i\)'th point of the ground truth shape. Finally, \(\mathcal{N}(s)\) is the distance between the mean eye points (pupils).

**Parameters**

- `shape` (*menpo.shape.PointCloud*) – The input shape (e.g. the final shape of a fitting procedure). It must have either 68 or 66 or 51 or 49 points.
- `gt_shape` (*menpo.shape.PointCloud*) – The ground truth shape. It must have either 68 or 66 or 51 or 49 points.

**Returns**

`normalised_error (float)` – The computed normalised Euclidean error.

**Raises**

- `ValueError` – Final shape must have 68 or 66 or 51 or 49 points
- `ValueError` – Ground truth shape must have 68 or 66 or 51 or 49 points

outer_eye_corner_68_euclidean_error

`menpofit.error.outer_eye_corner_68_euclidean_error(shape, gt_shape)`

Computes the Euclidean error based on 68 points normalised with the distance between the mean eye points (pupils), i.e.

\[
\frac{\mathcal{F}(s, s^*)}{\mathcal{N}(s^*)}
\]

where

\[
\mathcal{F}(s, s^*) = \frac{1}{68} \sum_{i=1}^{68} \sqrt{(s_{i,x} - s_{i,x}^*)^2 + (s_{i,y} - s_{i,y}^*)^2}
\]

where \( s \) and \( s^* \) are the final and ground truth shapes, respectively. \((s_{i,x}, s_{i,y})\) are the \(i\)'th point of the final shape, \((s_{i,x}^*, s_{i,y}^*)\) are the \(i\)'th point of the ground truth shape. Finally, \(\mathcal{N}(s^*)\) is the distance between the 36-th and 45-th points.

**Parameters**

- `shape` (*menpo.shape.PointCloud*) – The input shape (e.g. the final shape of a fitting procedure). It must have 68 points.
- `gt_shape` (*menpo.shape.PointCloud*) – The ground truth shape. It must have 68 points.

**Returns**

`normalised_error (float)` – The computed normalised Euclidean error.
Raises

- `ValueError` – Final shape must have 68 points
- `ValueError` – Ground truth shape must have 68 points

`outer_eye_corner_51_euclidean_error`

Computes the Euclidean error based on 51 points normalised with the distance between the mean eye points (pupils), i.e.

\[
\frac{\mathcal{F}(s, s^*)}{\mathcal{N}(s^*)}
\]

where

\[
\mathcal{F}(s, s^*) = \frac{1}{51} \sum_{i=1}^{51} \sqrt{(s_{i,x} - s_{i,x}^*)^2 + (s_{i,y} - s_{i,y}^*)^2}
\]

where \( s \) and \( s^* \) are the final and ground truth shapes, respectively. \((s_{i,x}, s_{i,y})\) are the \( x \) and \( y \) coordinates of the \( i \)’th point of the final shape, \((s_{i,x}^*, s_{i,y}^*)\) are the \( x \) and \( y \) coordinates of the \( i \)’th point of the ground truth shape. Finally, \( \mathcal{N}(s^*) \) is the distance between the 19-th and 28-th points.

Parameters

- `shape` ([`menpo.shape.PointCloud`]) – The input shape (e.g. the final shape of a fitting procedure). It must have 68 or 51 points.
- `gt_shape` ([`menpo.shape.PointCloud`]) – The ground truth shape. It must have 68 or 51 points.

Returns `normalised_error` (float) – The computed normalised Euclidean error.

Raises

- `ValueError` – Final shape must have 68 or 51 points
- `ValueError` – Ground truth shape must have 68 or 51 points

`outer_eye_corner_49_euclidean_error`

Computes the Euclidean error based on 49 points normalised with the distance between the mean eye points (pupils), i.e.

\[
\frac{\mathcal{F}(s, s^*)}{\mathcal{N}(s^*)}
\]

where

\[
\mathcal{F}(s, s^*) = \frac{1}{49} \sum_{i=1}^{49} \sqrt{(s_{i,x} - s_{i,x}^*)^2 + (s_{i,y} - s_{i,y}^*)^2}
\]

where \( s \) and \( s^* \) are the final and ground truth shapes, respectively. \((s_{i,x}, s_{i,y})\) are the \( x \) and \( y \) coordinates of the \( i \)’th point of the final shape, \((s_{i,x}^*, s_{i,y}^*)\) are the \( x \) and \( y \) coordinates of the \( i \)’th point of the ground truth shape. Finally, \( \mathcal{N}(s^*) \) is the distance between the 19-th and 28-th points.
Parameters

- **shape** (*menpo.shape.PointCloud*) – The input shape (e.g. the final shape of a fitting procedure). It must be 68 or 66 or 51 or 49 points.

- **gt_shape** (*menpo.shape.PointCloud*) – The ground truth shape. It must have 68 or 66 or 51 or 49 points.

Returns **normalised_error** (*float*) – The computed normalised Euclidean error.

Raises

- **ValueError** – Final shape must have 68 or 66 or 51 or 49 points

- **ValueError** – Ground truth shape must have 68 or 66 or 51 or 49 points

### 2.2.5 menpo.fit.fitter

Fitter Classes

**MultiScaleNonParametricFitter**

```python
class menpo.fit.fitter.MultiScaleNonParametricFitter(scales, reference_shape, holistic_features, algorithms)
```

Bases: object

Class for defining a multi-scale fitter for a non-parametric fitting method, i.e. a method that does not optimise over a parametric shape model.

Parameters

- **scales** (*list of int or float*) – The scale value of each scale. They must be provided in ascending order, i.e. from lowest to highest scale.

- **reference_shape** (*menpo.shape.PointCloud*) – The reference shape that will be used to normalise the size of an input image so that the scale of its initial fitting shape matches the scale of the reference shape.

- **holistic_features** (*list of closure*) – The features that will be extracted from the input image at each scale. They must be provided in ascending order, i.e. from lowest to highest scale.

- **algorithms** (*list of class*) – The list of algorithm objects that will perform the fitting per scale.

**fit_from_bb** (*image, bounding_box, max_iters=20, gt_shape=None, return_costs=False, **kwargs*)

Fits the multi-scale fitter to an image given an initial bounding box.

Parameters

- **image** (*menpo.image.Image or subclass*) – The image to be fitted.

- **bounding_box** (*menpo.shape.PointDirectedGraph*) – The initial bounding box from which the fitting procedure will start. Note that the bounding box is used in order to align the model’s reference shape.

- **max_iters** (*int or list of int, optional*) – The maximum number of iterations. If int, then it specifies the maximum number of iterations over all scales. If list of int, then specifies the maximum number of iterations per scale.

- **gt_shape** (*menpo.shape.PointCloud, optional*) – The ground truth shape associated to the image.
• **return_costs** *(bool, optional)* – If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned `fitting_result`. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.

• **kwargs** *(dict, optional)* – Additional keyword arguments that can be passed to specific implementations.

**Returns**

`fitting_result` *(MultiScaleNonParametricIterativeResult or subclass)* – The multi-scale fitting result containing the result of the fitting procedure.

**fit_from_shape**(image, initial_shape, max_iters=20, gt_shape=None, return_costs=False, **kwargs)

Fits the multi-scale fitter to an image given an initial shape.

**Parameters**

• **image** *(menpo.image.Image or subclass)* – The image to be fitted.

• **initial_shape** *(menpo.shape.PointCloud)* – The initial shape estimate from which the fitting procedure will start.

• **max_iters** *(int or list of int, optional)* – The maximum number of iterations. If int, then it specifies the maximum number of iterations over all scales. If list of int, then specifies the maximum number of iterations per scale.

• **gt_shape** *(menpo.shape.PointCloud, optional)* – The ground truth shape associated to the image.

• **return_costs** *(bool, optional)* – If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned `fitting_result`. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.

• **kwargs** *(dict, optional)* – Additional keyword arguments that can be passed to specific implementations.

**Returns**

`fitting_result` *(MultiScaleNonParametricIterativeResult or subclass)* – The multi-scale fitting result containing the result of the fitting procedure.

**property holistic_features**

The features that are extracted from the input image at each scale in ascending order, i.e. from lowest to highest scale.

Type **list of closure**

**property n_scales**

Returns the number of scales.

Type **int**

**property reference_shape**

The reference shape that is used to normalise the size of an input image so that the scale of its initial fitting shape matches the scale of this reference shape.

Type **menpo.shape.PointCloud**

**property scales**

The scale value of each scale in ascending order, i.e. from lowest to highest scale.

Type **list of int or float**
MultiScaleParametricFitter

```python
class menpo.fit.fitter.MultiScaleParametricFitter(scales, reference_shape, holistic_features, algorithms)
Bases: MultiScaleNonParametricFitter
```

Class for defining a multi-scale fitter for a parametric fitting method, i.e. a method that optimises over the parameters of a statistical shape model.

**Note:** When using a method with a parametric shape model, the first step is to **reconstruct the initial shape** using the shape model. The generated reconstructed shape is then used as initialisation for the iterative optimisation. This step takes place at each scale and it is not considered as an iteration, thus it is not counted for the provided `max_iters`.

**Parameters**

- `scales (list of int or float)` – The scale value of each scale. They must provided in ascending order, i.e. from lowest to highest scale.
- `reference_shape (menpo.shape.PointCloud)` – The reference shape that will be used to normalise the size of an input image so that the scale of its initial fitting shape matches the scale of the reference shape.
- `holistic_features (list of closure)` – The features that will be extracted from the input image at each scale. They must provided in ascending order, i.e. from lowest to highest scale.
- `algorithms (list of class)` – The list of algorithm objects that will perform the fitting per scale.

**fit_from_bb(image, bounding_box, max_iters=20, gt_shape=None, return_costs=False, **kwargs)**

Fits the multi-scale fitter to an image given an initial bounding box.

**Parameters**

- `image (menpo.image.Image or subclass)` – The image to be fitted.
- `bounding_box (menpo.shape.PointDirectedGraph)` – The initial bounding box from which the fitting procedure will start. Note that the bounding box is used in order to align the model’s reference shape.
- `max_iters (int or list of int, optional)` – The maximum number of iterations. If `int`, then it specifies the maximum number of iterations over all scales. If `list of int`, then specifies the maximum number of iterations per scale.
- `gt_shape (menpo.shape.PointCloud, optional)` – The ground truth shape associated to the image.
- `return_costs (bool, optional)` – If `True`, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned `fitting_result`. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.
- `kwargs (dict, optional)` – Additional keyword arguments that can be passed to specific implementations.

**Returns**

- `fitting_result (MultiScaleNonParametricIterativeResult or subclass)` – The multi-scale fitting result containing the result of the fitting procedure.
**fit_from_shape** *(image, initial_shape, max_iters=20, gt_shape=None, return_costs=False, \*\*kwargs)*

Fits the multi-scale fitter to an image given an initial shape.

**Parameters**

- **image** *(menpo.image.Image or subclass)* – The image to be fitted.
- **initial_shape** *(menpo.shape.PointCloud)* – The initial shape estimate from which the fitting procedure will start.
- **max_iters** *(int or list of int, optional)* – The maximum number of iterations. If int, then it specifies the maximum number of iterations over all scales. If list of int, then specifies the maximum number of iterations per scale.
- **gt_shape** *(menpo.shape.PointCloud, optional)* – The ground truth shape associated to the image.
- **return_costs** *(bool, optional)* – If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.
- **kwargs** *(dict, optional)* – Additional keyword arguments that can be passed to specific implementations.

**Returns**

- **fitting_result** *(MultiScaleNonParametricIterativeResult or subclass)* – The multi-scale fitting result containing the result of the fitting procedure.

**property holistic_features**

The features that are extracted from the input image at each scale in ascending order, i.e. from lowest to highest scale.

**Type** list of closure

**property n_scales**

Returns the number of scales.

**Type** int

**property reference_shape**

The reference shape that is used to normalise the size of an input image so that the scale of its initial fitting shape matches the scale of this reference shape.

**Type** menpo.shape.PointCloud

**property scales**

The scale value of each scale in ascending order, i.e. from lowest to highest scale.

**Type** list of int or float

**Perturb Functions**

Collection of functions that perform a kind of perturbation on a shape or bounding box.
**align_shape_with_bounding_box**

```python
menpo.fitter.align_shape_with_bounding_box(shape, bounding_box, alignment_transform_cls=<class 'menpo.transform.homogeneous.similarity.AlignmentSimilarity'>, **kwargs)
```

Aligns the provided shape with the bounding box using a particular alignment transform.

**Parameters**

- `shape` (*menpo.shape.PointCloud*) – The shape instance used in the alignment.
- `bounding_box` (*menpo.shape.PointDirectedGraph*) – The bounding box instance used in the alignment.
- `alignment_transform_cls` (*menpo.transform.Alignment*, optional) – The class of the alignment transform used to perform the alignment.

**Returns**

`noisy_shape` (*menpo.shape.PointCloud*) – The noisy shape

**generate_perturbations_from_gt**

```python
menpo.fitter.generate_perturbations_from_gt(images, n_perturbations, perturb_func, gt_group=None, bb_group_glob=None, verbose=False)
```

Function that returns a callable that generates perturbations of the bounding boxes of the provided images.

**Parameters**

- `images` (*list of menpo.image.Image*) – The list of images.
- `n_perturbations` (*int*) – The number of perturbed shapes to be generated per image.
- `perturb_func` (*callable*) – The function that will be used for generating the perturbations.
- `gt_group` (*str*) – The group of the ground truth shapes attached to the images.
- `bb_group_glob` (*str*) – The group of the bounding boxes attached to the images.
- `verbose` (*bool*, optional) – If `True`, then progress information is printed.

**Returns**

`generated_bb_func` (*callable*) – The function that generates the perturbations.

**noisy_alignment_similarity_transform**

```python
menpo.fitter.noisy_alignment_similarity_transform(source, target, noise_type='uniform', noise_percentage=0.1, allow_alignment_rotation=False)
```

Constructs and perturbs the optimal similarity transform between the source and target shapes by adding noise to its parameters.

**Parameters**

- `source` (*menpo.shape.PointCloud*) – The source pointcloud instance used in the alignment
- `target` (*menpo.shape.PointCloud*) – The target pointcloud instance used in the alignment
- `noise_type` (*['uniform', 'gaussian']*, optional) – The type of noise to be added.
MenpoFit Documentation, Release 0.6.1

- **noise_percentage** *(float in (0, 1) or list of len 3, optional)* – The standard percentage of noise to be added. If float, then the same amount of noise is applied to the scale, rotation and translation parameters of the optimal similarity transform. If list of float it must have length 3, where the first, second and third elements denote the amount of noise to be applied to the scale, rotation and translation parameters, respectively.

- **allow_alignment_rotation** *(bool, optional)* – If False, then the rotation is not considered when computing the optimal similarity transform between source and target.


noisy_shape_from_bounding_box

menpo.fit.noisy_shape_from_bounding_box(shape, bounding_box, noise_type='uniform', noise_percentage=0.05, allow_alignment_rotation=False)

Constructs and perturbs the optimal similarity transform between the bounding box of the source shape and the target bounding box, by adding noise to its parameters. It returns the noisy version of the provided shape.

**Parameters**

- **shape** *(menpo.shape.PointCloud)* – The source pointcloud instance used in the alignment. Note that the bounding box of the shape will be used.

- **bounding_box** *(menpo.shape.PointDirectedGraph)* – The target bounding box instance used in the alignment

- **noise_type** *(('uniform', 'gaussian'), optional)* – The type of noise to be added.

- **noise_percentage** *(float in (0, 1) or list of len 3, optional)* – The standard percentage of noise to be added. If float, then the same amount of noise is applied to the scale, rotation and translation parameters of the optimal similarity transform. If list of float it must have length 3, where the first, second and third elements denote the amount of noise to be applied to the scale, rotation and translation parameters, respectively.

- **allow_alignment_rotation** *(bool, optional)* – If False, then the rotation is not considered when computing the optimal similarity transform between source and target.

**Returns** noisy_shape *(menpo.shape.PointCloud)* – The noisy shape.

noisy_shape_from_shape

menpo.fit.noisy_shape_from_shape(reference_shape, shape, noise_type='uniform', noise_percentage=0.05, allow_alignment_rotation=False)

Constructs and perturbs the optimal similarity transform between the provided reference shape and the target shape, by adding noise to its parameters. It returns the noisy version of the reference shape.

**Parameters**

- **reference_shape** *(menpo.shape.PointCloud)* – The source reference shape instance used in the alignment.

- **shape** *(menpo.shape.PointDirectedGraph)* – The target shape instance used in the alignment
MenpoFit Documentation, Release 0.6.1

• **noise_type** ("uniform", 'gaussian', optional) – The type of noise to be added.

• **noise_percentage** (float in (0, 1) or list of len 3, optional) – The standard percentage of noise to be added. If float, then the same amount of noise is applied to the scale, rotation and translation parameters of the optimal similarity transform. If list of float it must have length 3, where the first, second and third elements denote the amount of noise to be applied to the scale, rotation and translation parameters, respectively.

• **allow_alignment_rotation** (bool, optional) – If False, then the rotation is not considered when computing the optimal similarity transform between source and target.


### noisy_target_alignment_transform

```python
menpofit.fitter.noisy_target_alignment_transform(source, target, alignment_transform_cls=<class 'menpo.transform.homogeneous.affine.AlignmentAffine'>, noise_std=0.1, **kwargs)
```

Constructs the optimal alignment transform between the source and a noisy version of the target obtained by adding white noise to each of its points.

**Parameters**

• **source** (menpo.shape.PointCloud) – The source pointcloud instance used in the alignment

• **target** (menpo.shape.PointCloud) – The target pointcloud instance used in the alignment

• **alignment_transform_cls** (menpo.transform.Alignment, optional) – The alignment transform class used to perform the alignment.

• **noise_std** (float or list of float, optional) – The standard deviation of the white noise to be added to each one of the target points. If float, then the same standard deviation is used for all points. If list, then it must define a value per point.

Returns **noisy_transform** (menpo.transform.Alignment) – The noisy Similarity Transform

### 2.2.6 menpofit.io

Menpofit includes the ability to save and load pre-trained models for specific tasks. This module contains code for pickling down, downloading, and restoring fitters efficiently.

If you make use of one of menpofit’s pre-trained models, you will find that the type that is provided to you is the **PickleWrappedFitter**. See it’s documentation to understand it’s purpose and how you can effectively use it.

**PickleWrappedFitter**

```python
class menpofit.io.PickleWrappedFitter(fitter_cls, fitter_args, fitter_kwags, fit_from_bb_kwargs, fit_from_shape_kwargs, image_preprocess=<function image_greyscale_crop_preprocess>)
```

Bases: object

Wrapper around a menpofit fitter so that we can a) efficiently pickle it and b) parametrize over both the fitter construction and the fit methods (e.g. `.fit_from_bb()` and `.fit_from_shape()`)

2.2. Internal API 323
Pickling menpofit fitters is a little tricky for a few reasons. Firstly, on construction of a fitter from a deformable model some amount of pre-processing takes place which allocates potentially large arrays. To ship a compact model we would therefore rather delay the construction of the fitter until load time on the client.

If this was the only issue, we could achieve this by simply saving a partial over the fitter constructor with all the args and kwargs the fitter constructor takes - after loading the pickle, invoking the partial with no args (it’s parameters being fully specified) would return the fitter and all would be well.

However, we also may want to choose fit-time parameters for the fitter for optimal usage, (for instance, a choice over the max_iters kwarg that we know to be efficient). This leaves us with a problem, as now we need to have some entity that can store state which we can pass to both the fitter and to the resulting fitters methods on the client at unpickle time.

This class is the solution to this problem. To use, you should pickle down a partial over this class specifying all arguments and kwargs needed for the fitter constructor and for the fit methods.

At load time, menpofit will invoke the partial, returning this object instantiated. This offers the same API as a menpofit fitter, and so can be used transparently to fit. If you wish to access the original fitter (without fit parameter customization) this can be accessed as the wrapped_fitter property.

**Parameters**

- `fitter_cls` *(Fitter)* - A menpofit fitter class that will be constructed at unpickle time, e.g. LucasKanadeAAMFitter
- `fitter_args` *(tuple)* - A tuple of all args that need to be passed to `fitter_cls` at construction time e.g. `(aam,)`
- `fitter_kwargs` *(dict)* - A dictionary of kwargs that will to be passed to `fitter_cls` at construction time e.g. `{ 'lk_algorithm_cls': WibergInverseCompositional }`
- `fit_from_bb_kwargs` *(dict)* - A dictionary of kwargs that will to be passed to `fitter_cls` at fit time. These in effect change the defaults that the original fitter offered, but can still be overridden at call time (e.g. `self.fit_from_bb(image, bbox, max_iters=[50, 50])` would take precedence over the max_iters in the above example)
- `fit_from_shape_kwargs` *(dict)* - A dictionary of kwargs that will to be passed to the wrapped fitter’s `fit_from_shape` method at fit time. These in effect change the defaults that the original fitter offered, but can still be overridden at call time (e.g. `self.fit_from_shape(image, shape, max_iters=[50, 50])` would take precedence over the max_iters in the above example)
- `image_preprocess` *(callable or None, optional)* - A pre-processing function to apply on the test image before fitting. The default option converts the image to greyscale. The function needs to have the following signature:

```python
new_image, transform = image_preprocess(image, pointcloud)
```

where `new_image` is the pre-processed image and `transform` is the `menpo.transform.Homogeneous` object that was applied on the image. If `None`, then no pre-processing is performed.

**Examples**

```python
from menpofit.io import PickleWrappedFitter, image_greyscale_crop_preprocess
from functools import partial
```
# LucasKanadeAAMFitter only takes one argument, a trained aam.
fitter_args = (aam, )

# kwargs for fitter construction. Note that here sampling is a
# list of numpy arrays we have already constructed (one per level)
fitter_kwargs = dict(lk_algorithm_cls=WibergInverseCompositional,
sampling=sampling)

# kwargs for fitter.fit_from_{bb, shape}
# (note here we reuse the same kwargs twice)
fit_kwargs = dict(max_iters=[25, 5])

# Partial over the PickleWrappedFitter to prepare an object that can be
# invoked at load time
fitter_wrapper = partial(PickleWrappedFitter, LucasKanadeAAMFitter,
fitter_args, fitter_kwargs,
fit_kwargs, fit_kwargs,
image_preprocess=image_greyscale_crop_preprocess)

# save the pickle down.
mio.export_pickle(fitter_wrapper, 'pretrained_aam.pkl')

# ----------------------- L O A D T I M E ---------------------------#
# at load time, invoke the partial to instantiate this class (and build
# the internally-held wrapped fitter)
fitter = mio.import_pickle('pretrained_aam.pkl')()

fit_from_bb(image, bounding_box, **kwargs)
Fits the fitter to an image given an initial bounding box, using the optimal parameters that we chosen for
this pickled fitter.

Parameters

- **image** (menpo.image.Image or subclass) – The image to be fitted.
- **bounding_box** (menpo.shape.PointDirectedGraph) – The initial bounding box from
  which the fitting procedure will start. Note that the bounding box is used in order to align
  the model’s reference shape.
- **kwargs** (dict, optional) – Other kwargs to override the optimal defaults. See the docu-
  mentation for .fit_from_bb() on the type of self.wrapped_fitter to see what can be
  provided here.

Returns fitting_result (FittingResult or subclass) – The fitting result containing the result
of the fitting procedure.

fit_from_shape(image, initial_shape, **kwargs)
Fits the fitter to an image given an initial shape, using the optimal parameters that we chosen for this
pickled fitter.

Parameters

- **image** (menpo.image.Image or subclass) – The image to be fitted.
- **initial_shape** (menpo.shape.PointCloud) – The initial shape estimate from which
  the fitting procedure will start.
• `kwargs (dict)` – Other kwargs to override the optimal defaults. See the documentation for `.fit_from_shape()` on the type of `self.wrapped_fitter` to see what can be provided here.

Returns `fitting_result` (`FittingResult` or subclass) – The fitting result containing the result of the fitting procedure.

### 2.2.7 menpofit.math

**Regression**

**IRLRegression**

class menpofit.math.IRLRegression

Bases: object

Class for training and applying Incremental Regularized Linear Regression.

Parameters

- `alpha` (`float`, optional) – The regularization parameter of the features.
- `bias` (`bool`, optional) – If True, a bias term is used.
- `incrementable` (`bool`, optional) – If True, then the regression model will have the ability to get incremented.

increment `(X, Y)`

Incrementally update the regression model.

Parameters

- `X` ((`n_features`, `n_samples`) `ndarray`) – The array of feature vectors.
- `Y` ((`n_dims`, `n_samples`) `ndarray`) – The array of target vectors.

Raises `ValueError` – Model is not incrementable

predict `(x)`

Makes a prediction using the trained regression model.

Parameters

- `x` ((`n_features`,) `ndarray`) – The input feature vector.

Returns `prediction` ((`n_dims`,) `ndarray`) – The prediction vector.

train `(X, Y)`

Train the regression model.

Parameters

- `X` ((`n_features`, `n_samples`) `ndarray`) – The array of feature vectors.
- `Y` ((`n_dims`, `n_samples`) `ndarray`) – The array of target vectors.

**IIRLRegression**

class menpofit.math.IIRLRegression

Bases: IRLRegression

Class for training and applying Indirect Incremental Regularized Linear Regression.

Parameters
• **alpha** (*float*, optional) – The regularization parameter.

• **bias** (*bool*, optional) – If True, a bias term is used.

• **alpha2** (*float*, optional) – The regularization parameter of the Hessian.

**increment** (*X, Y*)
Incrementally update the regression model.

**Parameters**

• **X** (*((n_features, n_samples) ndarray*) – The array of feature vectors.

• **Y** (*((n_dims, n_samples) ndarray*) – The array of target vectors.

**Raises** **ValueError** – Model is not incrementable

**predict** (*x*)
Makes a prediction using the trained regression model.

**Parameters**

• **x** (*((n_features,) ndarray*) – The input feature vector.

**Returns** **prediction** (*((n_dims,) ndarray*) – The prediction vector.

**train** (*X, Y*)
Train the regression model.

**Parameters**

• **X** (*((n_features, n_samples) ndarray*) – The array of feature vectors.

• **Y** (*((n_dims, n_samples) ndarray*) – The array of target vectors.

**PCRRegression**

**class** menpofit.math.PCRRegression (*variance=None, bias=True*)

**Bases:** object

Class for training and applying Multivariate Linear Regression using Principal Component Regression.

**Parameters**

• **variance** (*float* or *None*, optional) – The SVD variance.

• **bias** (*bool*, optional) – If True, a bias term is used.

**increment** (*X, Y*)
Incrementally update the regression model.

**Parameters**

• **X** (*((n_features, n_samples) ndarray*) – The array of feature vectors.

• **Y** (*((n_dims, n_samples) ndarray*) – The array of target vectors.

**Raises** **ValueError** – Model is not incrementable

**predict** (*x*)
Makes a prediction using the trained regression model.

**Parameters**

• **x** (*((n_features,) ndarray*) – The input feature vector.

**Returns** **prediction** (*((n_dims,) ndarray*) – The prediction vector.

**train** (*X, Y*)
Train the regression model.
OptimalLinearRegression

class menpofit.math.OptimalLinearRegression(variance=None, bias=True)
Bases: object

Class for training and applying Multivariate Linear Regression using optimal reconstructions.

Parameters

• variance (float or None, optional) – The SVD variance.
• bias (bool, optional) – If True, a bias term is used.

increment (X, Y)
Incrementally update the regression model.

Parameters

• X ((n_features, n_samples) ndarray) – The array of feature vectors.
• Y ((n_dims, n_samples) ndarray) – The array of target vectors.

Raises ValueError – Model is not incrementable

predict (x)
Makes a prediction using the trained regression model.

Parameters x ((n_features,) ndarray) – The input feature vector.

Returns prediction ((n_dims,) ndarray) – The prediction vector.

train (X, Y)
Train the regression model.

Parameters

• X ((n_features, n_samples) ndarray) – The array of feature vectors.
• Y ((n_dims, n_samples) ndarray) – The array of target vectors.

OPPRegression

class menpofit.math.OPPRegression(bias=True, whiten=False)
Bases: object

Class for training and applying Multivariate Linear Regression using Orthogonal Procrustes Problem reconstructions.

Parameters

• bias (bool, optional) – If True, a bias term is used.
• whiten (bool, optional) – Whether to use a whitened PCA model.

increment (X, Y)
Incrementally update the regression model.

Parameters

• X ((n_features, n_samples) ndarray) – The array of feature vectors.
• Y ((n_dims, n_samples) ndarray) – The array of target vectors.
• X((n_features, n_samples) ndarray) – The array of feature vectors.
• Y((n_dims, n_samples) ndarray) – The array of target vectors.

Raises ValueError – Model is not incrementable

predict(x)
Makes a prediction using the trained regression model.

Parameters
x ((n_features,) ndarray) – The input feature vector.

Returns prediction ((n_dims,) ndarray) – The prediction vector.

train(X, Y)
Train the regression model.

Parameters
• X ((n_features, n_samples) ndarray) – The array of feature vectors.
• Y ((n_dims, n_samples) ndarray) – The array of target vectors.

Correlation Filters

mccf

menpo.math.mccf(X, y, l=0.01, boundary='constant', crop_filter=True)
Multi-Channel Correlation Filter (MCCF).

Parameters
• X ((n_images, n_channels, image_h, image_w) ndarray) – The training images.
• y ((1, response_h, response_w) ndarray) – The desired response.
• l (float, optional) – Regularization parameter.
• boundary ({'constant', 'symmetric'}, optional) – Determines how the image is padded.
• crop_filter (bool, optional) – If True, the shape of the MOSSE filter is the same as the shape of the desired response. If False, the filter’s shape is equal to: X[0].shape + y.shape - 1

Returns
• f ((1, response_h, response_w) ndarray) – Multi-Channel Correlation Filter (MCCF) filter associated to the training images.
• sXY ((N,) ndarray) – The auto-correlation array, where N = (image_h+response_h-1) * (image_w+response_w-1) * n_channels.
• sXX ((N, N) ndarray) – The cross-correlation array, where N = (image_h+response_h-1) * (image_w+response_w-1) * n_channels.

References
**imccf**

```
menpofit.math.imccf(A, B, n_ab, X, y, l=0.01, boundary='constant', crop_filter=True, f=1.0)
```

Incremental Multi-Channel Correlation Filter (MCCF)

**Parameters**

- **A** (*N,* `ndarray`) – The current auto-correlation array, where \( N = (\text{patch}_h+\text{response}_h-1) \times (\text{patch}_w+\text{response}_w-1) \times \text{n_channels}. \)
- **B** (*N, N* `ndarray`) – The current cross-correlation array, where \( N = (\text{patch}_h+\text{response}_h-1) \times (\text{patch}_w+\text{response}_w-1) \times \text{n_channels}. \)
- **n_ab** (*int*) – The current number of images.
- **X** (*n_images, n_channels, image_h, image_w* `ndarray*) – The training images (patches).
- **y** (*1, response_h, response_w* `ndarray*) – The desired response.
- **l** (*float*, optional) – Regularization parameter.
- **boundary** (*{'constant', 'symmetric'}, optional*) – Determines how the image is padded.
- **crop_filter** (*bool*, optional) – If True, the shape of the MOSSE filter is the same as the shape of the desired response. If False, the filter’s shape is equal to: \( X[0].\text{shape} + y.\text{shape} - 1 \)
- **f** (*[0, 1]* `float`, optional) – Forgetting factor that weights the relative contribution of new samples vs old samples. If 1.0, all samples are weighted equally. If <1.0, more emphasis is put on the new samples.

**Returns**

- **f** (*1, response_h, response_w* `ndarray*) – Multi-Channel Correlation Filter (MCCF) filter associated to the training images.
- **sXY** (*N,* `ndarray*) – The auto-correlation array, where \( N = (\text{image}_h+\text{response}_h-1) \times (\text{image}_w+\text{response}_w-1) \times \text{n_channels}. \)
- **sXX** (*N, N* `ndarray*) – The cross-correlation array, where \( N = (\text{image}_h+\text{response}_h-1) \times (\text{image}_w+\text{response}_w-1) \times \text{n_channels}. \)

**mosse**

```
menpofit.math.mosse(X, y, l=0.01, boundary='constant', crop_filter=True)
```

Minimum Output Sum of Squared Errors (MOSSE) filter.

**Parameters**

- **X** (*n_images, n_channels, image_h, image_w* `ndarray*) – The training images.
- **y** (*1, response_h, response_w* `ndarray*) – The desired response.
- **l** (*float*, optional) – Regularization parameter.
• **boundary** ("constant", 'symmetric'), optional) – Determines how the image is padded.

• **crop_filter** (bool, optional) – If True, the shape of the MOSSE filter is the same as the shape of the desired response. If False, the filter’s shape is equal to: `X[0].shape + y.shape - 1`

**Returns**

• f((1, response_h, response_w) ndarray) – Minimum Output Sum of Squared Errors (MOSSE) filter associated to the training images.

• sXY ((N,) ndarray) – The auto-correlation array, where `N = (image_h+response_h-1) * (image_w+response_w-1) * n_channels`.

• sXX ((N, N) ndarray) – The cross-correlation array, where `N = (image_h+response_h-1) * (image_w+response_w-1) * n_channels`.

**References**

**imosse**

menpofit.math.imosse(A, B, n_ab, X, y, l=0.01, boundary='constant', crop_filter=True, f=1.0)

Incremental Minimum Output Sum of Squared Errors (iMOSSE) filter.

**Parameters**

• **A** (N,) ndarray – The current auto-correlation array, where `N = (patch_h+response_h-1) * (patch_w+response_w-1) * n_channels`.

• **B** (N, N) ndarray – The current cross-correlation array, where `N = (patch_h+response_h-1) * (patch_w+response_w-1) * n_channels`.

• **n_ab** (int) – The current number of images.

• **X**((n_images, n_channels, image_h, image_w) ndarray) – The training images (patches).

• **y**((1, response_h, response_w) ndarray) – The desired response.

• **l** (float, optional) – Regularization parameter.

• **boundary** ("constant", 'symmetric'), optional) – Determines how the image is padded.

• **crop_filter** (bool, optional) – If True, the shape of the MOSSE filter is the same as the shape of the desired response. If False, the filter's shape is equal to: `X[0].shape + y.shape - 1`

• **f**([0, 1] float, optional) – Forgetting factor that weights the relative contribution of new samples vs old samples. If 1.0, all samples are weighted equally. If <1.0, more emphasis is put on the new samples.

**Returns**

• f((1, response_h, response_w) ndarray) – Minimum Output Sum of Squared Errors (MOSSE) filter associated to the training images.

• sXY ((N,) ndarray) – The auto-correlation array, where `N = (image_h+response_h-1) * (image_w+response_w-1) * n_channels`.
• \( sXX \) \( ((N, N) \text{ ndarray}) \) – The cross-correlation array, where \( N = (image_h+response_h-1) \times (image_w+response_w-1) \times n\_channels \).

## 2.2.8 menpofit.modelinstance

### Abstract Classes

#### ModelInstance

class menpofit.modelinstance.ModelInstance(model)

Bases: Targetable, Vectorizable, DP

Base class for creating a model that can produce a target menpo.shape.PointCloud and knows how to compute its own derivative with respect to its parametrisation.

**Parameters**

- **model**
  - class
  - The trained model (e.g. menpo.model.PCAModel).

**as_vector(**kwargs)**

Returns a flattened representation of the object as a single vector.

- **Returns**
  - **vector** \( ((N, ) \text{ ndarray}) \) – The core representation of the object, flattened into a single vector. Note that this is always a view back on to the original object, but is not writable.

**copy()**

Generate an efficient copy of this object.

- **Returns**
  - **type(self)** – A copy of this object

**abstract \( d\_dp(\text{points}) \)**

The derivative of this spatial object with respect to the parametrisation changes evaluated at points.

**Parameters**

- **points** \( ((n\_points, n\_dims) \text{ ndarray}) \) – The spatial points at which the derivative should be evaluated.

**Returns**

- **d_dp((n_points, n\_parameters, n\_dims) \text{ ndarray})** – The Jacobian with respect to the parametrisation.

\( d\_dp[i, j, k] \) is the scalar differential change that the \( k \)'th dimension of the \( i \)'th point experiences due to a first order change in the \( j \)'th scalar in the parametrisation vector.

**from_vector(vector)**

Build a new instance of the object from it’s vectorized state.

- **self** is used to fill out the missing state required to rebuild a full object from it’s standardized flattened state. This is the default implementation, which is which is a deepcopy of the object followed by a call to from_vector_inplace(). This method can be overridden for a performance benefit if desired.

**Parameters**

- **vector** \( ((n\_parameters, ) \text{ ndarray}) \) – Flattened representation of the object.

**Returns**

- **object**(type(self)) – An new instance of this class.
from_vector_inplace(vector)
    Deprecated. Use the non-mutating API, from_vector.
    For internal usage in performance-sensitive spots, see _from_vector_inplace()

    Parameters vector((n_parameters,), ndarray) – Flattened representation of this object

has_nan_values()
    Tests if the vectorized form of the object contains nan values or not. This is particularly useful for objects
    with unknown values that have been mapped to nan values.

    Returns has_nan_values(bool) – If the vectorized object contains nan values.

set_target(new_target)
    Update this object so that it attempts to recreate the new_target.

    Parameters new_target(PointCloud) – The new target that this object should try and re-
    generate.

property n_dims
    The number of dimensions of the target.
    Type int

property n_parameters
    The length of the vector that this object produces.
    Type int

property n_points
    The number of points on the target.
    Type int

property n_weights
    The number of parameters in the linear model.
    Type int

property target
    The current menpo.shape.PointCloud that this object produces.
    Type menpo.shape.PointCloud

property weights
    The weights of the model.
    Type (n_weights,) ndarray

Similarity Model

similarity_2d_instance_model

menpofit.modelinstance.similarity_2d_instance_model(shape)
    Creates a menpo.model.MeanLinearModel that encodes the 2D similarity transforms that can be applied on a
    2D shape that consists of n_points.

    Parameters shape(menpo.shape.PointCloud) – The input 2D shape.

    Returns model(subclass of menpo.model.MeanLinearModel) – Linear model with four compo-
    nents, the linear combinations of which represent the original shape under a similarity trans-
    form. The model is exhaustive (that is, all possible similarity transforms can be expressed with
    the model).
GlobalSimilarityModel

```python
class menpo.fit.modelinstance.GlobalSimilarityModel(data, **kwargs)
    Bases: Targetable, Vectorizable

Class for creating a model that represents a global similarity transform (in-plane rotation, scaling, translation).

Parameters
- **data** (list of menpo.shape.PointCloud) – The list of shapes to use as training data.

as_vector(**kwargs)

Returns a flattened representation of the object as a single vector.

Returns
- **vector** ((N,) ndarray) – The core representation of the object, flattened into a single vector. Note that this is always a view back on to the original object, but is not writable.

copy()

Generate an efficient copy of this object.

Note that Numpy arrays and other Copyable objects on self will be deeply copied. Dictionaries and sets will be shallow copied, and everything else will be assigned (no copy will be made).

Classes that store state other than numpy arrays and immutable types should overwrite this method to ensure all state is copied.

Returns
- **type(self)** – A copy of this object

d_dp(_)

Returns the Jacobian of the similarity model reshaped in order to have the standard Jacobian shape, i.e. (n_points, n_weights, n_dims) which maps to (n_features, n_components, n_dims) on the linear model.

Returns
- **jacobian** ((n_features, n_components, n_dims) ndarray) – The Jacobian of the model in the standard Jacobian shape.

from_vector(vector)

Build a new instance of the object from it’s vectorized state.

self is used to fill out the missing state required to rebuild a full object from it’s standardized flattened state. This is the default implementation, which is which is a deepcopy of the object followed by a call to from_vector_inplace(). This method can be overridden for a performance benefit if desired.

Parameters
- **vector** ((n_parameters,) ndarray) – Flattened representation of the object.

Returns
- **object** (type(self)) – An new instance of this class.

from_vector_inplace(vector)

Deprecated. Use the non-mutating API, from_vector.

For internal usage in performance-sensitive spots, see _from_vector_inplace()

Parameters
- **vector** ((n_parameters,) ndarray) – Flattened representation of this object

has_nan_values()

Tests if the vectorized form of the object contains nan values or not. This is particularly useful for objects with unknown values that have been mapped to nan values.

Returns
- **has_nan_values** (bool) – If the vectorized object contains nan values.

set_target(new_target)

Update this object so that it attempts to recreate the new_target.

Parameters
- **new_target** (menpo.shape.PointCloud) – The new target that this object should try and regenerate.
property n_dims
The number of dimensions of the spatial instance of the model.
   Type int

property n_parameters
The length of the vector that this object produces.
   Type int

property n_points
The number of points on the target.
   Type int

property n_weights
The number of parameters in the linear model.
   Type int

property target
The current menpo.shape.PointCloud that this object produces.
   Type menpo.shape.PointCloud

property weights
The weights of the model.
   Type (n_weights,) ndarray

Point Distribution Model

PDM

class menpofit.modelinstance.PDM(data, max_n_components=None)
   Bases: ModelInstance

Class for building a Point Distribution Model. It is a specialised version of ModelInstance for use with spatial data.

Parameters

• data (list of menpo.shape.PointCloud or menpo.model.PCAModel instance) – If a list of menpo.shape.PointCloud, then a menpo.model.PCAModel will be trained from those training shapes. Otherwise, a trained menpo.model.PCAModel instance can be provided.

• max_n_components (int or None, optional) – The maximum number of components that the model will keep. If None, then all the components will be kept.

as_vector(**kwargs)
Returns a flattened representation of the object as a single vector.

Returns vector ((N,) ndarray) – The core representation of the object, flattened into a single vector. Note that this is always a view back on to the original object, but is not writable.

copy()
Generate an efficient copy of this object.

Note that Numpy arrays and other Copyable objects on self will be deeply copied. Dictionaries and sets will be shallow copied, and everything else will be assigned (no copy will be made).

Classes that store state other than numpy arrays and immutable types should overwrite this method to ensure all state is copied.

2.2. Internal API
Returns \( \text{type}(\text{self}) \) – A copy of this object

d_dp \((\text{points})\)

Returns the Jacobian of the similarity model reshaped in order to have the standard Jacobian shape, i.e. \((\text{n_points}, \text{n_weights}, \text{n_dims})\) which maps to \((\text{n_features}, \text{n_components}, \text{n_dims})\) on the linear model.

Returns \( \text{jacobian}((\text{n_features}, \text{n_components}, \text{n_dims}) \text{ndarray}) \) – The Jacobian of the model in the standard Jacobian shape.

from_vector \((\text{vector})\)

Build a new instance of the object from its vectorized state.

\(\text{self}\) is used to fill out the missing state required to rebuild a full object from its standardized flattened state. This is the default implementation, which is which is a deep copy of the object followed by a call to \text{from_vector_inplace()}\). This method can be overridden for a performance benefit if desired.

Parameters \(\text{vector}((\text{n_parameters},) \text{ndarray})\) – Flattened representation of the object.

Returns \(\text{object}(\text{type}(\text{self}))\) – An new instance of this class.

from_vector_inplace \((\text{vector})\)

Deprecated. Use the non-mutating API, \text{from_vector}.

For internal usage in performance-sensitive spots, see \text{from_vector_inplace()}

Parameters \(\text{vector}((\text{n_parameters},) \text{ndarray})\) – Flattened representation of this object

has_nan_values()

Tests if the vectorized form of the object contains \text{nan} values or not. This is particularly useful for objects with unknown values that have been mapped to \text{nan} values.

Returns \(\text{has_nan_values}(\text{bool})\) – If the vectorized object contains \text{nan} values.

increment \((\text{shapes}, \text{n_shapes}=\text{None}, \text{forgetting_factor}=1.0, \text{max_n_components}=\text{None}, \text{verbose}=\text{False})\)

Update the eigenvectors, eigenvalues and mean vector of this model by performing incremental PCA on the given samples.

Parameters

- \text{shapes (list of menpo.shape.PointCloud)} – List of new shapes to update the model from.
- \text{n_shapes (int or None, optional)} – If int, then \text{shapes} must be an iterator that yields \text{n_shapes}. If None, then \text{shapes} has to be a list (so we know how large the data matrix needs to be).
- \text{forgetting_factor ([0.0, 1.0] float, optional)} – Forgetting factor that weights the relative contribution of new samples vs old samples. If 1.0, all samples are weighted equally and, hence, the results is the exact same as performing batch PCA on the concatenated list of old and new samples. If <1.0, more emphasis is put on the new samples. See [1] for details.
- \text{max_n_components (int or None, optional)} – The maximum number of components that the model will keep. If None, then all the components will be kept.
- \text{verbose (bool, optional)} – If True, then information about the progress will be printed.

References

set_target \((\text{new_target})\)

Update this object so that it attempts to recreate the \text{new_target}. 
Parameters **new_target** ([PointCloud](#)) – The new target that this object should try and re-generate.

**property n_active_components**

The number of components currently in use on this model.

Type `int`

**property n_dims**

The number of dimensions of the spatial instance of the model

Type `int`

**property n_parameters**

The length of the vector that this object produces.

Type `int`

**property n_points**

The number of points on the target.

Type `int`

**property n_weights**

The number of parameters in the linear model.

Type `int`

**property target**

The current menpo.shape<PointCloud> that this object produces.

Type `menpo.shape.PointCloud`

**property weights**

The weights of the model.

Type `(n_weights,) ndarray`

GlobalPDM

class menpofit.modelinstance.GlobalPDM(data, global_transform_cls, max_n_components=None)

**class** `menpofit.modelinstance.GlobalPDM` *(data, global_transform_cls, max_n_components=None)*

**Bases:** `PDM`

Class for building a Point Distribution Model that also stores a Global Alignment transform. The final transform couples the Global Alignment transform to a statistical linear model, so that its weights are fully specified by both the weights of statistical model and the weights of the similarity transform.

**Parameters**

- **data** *(list of menpo.shape.PointCloud or menpo.model.PCAModel instance)* – If a list of menpo.shape.PointCloud, then a menpo.model.PCAModel instance will be trained from those training shapes. Otherwise, a trained menpo.model.PCAModel instance can be provided.

- **global_transform_cls** *(class)* – The Global Similarity transform class (e.g. DifferentiableAlignmentSimilarity).

- **max_n_components** *(int or None, optional)* – The maximum number of components that the model will keep. If None, then all the components will be kept.

**as_vector** (**kwargs)**

Returns a flattened representation of the object as a single vector.
**Returns** vector \((N,)\) ndarray – The core representation of the object, flattened into a single vector. Note that this is always a view back on to the original object, but is not writable.

**copy()**

Generate an efficient copy of this object.

Note that Numpy arrays and other *Copyable* objects on *self* will be deeply copied. Dictionaries and sets will be shallow copied, and everything else will be assigned (no copy will be made).

Classes that store state other than numpy arrays and immutable types should overwrite this method to ensure all state is copied.

**Returns** type(*self*) – A copy of this object

**d_dp(points)**

The derivative with respect to the parametrisation changes evaluated at points.

**Parameters**

points \((n\_points, n\_dims)\) ndarray – The spatial points at which the derivative should be evaluated.

**Returns** d_dp \((n\_points, n\_parameters, n\_dims)\) ndarray – The Jacobian with respect to the parametrisation.

**from_vector(vector)**

Build a new instance of the object from it’s vectorized state.

*self* is used to fill out the missing state required to rebuild a full object from it’s standardized flattened state. This is the default implementation, which is which is a *deepcopy* of the object followed by a call to *from_vector_inplace()* . This method can be overridden for a performance benefit if desired.

**Parameters**

vector \((n\_parameters,)\) ndarray – Flattened representation of the object.

**Returns** object (type(*self*)) – An new instance of this class.

**from_vector_inplace(vector)**

Deprecated. Use the non-mutating API, *from_vector*.

For internal usage in performance-sensitive spots, see *from_vector_inplace()*

**Parameters**

vector \((n\_parameters,)\) ndarray – Flattened representation of this object

**has_nan_values()**

Tests if the vectorized form of the object contains *nan* values or not. This is particularly useful for objects with unknown values that have been mapped to *nan* values.

**Returns** has_nan_values (bool) – If the vectorized object contains *nan* values.

**increment(shapes, n_shapes=None, forgetting_factor=1.0, max_n_components=None, verbose=False)**

Update the eigenvectors, eigenvalues and mean vector of this model by performing incremental PCA on the given samples.

**Parameters**

* shapes (list of menpo.shape.PointCloud) – List of new shapes to update the model from.

* n_shapes (int or None, optional) – If int, then *shapes* must be an iterator that yields *n_shapes*. If None, then *shapes* has to be a list (so we know how large the data matrix needs to be).

* forgetting_factor ([0.0, 1.0] float, optional) – Forgetting factor that weights the relative contribution of new samples vs old samples. If 1.0, all samples are weighted equally and, hence, the results is the exact same as performing batch PCA on the concatenated list of old and new simples. If <1.0, more emphasis is put on the new samples. See [1] for details.
• **max_n_components** *(int or None, optional)*  – The maximum number of components that the model will keep. If `None`, then all the components will be kept.

• **verbose** *(bool, optional)*  – If `True`, then information about the progress will be printed.

---

**References**

`set_target`(new_target)

Update this object so that it attempts to recreate the `new_target`.

**Parameters**

new_target *(PointCloud)*  – The new target that this object should try and regenerate.

**property global_parameters**

The parameters for the global transform.

**Type** *(n_global_parameters,) ndarray*

**property n_active_components**

The number of components currently in use on this model.

**Type** *int*

**property n_dims**

The number of dimensions of the spatial instance of the model.

**Type** *int*

**property n_global_parameters**

The number of parameters in the `global_transform`

**Type** *int*

**property n_parameters**

The length of the vector that this object produces.

**Type** *int*

**property n_points**

The number of points on the `target`.

**Type** *int*

**property n_weights**

The number of parameters in the linear model.

**Type** *int*

**property target**

The current `menpo.shape.PointCloud` that this object produces.

**Type** *menpo.shape.PointCloud*

**property weights**

The weights of the model.

**Type** *(n_weights,) ndarray*

---

**OrthoPDM**

**class** `menpofit.modelinstance.OrthoPDM`(data, max_n_components=None)

**Bases:** `GlobalPDM`
Class for building a Point Distribution Model that also stores a Global Alignment transform. The final transform couples the Global Alignment transform to a statistical linear model, so that its weights are fully specified by both the weights of statistical model and the weights of the similarity transform.

This transform (in contrast to the :map:`GlobalPDM`) additionally orthonormalises both the global and the model basis against each other, ensuring that orthogonality and normalization is enforced across the unified bases.

**Parameters**

- **data** (list of menpo.shape.PointCloud or menpo.model.PCAModel instance) – If a list of menpo.shape.PointCloud, then a menpo.model.PCAModel will be trained from those training shapes. Otherwise, a trained menpo.model.PCAModel instance can be provided.

- **max_n_components** (int or None, optional) – The maximum number of components that the model will keep. If None, then all the components will be kept.

**Methods**

- **as_vector(**kwargs)**
  
  Returns a flattened representation of the object as a single vector.

  **Returns** vector ((N,) ndarray) – The core representation of the object, flattened into a single vector. Note that this is always a view back on to the original object, but is not writable.

- **copy()**
  
  Generate an efficient copy of this object.

  Note that Numpy arrays and other Copyable objects on self will be deeply copied. Dictionaries and sets will be shallow copied, and everything else will be assigned (no copy will be made).

  Classes that store state other than numpy arrays and immutable types should overwrite this method to ensure all state is copied.

  **Returns** type(self) – A copy of this object

- **d_dp(points)**
  
  The derivative with respect to the parametrisation changes evaluated at points.

  **Parameters** points ((n_points, n_dims) ndarray) – The spatial points at which the derivative should be evaluated.

  **Returns** d_dp ((n_points, n_parameters, n_dims) ndarray) – The Jacobian with respect to the parametrisation.

- **from_vector(vector)**
  
  Build a new instance of the object from it’s vectorized state.

  self is used to fill out the missing state required to rebuild a full object from it’s standardized flattened state. This is the default implementation, which is which is a deepcopy of the object followed by a call to from_vector_inplace(). This method can be overridden for a performance benefit if desired.

  **Parameters** vector ((n_parameters,) ndarray) – Flattened representation of the object.

  **Returns** object (type(self)) – An new instance of this class.

- **from_vector_inplace(vector)**
  
  Deprecated. Use the non-mutating API, from_vector.

  For internal usage in performance-sensitive spots, see _from_vector_inplace()

  **Parameters** vector ((n_parameters,) ndarray) – Flattened representation of this object

- **has_nan_values()**
  
  Tests if the vectorized form of the object contains nan values or not. This is particularly useful for objects with unknown values that have been mapped to nan values.

  **Returns** has_nan_values (bool) – If the vectorized object contains nan values.
increment(shapes, n_shapes=None, forgetting_factor=1.0, max_n_components=None, verbose=False)

Update the eigenvectors, eigenvalues and mean vector of this model by performing incremental PCA on the given samples.

Parameters

• shapes (list of menpo.shape.PointCloud) – List of new shapes to update the model from.

• n_shapes (int or None, optional) – If int, then shapes must be an iterator that yields n_shapes. If None, then shapes has to be a list (so we know how large the data matrix needs to be).

• forgetting_factor ([0.0, 1.0] float, optional) – Forgetting factor that weights the relative contribution of new samples vs old samples. If 1.0, all samples are weighted equally and, hence, the results is the exact same as performing batch PCA on the concatenated list of old and new simples. If <1.0, more emphasis is put on the new samples. See [1] for details.

• max_n_components (int or None, optional) – The maximum number of components that the model will keep. If None, then all the components will be kept.

• verbose (bool, optional) – If True, then information about the progress will be printed.

References

set_target(new_target)

Update this object so that it attempts to recreate the new_target.

Parameters new_target (PointCloud) – The new target that this object should try and regenerate.

property global_parameters
The parameters for the global transform.

Type (n_global_parameters,) ndarray

property n_active_components
The number of components currently in use on this model.

Type int

property n_dims
The number of dimensions of the spatial instance of the model

Type int

property n_global_parameters
The number of parameters in the global_transform

Type int

property n_parameters
The length of the vector that this object produces.

Type int

property n_points
The number of points on the target.

Type int
property n_weights
The number of parameters in the linear model.
Type int

property target
The current menpo.shape.PointCloud that this object produces.
Type menpo.shape.PointCloud

property weights
The weights of the model.
Type (n_weights,) ndarray

2.2.9 menpofit.result

Basic Result

Class for defining a basic fitting result.

Result
class menpofit.result.Result (final_shape, image=None, initial_shape=None, gt_shape=None)
Bases: object

Class for defining a basic fitting result. It holds the final shape of a fitting process and, optionally, the initial
shape, ground truth shape and the image object.

Parameters
• final_shape (menpo.shape.PointCloud) – The final shape of the fitting process.

• image (menpo.image.Image or subclass or None, optional) – The image on which the
fitting process was applied. Note that a copy of the image will be assigned as an attribute. If
None, then no image is assigned.

• initial_shape (menpo.shape.PointCloud or None, optional) – The initial shape that
was provided to the fitting method to initialise the fitting process. If None, then no initial
shape is assigned.

• gt_shape (menpo.shape.PointCloud or None, optional) – The ground truth shape associ-
ed with the image. If None, then no ground truth shape is assigned.

final_error (compute_error=None)
Returns the final error of the fitting process, if the ground truth shape exists. This is the error computed
based on the final_shape.

Parameters compute_error (callable, optional) – Callable that computes the error between
the fitted and ground truth shapes.

Returns final_error (float) – The final error at the end of the fitting process.

Raises ValueError – Ground truth shape has not been set, so the final error cannot be com-
puted

initial_error (compute_error=None)
Returns the initial error of the fitting process, if the ground truth shape and initial shape exist. This is the
error computed based on the initial_shape.
Parameters **compute_error** *(callable, optional)* – Callable that computes the error between the initial and ground truth shapes.

Returns **initial_error** *(float)* – The initial error at the beginning of the fitting process.

Raises

- **ValueError** – Initial shape has not been set, so the initial error cannot be computed
- **ValueError** – Ground truth shape has not been set, so the initial error cannot be computed

**view** *(figure_id=None, new_figure=False, render_image=True, render_final_shape=True, render_initial_shape=False, render_gt_shape=False, subplots_enabled=True, channels=None, interpolation='bilinear', cmap_name=None, alpha=1.0, masked=True, final_marker_face_colour='r', final_marker_edge_colour='k', final_line_colour='r', initial_marker_face_colour='b', initial_marker_edge_colour='k', initial_line_colour='b', gt_marker_face_colour='y', gt_marker_edge_colour='k', gt_line_colour='y', render_lines=True, line_style='-', line_width=2, render_markers=True, marker_style='o', marker_size=4, marker_edge_width=1.0, render_numbering=False, numbers_horizontal_align='center', numbers_vertical_align='bottom', numbers_font_name='sans-serif', numbers_font_size=10, numbers_font_style='normal', numbers_font_weight='normal', numbers_font_colour='k', render_legend=True, legend_title='', legend_font_name='sans-serif', legend_font_style='normal', legend_font_size=10, legend_font_weight='normal', legend_marker_scale=None, legend_location=2, legend_bbox_to_anchor=(1.05, 1.0), legend_border_axes_pad=None, legend_n_columns=1, legend_horizontal_spacing=None, legend_vertical_spacing=None, legend_border=True, legend_border_padding=None, legend_shadow=False, legend_rounded_corners=False, render_axes=True, axes_font_name='sans-serif', axes_font_size=10, axes_font_style='normal', axes_font_weight='normal', axes_x_limits=None, axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None, figure_size=(7, 7))

Visualize the fitting result. The method renders the final fitted shape and optionally the initial shape, ground truth shape and the image, id they were provided.

Parameters

- **figure_id** *(object, optional)* – The id of the figure to be used.
- **new_figure** *(bool, optional)* – If True, a new figure is created.
- **render_image** *(bool, optional)* – If True and the image exists, then it gets rendered.
- **render_final_shape** *(bool, optional)* – If True, then the final fitting shape gets rendered.
- **render_initial_shape** *(bool, optional)* – If True and the initial fitting shape exists, then it gets rendered.
- **render_gt_shape** *(bool, optional)* – If True and the ground truth shape exists, then it gets rendered.
- **subplots_enabled** *(bool, optional)* – If True, then the requested final, initial and ground truth shapes get rendered on separate subplots.
- **channels** *(int or list of int or all or None)* – If int or list of int, the specified channel(s) will be rendered. If all, all the channels will be rendered in subplots. If None and the image is RGB, it will be rendered in RGB mode. If None and the image is not RGB, it is equivalent to all.
- **interpolation** *(See Below, optional)* – The interpolation used to render the image. For example, if *bilinear*, the image will be smooth and if *nearest*, the image will be pixelated. Example options
MenpoFit Documentation, Release 0.6.1

- **cmap_name** (*str*, optional) – If None, single channel and three channel images default to greyscale and rgb colormaps respectively.
- **alpha** (*float*, optional) – The alpha blending value, between 0 (transparent) and 1 (opaque).
- **masked** (*bool*, optional) – If True, then the image is rendered as masked.
- **final_marker_face_colour** (*See Below, optional*) – The face (filling) colour of the markers of the final fitting shape. Example options
  
  ```
  \{r, g, b, c, m, k, w\}
  or
  (3, ) ndarray
  ```

- **final_marker_edge_colour** (*See Below, optional*) – The edge colour of the markers of the final fitting shape. Example options
  
  ```
  \{r, g, b, c, m, k, w\}
  or
  (3, ) ndarray
  ```

- **final_line_colour** (*See Below, optional*) – The line colour of the final fitting shape. Example options
  
  ```
  \{r, g, b, c, m, k, w\}
  or
  (3, ) ndarray
  ```

- **initial_marker_face_colour** (*See Below, optional*) – The face (filling) colour of the markers of the initial shape. Example options
  
  ```
  \{r, g, b, c, m, k, w\}
  or
  (3, ) ndarray
  ```

- **initial_marker_edge_colour** (*See Below, optional*) – The edge colour of the markers of the initial shape. Example options
  
  ```
  \{r, g, b, c, m, k, w\}
  or
  (3, ) ndarray
  ```

- **initial_line_colour** (*See Below, optional*) – The line colour of the initial shape. Example options
  
  ```
  \{r, g, b, c, m, k, w\}
  or
  (3, ) ndarray
  ```

- **gt_marker_face_colour** (*See Below, optional*) – The face (filling) colour of the markers of the ground truth shape. Example options
  
  ```
  \{r, g, b, c, m, k, w\}
  or
  (3, ) ndarray
  ```

- **gt_marker_face_colour** (*See Below, optional*) – The face (filling) colour of the markers of the ground truth shape. Example options
  
  ```
  \{r, g, b, c, m, k, w\}
  or
  (3, ) ndarray
  ```
• **gt_marker_edge_colour** *(See Below, optional)* – The edge colour of the markers of the ground truth shape. Example options

```plaintext
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• **gt_line_colour** *(See Below, optional)* – The line colour of the ground truth shape. Example options

```plaintext
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• **render_lines** *(bool or list of bool, optional)* – If True, the lines will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per shape in *(final, initial, groundtruth)* order.

• **line_style** *(str or list of str, optional)* – The style of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per shape in *(final, initial, groundtruth)* order. Example options:

```plaintext
{ '-', '--', '-.', ':' }
```

• **line_width** *(float or list of float, optional)* – The width of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per shape in *(final, initial, groundtruth)* order.

• **render_markers** *(bool or list of bool, optional)* – If True, the markers will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per shape in *(final, initial, groundtruth)* order.

• **marker_style** *(str or list of str, optional)* – The style of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per shape in *(final, initial, groundtruth)* order. Example options:

```plaintext
{.,,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
```

• **marker_size** *(int or list of int, optional)* – The size of the markers in points. You can either provide a single value that will be used for all shapes or a list with a different value per shape in *(final, initial, groundtruth)* order.

• **marker_edge_width** *(float or list of float, optional)* – The width of the markers’ edge. You can either provide a single value that will be used for all shapes or a list with a different value per shape in *(final, initial, groundtruth)* order.

• **render_numbering** *(bool, optional)* – If True, the landmarks will be numbered.

• **numbers_horizontal_align** *(center, right, left), optional)* – The horizontal alignment of the numbers’ texts.

• **numbers_vertical_align** *(center, top, bottom, baseline), optional)* – The vertical alignment of the numbers’ texts.
• **numbers_font_name** *(See Below, optional)* – The font of the numbers. Example options

  ```
  [serif, sans-serif, cursive, fantasy, monospace]
  ```

• **numbers_font_size** *(int, optional)* – The font size of the numbers.

• **numbers_font_style** *(normal, italic, oblique), optional)* – The font style of the numbers.

• **numbers_font_weight** *(See Below, optional)* – The font weight of the numbers. Example options

  ```
  [ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black]
  ```

• **numbers_font_colour** *(See Below, optional)* – The font colour of the numbers. Example options

  ```
  [r, g, b, c, m, k, w]
  or
  (3, ) ndarray
  ```

• **render_legend** *(bool, optional)* – If True, the legend will be rendered.

• **legend_title** *(str, optional)* – The title of the legend.

• **legend_font_name** *(See below, optional)* – The font of the legend. Example options

  ```
  [serif, sans-serif, cursive, fantasy, monospace]
  ```

• **legend_font_style** *(normal, italic, oblique), optional)* – The font style of the legend.

• **legend_font_size** *(int, optional)* – The font size of the legend.

• **legend_font_weight** *(See Below, optional)* – The font weight of the legend. Example options

  ```
  [ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black]
  ```

• **legend_marker_scale** *(float, optional)* – The relative size of the legend markers with respect to the original

• **legend_location** *(int, optional)* – The location of the legend. The predefined values are:
| 'best'   | 0 |
| 'upper right' | 1 |
| 'upper left' | 2 |
| 'lower left' | 3 |
| 'lower right' | 4 |
| 'right' | 5 |
| 'center left' | 6 |
| 'center right' | 7 |
| 'lower center' | 8 |
| 'upper center' | 9 |
| 'center' | 10 |

- **legend_bbox_to_anchor** ((float, float) tuple, optional) – The bbox that the legend will be anchored.
- **legend_border_axes_pad** (float, optional) – The pad between the axes and legend border.
- **legend_n_columns** (int, optional) – The number of the legend’s columns.
- **legend_horizontal_spacing** (float, optional) – The spacing between the columns.
- **legend_vertical_spacing** (float, optional) – The vertical space between the legend entries.
- **legend_border** (bool, optional) – If True, a frame will be drawn around the legend.
- **legend_border_padding** (float, optional) – The fractional whitespace inside the legend border.
- **legend_shadow** (bool, optional) – If True, a shadow will be drawn behind legend.
- **legend_rounded_corners** (bool, optional) – If True, the frame’s corners will be rounded (fancybox).
- **render_axes** (bool, optional) – If True, the axes will be rendered.
- **axes_font_name** (See Below, optional) – The font of the axes. Example options

{serif, sans-serif, cursive, fantasy, monospace}

- **axes_font_size** (int, optional) – The font size of the axes.
- **axes_font_style** ([normal, italic, oblique], optional) – The font style of the axes.
- **axes_font_weight** (See Below, optional) – The font weight of the axes. Example options

{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}

- **axes_x_limits** (float or (float, float) or None, optional) – The limits of the x axis. If float, then it sets padding on the right and left of the Image as a percentage of the Image’s width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- **axes_y_limits** ((float, float) tuple or None, optional) – The limits of the y axis. If float, then it sets padding on the top and bottom of the Image as a percentage of the
Image’s height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.

- **axes_x_ticks** (list or tuple or None, optional) – The ticks of the x axis.
- **axes_y_ticks** (list or tuple or None, optional) – The ticks of the y axis.
- **figure_size** ((float, float) tuple or None optional) – The size of the figure in inches.

Returns **renderer** (class) – The renderer object.

**view_widget** (figure_size=(7, 7))

Visualizes the result object using an interactive widget.

Parameters **figure_size** ((int, int), optional) – The initial size of the rendered figure.

**property final_shape**

Returns the final shape of the fitting process.

Type `menpo.shape.PointCloud`

**property gt_shape**

Returns the ground truth shape associated with the image. In case there is not an attached ground truth shape, then None is returned.

Type `menpo.shape.PointCloud` or None

**property image**

Returns the image that the fitting was applied on, if it was provided. Otherwise, it returns None.

Type `menpo.shape.Image` or subclass or None

**property initial_shape**

Returns the initial shape that was provided to the fitting method to initialise the fitting process. In case the initial shape does not exist, then None is returned.

Type `menpo.shape.PointCloud` or None

**property is_iterative**

Flag whether the object is an iterative fitting result.

Type `bool`

**Iterative Result**

Classes for defining an iterative fitting result.

**NonParametricIterativeResult**

**class** `menpofit.result.NonParametricIterativeResult` (shapes, initial_shape=None, image=None, gt_shape=None, costs=None)

Bases: `Result`

Class for defining a non-parametric iterative fitting result, i.e. the result of a method that does not optimize over a parametric shape model. It holds the shapes of all the iterations of the fitting procedure. It can optionally store the image on which the fitting was applied, as well as its ground truth shape.

Parameters

- **shapes** (list of `menpo.shape.PointCloud`) – The list of shapes per iteration. Note that the list does not include the initial shape. The last member of the list is the final shape.
• **initial_shape** *(menpo.shape.PointCloud or None, optional)* – The initial shape from which the fitting process started. If `None`, then no initial shape is assigned.

• **image** *(menpo.image.Image or subclass or None, optional)* – The image on which the fitting process was applied. Note that a copy of the image will be assigned as an attribute. If `None`, then no image is assigned.

• **gt_shape** *(menpo.shape.PointCloud or None, optional)* – The ground truth shape associated with the image. If `None`, then no ground truth shape is assigned.

• **costs** *(list of float or None, optional)* – The list of cost per iteration. If `None`, then it is assumed that the cost function cannot be computed for the specific algorithm. It must have the same length as `shapes`.

**displacements**

A list containing the displacement between the shape of each iteration and the shape of the previous one.

Type `list of ndarray`

**displacements_stats** *(stat_type='mean')*

A list containing a statistical metric on the displacements between the shape of each iteration and the shape of the previous one.

Parameters **stat_type** *({'mean', 'median', 'min', 'max'}, optional)* – Specifies a statistic metric to be extracted from the displacements.

Returns **displacements_stat** *(list of float)* – The statistical metric on the points displacements for each iteration.

Raises **ValueError** – type must be ‘mean’, ‘median’, ‘min’ or ‘max’

**errors** *(compute_error=None)*

Returns a list containing the error at each fitting iteration, if the ground truth shape exists.

Parameters **compute_error** *(callable, optional)* – Callable that computes the error between the shape at each iteration and the ground truth shape.

Returns **errors** *(list of float)* – The error at each iteration of the fitting process.

Raises **ValueError** – Ground truth shape has not been set, so the final error cannot be computed

**final_error** *(compute_error=None)*

Returns the final error of the fitting process, if the ground truth shape exists. This is the error computed based on the `final_shape`.

Parameters **compute_error** *(callable, optional)* – Callable that computes the error between the fitted and ground truth shapes.

Returns **final_error** *(float)* – The final error at the end of the fitting process.

Raises **ValueError** – Ground truth shape has not been set, so the final error cannot be computed

**initial_error** *(compute_error=None)*

Returns the initial error of the fitting process, if the ground truth shape and initial shape exist. This is the error computed based on the `initial_shape`.

Parameters **compute_error** *(callable, optional)* – Callable that computes the error between the initial and ground truth shapes.

Returns **initial_error** *(float)* – The initial error at the beginning of the fitting process.

Raises
• **ValueError** – Initial shape has not been set, so the initial error cannot be computed
• **ValueError** – Ground truth shape has not been set, so the initial error cannot be computed

```python
plot_costs(figure_id=None, new_figure=False, render_lines=True, line_colour='b', line_style='-', line_width=2, render_markers=True, marker_style='o', marker_size=4, marker_face_colour='b', marker_edge_colour='k', marker_edge_width=1.0, render_axes=True, axes_font_name='sans-serif', axes_font_size=10, axes_font_style='normal', axes_font_weight='normal', axes_x_limits=0.0, axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None, figure_size=(10, 6), render_grid=True, grid_line_style='–', grid_line_width=0.5)
```

Plot of the cost function evolution at each fitting iteration.

### Parameters

- **figure_id** *(object, optional)* – The id of the figure to be used.
- **new_figure** *(bool, optional)* – If True, a new figure is created.
- **render_lines** *(bool, optional)* – If True, the line will be rendered.
- **line_colour** *(colour or None, optional)* – The colour of the line. If None, the colour is sampled from the jet colormap. Example colour options are
  ```python
  {'r', 'g', 'b', 'c', 'm', 'k', 'w'}
  or
  (3, ) ndarray
  ```
- **line_style** *(('‐', '–‐', '‐‐', '‐‐‐', ':'), optional)* – The style of the lines.
- **line_width** *(float, optional)* – The width of the lines.
- **render_markers** *(bool, optional)* – If True, the markers will be rendered.
- **marker_style** *(marker, optional)* – The style of the markers. Example marker options
  ```python
  {'.', ',', 'o', 'v', '^', '<', '>', '+', 'x', 'D', 'd', 's', 'p', 'P', 'H', 'h', '1', '2', '3', '4', '8'}
  ```
- **marker_size** *(int, optional)* – The size of the markers in points.
- **marker_face_colour** *(colour or None, optional)* – The face (filling) colour of the markers. If None, the colour is sampled from the jet colormap. Example colour options are
  ```python
  {'r', 'g', 'b', 'c', 'm', 'k', 'w'}
  or
  (3, ) ndarray
  ```
- **marker_edge_colour** *(colour or None, optional)* – The edge colour of the markers. If None, the colour is sampled from the jet colormap. Example colour options are
  ```python
  {'r', 'g', 'b', 'c', 'm', 'k', 'w'}
  or
  (3, ) ndarray
  ```
- **marker_edge_width** *(float, optional)* – The width of the markers’ edge.
- **render_axes** *(bool, optional)* – If True, the axes will be rendered.
• **axes_font_name** (*See below, optional*) – The font of the axes. Example options

```python
['serif', 'sans-serif', 'cursive', 'fantasy', 'monospace']
```

• **axes_font_size** (*int, optional*) – The font size of the axes.

• **axes_font_style** (*{'normal', 'italic', 'oblique'}, optional*) – The font style of the axes.

• **axes_font_weight** (*See below, optional*) – The font weight of the axes. Example options

```python
['ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black']
```

• **axes_x_limits** (*float or (float, float) or None, optional*) – The limits of the x axis. If float, then it sets padding on the right and left of the graph as a percentage of the curves’ width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.

• **axes_y_limits** (*float or (float, float) or None, optional*) – The limits of the y axis. If float, then it sets padding on the top and bottom of the graph as a percentage of the curves’ height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.

• **axes_x_ticks** (*list or tuple or None, optional*) – The ticks of the x axis.

• **axes_y_ticks** (*list or tuple or None, optional*) – The ticks of the y axis.

• **figure_size** (*((float, float) or None, optional*) – The size of the figure in inches.

• **render_grid** (*bool, optional*) – If True, the grid will be rendered.

• **grid_line_style** (*{‘-‘, ‘--’, ‘.-‘, ‘:’}, optional*) – The style of the grid lines.

• **grid_line_width** (*float, optional*) – The width of the grid lines.

**Returns** renderer (*menpo.visualize.GraphPlotter*) – The renderer object.

```python
plot_displacements(stat_type='mean', figure_id=None, new_figure=False, render_lines=True,
                   lineColour='b', lineStyle='-', lineWidth=2, render_markers=True,
                   markerStyle='o', markerSize=4, markerFaceColour='b',
                   markerEdgeColour='k', markerEdgeWidth=1.0, render_axes=True,
                   axes_fontName='sans-serif', axes_fontSize=10, axes_fontStyle='normal',
                   axes_fontWeight='normal', axes_x_limits=0.0, axes_y_limits=None,
                   axes_x_ticks=None, axes_y_ticks=None, figure_size=(10, 6),
                   render_grid=True, grid_lineStyle='-', grid_lineWidth=0.5)
```

Plot of a statistic metric of the displacement between the shape of each iteration and the shape of the previous one.

**Parameters**

• **stat_type** (*{mean, median, min, max}, optional*) – Specifies a statistic metric to be extracted from the displacements (see also `displacements_stats` method).

• **figure_id** (*object, optional*) – The id of the figure to be used.

• **new_figure** (*bool, optional*) – If True, a new figure is created.

• **render_lines** (*bool, optional*) – If True, the line will be rendered.
• **line_colour** *(colour or None (See below), optional)* – The colour of the line. If None, the colour is sampled from the jet colormap. Example colour options are

\[
[r, g, b, c, m, k, w] \\
\text{or} \\
(3, ) \text{ ndarray}
\]

• **line_style** *(str (See below), optional)* – The style of the lines. Example options:

\[
{-, --, -. , :}
\]

• **line_width** *(float, optional)* – The width of the lines.

• **render_markers** *(bool, optional)* – If True, the markers will be rendered.

• **marker_style** *(str (See below), optional)* – The style of the markers. Example marker options

\[
{., ,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
\]

• **marker_size** *(int, optional)* – The size of the markers in points.

• **marker_face_colour** *(colour or None, optional)* – The face (filling) colour of the markers. If None, the colour is sampled from the jet colormap. Example colour options are

\[
[r, g, b, c, m, k, w] \\
\text{or} \\
(3, ) \text{ ndarray}
\]

• **marker_edge_colour** *(colour or None, optional)* – The edge colour of the markers. If None, the colour is sampled from the jet colormap. Example colour options are

\[
[r, g, b, c, m, k, w] \\
\text{or} \\
(3, ) \text{ ndarray}
\]

• **marker_edge_width** *(float, optional)* – The width of the markers’ edge.

• **render_axes** *(bool, optional)* – If True, the axes will be rendered.

• **axes_font_name** *(str (See below), optional)* – The font of the axes. Example options

\[
\{\text{serif, sans-serif, cursive, fantasy, monospace}\}
\]

• **axes_font_size** *(int, optional)* – The font size of the axes.

• **axes_font_style** *(str (See below), optional)* – The font style of the axes. Example options

\[
\{\text{normal, italic, oblique}\}
\]

• **axes_font_weight** *(str (See below), optional)* – The font weight of the axes. Example options

\[
\{\text{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}\}
\]

• **axes_x_limits** *(float or (float, float) or None, optional)* – The limits of the x axis. If float, then it sets padding on the right and left of the graph as a percentage of the curves’
width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.

• **axes_y_limits** (float or (float, float) or None, optional) – The limits of the y axis. If float, then it sets padding on the top and bottom of the graph as a percentage of the curves’ height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.

• **axes_x_ticks** (list or tuple or None, optional) – The ticks of the x axis.

• **axes_y_ticks** (list or tuple or None, optional) – The ticks of the y axis.

• **figure_size** ((float, float) or None, optional) – The size of the figure in inches.

• **render_grid** (bool, optional) – If True, the grid will be rendered.

• **grid_line_style** (\{'-', '--', '-.', ':'\}, optional) – The style of the grid lines.

• **grid_line_width** (float, optional) – The width of the grid lines.

Returns **renderer** (menpo.visualize.GraphPlotter) – The renderer object.

plot_errors (compute_error=None, figure_id=None, new_figure=False, render_lines=True, line_colour='b', line_style=':', line_width=2, render_markers=True, marker_style='o', marker_size=4, marker_face_colour='b', marker_edge_colour='k', marker_edge_width=1.0, render_axes=True, axes_font_name='sans-serif', axes_font_size=10, axes_font_style='normal', axes_font_weight='normal', axes_x_limits=0.0, axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None, figure_size=(10, 6), render_grid=True, grid_line_style='-', grid_line_width=0.5)

Plot of the error evolution at each fitting iteration.

Parameters

• **compute_error** (callable, optional) – Callable that computes the error between the shape at each iteration and the ground truth shape.

• **figure_id** (object, optional) – The id of the figure to be used.

• **new_figure** (bool, optional) – If True, a new figure is created.

• **render_lines** (bool, optional) – If True, the line will be rendered.

• **line_colour** (colour or None (See below), optional) – The colour of the line. If None, the colour is sampled from the jet colormap. Example colour options are

\{r, g, b, c, m, k, w\}

or

(3,) ndarray

• **line_style** (str (See below), optional) – The style of the lines. Example options:

\{-, --, -. ., :\}

• **line_width** (float, optional) – The width of the lines.

• **render_markers** (bool, optional) – If True, the markers will be rendered.

• **marker_style** (str (See below), optional) – The style of the markers. Example marker options

\{., ., o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8\}

• **marker_size** (int, optional) – The size of the markers in points.
• **marker_face_colour** *(colour or None, optional)* – The face (filling) colour of the markers. If None, the colour is sampled from the jet colormap. Example *colour* options are

\[
\{r, g, b, c, m, k, w\}
\text{ or }
\begin{bmatrix}
3,
\end{bmatrix}
\text{ ndarray}
\]

• **marker_edge_colour** *(colour or None, optional)* – The edge colour of the markers. If None, the colour is sampled from the jet colormap. Example *colour* options are

\[
\{r, g, b, c, m, k, w\}
\text{ or }
\begin{bmatrix}
3,
\end{bmatrix}
\text{ ndarray}
\]

• **marker_edge_width** *(float, optional)* – The width of the markers’ edge.

• **render_axes** *(bool, optional)* – If True, the axes will be rendered.

• **axes_font_name** *(str (See below), optional)* – The font of the axes. Example options

\[
\{\text{serif, sans-serif, cursive, fantasy, monospace}\}
\]

• **axes_font_size** *(int, optional)* – The font size of the axes.

• **axes_font_style** *(str (See below), optional)* – The font style of the axes. Example options

\[
\{\text{normal, italic, oblique}\}
\]

• **axes_font_weight** *(str (See below), optional)* – The font weight of the axes. Example options

\[
\{\text{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}\}
\]

• **axes_x_limits** *(float or (float, float) or None, optional)* – The limits of the x axis. If *float*, then it sets padding on the right and left of the graph as a percentage of the curves’ width. If *tuple* or *list*, then it defines the axis limits. If None, then the limits are set automatically.

• **axes_y_limits** *(float or (float, float) or None, optional)* – The limits of the y axis. If *float*, then it sets padding on the top and bottom of the graph as a percentage of the curves’ height. If *tuple* or *list*, then it defines the axis limits. If None, then the limits are set automatically.

• **axes_x_ticks** *(list or tuple or None, optional)* – The ticks of the x axis.

• **axes_y_ticks** *(list or tuple or None, optional)* – The ticks of the y axis.

• **figure_size** *((float, float) or None, optional)* – The size of the figure in inches.

• **render_grid** *(bool, optional)* – If True, the grid will be rendered.

• **grid_line_style** *(('-', ' --', ' -. ', ':')*, optional)* – The style of the grid lines.

• **grid_line_width** *(float, optional)* – The width of the grid lines.

Returns **renderer** *(menpo.visualize.GraphPlotter)* – The renderer object.
to_result (pass_image=True, pass_initial_shape=True, pass_gt_shape=True)

Returns a Result instance of the object, i.e. a fitting result object that does not store the iterations. This can be useful for reducing the size of saved fitting results.

Parameters

• pass_image (bool, optional) – If True, then the image will get passed (if it exists).

• pass_initial_shape (bool, optional) – If True, then the initial shape will get passed (if it exists).

• pass_gt_shape (bool, optional) – If True, then the ground truth shape will get passed (if it exists).

Returns result (Result) – The final “lightweight” fitting result.

view (figure_id=None, new_figure=False, render_image=True, render_final_shape=True, render_initial_shape=True, render_gt_shape=True, subplots_enabled=True, channels=None, interpolation='bilinear', cmap_name=None, alpha=1.0, masked=True, final_marker_face_colour='r', final_marker_edge_colour='k', final_line_colour='r', initial_marker_face_colour='b', initial_marker_edge_colour='k', initial_line_colour='b', gt_marker_face_colour='y', gt_marker_edge_colour='k', gt_line_colour='y', render_lines=True, line_style='-', line_width=2, render_markers=True, marker_style='o', marker_size=4, marker_edge_width=1.0, render_numbering=False, numbers_horizontal_align='center', numbers_vertical_align='bottom', numbers_font_name='sans-serif', numbers_font_size=10, numbers_font_style='normal', numbers_font_weight='normal', numbers_font_colour='k', render_legend=True, legend_title='', legend_font_name='sans-serif', legend_font_style='normal', legend_font_size=10, legend_font_weight='normal', legend_marker_scale=None, legend_location=2, legend_bbox_to_anchor=(1.05, 1.0), legend_border_axes_pad=None, legend_n_columns=1, legend_horizontal_spacing=None, legend_vertical_spacing=None, legend_border=True, legend_border_padding=None, legend_shadow=False, legend_rounded_corners=False, render_axes=False, axes_font_name='sans-serif', axes_font_size=10, axes_font_style='normal', axes_font_weight='normal', axes_x_limits=None, axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None, figure_size=(7, 7))

Visualize the fitting result. The method renders the final fitted shape and optionally the initial shape, ground truth shape and the image, if they were provided.

Parameters

• figure_id (object, optional) – The id of the figure to be used.

• new_figure (bool, optional) – If True, a new figure is created.

• render_image (bool, optional) – If True and the image exists, then it gets rendered.

• render_final_shape (bool, optional) – If True, then the final fitting shape gets rendered.

• render_initial_shape (bool, optional) – If True and the initial fitting shape exists, then it gets rendered.

• render_gt_shape (bool, optional) – If True and the ground truth shape exists, then it gets rendered.

• subplots_enabled (bool, optional) – If True, then the requested final, initial and ground truth shapes get rendered on separate subplots.

• channels (int or list of int or all or None) – If int or list of int, the specified channel(s) will be rendered. If all, all the channels will be rendered in subplots. If None and the image is RGB, it will be rendered in RGB mode. If None and the image is not RGB, it is equivalent to all.
• **interpolation** *(See Below, optional)* – The interpolation used to render the image. For example, if `bilinear`, the image will be smooth and if `nearest`, the image will be pixelated. Example options

```python
{none, nearest, bilinear, bicubic, spline16, spline36, hanning, hamming, hermite, kaiser, quadric, catrom, gaussian, bessel, mitchell, sinc, lanczos}
```

• **cmap_name** *(str, optional)* – If `None`, single channel and three channel images default to greyscale and rgb colormaps respectively.

• **alpha** *(float, optional)* – The alpha blending value, between 0 (transparent) and 1 (opaque).

• **masked** *(bool, optional)* – If `True`, then the image is rendered as masked.

• **final_marker_face_colour** *(See Below, optional)* – The face (filling) colour of the markers of the final fitting shape. Example options

```python
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• **final_marker_edge_colour** *(See Below, optional)* – The edge colour of the markers of the final fitting shape. Example options

```python
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• **final_line_colour** *(See Below, optional)* – The line colour of the final fitting shape. Example options

```python
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• **initial_marker_face_colour** *(See Below, optional)* – The face (filling) colour of the markers of the initial shape. Example options

```python
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• **initial_marker_edge_colour** *(See Below, optional)* – The edge colour of the markers of the initial shape. Example options

```python
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• **initial_line_colour** *(See Below, optional)* – The line colour of the initial shape. Example options

```python
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```
- **gt_marker_face_colour** *(See Below, optional)* – The face (filling) colour of the markers of the ground truth shape. Example options

```
[r, g, b, c, m, k, w]
or
(3, ) ndarray
```

- **gt_marker_edge_colour** *(See Below, optional)* – The edge colour of the markers of the ground truth shape. Example options

```
[r, g, b, c, m, k, w]
or
(3, ) ndarray
```

- **gt_line_colour** *(See Below, optional)* – The line colour of the ground truth shape. Example options

```
[r, g, b, c, m, k, w]
or
(3, ) ndarray
```

- **render_lines** *(bool or list of bool, optional)* – If True, the lines will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per shape in *(final, initial, groundtruth)* order.

- **line_style** *(str or list of str, optional)* – The style of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per shape in *(final, initial, groundtruth)* order. Example options:

```
{',-', '--', '-.', ':'}
```

- **line_width** *(float or list of float, optional)* – The width of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per shape in *(final, initial, groundtruth)* order.

- **render_markers** *(bool or list of bool, optional)* – If True, the markers will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per shape in *(final, initial, groundtruth)* order.

- **marker_style** *(str or list of str, optional)* – The style of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per shape in *(final, initial, groundtruth)* order. Example options:

```
{., ,, o, v, ^, <, >, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
```

- **marker_size** *(int or list of int, optional)* – The size of the markers in points. You can either provide a single value that will be used for all shapes or a list with a different value per shape in *(final, initial, groundtruth)* order.

- **marker_edge_width** *(float or list of float, optional)* – The width of the markers’ edge. You can either provide a single value that will be used for all shapes or a list with a different value per shape in *(final, initial, groundtruth)* order.

- **render_numbering** *(bool, optional)* – If True, the landmarks will be numbered.

- **numbers_horizontal_align** *(center, right, left), optional)* – The horizontal alignment of the numbers’ texts.

- **numbers_vertical_align** *(center, top, bottom, baseline), optional)* – The vertical alignment of the numbers’ texts.
• **numbers_font_name** *(See Below, optional)* – The font of the numbers. Example options

```plaintext
{serif, sans-serif, cursive, fantasy, monospace}
```

• **numbers_font_size** *(int, optional)* – The font size of the numbers.

• **numbers_font_style** *(normal, italic, oblique), optional)* – The font style of the numbers.

• **numbers_font_weight** *(See Below, optional)* – The font weight of the numbers. Example options

```plaintext
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```

• **numbers_font_colour** *(See Below, optional)* – The font colour of the numbers. Example options

```plaintext
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• **render_legend** *(bool, optional)* – If True, the legend will be rendered.

• **legend_title** *(str, optional)* – The title of the legend.

• **legend_font_name** *(See below, optional)* – The font of the legend. Example options

```plaintext
{serif, sans-serif, cursive, fantasy, monospace}
```

• **legend_font_style** *(normal, italic, oblique), optional)* – The font style of the legend.

• **legend_font_size** *(int, optional)* – The font size of the legend.

• **legend_font_weight** *(See Below, optional)* – The font weight of the legend. Example options

```plaintext
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```

• **legend_marker_scale** *(float, optional)* – The relative size of the legend markers with respect to the original

• **legend_location** *(int, optional)* – The location of the legend. The predefined values are:
<table>
<thead>
<tr>
<th>'best'</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>'upper right'</td>
<td>1</td>
</tr>
<tr>
<td>'upper left'</td>
<td>2</td>
</tr>
<tr>
<td>'lower left'</td>
<td>3</td>
</tr>
<tr>
<td>'lower right'</td>
<td>4</td>
</tr>
<tr>
<td>'right'</td>
<td>5</td>
</tr>
<tr>
<td>'center left'</td>
<td>6</td>
</tr>
<tr>
<td>'center right'</td>
<td>7</td>
</tr>
<tr>
<td>'lower center'</td>
<td>8</td>
</tr>
<tr>
<td>'upper center'</td>
<td>9</td>
</tr>
<tr>
<td>'center'</td>
<td>10</td>
</tr>
</tbody>
</table>

- **legend_bbox_to_anchor** ((float, float) tuple, optional) – The bbox that the legend will be anchored.
- **legend_border_axes_pad** (float, optional) – The pad between the axes and legend border.
- **legend_n_columns** (int, optional) – The number of the legend’s columns.
- **legend_horizontal_spacing** (float, optional) – The spacing between the columns.
- **legend_vertical_spacing** (float, optional) – The vertical space between the legend entries.
- **legend_border** (bool, optional) – If True, a frame will be drawn around the legend.
- **legend_border_padding** (float, optional) – The fractional whitespace inside the legend border.
- **legend_shadow** (bool, optional) – If True, a shadow will be drawn behind legend.
- **legend_rounded_corners** (bool, optional) – If True, the frame’s corners will be rounded (fancybox).
- **render_axes** (bool, optional) – If True, the axes will be rendered.
- **axes_font_name** (See Below, optional) – The font of the axes. Example options

{serif, sans-serif, cursive, fantasy, monospace}

- **axes_font_size** (int, optional) – The font size of the axes.
- **axes_font_style** ((normal, italic, oblique), optional) – The font style of the axes.
- **axes_font_weight** (See Below, optional) – The font weight of the axes. Example options

{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}

- **axes_x_limits** (float or (float, float) or None, optional) – The limits of the x axis. If float, then it sets padding on the right and left of the Image as a percentage of the Image’s width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- **axes_y_limits** ((float, float) tuple or None, optional) – The limits of the y axis. If float, then it sets padding on the top and bottom of the Image as a percentage of the
Image’s height. If *tuple* or *list*, then it defines the axis limits. If *None*, then the limits are set automatically.

- **axes_x_ticks** (*list* or *tuple* or *None*, optional) – The ticks of the x axis.
- **axes_y_ticks** (*list* or *tuple* or *None*, optional) – The ticks of the y axis.
- **figure_size** (*float, float* *tuple* or *None* optional) – The size of the figure in inches.

Returns `renderer` (*class*) – The renderer object.

```python
view_iterations(figure_id=None, new_figure=False, iters=None, render_image=True, subplots_enabled=False, channels=None, interpolation='bilinear', cmap_name=None, alpha=1.0, masked=True, render_lines=True, line_style='-', line_width=2, line_colour=None, render_markers=True, marker_edge_colour=None, marker_face_colour=None, marker_style='o', marker_size=4, marker_edge_width=1.0, render_numbering=False, numbers_horizontal_align='center', numbers_vertical_align='bottom', numbers_font_name='sans-serif', numbers_font_size=10, numbers_font_style='normal', numbers_font_weight='normal', numbers_font_colour='k', render_legend=True, legend_title='', legend_font_name='sans-serif', legend_font_style='normal', legend_font_size=10, legend_font_weight='normal', legend_marker_scale=None, legend_location=2, legend_bbox_to_anchor=(1.05, 1.0), legend_border_axes_pad=None, legend_n_columns=1, legend_horizontal_spacing=None, legend_vertical_spacing=None, legend_border=True, legend_border_padding=None, legend_shadow=False, legend_rounded_corners=False, axes_font_name='sans-serif', axes_font_size=10, axes_font_style='normal', axes_font_weight='normal', axes_x_limits=None, axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None, figure_size=(7, 7))
```

Visualize the iterations of the fitting process.

**Parameters**

- **figure_id** (*object*, optional) – The id of the figure to be used.
- **new_figure** (*bool*, optional) – If *True*, a new figure is created.
- **iters** (*int* or *list* of *int* or *None*, optional) – The iterations to be visualized. If *None*, then all the iterations are rendered.

<table>
<thead>
<tr>
<th>No.</th>
<th>Visualised shape</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>self.initial_shape</td>
<td>Initial shape</td>
</tr>
<tr>
<td>1</td>
<td>self.shapes[1]</td>
<td>Iteration 1</td>
</tr>
<tr>
<td>i</td>
<td>self.shapes[i]</td>
<td>Iteration i</td>
</tr>
<tr>
<td>n_iters</td>
<td>self.final_shape</td>
<td>Final shape</td>
</tr>
</tbody>
</table>

- **render_image** (*bool*, optional) – If *True* and the image exists, then it gets rendered.
- **subplots_enabled** (*bool*, optional) – If *True*, then the requested final, initial and ground truth shapes get rendered on separate subplots.
- **channels** (*int* or *list* of *int* or *all* or *None*) – If *int* or *list* of *int*, the specified channel(s) will be rendered. If *all*, all the channels will be rendered in subplots. If *None* and the image is RGB, it will be rendered in RGB mode. If *None* and the image is not RGB, it is equivalent to *all*. 

360 Chapter 2. The MenpoFit API
• **interpolation** *(str, optional)* – The interpolation used to render the image. For example, if `bilinear`, the image will be smooth and if `nearest`, the image will be pixelated. Example options

```python
none, nearest, bilinear, bicubic, spline16, spline36, hanning,
hamming, hermite, kaiser, quadric, catrom, gaussian, bessel,
mitchell, sinc, lanczos
```

• **cmap_name** *(str, optional)* – If None, single channel and three channel images default to greyscale and rgb colormaps respectively.

• **alpha** *(float, optional)* – The alpha blending value, between 0 (transparent) and 1 (opaque).

• **masked** *(bool, optional)* – If True, then the image is rendered as masked.

• **render_lines** *(bool or list of bool, optional)* – If True, the lines will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.

• **line_style** *(str or list of str, optional)* – The style of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options:

```python
{-, --, -. , :}
```

• **line_width** *(float or list of float, optional)* – The width of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.

• **line_colour** *(colour or list of colour, optional)* – The colour of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```python
[r, g, b, c, m, k, w]
or
(3, ) ndarray
```

• **render_markers** *(bool or list of bool, optional)* – If True, the markers will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.

• **marker_style** *(str or list of str, optional)* – The style of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```python
{., , ., o, v, ^, <, >, +, x, D, d, m, p, *, h, H, 1, 2, 3, 4, 8}
```

• **marker_size** *(int or list of int, optional)* – The size of the markers in points. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.

• **marker_edge_colour** *(colour or list of colour, optional)* – The edge colour of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```python
[r, g, b, c, m, k, w]
or
(3, ) ndarray
```
• **marker_face_colour** *(colour or list of colour)* (See Below, optional) – The face (filling) colour of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```
[r, g, b, c, m, k, w]
or
(3, ) ndarray
```

• **marker_edge_width** *(float or list of float)*, optional) – The width of the markers’ edge. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.

• **render_numbering** *(bool, optional)* – If True, the landmarks will be numbered.

• **numbers_horizontal_align** *(str)* (See below, optional) – The horizontal alignment of the numbers’ texts. Example options

```
{center, right, left}
```

• **numbers_vertical_align** *(str)* (See below, optional) – The vertical alignment of the numbers’ texts. Example options

```
{center, top, bottom, baseline}
```

• **numbers_font_name** *(str)* (See below, optional) – The font of the numbers. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

• **numbers_font_size** *(int)*, optional) – The font size of the numbers.

• **numbers_font_style** *(str)* (See below, optional) – The font style of the numbers.

• **numbers_font_weight** *(str)* (See below, optional) – The font weight of the numbers. Example options

```
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```

• **numbers_font_colour** *(See Below, optional)* – The font colour of the numbers. Example options

```
[r, g, b, c, m, k, w]
or
(3, ) ndarray
```

• **render_legend** *(bool, optional)* – If True, the legend will be rendered.

• **legend_title** *(str, optional)* – The title of the legend.

• **legend_font_name** *(str)* (See below, optional) – The font of the legend. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

• **legend_font_style** *(str)* (See below, optional) – The font style of the legend. Example options
• **legend_font_size** *(int, optional)* – The font size of the legend.

• **legend_font_weight** *(str (See below), optional)* – The font weight of the legend.
  Example options

  *(ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black)*

• **legend_marker_scale** *(float, optional)* – The relative size of the legend markers with respect to the original

• **legend_location** *(int, optional)* – The location of the legend. The predefined values are:

<table>
<thead>
<tr>
<th>Legend Location</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>'best'</td>
<td>0</td>
</tr>
<tr>
<td>'upper right'</td>
<td>1</td>
</tr>
<tr>
<td>'upper left'</td>
<td>2</td>
</tr>
<tr>
<td>'lower left'</td>
<td>3</td>
</tr>
<tr>
<td>'lower right'</td>
<td>4</td>
</tr>
<tr>
<td>'right'</td>
<td>5</td>
</tr>
<tr>
<td>'center left'</td>
<td>6</td>
</tr>
<tr>
<td>'center right'</td>
<td>7</td>
</tr>
<tr>
<td>'lower center'</td>
<td>8</td>
</tr>
<tr>
<td>'upper center'</td>
<td>9</td>
</tr>
<tr>
<td>'center'</td>
<td>10</td>
</tr>
</tbody>
</table>

• **legend_bbox_to_anchor** *(float, float) tuple, optional)* – The bbox that the legend will be anchored.

• **legend_border_axes_pad** *(float, optional)* – The pad between the axes and legend border.

• **legend_n_columns** *(int, optional)* – The number of the legend’s columns.

• **legend_horizontal_spacing** *(float, optional)* – The spacing between the columns.

• **legend_vertical_spacing** *(float, optional)* – The vertical space between the legend entries.

• **legend_border** *(bool, optional)* – If True, a frame will be drawn around the legend.

• **legend_border_padding** *(float, optional)* – The fractional whitespace inside the legend border.

• **legend_shadow** *(bool, optional)* – If True, a shadow will be drawn behind legend.

• **legend_rounded_corners** *(bool, optional)* – If True, the frame’s corners will be rounded (fancybox).

• **render_axes** *(bool, optional)* – If True, the axes will be rendered.

• **axes_font_name** *(str (See below), optional)* – The font of the axes. Example options

  *(serif, sans-serif, cursive, fantasy, monospace)*

• **axes_font_size** *(int, optional)* – The font size of the axes.
• **axes_font_style** ((normal, italic, oblique), optional) – The font style of the axes.

• **axes_font_weight** (str (See below), optional) – The font weight of the axes. Example options
  
  {ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}

• **axes_x_limits** (float or (float, float) or None, optional) – The limits of the x axis. If float, then it sets padding on the right and left of the Image as a percentage of the Image’s width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.

• **axes_y_limits** ((float, float) tuple or None, optional) – The limits of the y axis. If float, then it sets padding on the top and bottom of the Image as a percentage of the Image’s height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.

• **axes_x_ticks** (list or tuple or None, optional) – The ticks of the x axis.

• **axes_y_ticks** (list or tuple or None, optional) – The ticks of the y axis.

• **figure_size** ((float, float) tuple or None optional) – The size of the figure in inches.

Returns **renderer** (class) – The renderer object.

**view_widget** (figure_size=(7, 7))

Visualizes the result object using an interactive widget.

Parameters **figure_size** ((int, int), optional) – The initial size of the rendered figure.

**property costs**

Returns a list with the cost per iteration. It returns None if the costs are not computed.

Type list of float or None

**property final_shape**

Returns the final shape of the fitting process.

Type menpo.shape.PointCloud

**property gt_shape**

Returns the ground truth shape associated with the image. In case there is not an attached ground truth shape, then None is returned.

Type menpo.shape.PointCloud or None

**property image**

Returns the image that the fitting was applied on, if it was provided. Otherwise, it returns None.

Type menpo.shape.Image or subclass or None

**property initial_shape**

Returns the initial shape that was provided to the fitting method to initialise the fitting process. In case the initial shape does not exist, then None is returned.

Type menpo.shape.PointCloud or None

**property is_iterative**

Flag whether the object is an iterative fitting result.

Type bool
property n_iters
Returns the total number of iterations of the fitting process.

Type int

property shapes
Returns the list of shapes obtained at each iteration of the fitting process. The list includes the initial_shape (if it exists) and final_shape.

Type list of menpo.shape.PointCloud

ParametricIterativeResult
class menpofit.result.ParametricIterativeResult(shapes, shape_parameters, initial_shape=None, image=None, gt_shape=None, costs=None)

Bases: NonParametricIterativeResult

Class for defining a parametric iterative fitting result, i.e. the result of a method that optimizes the parameters of a shape model. It holds the shapes and shape parameters of all the iterations of the fitting procedure. It can optionally store the image on which the fitting was applied, as well as its ground truth shape.

Note: When using a method with a parametric shape model, the first step is to reconstruct the initial shape using the shape model. The generated reconstructed shape is then used as initialisation for the iterative optimisation. This step is not counted in the number of iterations.

Parameters

• shapes (list of menpo.shape.PointCloud) – The list of shapes per iteration. Note that the list does not include the initial shape. However, it includes the reconstruction of the initial shape. The last member of the list is the final shape.

• shape_parameters (list of ndarray) – The list of shape parameters per iteration. Note that the list includes the parameters of the projection of the initial shape. The last member of the list corresponds to the final shape’s parameters. It must have the same length as shapes.

• initial_shape (menpo.shape.PointCloud or None, optional) – The initial shape from which the fitting process started. If None, then no initial shape is assigned.

• image (menpo.image.Image or subclass or None, optional) – The image on which the fitting process was applied. Note that a copy of the image will be assigned as an attribute. If None, then no image is assigned.

• gt_shape (menpo.shape.PointCloud or None, optional) – The ground truth shape associated with the image. If None, then no ground truth shape is assigned.

• costs (list of float or None, optional) – The list of cost per iteration. If None, then it is assumed that the cost function cannot be computed for the specific algorithm. It must have the same length as shapes.

displacements ()
A list containing the displacement between the shape of each iteration and the shape of the previous one.

Type list of ndarray
displacements_stats (stat_type='mean')
A list containing a statistical metric on the displacements between the shape of each iteration and the shape of the previous one.
Parameters `stat_type` ({'mean', 'median', 'min', 'max'}, optional) – Specifies a statistic metric to be extracted from the displacements.

Returns `displacements_stat` (list of float) – The statistical metric on the points displacements for each iteration.

Raises `ValueError` – type must be ‘mean’, ‘median’, ‘min’ or ‘max’

`errors` (compute_error=None)
Returns a list containing the error at each fitting iteration, if the ground truth shape exists.

Parameters `compute_error` (callable, optional) – Callable that computes the error between the shape at each iteration and the ground truth shape.

Returns `errors` (list of float) – The error at each iteration of the fitting process.

Raises `ValueError` – Ground truth shape has not been set, so the final error cannot be computed

`final_error` (compute_error=None)
Returns the final error of the fitting process, if the ground truth shape exists. This is the error computed based on the `final_shape`.

Parameters `compute_error` (callable, optional) – Callable that computes the error between the fitted and ground truth shapes.

Returns `final_error` (float) – The final error at the end of the fitting process.

Raises `ValueError` – Ground truth shape has not been set, so the final error cannot be computed

`initial_error` (compute_error=None)
Returns the initial error of the fitting process, if the ground truth shape and initial shape exist. This is the error computed based on the `initial_shape`.

Parameters `compute_error` (callable, optional) – Callable that computes the error between the initial and ground truth shapes.

Returns `initial_error` (float) – The initial error at the beginning of the fitting process.

Raises

- `ValueError` – Initial shape has not been set, so the initial error cannot be computed.
- `ValueError` – Ground truth shape has not been set, so the initial error cannot be computed.

`plot_costs` (figure_id=None, new_figure=False, render_lines=True, line_colour='b', line_style='-', line_width=2, render_markers=True, marker_style='o', marker_size=4, marker_face_colour='b', marker_edge_colour='k', marker_edge_width=1.0, render_axes=True, axes_font_name='sans-serif', axes_font_size=10, axes_font_style='normal', axes_font_weight='normal', axes_x_limits=0.0, axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None, figure_size=(10, 6), render_grid=True, grid_line_style='–', grid_line_width=0.5)
Plot of the cost function evolution at each fitting iteration.

Parameters

- `figure_id` (object, optional) – The id of the figure to be used.
- `new_figure` (bool, optional) – If True, a new figure is created.
- `render_lines` (bool, optional) – If True, the line will be rendered.
MenpoFit Documentation, Release 0.6.1

• **line_colour** *(colour or None, optional)* – The colour of the line. If `None`, the colour is sampled from the jet colormap. Example `colour` options are

```python
{'r', 'g', 'b', 'c', 'm', 'k', 'w'}
```

or

```python
(3, ) ndarray
```

• **line_style** *(str, optional)* – The style of the lines.

• **line_width** *(float, optional)* – The width of the lines.

• **render_markers** *(bool, optional)* – If `True`, the markers will be rendered.

• **marker_style** *(marker, optional)* – The style of the markers. Example `marker` options are

```python
{'.', ',', 'o', 'v', '^', '<', '>', '+', 'x', 'D', 'd', 's', 'p', 's', 'h', 'H', '1', '2', '3', '4', '8'}
```

• **marker_size** *(int, optional)* – The size of the markers in points.

• **marker_face_colour** *(colour or None, optional)* – The face (filling) colour of the markers. If `None`, the colour is sampled from the jet colormap. Example `colour` options are

```python
{'r', 'g', 'b', 'c', 'm', 'k', 'w'}
```

or

```python
(3, ) ndarray
```

• **marker_edge_colour** *(colour or None, optional)* – The edge colour of the markers. If `None`, the colour is sampled from the jet colormap. Example `colour` options are

```python
{'r', 'g', 'b', 'c', 'm', 'k', 'w'}
```

or

```python
(3, ) ndarray
```

• **marker_edge_width** *(float, optional)* – The width of the markers’ edge.

• **render_axes** *(bool, optional)* – If `True`, the axes will be rendered.

• **axes_font_name** *(See below, optional)* – The font of the axes. Example options

```python
{'serif', 'sans-serif', 'cursive', 'fantasy', 'monospace'}
```

• **axes_font_size** *(int, optional)* – The font size of the axes.

• **axes_font_style** *(See below, optional)* – The font style of the axes.

• **axes_font_weight** *(See below, optional)* – The font weight of the axes. Example options

```python
{'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'}
```

• **axes_x_limits** *(float or (float, float) or None, optional)* – The limits of the x axis. If `float`, then it sets padding on the right and left of the graph as a percentage of the curves’ width. If `tuple` or `list`, then it defines the axis limits. If `None`, then the limits are set automatically.
MenpoFit Documentation, Release 0.6.1

- **axes_y_limits** *(float or (float, float) or None, optional)* – The limits of the y axis. If *float*, then it sets padding on the top and bottom of the graph as a percentage of the curves’ height. If *tuple* or *list*, then it defines the axis limits. If *None*, then the limits are set automatically.

- **axes_x_ticks** *(list or tuple or None, optional)* – The ticks of the x axis.

- **axes_y_ticks** *(list or tuple or None, optional)* – The ticks of the y axis.

- **figure_size** *((float, float) or None, optional)* – The size of the figure in inches.

- **render_grid** *(bool, optional)* – If *True*, the grid will be rendered.

- **grid_line_style** *((str), optional)* – The style of the grid lines.

- **grid_line_width** *(float, optional)* – The width of the grid lines.

Returns *renderer* *(menpo.visualize.GraphPlotter)* – The renderer object.

**plot_displacements** *(stat_type='mean', figure_id=None, new_figure=False, render_lines=True, line_colour='b', line_style='-', line_width=2, render_markers=True, marker_style='o', marker_size=4, marker_face_colour='b', marker_edge_colour='k', marker_edge_width=1.0, render_axes=True, axes_font_name='sans-serif', axes_font_size=10, axes_font_style='normal', axes_font_weight='normal', axes_x_limits=0.0, axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None, figure_size=(10, 6), render_grid=True, grid_line_style='--', grid_line_width=0.5)*

Plot of a statistic metric of the displacement between the shape of each iteration and the shape of the previous one.

Parameters

- **stat_type** *(mean, median, min, max), optional)* – Specifies a statistic metric to be extracted from the displacements (see also *displacements_stats()* method).

- **figure_id** *(object, optional)* – The id of the figure to be used.

- **new_figure** *(bool, optional)* – If *True*, a new figure is created.

- **render_lines** *(bool, optional)* – If *True*, the line will be rendered.

- **line_colour** *(colour or None (See below), optional)* – The colour of the line. If *None*, the colour is sampled from the jet colormap. Example *colour* options are

```
[r, g, b, c, m, k, w]
or
(3,) ndarray
```

- **line_style** *(str (See below), optional)* – The style of the lines. Example options:

```
[\-\-, \-\-, \-, :, ]
```

- **line_width** *(float, optional)* – The width of the lines.

- **render_markers** *(bool, optional)* – If *True*, the markers will be rendered.

- **marker_style** *(str (See below), optional)* – The style of the markers. Example *marker* options

```
[\., ,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8]
```

- **marker_size** *(int, optional)* – The size of the markers in points.
MenpoFit Documentation, Release 0.6.1

- **marker_face_colour** *(colour or None, optional)* – The face (filling) colour of the markers. If None, the colour is sampled from the jet colormap. Example *colour* options are

  ```
  {r, g, b, c, m, k, w}
  or
  (3, ) ndarray
  ```

- **marker_edge_colour** *(colour or None, optional)* – The edge colour of the markers. If None, the colour is sampled from the jet colormap. Example *colour* options are

  ```
  {r, g, b, c, m, k, w}
  or
  (3, ) ndarray
  ```

- **marker_edge_width** *(float, optional)* – The width of the markers’ edge.

- **render_axes** *(bool, optional)* – If True, the axes will be rendered.

- **axes_font_name** *(str (See below), optional)* – The font of the axes. Example options are

  ```
  {serif, sans-serif, cursive, fantasy, monospace}
  ```

- **axes_font_size** *(int, optional)* – The font size of the axes.

- **axes_font_style** *(str (See below), optional)* – The font style of the axes. Example options are

  ```
  {normal, italic, oblique}
  ```

- **axes_font_weight** *(str (See below), optional)* – The font weight of the axes. Example options are

  ```
  {ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
  ```

- **axes_x_limits** *(float or (float, float) or None, optional)* – The limits of the x axis. If float, then it sets padding on the right and left of the graph as a percentage of the curves’ width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.

- **axes_y_limits** *(float or (float, float) or None, optional)* – The limits of the y axis. If float, then it sets padding on the top and bottom of the graph as a percentage of the curves’ height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.

- **axes_x_ticks** *(list or tuple or None, optional)* – The ticks of the x axis.

- **axes_y_ticks** *(list or tuple or None, optional)* – The ticks of the y axis.

- **figure_size** *((float, float) or None, optional)* – The size of the figure in inches.

- **render_grid** *(bool, optional)* – If True, the grid will be rendered.

- **grid_line_style** *(('-', '--', '-', ':'), optional)* – The style of the grid lines.

- **grid_line_width** *(float, optional)* – The width of the grid lines.

**Returns** `renderer` *(menpo.visualize.GraphPlotter)* – The renderer object.
**plot_errors** (compute_error=None, figure_id=None, new_figure=False, render_lines=True, line_colour='b', line_style='-', line_width=2, render_markers=True, marker_style='o', marker_size=4, marker_face_colour='b', marker_edge_colour='k', marker_edge_width=1.0, render_axes=True, axes_font_name='sans-serif', axes_font_size=10, axes_font_style='normal', axes_font_weight='normal', axes_x_limits=0.0, axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None, figure_size=(10, 6), render_grid=True, grid_line_style='–', grid_line_width=0.5)

Plot of the error evolution at each fitting iteration.

**Parameters**

- **compute_error** (callable, optional) – Callable that computes the error between the shape at each iteration and the ground truth shape.

- **figure_id** (object, optional) – The id of the figure to be used.

- **new_figure** (bool, optional) – If True, a new figure is created.

- **render_lines** (bool, optional) – If True, the line will be rendered.

- **line_colour** (colour or None (See below), optional) – The colour of the line. If None, the colour is sampled from the jet colormap. Example colour options are

  ```plaintext
  {r, g, b, c, m, k, w}
  or
  (3, ) ndarray
  ```

- **line_style** (str (See below), optional) – The style of the lines. Example options:

  ```plaintext
  [-, --, -, :]
  ```

- **line_width** (float, optional) – The width of the lines.

- **render_markers** (bool, optional) – If True, the markers will be rendered.

- **marker_style** (str (See below), optional) – The style of the markers. Example marker options

  ```plaintext
  {., ,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
  ```

- **marker_size** (int, optional) – The size of the markers in points.

- **marker_face_colour** (colour or None, optional) – The face (filling) colour of the markers. If None, the colour is sampled from the jet colormap. Example colour options are

  ```plaintext
  {r, g, b, c, m, k, w}
  or
  (3, ) ndarray
  ```

- **marker_edge_colour** (colour or None, optional) – The edge colour of the markers. If None, the colour is sampled from the jet colormap. Example colour options are

  ```plaintext
  {r, g, b, c, m, k, w}
  or
  (3, ) ndarray
  ```

- **marker_edge_width** (float, optional) – The width of the markers’ edge.

- **render_axes** (bool, optional) – If True, the axes will be rendered.

- **axes_font_name** (str (See below), optional) – The font of the axes. Example options
MenpoFit Documentation, Release 0.6.1

• axes_font_size (int, optional) – The font size of the axes.

• axes_font_style (str (See below), optional) – The font style of the axes. Example options

{normal, italic, oblique}

• axes_font_weight (str (See below), optional) – The font weight of the axes. Example options

{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}

• axes_x_limits (float or (float, float) or None, optional) – The limits of the x axis. If float, then it sets padding on the right and left of the graph as a percentage of the curves’ width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.

• axes_y_limits (float or (float, float) or None, optional) – The limits of the y axis. If float, then it sets padding on the top and bottom of the graph as a percentage of the curves’ height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.

• axes_x_ticks (list or tuple or None, optional) – The ticks of the x axis.

• axes_y_ticks (list or tuple or None, optional) – The ticks of the y axis.

• figure_size ((float, float) or None, optional) – The size of the figure in inches.

• render_grid (bool, optional) – If True, the grid will be rendered.

• grid_line_style ({'-', '--', '-.', ':'}, optional) – The style of the grid lines.

• grid_line_width (float, optional) – The width of the grid lines.

Returns renderer (menpo.visualize.GraphPlotter) – The renderer object.

reconstructed_initial_error (compute_error=None)

Returns the error of the reconstructed initial shape of the fitting process, if the ground truth shape exists. This is the error computed based on the reconstructed_initial_shape.

Parameters compute_error (callable, optional) – Callable that computes the error between the reconstructed initial and ground truth shapes.

Returns reconstructed_initial_error (float) – The error that corresponds to the initial shape’s reconstruction.

Raises ValueError – Ground truth shape has not been set, so the reconstructed initial error cannot be computed

to_result (pass_image=True, pass_initial_shape=True, pass_gt_shape=True)

Returns a Result instance of the object, i.e. a fitting result object that does not store the iterations. This can be useful for reducing the size of saved fitting results.

Parameters

• pass_image (bool, optional) – If True, then the image will get passed (if it exists).

• pass_initial_shape (bool, optional) – If True, then the initial shape will get passed (if it exists).
• **pass_gt_shape** *(bool, optional)* – If True, then the ground truth shape will get passed (if it exists).

**Returns**

result *(Result)* – The final “lightweight” fitting result.

view *(figure_id=None, new_figure=False, render_image=True, render_final_shape=True, render_initial_shape=False, render_gt_shape=False, subplots_enabled=True, channels=None, interpolation='bilinear', cmap_name=None, alpha=1.0, masked=False, final_marker_face_colour='r', final_marker_edge_colour='k', final_line_colour='b', gt_marker_face_colour='y', gt_marker_edge_colour='k', gt_line_colour='y', render_lines=True, line_style='-', line_width=2, render_markers=True, marker_style='o', marker_size=4, marker_edge_width=1.0, render_numbering=False, numbers_horizontal_align='center', numbers_vertical_align='bottom', numbers_font_name='sans-serif', numbers_font_size=10, numbers_font_style='normal', numbers_font_weight='normal', numbers_font_colour='k', render_legend=True, legend_title='', legend_font_name='sans-serif', legend_font_style='normal', legend_font_size=10, legend_font_weight='normal', legend_marker_scale=None, legend_location=2, legend_bbox_to_anchor=(1.05, 1.0), legend_horizontal_spacing=None, legend_vertical_spacing=None, legend_border=True, legend_border_padding=None, legend_shadow=False, legend_rounded_corners=False, render_axes=True, axes_font_name='sans-serif', axes_font_size=10, axes_font_style='normal', axes_font_weight='normal', axes_x_limits=None, axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None, figure_size=(7, 7))

Visualize the fitting result. The method renders the final fitted shape and optionally the initial shape, ground truth shape and the image, if they were provided.

**Parameters**

• **figure_id** *(object, optional)* – The id of the figure to be used.

• **new_figure** *(bool, optional)* – If True, a new figure is created.

• **render_image** *(bool, optional)* – If True and the image exists, then it gets rendered.

• **render_final_shape** *(bool, optional)* – If True, then the final fitting shape gets rendered.

• **render_initial_shape** *(bool, optional)* – If True and the initial fitting shape exists, then it gets rendered.

• **render_gt_shape** *(bool, optional)* – If True and the ground truth shape exists, then it gets rendered.

• **subplots_enabled** *(bool, optional)* – If True, then the requested final, initial and ground truth shapes get rendered on separate subplots.

• **channels** *(int or list of int or all or None)* – If int or list of int, the specified channel(s) will be rendered. If all, all the channels will be rendered in subplots. If None and the image is RGB, it will be rendered in RGB mode. If None and the image is not RGB, it is equivalent to all.

• **interpolation** *(See Below, optional)* – The interpolation used to render the image. For example, if bilinear, the image will be smooth and if nearest, the image will be pixelated. Example options

```python
{none, nearest, bilinear, bicubic, spline16, spline36, hanning, hamming, hermite, kaiser, quadric, catrom, gaussian, bessel, mitchell, sinc, lanczos}
```
- **cmap_name** (*str*, optional) – If None, single channel and three channel images default to greyscale and rgb colormaps respectively.
- **alpha** (*float*, optional) – The alpha blending value, between 0 (transparent) and 1 (opaque).
- **masked** (*bool*, optional) – If True, then the image is rendered as masked.
- **final_marker_face_colour** (*See Below, optional*) – The face (filling) colour of the markers of the final fitting shape. Example options
  
  ```
  [r, g, b, c, m, k, w]
  or
  (3, ) ndarray
  ```
- **final_marker_edge_colour** (*See Below, optional*) – The edge colour of the markers of the final fitting shape. Example options
  
  ```
  [r, g, b, c, m, k, w]
  or
  (3, ) ndarray
  ```
- **final_line_colour** (*See Below, optional*) – The line colour of the final fitting shape. Example options
  
  ```
  [r, g, b, c, m, k, w]
  or
  (3, ) ndarray
  ```
- **initial_marker_face_colour** (*See Below, optional*) – The face (filling) colour of the markers of the initial shape. Example options
  
  ```
  [r, g, b, c, m, k, w]
  or
  (3, ) ndarray
  ```
- **initial_marker_edge_colour** (*See Below, optional*) – The edge colour of the markers of the initial shape. Example options
  
  ```
  [r, g, b, c, m, k, w]
  or
  (3, ) ndarray
  ```
- **initial_line_colour** (*See Below, optional*) – The line colour of the initial shape. Example options
  
  ```
  [r, g, b, c, m, k, w]
  or
  (3, ) ndarray
  ```
- **gt_marker_face_colour** (*See Below, optional*) – The face (filling) colour of the markers of the ground truth shape. Example options
  
  ```
  [r, g, b, c, m, k, w]
  or
  (3, ) ndarray
  ```
- **gt_marker_edge_colour** (*See Below, optional*) – The edge colour of the markers of the ground truth shape. Example options
  
  ```
  [r, g, b, c, m, k, w]
  or
  (3, ) ndarray
  ```
• **gt_line_colour** (*See Below, optional*) – The line colour of the ground truth shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• **render_lines** (*bool or list of bool, optional*) – If True, the lines will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (*final, initial, groundtruth*) order.

• **line_style** (*str or list of str, optional*) – The style of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (*final, initial, groundtruth*) order. Example options:

```
{'-', '--', '-.', ':'}
```

• **line_width** (*float or list of float, optional*) – The width of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (*final, initial, groundtruth*) order.

• **render_markers** (*bool or list of bool, optional*) – If True, the markers will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (*final, initial, groundtruth*) order.

• **marker_style** (*str or list of str, optional*) – The style of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (*final, initial, groundtruth*) order. Example options:

```
{., ,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
```

• **marker_size** (*int or list of int, optional*) – The size of the markers in points. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (*final, initial, groundtruth*) order.

• **marker_edge_width** (*float or list of float, optional*) – The width of the markers’ edge. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (*final, initial, groundtruth*) order.

• **render_numbering** (*bool, optional*) – If True, the landmarks will be numbered.

• **numbers_horizontal_align** (*{center, right, left}, optional*) – The horizontal alignment of the numbers’ texts.

• **numbers_vertical_align** (*{center, top, bottom, baseline}, optional*) – The vertical alignment of the numbers’ texts.

• **numbers_font_name** (*See Below, optional*) – The font of the numbers. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

• **numbers_font_size** (*int, optional*) – The font size of the numbers.

• **numbers_font_style** (*{normal, italic, oblique}, optional*) – The font style of the numbers.
• **numbers_font_weight** *(See Below, optional)* – The font weight of the numbers. Example options

{ultralight, light, normal, regular, book, medium, roman,
semibold, demibold, demi, bold, heavy, extra bold, black}

• **numbers_font_colour** *(See Below, optional)* – The font colour of the numbers. Example options

[r, g, b, c, m, k, w]
or
(3, ) ndarray

• **render_legend** *(bool, optional)* – If True, the legend will be rendered.

• **legend_title** *(str, optional)* – The title of the legend.

• **legend_font_name** *(See below, optional)* – The font of the legend. Example options

{serif, sans-serif, cursive, fantasy, monospace}

• **legend_font_style** {{normal, italic, oblique}, optional} – The font style of the legend.

• **legend_font_size** *(int, optional)* – The font size of the legend.

• **legend_font_weight** *(See Below, optional)* – The font weight of the legend. Example options

{ultralight, light, normal, regular, book, medium, roman,
semibold, demibold, demi, bold, heavy, extra bold, black}

• **legend_marker_scale** *(float, optional)* – The relative size of the legend markers with respect to the original

• **legend_location** *(int, optional)* – The location of the legend. The predefined values are:

<table>
<thead>
<tr>
<th>Location</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>'best'</td>
<td>0</td>
</tr>
<tr>
<td>'upper right'</td>
<td>1</td>
</tr>
<tr>
<td>'upper left'</td>
<td>2</td>
</tr>
<tr>
<td>'lower left'</td>
<td>3</td>
</tr>
<tr>
<td>'lower right'</td>
<td>4</td>
</tr>
<tr>
<td>'right'</td>
<td>5</td>
</tr>
<tr>
<td>'center left'</td>
<td>6</td>
</tr>
<tr>
<td>'center right'</td>
<td>7</td>
</tr>
<tr>
<td>'lower center'</td>
<td>8</td>
</tr>
<tr>
<td>'upper center'</td>
<td>9</td>
</tr>
<tr>
<td>'center'</td>
<td>10</td>
</tr>
</tbody>
</table>

• **legend_bbox_to_anchor** {{float, float} tuple, optional} – The bbox that the legend will be anchored.

• **legend_border_axes_pad** *(float, optional)* – The pad between the axes and legend border.

• **legend_n_columns** *(int, optional)* – The number of the legend’s columns.
- `legend_horizontal_spacing (float, optional)` – The spacing between the columns.
- `legend_vertical_spacing (float, optional)` – The vertical space between the legend entries.
- `legend_border (bool, optional)` – If True, a frame will be drawn around the legend.
- `legend_border_padding (float, optional)` – The fractional whitespace inside the legend border.
- `legend_shadow (bool, optional)` – If True, a shadow will be drawn behind legend.
- `legend_rounded_corners (bool, optional)` – If True, the frame’s corners will be rounded (fancybox).
- `render_axes (bool, optional)` – If True, the axes will be rendered.
- `axes_font_name (See Below, optional)` – The font of the axes. Example options
  
  ```
  {serif, sans-serif, cursive, fantasy, monospace}
  ```

- `axes_font_size (int, optional)` – The font size of the axes.
- `axes_font_style (normal, italic, oblique, optional)` – The font style of the axes.
- `axes_font_weight (See Below, optional)` – The font weight of the axes. Example options
  
  ```
  {ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
  ```

- `axes_x_limits (float or (float, float) or None, optional)` – The limits of the x axis. If float, then it sets padding on the right and left of the Image as a percentage of the Image’s width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- `axes_y_limits ((float, float) tuple or None, optional)` – The limits of the y axis. If float, then it sets padding on the top and bottom of the Image as a percentage of the Image’s height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- `axes_x_ticks (list or tuple or None, optional)` – The ticks of the x axis.
- `axes_y_ticks (list or tuple or None, optional)` – The ticks of the y axis.
- `figure_size ((float, float) tuple or None optional)` – The size of the figure in inches.

Returns `renderer (class)` – The renderer object.
view_iterations (figure_id=None, new_figure=False, iters=None, render_image=True, subplots_enabled=False, channels=None, interpolation='bilinear', cmap_name=None, alpha=1.0, masked=True, render_lines=True, line_style='-', line_width=2, line_colour=None, render_markers=True, marker_edge_colour=None, marker_face_colour=None, marker_style='o', marker_size=4, marker_edge_width=1.0, render_numbering=False, numbers_horizontal_align='center', numbers_vertical_align='bottom', numbers_font_name='sans-serif', numbers_font_size=10, numbers_font_style='normal', numbers_font_weight='normal', numbers_font_colour='k', render_legend=True, legend_title='', legend_font_name='sans-serif', legend_font_style='normal', legend_font_size=10, legend_font_weight='normal', legend_marker_scale=None, legend_location=2, legend_bbox_to_anchor=(1.05, 1.0), legend_border_axes_pad=None, legend_n_columns=1, legend_horizontal_spacing=None, legend_vertical_spacing=None, legend_border=True, legend_border_padding=None, legend_shadow=False, legend_rounded_corners=False, render_axes=False, axes_font_name='sans-serif', axes_font_size=10, axes_font_style='normal', axes_font_weight='normal', axes_x_limits=None, axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None, figure_size=(7, 7))

Visualize the iterations of the fitting process.

Parameters

- **figure_id** *(object, optional)* – The id of the figure to be used.
- **new_figure** *(bool, optional)* – If True, a new figure is created.
- **iters** *(int or list of int or None, optional)* – The iterations to be visualized. If None, then all the iterations are rendered.

<table>
<thead>
<tr>
<th>No.</th>
<th>Visualised shape</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>self.initial_shape</td>
<td>Initial shape</td>
</tr>
<tr>
<td>1</td>
<td>self.reconstructed_initial_shape</td>
<td>Reconstructed initial</td>
</tr>
<tr>
<td>2</td>
<td>self.shapes[2]</td>
<td>Iteration 1</td>
</tr>
<tr>
<td>i</td>
<td>self.shapes[i]</td>
<td>Iteration i-1</td>
</tr>
<tr>
<td>n_iters+1</td>
<td>self.final_shape</td>
<td>Final shape</td>
</tr>
</tbody>
</table>

- **render_image** *(bool, optional)* – If True and the image exists, then it gets rendered.
- **subplots_enabled** *(bool, optional)* – If True, then the requested final, initial and ground truth shapes get rendered on separate subplots.
- **channels** *(int or list of int or all or None, optional)* – If int or list of int, the specified channel(s) will be rendered. If all, all the channels will be rendered in subplots. If None and the image is RGB, it will be rendered in RGB mode. If None and the image is not RGB, it is equivalent to all.
- **interpolation** *(str (See Below), optional)* – The interpolation used to render the image. For example, if bilinear, the image will be smooth and if nearest, the image will be pixelated. Example options

   {none, nearest, bilinear, bicubic, spline16, spline36, hanning, hamming, hermite, kaiser, quadric, catrom, gaussian, bessel, mitchell, sinc, lanczos}

- **cmap_name** *(str, optional,)* – If None, single channel and three channel images default to greyscale and rgb colormaps respectively.
• **alpha** *(float, optional)* – The alpha blending value, between 0 (transparent) and 1 (opaque).

• **masked** *(bool, optional)* – If True, then the image is rendered as masked.

• **render_lines** *(bool or list of bool, optional)* – If True, the lines will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.

• **line_style** *(str or list of str (See below), optional)* – The style of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options:

```plaintext
(-, --, -. , :)
```

• **line_width** *(float or list of float, optional)* – The width of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.

• **line_colour** *(colour or list of colour (See Below), optional)* – The colour of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```plaintext
{r, g, b, c, m, k, w}
```

• **render_markers** *(bool or list of bool, optional)* – If True, the markers will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.

• **marker_style** *(str or list of str (See below), optional)* – The style of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```plaintext
{., ,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
```

• **marker_size** *(int or list of int, optional)* – The size of the markers in points. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.

• **marker_edge_colour** *(colour or list of colour (See Below), optional)* – The edge colour of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```plaintext
{r, g, b, c, m, k, w}
```

• **marker_face_colour** *(colour or list of colour (See Below), optional)* – The face (filling) colour of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```plaintext
{r, g, b, c, m, k, w}
```

• **marker_edge_width** *(float or list of float, optional)* – The width of the markers’ edge. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.
• **render_numbering** *(bool, optional)* – If True, the landmarks will be numbered.

• **numbers_horizontal_align** *(str (See below), optional)* – The horizontal alignment of the numbers’ texts. Example options

  `{center, right, left}`

• **numbers_vertical_align** *(str (See below), optional)* – The vertical alignment of the numbers’ texts. Example options

  `{center, top, bottom, baseline}`

• **numbers_font_name** *(str (See below), optional)* – The font of the numbers. Example options

  `{serif, sans-serif, cursive, fantasy, monospace}`

• **numbers_font_size** *(int, optional)* – The font size of the numbers.

• **numbers_font_style** *(str (normal, italic, oblique), optional)* – The font style of the numbers.

• **numbers_font_weight** *(str (See below), optional)* – The font weight of the numbers. Example options

  `{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}`

• **numbers_font_colour** *(See Below, optional)* – The font colour of the numbers. Example options

  `{r, g, b, c, m, k, w}`
  
  or
  
  `(3, ) ndarray`

• **render_legend** *(bool, optional)* – If True, the legend will be rendered.

• **legend_title** *(str, optional)* – The title of the legend.

• **legend_font_name** *(See below, optional)* – The font of the legend. Example options

  `{serif, sans-serif, cursive, fantasy, monospace}`

• **legend_font_style** *(str (See below), optional)* – The font style of the legend. Example options

  `{normal, italic, oblique}`

• **legend_font_size** *(int, optional)* – The font size of the legend.

• **legend_font_weight** *(str (See below), optional)* – The font weight of the legend. Example options

  `{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}`
• **legend_marker_scale** *(float, optional)* – The relative size of the legend markers with respect to the original

• **legend_location** *(int, optional)* – The location of the legend. The predefined values are:

<table>
<thead>
<tr>
<th>Location</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>'best'</td>
<td>0</td>
</tr>
<tr>
<td>'upper right'</td>
<td>1</td>
</tr>
<tr>
<td>'upper left'</td>
<td>2</td>
</tr>
<tr>
<td>'lower left'</td>
<td>3</td>
</tr>
<tr>
<td>'lower right'</td>
<td>4</td>
</tr>
<tr>
<td>'right'</td>
<td>5</td>
</tr>
<tr>
<td>'center left'</td>
<td>6</td>
</tr>
<tr>
<td>'center right'</td>
<td>7</td>
</tr>
<tr>
<td>'lower center'</td>
<td>8</td>
</tr>
<tr>
<td>'upper center'</td>
<td>9</td>
</tr>
<tr>
<td>'center'</td>
<td>10</td>
</tr>
</tbody>
</table>

• **legend_bbox_to_anchor** *(float, float tuple, optional)* – The bbox that the legend will be anchored.

• **legend_border_axes_pad** *(float, optional)* – The pad between the axes and legend border.

• **legend_n_columns** *(int, optional)* – The number of the legend’s columns.

• **legend_horizontal_spacing** *(float, optional)* – The spacing between the columns.

• **legend_vertical_spacing** *(float, optional)* – The vertical space between the legend entries.

• **legend_border** *(bool, optional)* – If True, a frame will be drawn around the legend.

• **legend_border_padding** *(float, optional)* – The fractional whitespace inside the legend border.

• **legend_shadow** *(bool, optional)* – If True, a shadow will be drawn behind legend.

• **legend_rounded_corners** *(bool, optional)* – If True, the frame’s corners will be rounded (fancybox).

• **render_axes** *(bool, optional)* – If True, the axes will be rendered.

• **axes_font_name** *(str (See below), optional)* – The font of the axes. Example options:

```
{serif, sans-serif, cursive, fantasy, monospace}
```

• **axes_font_size** *(int, optional)* – The font size of the axes.

• **axes_font_style** *(normal, italic, oblique, optional)* – The font style of the axes.

• **axes_font_weight** *(str (See below), optional)* – The font weight of the axes. Example options:

```
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```

• **axes_x_limits** *(float or (float, float) or None, optional)* – The limits of the x axis. If float, then it sets padding on the right and left of the Image as a percentage of the Image’s
width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.

- **axes_y_limits** ((float, float) tuple or None, optional) – The limits of the y axis. If float, then it sets padding on the top and bottom of the Image as a percentage of the Image’s height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.

- **axes_x_ticks** (list or tuple or None, optional) – The ticks of the x axis.

- **axes_y_ticks** (list or tuple or None, optional) – The ticks of the y axis.

- **figure_size** ((float, float) tuple or None optional) – The size of the figure in inches.

  Returns **renderer**(class) – The renderer object.

  **view_widget**(figure_size=(7, 7))

  Visualizes the result object using an interactive widget.

  Parameters **figure_size**(int, int, optional) – The initial size of the rendered figure.

  **property costs**

  Returns a list with the cost per iteration. It returns None if the costs are not computed.

  Type list of float or None

  **property final_shape**

  Returns the final shape of the fitting process.

  Type menpo.shape.PointCloud

  **property gt_shape**

  Returns the ground truth shape associated with the image. In case there is not an attached ground truth shape, then None is returned.

  Type menpo.shape.PointCloud or None

  **property image**

  Returns the image that the fitting was applied on, if it was provided. Otherwise, it returns None.

  Type menpo.shape.Image or subclass or None

  **property initial_shape**

  Returns the initial shape that was provided to the fitting method to initialise the fitting process. In case the initial shape does not exist, then None is returned.

  Type menpo.shape.PointCloud or None

  **property is_iterative**

  Flag whether the object is an iterative fitting result.

  Type bool

  **property n_iters**

  Returns the total number of iterations of the fitting process.

  Type int

  **property reconstructed_initial_shape**

  Returns the initial shape’s reconstruction with the shape model that was used to initialise the iterative optimisation process.

  Type menpo.shape.PointCloud
property `shape_parameters`
Returns the list of shape parameters obtained at each iteration of the fitting process. The list includes the parameters of the `reconstructed_initial_shape` and `final_shape`.

Type `list of (n_params,) ndarray`

property `shapes`
Returns the list of shapes obtained at each iteration of the fitting process. The list includes the `initial_shape` (if it exists), `reconstructed_initial_shape` and `final_shape`.

Type `list of menpo.shape.PointCloud`

Multi-Scale Iterative Result

Classes for defining a multi-scale iterative fitting result.

MultiScaleNonParametricIterativeResult

class `menpofit.result.MultiScaleNonParametricIterativeResult`

Bases: `NonParametricIterativeResult`

Class for defining a multi-scale non-parametric iterative fitting result, i.e. the result of a multi-scale method that does not optimise over a parametric shape model. It holds the shapes of all the iterations of the fitting procedure, as well as the scales. It can optionally store the image on which the fitting was applied, as well as its ground truth shape.

Parameters

- `results` (list of `NonParametricIterativeResult`) – The list of non parametric iterative results per scale.
- `scales` (list of float) – The scale values (normally small to high).
- `affine_transforms` (list of `menpo.transform.Affine`) – The list of affine transforms per scale that transform the shapes into the original image space.
- `scale_transforms` (list of `menpo.shape.Scale`) – The list of scaling transforms per scale.
- `image` (`menpo.image.Image` or subclass or None, optional) – The image on which the fitting process was applied. Note that a copy of the image will be assigned as an attribute. If `None`, then no image is assigned.
- `gt_shape` (`menpo.shape.PointCloud` or `None`, optional) – The ground truth shape associated with the image. If `None`, then no ground truth shape is assigned.

`displacements()`
A list containing the displacement between the shape of each iteration and the shape of the previous one.

Type `list of ndarray`

`displacements_stats` (stat_type='mean')
A list containing a statistical metric on the displacements between the shape of each iteration and the shape of the previous one.
Parameters `stat_type` ({$'mean$', $'median$', $'min$', $'max$}, optional) – Specifies a statistic metric to be extracted from the displacements.

Returns `displacements_stat` (list of float) – The statistical metric on the points displacements for each iteration.

Raises `ValueError` – type must be ‘mean’, ‘median’, ‘min’ or ‘max’

`errors` (compute_error=None)

Returns a list containing the error at each fitting iteration, if the ground truth shape exists.

Parameters `compute_error` (callable, optional) – Callable that computes the error between the shape at each iteration and the ground truth shape.

Returns `errors` (list of float) – The error at each iteration of the fitting process.

Raises `ValueError` – Ground truth shape has not been set, so the final error cannot be computed

`final_error` (compute_error=None)

Returns the final error of the fitting process, if the ground truth shape exists. This is the error computed based on the final_shape.

Parameters `compute_error` (callable, optional) – Callable that computes the error between the fitted and ground truth shapes.

Returns `final_error` (float) – The final error at the end of the fitting process.

Raises `ValueError` – Ground truth shape has not been set, so the final error cannot be computed

`initial_error` (compute_error=None)

Returns the initial error of the fitting process, if the ground truth shape and initial shape exist. This is the error computed based on the initial_shape.

Parameters `compute_error` (callable, optional) – Callable that computes the error between the initial and ground truth shapes.

Returns `initial_error` (float) – The initial error at the beginning of the fitting process.

Raises

- `ValueError` – Initial shape has not been set, so the initial error cannot be computed
- `ValueError` – Ground truth shape has not been set, so the initial error cannot be computed

`plot_costs` (figure_id=None, new_figure=False, render_lines=True, lineColour='b', lineStyle='-', lineWidth=2, render_markers=True, markerStyle='o', marker_size=4, marker_faceColour='b', marker_edgeColour='k', marker_edge_width=1.0, render_axes=True, axes_fontName='sans-serif', axes_font_size=10, axes_font_style='normal', axes_font_weight='normal', axes_xLimits=0.0, axes_yLimits=None, axes_x_ticks=None, axes_y_ticks=None, figure_size=(10, 6), render_grid=True, grid_line_style='–', grid_line_width=0.5)

Plot of the cost function evolution at each fitting iteration.

Parameters

- `figure_id` (object, optional) – The id of the figure to be used.
- `new_figure` (bool, optional) – If True, a new figure is created.
- `render_lines` (bool, optional) – If True, the line will be rendered.
- **line_colour (colour or None, optional)** – The colour of the line. If None, the colour is sampled from the jet colormap. Example `colour` options are

```plaintext
['r', 'g', 'b', 'c', 'm', 'k', 'w']
```

- **line_style (str, optional)** – The style of the lines.
- **line_width (float, optional)** – The width of the lines.
- **render_markers (bool, optional)** – If True, the markers will be rendered.
- **marker_style (marker, optional)** – The style of the markers. Example `marker` options are

```plaintext
['.', ',', 'o', 'v', '^', '<', '>', '+', 'x', 'D', 'd', 's', 'p', '*', 'h', 'H', '1', '2', '3', '4', '8']
```

- **marker_size (int, optional)** – The size of the markers in points.
- **marker_face_colour (colour or None, optional)** – The face (filling) colour of the markers. If None, the colour is sampled from the jet colormap. Example `colour` options are

```plaintext
['r', 'g', 'b', 'c', 'm', 'k', 'w']
```

- **marker_edge_colour (colour or None, optional)** – The edge colour of the markers. If None, the colour is sampled from the jet colormap. Example `colour` options are

```plaintext
['r', 'g', 'b', 'c', 'm', 'k', 'w']
```

- **marker_edge_width (float, optional)** – The width of the markers’ edge.
- **render_axes (bool, optional)** – If True, the axes will be rendered.
- **axes_font_name (See below, optional)** – The font of the axes. Example options

```plaintext
['serif', 'sans-serif', 'cursive', 'fantasy', 'monospace']
```

- **axes_font_size (int, optional)** – The font size of the axes.
- **axes_font_style ([normal, italic, oblique], optional)** – The font style of the axes.
- **axes_font_weight (See below, optional)** – The font weight of the axes. Example options

```plaintext
['ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black']
```

- **axes_x_limits (float or (float, float) or None, optional)** – The limits of the x axis. If float, then it sets padding on the right and left of the graph as a percentage of the curves’ width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
• **axes_y_limits** *(float or (float, float) or None, optional)* – The limits of the y axis. If `float`, then it sets padding on the top and bottom of the graph as a percentage of the curves’ height. If `tuple` or `list`, then it defines the axis limits. If `None`, then the limits are set automatically.

• **axes_x_ticks** *(list or tuple or None, optional)* – The ticks of the x axis.

• **axes_y_ticks** *(list or tuple or None, optional)* – The ticks of the y axis.

• **figure_size** *((float, float) or None, optional)* – The size of the figure in inches.

• **render_grid** *(bool, optional)* – If `True`, the grid will be rendered.

• **grid_line_style** *({'-', '--', '-.', ':'}, optional)* – The style of the grid lines.

• **grid_line_width** *(float, optional)* – The width of the grid lines.

**Returns** `renderer` *(menpo.visualize.GraphPlotter)* – The renderer object.

**plot_displacements** *(stat_type='mean', figure_id=None, new_figure=False, render_lines=True, line_colour='b', line_style='-', line_width=2, render_markers=True, marker_style='o', marker_size=4, marker_face_colour='b', marker_edge_colour='k', marker_edge_width=1.0, render_axes=True, axes_font_name='sans-serif', axes_font_size=10, axes_font_style='normal', axes_font_weight='normal', axes_x_limits=0.0, axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None, figure_size=(10, 6), render_grid=True, grid_line_style='-', grid_line_width=0.5)*

Plot of a statistical metric of the displacement between the shape of each iteration and the shape of the previous one.

**Parameters**

• **stat_type** *((mean, median, min, max), optional)* – Specifies a statistic metric to be extracted from the displacements (see also `displacements_stats()` method).

• **figure_id** *(object, optional)* – The id of the figure to be used.

• **new_figure** *(bool, optional)* – If `True`, a new figure is created.

• **render_lines** *(bool, optional)* – If `True`, the line will be rendered.

• **line_colour** *(colour or None (See below), optional)* – The colour of the line. If `None`, the colour is sampled from the jet colormap. Example `colour` options are

```plaintext
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• **line_style** *(str (See below), optional)* – The style of the lines. Example options:

```plaintext
{-, --, -, :}
```

• **line_width** *(float, optional)* – The width of the lines.

• **render_markers** *(bool, optional)* – If `True`, the markers will be rendered.

• **marker_style** *(str (See below), optional)* – The style of the markers. Example `marker` options

```plaintext
{.,,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
```

• **marker_size** *(int, optional)* – The size of the markers in points.
• **marker_face_colour** (*colour* or *None*, optional) – The face (filling) colour of the markers. If *None*, the colour is sampled from the jet colormap. Example *colour* options are

```
{r, g, b, c, m, k, w}
```

or

```
(3, ) ndarray
```

• **marker_edge_colour** (*colour* or *None*, optional) – The edge colour of the markers. If *None*, the colour is sampled from the jet colormap. Example *colour* options are

```
{r, g, b, c, m, k, w}
```

or

```
(3, ) ndarray
```

• **marker_edge_width** (*float*, optional) – The width of the markers’ edge.

• **render_axes** (*bool*, optional) – If *True*, the axes will be rendered.

• **axes_font_name** (*str* (See below), optional) – The font of the axes. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

• **axes_font_size** (*int*, optional) – The font size of the axes.

• **axes_font_style** (*str* (See below), optional) – The font style of the axes. Example options

```
{normal, italic, oblique}
```

• **axes_font_weight** (*str* (See below), optional) – The font weight of the axes. Example options

```
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```

• **axes_x_limits** (*float* or (*float*, *float*) or *None*, optional) – The limits of the x axis. If *float*, then it sets padding on the right and left of the graph as a percentage of the curves’ width. If *tuple* or *list*, then it defines the axis limits. If *None*, then the limits are set automatically.

• **axes_y_limits** (*float* or (*float*, *float*) or *None*, optional) – The limits of the y axis. If *float*, then it sets padding on the top and bottom of the graph as a percentage of the curves’ height. If *tuple* or *list*, then it defines the axis limits. If *None*, then the limits are set automatically.

• **axes_x_ticks** (*list* or *tuple* or *None*, optional) – The ticks of the x axis.

• **axes_y_ticks** (*list* or *tuple* or *None*, optional) – The ticks of the y axis.

• **figure_size** (*((float, float) or None, optional) – The size of the figure in inches.

• **render_grid** (*bool*, optional) – If *True*, the grid will be rendered.

• **grid_line_style** (*{'-', '--', '-.', ':'}, optional) – The style of the grid lines.

• **grid_line_width** (*float*, optional) – The width of the grid lines.

Returns **renderer** (*menpo.visualize.GraphPlotter*) – The renderer object.
plot_errors(compute_error=None, figure_id=None, new_figure=False, render_lines=True, line_colour='b', line_style='-', line_width=2, render_markers=True, marker_style='o', marker_size=4, marker_face_colour='b', marker_edge_colour='k', marker_edge_width=1.0, render_axes=True, axes_font_name='sans-serif', axes_font_size=10, axes_font_style='normal', axes_font_weight='normal', axes_x_limits=0.0, axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None, figure_size=(10, 6), render_grid=True, grid_line_style='–', grid_line_width=0.5)

Plot of the error evolution at each fitting iteration.

Parameters

• compute_error (callable, optional) – Callable that computes the error between the shape at each iteration and the ground truth shape.

• figure_id (object, optional) – The id of the figure to be used.

• new_figure (bool, optional) – If True, a new figure is created.

• render_lines (bool, optional) – If True, the line will be rendered.

• line_colour (colour or None (See below), optional) – The colour of the line. If None, the colour is sampled from the jet colormap. Example colour options are

```python
{r, g, b, c, m, k, w}
or(3, ) ndarray
```

• line_style (str (See below), optional) – The style of the lines. Example options:

```python
[-, --, -., :]
```

• line_width (float, optional) – The width of the lines.

• render_markers (bool, optional) – If True, the markers will be rendered.

• marker_style (str (See below), optional) – The style of the markers. Example marker options

```python
{., ,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
```

• marker_size (int, optional) – The size of the markers in points.

• marker_face_colour (colour or None, optional) – The face (filling) colour of the markers. If None, the colour is sampled from the jet colormap. Example colour options are

```python
{r, g, b, c, m, k, w}
or(3, ) ndarray
```

• marker_edge_colour (colour or None, optional) – The edge colour of the markers. If None, the colour is sampled from the jet colormap. Example colour options are

```python
{r, g, b, c, m, k, w}
or(3, ) ndarray
```

• marker_edge_width (float, optional) – The width of the markers’ edge.

• render_axes (bool, optional) – If True, the axes will be rendered.

• axes_font_name (str (See below), optional) – The font of the axes. Example options
• **axes_font_size** (*int*, optional) – The font size of the axes.

• **axes_font_style** (*str* (See below), optional) – The font style of the axes. Example options

{serif, sans-serif, cursive, fantasy, monospace}

• **axes_font_weight** (*str* (See below), optional) – The font weight of the axes. Example options

{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}

• **axes_x_limits** (*float* or (*float, float*) or *None*, optional) – The limits of the x axis. If *float*, then it sets padding on the right and left of the graph as a percentage of the curves’ width. If *tuple* or *list*, then it defines the axis limits. If *None*, then the limits are set automatically.

• **axes_y_limits** (*float* or (*float, float*) or *None*, optional) – The limits of the y axis. If *float*, then it sets padding on the top and bottom of the graph as a percentage of the curves’ height. If *tuple* or *list*, then it defines the axis limits. If *None*, then the limits are set automatically.

• **axes_x_ticks** (*list* or *tuple* or *None*, optional) – The ticks of the x axis.

• **axes_y_ticks** (*list* or *tuple* or *None*, optional) – The ticks of the y axis.

• **figure_size** (*(*float, float*)* or *None*, optional) – The size of the figure in inches.

• **render_grid** (*bool*, optional) – If *True*, the grid will be rendered.

• **grid_line_style** (*{‘-’, ‘--’, ‘.-’, ‘:’}*, optional) – The style of the grid lines.

• **grid_line_width** (*float*, optional) – The width of the grid lines.

Returns **renderer** (*menpo.visualize.GraphPlotter*) – The renderer object.

**to_result** (*pass_image=True, pass_initial_shape=True, pass_gt_shape=True*)

Returns a **Result** instance of the object, i.e. a fitting result object that does not store the iterations. This can be useful for reducing the size of saved fitting results.

Parameters

• **pass_image** (*bool*, optional) – If *True*, then the image will get passed (if it exists).

• **pass_initial_shape** (*bool*, optional) – If *True*, then the initial shape will get passed (if it exists).

• **pass_gt_shape** (*bool*, optional) – If *True*, then the ground truth shape will get passed (if it exists).

Returns **result** (*Result*) – The final “lightweight” fitting result.
Visualize the fitting result. The method renders the final fitted shape and optionally the initial shape, ground truth shape and the image, if they were provided.

Parameters

- **figure_id** (object, optional) – The id of the figure to be used.
- **new_figure** (bool, optional) – If True, a new figure is created.
- **render_image** (bool, optional) – If True and the image exists, then it gets rendered.
- **render_final_shape** (bool, optional) – If True, then the final fitting shape gets rendered.
- **render_initial_shape** (bool, optional) – If True and the initial fitting shape exists, then it gets rendered.
- **render_gt_shape** (bool, optional) – If True and the ground truth shape exists, then it gets rendered.
- **subplots_enabled** (bool, optional) – If True, then the requested final, initial and ground truth shapes get rendered on separate subplots.
- **channels** (int or list of int or all or None) – If int or list of int, the specified channel(s) will be rendered. If all, all the channels will be rendered in subplots. If None and the image is RGB, it will be rendered in RGB mode. If None and the image is not RGB, it is equivalent to all.
- **interpolation** (See Below, optional) – The interpolation used to render the image. For example, if bilinear, the image will be smooth and if nearest, the image will be pixelated. Example options

```
{none, nearest, bilinear, bicubic, spline16, spline36, hanning, hamming, hermite, kaiser, quadric, catrom, gaussian, bessel, mitchell, sinc, lanczos}
```

- **cmap_name** (str, optional) – If None, single channel and three channel images default to greyscale and RGB colormaps respectively.
- **alpha** (float, optional) – The alpha blending value, between 0 (transparent) and 1 (opaque).
• **masked** *(bool, optional) – If True, then the image is rendered as masked.*

• **final_marker_face_colour** *(See Below, optional) – The face (filling) colour of the markers of the final fitting shape. Example options*

  ```
  {r, g, b, c, m, k, w}
  or
  (3, ) ndarray
  ```

• **final_marker_edge_colour** *(See Below, optional) – The edge colour of the markers of the final fitting shape. Example options*

  ```
  {r, g, b, c, m, k, w}
  or
  (3, ) ndarray
  ```

• **final_line_colour** *(See Below, optional) – The line colour of the final fitting shape. Example options*

  ```
  {r, g, b, c, m, k, w}
  or
  (3, ) ndarray
  ```

• **initial_marker_face_colour** *(See Below, optional) – The face (filling) colour of the markers of the initial shape. Example options*

  ```
  {r, g, b, c, m, k, w}
  or
  (3, ) ndarray
  ```

• **initial_marker_edge_colour** *(See Below, optional) – The edge colour of the markers of the initial shape. Example options*

  ```
  {r, g, b, c, m, k, w}
  or
  (3, ) ndarray
  ```

• **initial_line_colour** *(See Below, optional) – The line colour of the initial shape. Example options*

  ```
  {r, g, b, c, m, k, w}
  or
  (3, ) ndarray
  ```

• **gt_marker_face_colour** *(See Below, optional) – The face (filling) colour of the markers of the ground truth shape. Example options*

  ```
  {r, g, b, c, m, k, w}
  or
  (3, ) ndarray
  ```

• **gt_marker_edge_colour** *(See Below, optional) – The edge colour of the markers of the ground truth shape. Example options*

  ```
  {r, g, b, c, m, k, w}
  or
  (3, ) ndarray
  ```
• **gt_line_colour**(See Below, optional) – The line colour of the ground truth shape. Example options

```python
[r, g, b, c, m, k, w]  
or  
(3, ) ndarray
```

• **render_lines**(bool or list of bool, optional) – If True, the lines will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (**final**, **initial**, **groundtruth**) order.

• **line_style**(str or list of str, optional) – The style of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (**final**, **initial**, **groundtruth**) order. Example options:

```python
['-', '--', '-.', ':']
```

• **line_width**(float or list of float, optional) – The width of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (**final**, **initial**, **groundtruth**) order.

• **render_markers**(bool or list of bool, optional) – If True, the markers will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (**final**, **initial**, **groundtruth**) order.

• **marker_style**(str or list of str, optional) – The style of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (**final**, **initial**, **groundtruth**) order. Example options:

```python
(.,,, o, v, ^, <, >, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8)
```

• **marker_size**(int or list of int, optional) – The size of the markers in points. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (**final**, **initial**, **groundtruth**) order.

• **marker_edge_width**(float or list of float, optional) – The width of the markers’ edge. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (**final**, **initial**, **groundtruth**) order.

• **render_numbering**(bool, optional) – If True, the landmarks will be numbered.

• **numbers_horizontal_align**(center, right, left, optional) – The horizontal alignment of the numbers’ texts.

• **numbers_vertical_align**(center, top, bottom, baseline, optional) – The vertical alignment of the numbers’ texts.

• **numbers_font_name**(See Below, optional) – The font of the numbers. Example options

```python
[serif, sans-serif, cursive, fantasy, monospace]
```

• **numbers_font_size**(int, optional) – The font size of the numbers.

• **numbers_font_style**(normal, italic, oblique, optional) – The font style of the numbers.

• **numbers_font_weight**(See Below, optional) – The font weight of the numbers. Example options
• **numbers_font_colour** *(See Below, optional)* – The font colour of the numbers. Example options

```
{r, g, b, c, m, k, w}
```

or

```
(3, ) ndarray
```

• **render_legend** *(bool, optional)* – If True, the legend will be rendered.

• **legend_title** *(str, optional)* – The title of the legend.

• **legend_font_name** *(See below, optional)* – The font of the legend. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

• **legend_font_style** *(normal, italic, oblique), optional)* – The font style of the legend.

• **legend_font_size** *(int, optional)* – The font size of the legend.

• **legend_font_weight** *(See Below, optional)* – The font weight of the legend. Example options

```
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```

• **legend_marker_scale** *(float, optional)* – The relative size of the legend markers with respect to the original

• **legend_location** *(int, optional)* – The location of the legend. The predefined values are:

```
'best' 0
'upper right' 1
'upper left' 2
'lower left' 3
'lower right' 4
'right' 5
'center left' 6
'center right' 7
'lower center' 8
'upper center' 9
'center' 10
```

• **legend_bbox_to_anchor** *((float, float) tuple, optional)* – The bbox that the legend will be anchored.

• **legend_border_axes_pad** *(float, optional)* – The pad between the axes and legend border.

• **legend_n_columns** *(int, optional)* – The number of the legend’s columns.

• **legend_horizontal_spacing** *(float, optional)* – The spacing between the columns.
• `legend_vertical_spacing` (`float`, optional) – The vertical space between the legend entries.

• `legend_border` (`bool`, optional) – If True, a frame will be drawn around the legend.

• `legend_border_padding` (`float`, optional) – The fractional whitespace inside the legend border.

• `legend_shadow` (`bool`, optional) – If True, a shadow will be drawn behind legend.

• `legend_rounded_corners` (`bool`, optional) – If True, the frame’s corners will be rounded (fancybox).

• `render_axes` (`bool`, optional) – If True, the axes will be rendered.

• `axes_font_name` (*See Below, optional*) – The font of the axes. Example options

  ```
  {serif, sans-serif, cursive, fantasy, monospace}
  ```

• `axes_font_size` (`int`, optional) – The font size of the axes.

• `axes_font_style` (*{normal, italic, oblique}*), optional) – The font style of the axes.

• `axes_font_weight` (*See Below, optional*) – The font weight of the axes. Example options

  ```
  {ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
  ```

• `axes_x_limits` (`float` or (`float`, `float`) or None, optional) – The limits of the x axis. If `float`, then it sets padding on the right and left of the Image as a percentage of the Image’s width. If `tuple` or `list`, then it defines the axis limits. If `None`, then the limits are set automatically.

• `axes_y_limits` ((`float`, `float`) `tuple` or `None`, optional) – The limits of the y axis. If `float`, then it sets padding on the top and bottom of the Image as a percentage of the Image’s height. If `tuple` or `list`, then it defines the axis limits. If `None`, then the limits are set automatically.

• `axes_x_ticks` (`list` or `tuple` or None, optional) – The ticks of the x axis.

• `axes_y_ticks` (`list` or `tuple` or None, optional) – The ticks of the y axis.

• `figure_size` ((`float`, `float`) `tuple` or `None` optional) – The size of the figure in inches.

Returns `renderer` (*class*) – The renderer object.
view_iterations

Parameters

- **figure_id** *(object, optional)* – The id of the figure to be used.
- **new_figure** *(bool, optional)* – If True, a new figure is created.
- **iters** *(int or list of int or None, optional)* – The iterations to be visualized. If None, then all the iterations are rendered.

<table>
<thead>
<tr>
<th>No.</th>
<th>Visualised shape</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>self.initial_shape</td>
<td>Initial shape</td>
</tr>
<tr>
<td>1</td>
<td>self.shapes[1]</td>
<td>Iteration 1</td>
</tr>
<tr>
<td>i</td>
<td>self.shapes[i]</td>
<td>Iteration i</td>
</tr>
<tr>
<td>n_iters</td>
<td>self.final_shape</td>
<td>Final shape</td>
</tr>
</tbody>
</table>

- **render_image** *(bool, optional)* – If True and the image exists, then it gets rendered.
- **subplots_enabled** *(bool, optional)* – If True, then the requested final, initial and ground truth shapes get rendered on separate subplots.
- **channels** *(int or list of int or all or None)* – If int or list of int, the specified channel(s) will be rendered. If all, all the channels will be rendered in subplots. If None and the image is RGB, it will be rendered in RGB mode. If None and the image is not RGB, it is equivalent to all.
- **interpolation** *(str (See Below), optional)* – The interpolation used to render the image. For example, if bilinear, the image will be smooth and if nearest, the image will be pixelated. Example options

    {none, nearest, bilinear, bicubic, spline16, spline36, hanning, hamming, hermite, kaiser, quadric, catrom, gaussian, bessel, mitchell, sinc, lanczos}

- **cmap_name** *(str, optional,)* – If None, single channel and three channel images default to greyscale and rgb colormaps respectively.

Visualize the iterations of the fitting process.
• **alpha** (*float*, optional) – The alpha blending value, between 0 (transparent) and 1 (opaque).

• **masked** (*bool*, optional) – If True, then the image is rendered as masked.

• **render_lines** (*bool* or *list* of *bool*, optional) – If True, the lines will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.

• **line_style** (*str* or *list* of *str* (See below), optional) – The style of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options:

  ```
  (-, --, -, :)
  ```

• **line_width** (*float* or *list* of *float*, optional) – The width of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.

• **line_colour** (*colour* or *list* of *colour* (See Below), optional) – The colour of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

  ```
  {r, g, b, c, m, k, w}
  or
  (3,) ndarray
  ```

• **render_markers** (*bool* or *list* of *bool*, optional) – If True, the markers will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.

• **marker_style** (*str* or *list* of *str* (See below), optional) – The style of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

  ```
  {., ,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
  ```

• **marker_size** (*int* or *list* of *int*, optional) – The size of the markers in points. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.

• **marker_edge_colour** (*colour* or *list* of *colour* (See Below), optional) – The edge colour of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

  ```
  {r, g, b, c, m, k, w}
  or
  (3,) ndarray
  ```

• **marker_face_colour** (*colour* or *list* of *colour* (See Below), optional) – The face (filling) colour of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

  ```
  {r, g, b, c, m, k, w}
  or
  (3,) ndarray
  ```

• **marker_edge_width** (*float* or *list* of *float*, optional) – The width of the markers’ edge. You can either provide a single value that will be used for all shapes or a list with a different
value per iteration shape.

- **render_numbering** (*bool*, optional) – If True, the landmarks will be numbered.

- **numbers_horizontal_align** (*str* (See below), optional) – The horizontal alignment of the numbers’ texts. Example options

  
  {center, right, left}

- **numbers_vertical_align** (*str* (See below), optional) – The vertical alignment of the numbers’ texts. Example options

  
  {center, top, bottom, baseline}

- **numbers_font_name** (*str* (See below), optional) – The font of the numbers. Example options

  
  {serif, sans-serif, cursive, fantasy, monospace}

- **numbers_font_size** (*int*, optional) – The font size of the numbers.

- **numbers_font_style** (*{normal, italic, oblique}*), optional) – The font style of the numbers.

- **numbers_font_weight** (*str* (See below), optional) – The font weight of the numbers. Example options

  
  {ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}

- **numbers_font_colour** (*See Below, optional*) – The font colour of the numbers. Example options

  
  {r, g, b, c, m, k, w} or (3,) ndarray

- **render_legend** (*bool*, optional) – If True, the legend will be rendered.

- **legend_title** (*str*, optional) – The title of the legend.

- **legend_font_name** (*See below, optional*) – The font of the legend. Example options

  
  {serif, sans-serif, cursive, fantasy, monospace}

- **legend_font_style** (*str* (See below), optional) – The font style of the legend. Example options

  
  {normal, italic, oblique}

- **legend_font_size** (*int*, optional) – The font size of the legend.

- **legend_font_weight** (*str* (See below), optional) – The font weight of the legend. Example options

  
  {ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
• **legend_marker_scale** (*float*, optional) – The relative size of the legend markers with respect to the original

• **legend_location** (*int*, optional) – The location of the legend. The predefined values are:

```plaintext
<table>
<thead>
<tr>
<th>Location</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>'best'</td>
<td>0</td>
</tr>
<tr>
<td>'upper right'</td>
<td>1</td>
</tr>
<tr>
<td>'upper left'</td>
<td>2</td>
</tr>
<tr>
<td>'lower left'</td>
<td>3</td>
</tr>
<tr>
<td>'lower right'</td>
<td>4</td>
</tr>
<tr>
<td>'right'</td>
<td>5</td>
</tr>
<tr>
<td>'center left'</td>
<td>6</td>
</tr>
<tr>
<td>'center right'</td>
<td>7</td>
</tr>
<tr>
<td>'lower center'</td>
<td>8</td>
</tr>
<tr>
<td>'upper center'</td>
<td>9</td>
</tr>
<tr>
<td>'center'</td>
<td>10</td>
</tr>
</tbody>
</table>
```

• **legend_bbox_to_anchor** (*(*float*, float)* tuple*, optional) – The bbox that the legend will be anchored.

• **legend_border_axes_pad** (*float*, optional) – The pad between the axes and legend border.

• **legend_n_columns** (*int*, optional) – The number of the legend’s columns.

• **legend_horizontal_spacing** (*float*, optional) – The spacing between the columns.

• **legend_vertical_spacing** (*float*, optional) – The vertical space between the legend entries.

• **legend_border** (*bool*, optional) – If True, a frame will be drawn around the legend.

• **legend_border_padding** (*float*, optional) – The fractional whitespace inside the legend border.

• **legend_shadow** (*bool*, optional) – If True, a shadow will be drawn behind legend.

• **legend_rounded_corners** (*bool*, optional) – If True, the frame’s corners will be rounded (fancybox).

• **render_axes** (*bool*, optional) – If True, the axes will be rendered.

• **axes_font_name** (*str* (See below), optional) – The font of the axes. Example options

```plaintext
{serif, sans-serif, cursive, fantasy, monospace}
```

• **axes_font_size** (*int*, optional) – The font size of the axes.

• **axes_font_style** (*{normal, italic, oblique}*), optional) – The font style of the axes.

• **axes_font_weight** (*str* (See below), optional) – The font weight of the axes. Example options

```plaintext
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```

• **axes_x_limits** (*float* or (*float*, float) or None, optional) – The limits of the x axis. If *float*, then it sets padding on the right and left of the Image as a percentage of the Image’s
width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.

- **axes_y_limits** ((float, float) tuple or None, optional) – The limits of the y axis. If float, then it sets padding on the top and bottom of the Image as a percentage of the Image’s height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.

- **axes_x_ticks** (list or tuple or None, optional) – The ticks of the x axis.

- **axes_y_ticks** (list or tuple or None, optional) – The ticks of the y axis.

- **figure_size** ((float, float) tuple or None optional) – The size of the figure in inches.

  Returns renderer (class) – The renderer object.

  `view_widget` (figure_size=(7, 7))

  Visualizes the result object using an interactive widget.

  Parameters `figure_size` ((int, int), optional) – The initial size of the rendered figure.

  property costs
  Returns a list with the cost per iteration. It returns None if the costs are not computed.

  Type list of float or None

  property final_shape
  Returns the final shape of the fitting process.

  Type menpo.shape.PointCloud

  property gt_shape
  Returns the ground truth shape associated with the image. In case there is not an attached ground truth shape, then None is returned.

  Type menpo.shape.PointCloud or None

  property image
  Returns the image that the fitting was applied on, if it was provided. Otherwise, it returns None.

  Type menpo.shape.Image or subclass or None

  property initial_shape
  Returns the initial shape that was provided to the fitting method to initialise the fitting process. In case the initial shape does not exist, then None is returned.

  Type menpo.shape.PointCloud or None

  property is_iterative
  Flag whether the object is an iterative fitting result.

  Type bool

  property n_iters
  Returns the total number of iterations of the fitting process.

  Type int

  property n_iters_per_scale
  Returns the number of iterations per scale of the fitting process.

  Type list of int

  property n_scales
  Returns the number of scales used during the fitting process.
Type `int`

**property shapes**
Returns the list of shapes obtained at each iteration of the fitting process. The list includes the initial_shape (if it exists) and final_shape.

Type `list of menpo.shape.PointCloud`

MultiScaleParametricIterativeResult

class menpofit.result.MultiScaleParametricIterativeResult (results, scales, affine_transforms, scale_transforms, image=None, gt_shape=None)

Bases: MultiScaleNonParametricIterativeResult

Class for defining a multi-scale parametric iterative fitting result, i.e. the result of a multi-scale method that optimizes over a parametric shape model. It holds the shapes of all the iterations of the fitting procedure, as well as the scales. It can optionally store the image on which the fitting was applied, as well as its ground truth shape.

**Note:** When using a method with a parametric shape model, the first step is to reconstruct the initial shape using the shape model. The generated reconstructed shape is then used as initialisation for the iterative optimisation. This step is not counted in the number of iterations.

**Parameters**

- • **results** (list of ParametricIterativeResult) – The list of parametric iterative results per scale.
- • **scales** (list of float) – The scale values (normally small to high).
- • **affine_transforms** (list of menpo.transform.Affine) – The list of affine transforms per scale that transform the shapes into the original image space.
- • **scale_transforms** (list of menpo.shape.Scale) – The list of scaling transforms per scale.
- • **image** (menpo.image.Image or subclass or None, optional) – The image on which the fitting process was applied. Note that a copy of the image will be assigned as an attribute. If None, then no image is assigned.
- • **gt_shape** (menpo.shape.PointCloud or None, optional) – The ground truth shape associated with the image. If None, then no ground truth shape is assigned.

**displacements**()
A list containing the displacement between the shape of each iteration and the shape of the previous one.

Type `list of ndarray`

**displacements_stats** (stat_type='mean')
A list containing a statistical metric on the displacements between the shape of each iteration and the shape of the previous one.

**Parameters**

**stat_type** (\{'mean', 'median', 'min', 'max'\}, optional) – Specifies a statistic metric to be extracted from the displacements.
Returns `displacements_stat` ([list of float]) – The statistical metric on the points displacements for each iteration.

Raises `ValueError` – type must be ‘mean’, ‘median’, ‘min’ or ‘max’

`errors` (*compute_error=None*)

Returns a list containing the error at each fitting iteration, if the ground truth shape exists.

Parameters `compute_error` ([callable, optional]) – Callable that computes the error between the shape at each iteration and the ground truth shape.

Returns `errors` ([list of float]) – The error at each iteration of the fitting process.

Raises `ValueError` – Ground truth shape has not been set, so the final error cannot be computed

`final_error` (*compute_error=None*)

Returns the final error of the fitting process, if the ground truth shape exists. This is the error computed based on the `final_shape`.

Parameters `compute_error` ([callable, optional]) – Callable that computes the error between the fitted and ground truth shapes.

Returns `final_error` ([float]) – The final error at the end of the fitting process.

Raises `ValueError` – Ground truth shape has not been set, so the final error cannot be computed

`initial_error` (*compute_error=None*)

Returns the initial error of the fitting process, if the ground truth shape and initial shape exist. This is the error computed based on the `initial_shape`.

Parameters `compute_error` ([callable, optional]) – Callable that computes the error between the initial and ground truth shapes.

Returns `initial_error` ([float]) – The initial error at the beginning of the fitting process.

Raises

- `ValueError` – Initial shape has not been set, so the initial error cannot be computed
- `ValueError` – Ground truth shape has not been set, so the initial error cannot be computed

`plot_costs` (*figure_id=None, new_figure=False, render_lines=True, line_colour='b', line_style='-', line_width=2, render_markers=True, marker_style='o', marker_size=4, marker_face_colour='b', marker_edge_colour='k', marker_edge_width=1.0, render_axes=True, axes_font_name='sans-serif', axes_font_size=10, axes_font_style='normal', axes_font_weight='normal', axes_x_limits=0.0, axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None, figure_size=(10, 6), render_grid=True, grid_line_style='–', grid_line_width=0.5)

Plot of the cost function evolution at each fitting iteration.

Parameters

- `figure_id` ([object, optional]) – The id of the figure to be used.
- `new_figure` ([bool, optional]) – If True, a new figure is created.
- `render_lines` ([bool, optional]) – If True, the line will be rendered.
- `line_colour` ([colour or None, optional]) – The colour of the line. If None, the colour is sampled from the jet colormap. Example colour options are
MenpoFit Documentation, Release 0.6.1

{'r', 'g', 'b', 'c', 'm', 'k', 'w'}
or
(3, ) ndarray

- **line_style**({'-', '--', '-.', ':'}, optional) – The style of the lines.
- **line_width** (float, optional) – The width of the lines.
- **render_markers** (bool, optional) – If True, the markers will be rendered.
- **marker_style** (marker, optional) – The style of the markers. Example marker options

   
   

- **marker_size** (int, optional) – The size of the markers in points.
- **marker_face_colour** (colour or None, optional) – The face (filling) colour of the markers. If None, the colour is sampled from the jet colormap. Example colour options are

   
   

- **marker_edge_colour** (colour or None, optional) – The edge colour of the markers. If None, the colour is sampled from the jet colormap. Example colour options are

   
   

- **marker_edge_width** (float, optional) – The width of the markers’ edge.
- **render_axes** (bool, optional) – If True, the axes will be rendered.
- **axes_font_name** (See below, optional) – The font of the axes. Example options

   


- **axes_font_size** (int, optional) – The font size of the axes.
- **axes_font_style** [{'normal', 'italic', 'oblique'}, optional] – The font style of the axes.
- **axes_font_weight** (See below, optional) – The font weight of the axes. Example options

   


- **axes_x_limits** (float or (float, float) or None, optional) – The limits of the x axis. If float, then it sets padding on the right and left of the graph as a percentage of the curves’ width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- **axes_y_limits** (float or (float, float) or None, optional) – The limits of the y axis. If float, then it sets padding on the top and bottom of the graph as a percentage of the
curves’ height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.

- **axes_x_ticks** *(list or tuple or None, optional)* – The ticks of the x axis.
- **axes_y_ticks** *(list or tuple or None, optional)* – The ticks of the y axis.
- **figure_size** *((float, float) or None, optional)* – The size of the figure in inches.
- **render_grid** *(bool, optional)* – If True, the grid will be rendered.
- **grid_line_style** *((',', '--', '-.', ':')*, optional)* – The style of the grid lines.
- **grid_line_width** *(float, optional)* – The width of the grid lines.

Returns **renderer** *(menpo.visualize.GraphPlotter)* – The renderer object.

**plot_displacements** *(stat_type='mean', figure_id=None, new_figure=False, render_lines=True, line_colour='b', line_style='-', line_width=2., render_markers=True, marker_style='o', marker_size=4., marker_face_colour='b', marker_edge_colour='k', marker_edge_width=1.0, render_axes=True, axes_font_name='sans-serif', axes_font_size=10, axes_font_style='normal', axes_font_weight='normal', axes_x_limits=0.0, axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None, figure_size=(10, 6), render_grid=True, grid_line_style='--', grid_line_width=0.5)*

Plot of a statistical metric of the displacement between the shape of each iteration and the shape of the previous one.

Parameters

- **stat_type** *((mean, median, min, max), optional)* – Specifies a statistic metric to be extracted from the displacements (see also `displacements_stats()` method).
- **figure_id** *(object, optional)* – The id of the figure to be used.
- **new_figure** *(bool, optional)* – If True, a new figure is created.
- **render_lines** *(bool, optional)* – If True, the line will be rendered.
- **line_colour** *(colour or None (See below), optional)* – The colour of the line. If None, the colour is sampled from the jet colormap. Example colour options are

```text
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- **line_style** *(str (See below), optional)* – The style of the lines. Example options:

```text
[-, --, -, ., :]
```

- **line_width** *(float, optional)* – The width of the lines.
- **render_markers** *(bool, optional)* – If True, the markers will be rendered.
- **marker_style** *(str (See below), optional)* – The style of the markers. Example marker options

```text
{., ,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
```

- **marker_size** *(int, optional)* – The size of the markers in points.
• **marker_face_colour** *(colour or None, optional)* – The face (filling) colour of the markers. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• **marker_edge_colour** *(colour or None, optional)* – The edge colour of the markers. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• **marker_edge_width** *(float, optional)* – The width of the markers’ edge.

• **render_axes** *(bool, optional)* – If True, the axes will be rendered.

• **axes_font_name** *(str (See below), optional)* – The font of the axes. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

• **axes_font_size** *(int, optional)* – The font size of the axes.

• **axes_font_style** *(str (See below), optional)* – The font style of the axes. Example options

```
{normal, italic, oblique}
```

• **axes_font_weight** *(str (See below), optional)* – The font weight of the axes. Example options

```
{ultralight, light, normal, regular, book, medium, roman,
 semibold, demibold, demi, bold, heavy, extra bold, black}
```

• **axes_x_limits** *(float or (float, float) or None, optional)* – The limits of the x axis. If float, then it sets padding on the right and left of the graph as a percentage of the curves’ width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.

• **axes_y_limits** *(float or (float, float) or None, optional)* – The limits of the y axis. If float, then it sets padding on the top and bottom of the graph as a percentage of the curves’ height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.

• **axes_x_ticks** *(list or tuple or None, optional)* – The ticks of the x axis.

• **axes_y_ticks** *(list or tuple or None, optional)* – The ticks of the y axis.

• **figure_size** *((float, float) or None, optional)* – The size of the figure in inches.

• **render_grid** *(bool, optional)* – If True, the grid will be rendered.

• **grid_line_style** *((',', '--', '-', ':')*, optional)* – The style of the grid lines.

• **grid_line_width** *(float, optional)* – The width of the grid lines.

**Returns** *renderer* *(menpo.visualize.GraphPlotter)* – The renderer object.
```python
plot_errors(compute_error=None, figure_id=None, new_figure=False, render_lines=True,
    line_colour='b', line_style='-', line_width=2, render_markers=True,
    marker_style='o', marker_size=4, marker_face_colour='b', marker_edge_colour='k',
    marker_edge_width=1.0, render_axes=True, axes_font_name='sans-serif',
    axes_font_size=10, axes_font_style='normal', axes_font_weight='normal',
    axes_x_limits=0.0, axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None,
    figure_size=(10, 6), render_grid=True, grid_line_style='–', grid_line_width=0.5)
```

Plot of the error evolution at each fitting iteration.

**Parameters**

- **compute_error** *(callable, optional)* — Callable that computes the error between the shape at each iteration and the ground truth shape.
- **figure_id** *(object, optional)* — The id of the figure to be used.
- **new_figure** *(bool, optional)* — If True, a new figure is created.
- **render_lines** *(bool, optional)* — If True, the line will be rendered.
- **line_colour** *(colour or None (See below), optional)* — The colour of the line. If None, the colour is sampled from the jet colormap. Example colour options are
  ```python
  {r, g, b, c, m, k, w}
  or
  (3, ) ndarray
  ```
- **line_style** *(str (See below), optional)* — The style of the lines. Example options:
  ```python
  {-, --, -, ., :}
  ```
- **line_width** *(float, optional)* — The width of the lines.
- **render_markers** *(bool, optional)* — If True, the markers will be rendered.
- **marker_style** *(str (See below), optional)* — The style of the markers. Example marker options
  ```python
  {., , , o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
  ```
- **marker_size** *(int, optional)* — The size of the markers in points.
- **marker_face_colour** *(colour or None, optional)* — The face (filling) colour of the markers. If None, the colour is sampled from the jet colormap. Example colour options are
  ```python
  {r, g, b, c, m, k, w}
  or
  (3, ) ndarray
  ```
- **marker_edge_colour** *(colour or None, optional)* — The edge colour of the markers. If None, the colour is sampled from the jet colormap. Example colour options are
  ```python
  {r, g, b, c, m, k, w}
  or
  (3, ) ndarray
  ```
- **marker_edge_width** *(float, optional)* — The width of the markers’ edge.
- **render_axes** *(bool, optional)* — If True, the axes will be rendered.
- **axes_font_name** *(str (See below), optional)* — The font of the axes. Example options
  ```python
  ```
• **axes_font_size** (*int*, optional) – The font size of the axes.

• **axes_font_style** (*str* (See below), optional) – The font style of the axes. Example options

  
  *{normal, italic, oblique}*

• **axes_font_weight** (*str* (See below), optional) – The font weight of the axes. Example options

  
  *{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}*

• **axes_x_limits** (*float* or (*float*, *float*) or *None*, optional) – The limits of the x axis. If *float*, then it sets padding on the right and left of the graph as a percentage of the curves’ width. If *tuple* or *list*, then it defines the axis limits. If *None*, then the limits are set automatically.

• **axes_y_limits** (*float* or (*float*, *float*) or *None*, optional) – The limits of the y axis. If *float*, then it sets padding on the top and bottom of the graph as a percentage of the curves’ height. If *tuple* or *list*, then it defines the axis limits. If *None*, then the limits are set automatically.

• **axes_x_ticks** (*list* or *tuple* or *None*, optional) – The ticks of the x axis.

• **axes_y_ticks** (*list* or *tuple* or *None*, optional) – The ticks of the y axis.

• **figure_size** (*tuple* or *None*, optional) – The size of the figure in inches.

• **render_grid** (*bool*, optional) – If *True*, the grid will be rendered.

• **grid_line_style** (*{'-', '--', '-.', ':'}*, optional) – The style of the grid lines.

• **grid_line_width** (*float*, optional) – The width of the grid lines.

Returns **renderer** (*menpo.visualize.GraphPlotter*) – The renderer object.

**reconstructed_initial_error**(compute_error=None)

Returns the error of the reconstructed initial shape of the fitting process, if the ground truth shape exists. This is the error computed based on the reconstructed initial shapes[0].

Parameters **compute_error** (*callable*, optional) – Callable that computes the error between the reconstructed initial and ground truth shapes.

Returns **reconstructed_initial_error** (*float*) – The error that corresponds to the initial shape’s reconstruction.

Raises **ValueError** – Ground truth shape has not been set, so the reconstructed initial error cannot be computed

**to_result**(pass_image=True, pass_initial_shape=True, pass_gt_shape=True)

Returns a **Result** instance of the object, i.e. a fitting result object that does not store the iterations. This can be useful for reducing the size of saved fitting results.

Parameters

• **pass_image** (*bool*, optional) – If *True*, then the image will get passed (if it exists).

• **pass_initial_shape** (*bool*, optional) – If *True*, then the initial shape will get passed (if it exists).
• **pass_gt_shape** *(bool, optional)* – If True, then the ground truth shape will get passed (if it exists).

**Returns** **result** *(Result)* – The final “lightweight” fitting result.

**view**(figure_id=None, new_figure=False, render_image=True, render_final_shape=True, render_initial_shape=False, render_gt_shape=False, subplots_enabled=True, channels=None, interpolation='bilinear', cmap_name=None, alpha=1.0, masked=True, final_marker_face_colour='r', final_marker_edge_colour='k', final_line_colour='b', initial_marker_face_colour='b', initial_marker_edge_colour='k', initial_line_colour='b', gt_marker_face_colour='y', gt_marker_edge_colour='k', gt_line_colour='y', render_lines=True, line_style='-', line_width=2, render_markers=True, marker_style='o', marker_size=4, marker_edge_width=1.0, render_numbering=False, numbers_horizontal_align='center', numbers_vertical_align='bottom', numbers_font_name='sans-serif', numbers_font_style='normal', numbers_font_size=10, numbers_font_weight='normal', numbers_font_colour='k', render_legend=True, legend_title='', legend_font_name='sans-serif', legend_font_style='normal', legend_font_size=10, legend_font_weight='normal', legend_marker_scale=None, legend_location=2, legend_bbox_to_anchor=(1.05, 1.0), legend_border_axes_pad=None, legend_n_columns=1, legend_horizontal_spacing=None, legend_vertical_spacing=None, legend_border=True, legend_border_padding=None, legend_shadow=False, legend_rounded_corners=False, render_axes=False, axes_font_name='sans-serif', axes_font_size=10, axes_font_style='normal', axes_font_weight='normal', axes_x_limits=None, axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None, figure_size=(7, 7))

Visualize the fitting result. The method renders the final fitted shape and optionally the initial shape, ground truth shape and the image, if they were provided.

**Parameters**

• **figure_id**(object, optional) – The id of the figure to be used.

• **new_figure**(bool, optional) – If True, a new figure is created.

• **render_image**(bool, optional) – If True and the image exists, then it gets rendered.

• **render_final_shape**(bool, optional) – If True, then the final fitting shape gets rendered.

• **render_initial_shape**(bool, optional) – If True and the initial fitting shape exists, then it gets rendered.

• **render_gt_shape**(bool, optional) – If True and the ground truth shape exists, then it gets rendered.

• **subplots_enabled**(bool, optional) – If True, then the requested final, initial and ground truth shapes get rendered on separate subplots.

• **channels**(int or list of int or all or None) – If int or list of int, the specified channel(s) will be rendered. If all, all the channels will be rendered in subplots. If None and the image is RGB, it will be rendered in RGB mode. If None and the image is not RGB, it is equivalent to all.

• **interpolation**(See Below, optional) – The interpolation used to render the image. For example, if 'bilinear', the image will be smooth and if 'nearest', the image will be pixelated. Example options

```
{none, nearest, bilinear, bicubic, spline16, spline36, hanning, hamming, hermite, kaiser, quadric, catrom, gaussian, bessel,Mitchell, sinc, lanczos}
```
• **cmap_name** *(str, optional)* – If None, single channel and three channel images default to greyscale and rgb colormaps respectively.

• **alpha** *(float, optional)* – The alpha blending value, between 0 (transparent) and 1 (opaque).

• **masked** *(bool, optional)* – If True, then the image is rendered as masked.

• **final_marker_face_colour** *(See Below, optional)* – The face (filling) colour of the markers of the final fitting shape. Example options

```
[r, g, b, c, m, k, w]
or
(3,) ndarray
```

• **final_marker_edge_colour** *(See Below, optional)* – The edge colour of the markers of the final fitting shape. Example options

```
[r, g, b, c, m, k, w]
or
(3,) ndarray
```

• **final_line_colour** *(See Below, optional)* – The line colour of the final fitting shape. Example options

```
[r, g, b, c, m, k, w]
or
(3,) ndarray
```

• **initial_marker_face_colour** *(See Below, optional)* – The face (filling) colour of the markers of the initial shape. Example options

```
[r, g, b, c, m, k, w]
or
(3,) ndarray
```

• **initial_marker_edge_colour** *(See Below, optional)* – The edge colour of the markers of the initial shape. Example options

```
[r, g, b, c, m, k, w]
or
(3,) ndarray
```

• **initial_line_colour** *(See Below, optional)* – The line colour of the initial shape. Example options

```
[r, g, b, c, m, k, w]
or
(3,) ndarray
```

• **gt_marker_face_colour** *(See Below, optional)* – The face (filling) colour of the markers of the ground truth shape. Example options

```
[r, g, b, c, m, k, w]
or
(3,) ndarray
```

• **gt_marker_edge_colour** *(See Below, optional)* – The edge colour of the markers of the ground truth shape. Example options
• `gt_line_colour` *(See Below, optional)* – The line colour of the ground truth shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• `render_lines` *(bool or list of bool, optional)* – If True, the lines will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per shape in `(final, initial, groundtruth)` order.

• `line_style` *(str or list of str, optional)* – The style of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per shape in `(final, initial, groundtruth)` order. Example options:

```
{'-', '--', '-.', ':'}
```

• `line_width` *(float or list of float, optional)* – The width of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per shape in `(final, initial, groundtruth)` order.

• `render_markers` *(bool or list of bool, optional)* – If True, the markers will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per shape in `(final, initial, groundtruth)` order.

• `marker_style` *(str or list of str, optional)* – The style of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per shape in `(final, initial, groundtruth)` order. Example options:

```
{., ,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
```

• `marker_size` *(int or list of int, optional)* – The size of the markers in points. You can either provide a single value that will be used for all shapes or a list with a different value per shape in `(final, initial, groundtruth)` order.

• `marker_edge_width` *(float or list of float, optional)* – The width of the markers’ edge. You can either provide a single value that will be used for all shapes or a list with a different value per shape in `(final, initial, groundtruth)` order.

• `render_numbering` *(bool, optional)* – If True, the landmarks will be numbered.

• `numbers_horizontal_align` *(center, right, left), optional)* – The horizontal alignment of the numbers’ texts.

• `numbers_vertical_align` *(center, top, bottom, baseline), optional)* – The vertical alignment of the numbers’ texts.

• `numbers_font_name` *(See Below, optional)* – The font of the numbers. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

• `numbers_font_size` *(int, optional)* – The font size of the numbers.

• `numbers_font_style` *(normal, italic, oblique), optional)* – The font style of the numbers.
- **numbers_font_weight** *(See Below, optional)* – The font weight of the numbers. Example options

   {ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}

- **numbers_font_colour** *(See Below, optional)* – The font colour of the numbers. Example options

   \{r, g, b, c, m, k, w\}
   or
   \(3, \) ndarray

- **render_legend** *(bool, optional)* – If True, the legend will be rendered.

- **legend_title** *(str, optional)* – The title of the legend.

- **legend_font_name** *(See below, optional)* – The font of the legend. Example options

   {serif, sans-serif, cursive, fantasy, monospace}

- **legend_font_style** *(\{normal, italic, oblique\}, optional)* – The font style of the legend.

- **legend_font_size** *(int, optional)* – The font size of the legend.

- **legend_font_weight** *(See Below, optional)* – The font weight of the legend. Example options

   {ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}

- **legend_marker_scale** *(float, optional)* – The relative size of the legend markers with respect to the original

- **legend_location** *(int, optional)* – The location of the legend. The predefined values are:

   
<table>
<thead>
<tr>
<th>Location</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>'best'</td>
<td>0</td>
</tr>
<tr>
<td>'upper right'</td>
<td>1</td>
</tr>
<tr>
<td>'upper left'</td>
<td>2</td>
</tr>
<tr>
<td>'lower left'</td>
<td>3</td>
</tr>
<tr>
<td>'lower right'</td>
<td>4</td>
</tr>
<tr>
<td>'right'</td>
<td>5</td>
</tr>
<tr>
<td>'center left'</td>
<td>6</td>
</tr>
<tr>
<td>'center right'</td>
<td>7</td>
</tr>
<tr>
<td>'lower center'</td>
<td>8</td>
</tr>
<tr>
<td>'upper center'</td>
<td>9</td>
</tr>
<tr>
<td>'center'</td>
<td>10</td>
</tr>
</tbody>
</table>

- **legend_bbox_to_anchor** *(\{float, float\} tuple, optional)* – The bbox that the legend will be anchored.

- **legend_border_axes_pad** *(float, optional)* – The pad between the axes and legend border.

- **legend_n_columns** *(int, optional)* – The number of the legend’s columns.
• **legend_horizontal_spacing** *(float, optional)* – The spacing between the columns.

• **legend_vertical_spacing** *(float, optional)* – The vertical space between the legend entries.

• **legend_border** *(bool, optional)* – If True, a frame will be drawn around the legend.

• **legend_border_padding** *(float, optional)* – The fractional whitespace inside the legend border.

• **legend_shadow** *(bool, optional)* – If True, a shadow will be drawn behind legend.

• **legend_rounded_corners** *(bool, optional)* – If True, the frame’s corners will be rounded (fancybox).

• **render_axes** *(bool, optional)* – If True, the axes will be rendered.

• **axes_font_name** *(See Below, optional)* – The font of the axes. Example options

  ```
  {serif, sans-serif, cursive, fantasy, monospace}
  ```

• **axes_font_size** *(int, optional)* – The font size of the axes.

• **axes_font_style** *(normal, italic, oblique), optional)* – The font style of the axes.

• **axes_font_weight** *(See Below, optional)* – The font weight of the axes. Example options

  ```
  {ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
  ```

• **axes_x_limits** *(float or (float, float) or None, optional)* – The limits of the x axis. If float, then it sets padding on the right and left of the Image as a percentage of the Image’s width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.

• **axes_y_limits** *(float, float) tuple or None, optional)* – The limits of the y axis. If float, then it sets padding on the top and bottom of the Image as a percentage of the Image’s height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.

• **axes_x_ticks** *(list or tuple or None, optional)* – The ticks of the x axis.

• **axes_y_ticks** *(list or tuple or None, optional)* – The ticks of the y axis.

• **figure_size** *(float, float) tuple or None optional)* – The size of the figure in inches.

  Returns **renderer** *(class)* – The renderer object.
view_iterations (figure_id=None, new_figure=False, iters=None, render_image=True, subplots_enabled=False, channels=None, interpolation='bilinear', cmap_name=None, alpha=1.0, masked=True, line_style='-', line_width=2, line_colour=None, render_markers=True, marker_edge_colour=None, marker_face_colour=None, marker_style='o', marker_size=4, marker_edge_width=1.0, render_numbering=False, numbers_horizontal_align='center', numbers_vertical_align='bottom', numbers_font_name='sans-serif', numbers_font_size=10, numbers_font_style='normal', numbers_font_weight='normal', numbers_font_colour='k', render_legend=True, legend_title='', legend_font_name='sans-serif', legend_font_style='normal', legend_font_size=10, legend_font_weight='normal', legend_marker_scale=None, legend_location=2, legend_bbox_to_anchor=(1.05, 1.0), legend_border_axes_pad=None, legend_n_columns=1, legend_horizontal_spacing=None, legend_vertical_spacing=None, legend_border=True, legend_border_padding=None, legend_shadow=False, legend_rounded_corners=False, render_axes=False, axes_font_name='sans-serif', axes_font_size=10, axes_font_style='normal', axes_font_weight='normal', axes_x_limits=None, axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None, figure_size=(7, 7))

Visualize the iterations of the fitting process.

Parameters

- **figure_id** (object, optional) – The id of the figure to be used.
- **new_figure** (bool, optional) – If True, a new figure is created.
- **iters** (int or list of int or None, optional) – The iterations to be visualized. If None, then all the iterations are rendered.

<table>
<thead>
<tr>
<th>No.</th>
<th>Visualised shape</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>self.initial_shape</td>
<td>Initial shape</td>
</tr>
<tr>
<td>1</td>
<td>self.reconstructed_initial_shape</td>
<td>Reconstructed initial</td>
</tr>
<tr>
<td>2</td>
<td>self.shapes[2]</td>
<td>Iteration 1</td>
</tr>
<tr>
<td>i</td>
<td>self.shapes[i]</td>
<td>Iteration i-1</td>
</tr>
<tr>
<td>n_iters+1</td>
<td>self.final_shape</td>
<td>Final shape</td>
</tr>
</tbody>
</table>

- **render_image** (bool, optional) – If True and the image exists, then it gets rendered.
- **subplots_enabled** (bool, optional) – If True, then the requested final, initial and ground truth shapes get rendered on separate subplots.
- **channels** (int or list of int or all or None) – If int or list of int, the specified channel(s) will be rendered. If all, all the channels will be rendered in subplots. If None and the image is RGB, it will be rendered in RGB mode. If None and the image is not RGB, it is equivalent to all.
- **interpolation** (str (See Below), optional) – The interpolation used to render the image. For example, if 'bilinear', the image will be smooth and if 'nearest', the image will be pixelated. Example options:

  {none, nearest, bilinear, bicubic, spline16, spline36, hanning, hamming, hermite, kaiser, quadric, catrom, gaussian, bessel, mitchell, sinc, lanczos}

- **cmap_name** (str, optional.) – If None, single channel and three channel images default to greyscale and rgb colormaps respectively.

2.2. Internal API
• **alpha** (*float*, optional) – The alpha blending value, between 0 (transparent) and 1 (opaque).

• **masked** (*bool*, optional) – If *True*, then the image is rendered as masked.

• **render_lines** (*bool* or list of *bool*, optional) – If *True*, the lines will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.

• **line_style** (*str* or list of *str* (See below), optional) – The style of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options:

```
(-, --, -., :)  
```

• **line_width** (*float* or list of *float*, optional) – The width of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.

• **line_colour** (*colour* or list of *colour* (See Below), optional) – The colour of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```
{r, g, b, c, m, k, w}  
or  
(3, ) ndarray  
```

• **render_markers** (*bool* or list of *bool*, optional) – If *True*, the markers will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.

• **marker_style** (*str* or list of *str* (See below), optional) – The style of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```
{., ,, o, v, ^, <, >, _, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}  
```

• **marker_size** (*int* or list of *int*, optional) – The size of the markers in points. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.

• **marker_edge_colour** (*colour* or list of *colour* (See Below), optional) – The edge colour of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```
{r, g, b, c, m, k, w}  
or  
(3, ) ndarray  
```

• **marker_face_colour** (*colour* or list of *colour* (See Below), optional) – The face (filling) colour of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```
{r, g, b, c, m, k, w}  
or  
(3, ) ndarray  
```

• **marker_edge_width** (*float* or list of *float*, optional) – The width of the markers’ edge. You can either provide a single value that will be used for all shapes or a list with a different
value per iteration shape.

- **render_numbering** *(bool, optional) –* If True, the landmarks will be numbered.

- **numbers_horizontal_align** *(str (See below), optional) –* The horizontal alignment of the numbers’ texts. Example options

  ```
  {center, right, left}
  ```

- **numbers_vertical_align** *(str (See below), optional) –* The vertical alignment of the numbers’ texts. Example options

  ```
  {center, top, bottom, baseline}
  ```

- **numbers_font_name** *(str (See below), optional) –* The font of the numbers. Example options

  ```
  {serif, sans-serif, cursive, fantasy, monospace}
  ```

- **numbers_font_size** *(int, optional) –* The font size of the numbers.

- **numbers_font_style** *(str (normal, italic, oblique), optional) –* The font style of the numbers.

- **numbers_font_weight** *(str (See below), optional) –* The font weight of the numbers. Example options

  ```
  {ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
  ```

- **numbers_font_colour** *(See Below, optional) –* The font colour of the numbers. Example options

  ```
  {r, g, b, c, m, k, w} or (3,) ndarray
  ```

- **render_legend** *(bool, optional) –* If True, the legend will be rendered.

- **legend_title** *(str, optional) –* The title of the legend.

- **legend_font_name** *(See below, optional) –* The font of the legend. Example options

  ```
  {serif, sans-serif, cursive, fantasy, monospace}
  ```

- **legend_font_style** *(str (See below), optional) –* The font style of the legend. Example options

  ```
  {normal, italic, oblique}
  ```

- **legend_font_size** *(int, optional) –* The font size of the legend.

- **legend_font_weight** *(str (See below), optional) –* The font weight of the legend. Example options

  ```
  {ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
  ```
• **legend_marker_scale** (*float*, optional) – The relative size of the legend markers with respect to the original.

• **legend_location** (*int*, optional) – The location of the legend. The predefined values are:

<table>
<thead>
<tr>
<th>Location</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>'best'</td>
<td>0</td>
</tr>
<tr>
<td>'upper right'</td>
<td>1</td>
</tr>
<tr>
<td>'upper left'</td>
<td>2</td>
</tr>
<tr>
<td>'lower left'</td>
<td>3</td>
</tr>
<tr>
<td>'lower right'</td>
<td>4</td>
</tr>
<tr>
<td>'right'</td>
<td>5</td>
</tr>
<tr>
<td>'center left'</td>
<td>6</td>
</tr>
<tr>
<td>'center right'</td>
<td>7</td>
</tr>
<tr>
<td>'lower center'</td>
<td>8</td>
</tr>
<tr>
<td>'upper center'</td>
<td>9</td>
</tr>
<tr>
<td>'center'</td>
<td>10</td>
</tr>
</tbody>
</table>

• **legend_bbox_to_anchor** (*((float, float) tuple*, optional) – The bbox that the legend will be anchored.

• **legend_border_axes_pad** (*float*, optional) – The pad between the axes and legend border.

• **legend_n_columns** (*int*, optional) – The number of the legend’s columns.

• **legend_horizontal_spacing** (*float*, optional) – The spacing between the columns.

• **legend_vertical_spacing** (*float*, optional) – The vertical space between the legend entries.

• **legend_border** (*bool*, optional) – If `True`, a frame will be drawn around the legend.

• **legend_border_padding** (*float*, optional) – The fractional whitespace inside the legend border.

• **legend_shadow** (*bool*, optional) – If `True`, a shadow will be drawn behind legend.

• **legend_rounded_corners** (*bool*, optional) – If `True`, the frame’s corners will be rounded (fancybox).

• **render_axes** (*bool*, optional) – If `True`, the axes will be rendered.

• **axes_font_name** (*str* (See below), optional) – The font of the axes. Example options

```plaintext
{serif, sans-serif, cursive, fantasy, monospace}
```

• **axes_font_size** (*int*, optional) – The font size of the axes.

• **axes_font_style** (*{normal, italic, oblique}* , optional) – The font style of the axes.

• **axes_font_weight** (*str* (See below), optional) – The font weight of the axes. Example options

```plaintext
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```

• **axes_x_limits** (*float or (float, float) or None*, optional) – The limits of the x axis. If `float`, then it sets padding on the right and left of the Image as a percentage of the Image’s
width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.

- **axes_y_limits** ((float, float) tuple or None, optional) – The limits of the y axis. If float, then it sets padding on the top and bottom of the Image as a percentage of the Image’s height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.

- **axes_x_ticks** (list or tuple or None, optional) – The ticks of the x axis.

- **axes_y_ticks** (list or tuple or None, optional) – The ticks of the y axis.

- **figure_size** ((float, float) tuple or None optional) – The size of the figure in inches.

Returns *renderer* (class) – The renderer object.

**view_widget** *(figure_size=(7, 7))*

Visualizes the result object using an interactive widget.

Parameters **figure_size** ((int, int), optional) – The initial size of the rendered figure.

**property costs**

Returns a list with the cost per iteration. It returns None if the costs are not computed.

*Type* list of float or None

**property final_shape**

Returns the final shape of the fitting process.

*Type* menpo.shape.PointCloud

**property gt_shape**

Returns the ground truth shape associated with the image. In case there is not an attached ground truth shape, then None is returned.

*Type* menpo.shape.PointCloud or None

**property image**

Returns the image that the fitting was applied on, if it was provided. Otherwise, it returns None.

*Type* menpo.shape.Image or subclass or None

**property initial_shape**

Returns the initial shape that was provided to the fitting method to initialise the fitting process. In case the initial shape does not exist, then None is returned.

*Type* menpo.shape.PointCloud or None

**property is_iterative**

Flag whether the object is an iterative fitting result.

*Type* bool

**property n_iters**

Returns the total number of iterations of the fitting process.

*Type* int

**property n_iters_per_scale**

Returns the number of iterations per scale of the fitting process.

*Type* list of int

**property n_scales**

Returns the number of scales used during the fitting process.
property reconstructed_initial_shapes
Returns the result of the reconstruction step that takes place at each scale before applying the iterative optimisation.

Type list of menpo.shape.PointCloud

property shape_parameters
Returns the list of shape parameters obtained at each iteration of the fitting process. The list includes the parameters of the initial_shape (if it exists) and final_shape.

Type list of (n_params,) ndarray

property shapes
Returns the list of shapes obtained at each iteration of the fitting process. The list includes the initial_shape (if it exists) and final_shape.

Type list of menpo.shape.PointCloud

2.2.10 menpofit.transform

Model Driven Transforms

OrthoMDTransform
class menpofit.transform.OrthoMDTransform(model, transform_cls, source=None)
Bases: GlobalMDTransform

A transform that couples an alignment transform to a statistical model together with a global similarity transform, such that the weights of the transform are fully specified by both the weights of statistical model and the weights of the similarity transform. The model is assumed to generate an instance which is then transformed by the similarity transform; the result defines the target landmarks of the transform. If no source is provided, the mean of the model is defined as the source landmarks of the transform.

This transform (in contrast to the GlobalMDTransform) additionally orthonormalises both the global and the model basis against each other, ensuring that orthogonality and normalization is enforced across the unified bases.

Parameters

- **model** (OrthoPDM or subclass) – A linear statistical shape model (Point Distribution Model) that also has a global similarity transform that is orthonormalised with the shape bases.

- **transform_cls** (subclass of menpo.transform.Alignment) – A class of menpo.transform.Alignment. The align constructor will be called on this with the source and target landmarks. The target is set to the points generated from the model using the provided weights - the source is either given or set to the model’s mean.

- **source** (menpo.shape.PointCloud or None, optional) – The source landmarks of the transform. If None, the mean of the model is used.

\( J_p() \)

Compute the parameters’ Jacobian, as shown in [1].

Returns \( J_p((n\_params, n\_params) \text{ ndarray}) \) – The parameters’ Jacobian.
apply \((x, \text{batch\_size}=\text{None}, **\text{kwargs})\)
Applies this transform to \(x\).

If \(x\) is \text{Transformable}, \(x\) will be handed this transform object to transform itself non-destructively (a transformed copy of the object will be returned).

If not, \(x\) is assumed to be an \text{ndarray}. The transformation will be non-destructive, returning the transformed version.

Any \text{kwargs} will be passed to the specific transform \_apply() method.

Parameters

- \(x\) (\text{Transformable or (n\_points, n\_dims) ndarray})\) – The array or object to be transformed.
- \text{batch\_size} (\text{int}, \text{optional}) – If not \text{None}, this determines how many items from the numpy array will be passed through the transform at a time. This is useful for operations that require large intermediate matrices to be computed.
- \text{kwargs} (\text{dict}) – Passed through to \_apply().

Returns transformed (\text{type } (x)) – The transformed object or array

apply\_inplace (*\text{args}, **\text{kwargs})
Deprecated as public supported API, use the non-mutating \text{apply()} instead.

For internal performance-specific uses, see \_apply\_inplace().

as\_vector (**\text{kwargs})
Returns a flattened representation of the object as a single vector.

Returns vector ((N,) \text{ndarray}) – The core representation of the object, flattened into a single vector. Note that this is always a view back on to the original object, but is not writable.

compose\_after (\text{transform})
Returns a \text{TransformChain} that represents this transform composed after the given transform:

```python
a = a.compose_after(b)
c.apply(p) == a.apply(b.apply(p))
```

\(a\) and \(b\) are left unchanged.

This corresponds to the usual mathematical formalism for the compose operator, \(\circ\).

Parameters \text{transform} (\text{Transform}) – Transform to be applied before self

Returns \text{transform} (\text{TransformChain}) – The resulting transform chain.

compose\_after\_from\_vector\_inplace (\text{delta})
Composes two transforms together based on the first order approximation proposed in [1].

Parameters \text{delta} ((N, ) \text{ndarray}) – Vectorized \text{ModelDrivenTransform} to be applied before self.

Returns \text{transform} (\text{self}) – self, updated to the result of the composition

References
compose_before(transform)

Returns a TransformChain that represents this transform composed before the given transform:

```python
c = a.compose_before(b)
c.apply(p) == b.apply(a.apply(p))
```

a and b are left unchanged.

**Parameters**

- **transform (Transform)** – Transform to be applied after self

**Returns**

- **transform (TransformChain)** – The resulting transform chain.

copy()

Generate an efficient copy of this object.

Note that Numpy arrays and other Copyable objects on self will be deeply copied. Dictionaries and sets will be shallow copied, and everything else will be assigned (no copy will be made).

Classes that store state other than numpy arrays and immutable types should overwrite this method to ensure all state is copied.

**Returns**

- **type(self)** – A copy of this object

d_dp(points)

The derivative of this ModelDrivenTransform with respect to the parametrisation changes evaluated at points.

This is done by chaining the derivative of points wrt the source landmarks on the transform (dW/dL) together with the Jacobian of the linear model wrt its weights (dX/dp).

**Parameters**

- **points ((n_points, n_dims) ndarray)** – The spatial points at which the derivative should be evaluated.

**Returns**

- **d_dp ((n_points, n_parameters, n_dims) ndarray)** – The Jacobian with respect to the parametrisation.

from_vector(vector)

Build a new instance of the object from it’s vectorized state.

`self` is used to fill out the missing state required to rebuild a full object from it’s standardized flattened state. This is the default implementation, which is which is a deepcopy of the object followed by a call to from_vector_inplace(). This method can be overridden for a performance benefit if desired.

**Parameters**

- **vector ((n_parameters,) ndarray)** – Flattened representation of the object.

**Returns**

- **object (type(self))** – An new instance of this class.

from_vector_inplace(vector)

Deprecated. Use the non-mutating API, from_vector.

For internal usage in performance-sensitive spots, see from_vector_inplace()

**Parameters**

- **vector ((n_parameters,) ndarray)** – Flattened representation of this object

has_nan_values()

Tests if the vectorized form of the object contains nan values or not. This is particularly useful for objects with unknown values that have been mapped to nan values.

**Returns**

- **has_nan_values (bool)** – If the vectorized object contains nan values.

pseudoinverse()

The pseudoinverse of the transform - that is, the transform that results from swapping source and target, or more formally, negating the transforms parameters. If the transform has a true inverse this is returned instead.
MenpoFit Documentation, Release 0.6.1

Type `type(self)`

**pseudoinverse_vector** *(vector)*
The vectorized pseudoinverse of a provided vector instance. Syntactic sugar for `self.from_vector(vector).pseudoinverse.as_vector()`. On `ModelDrivenTransform` this is especially fast - we just negate the vector provided.

Parameters `vector` ((P,) ndarray) – A vectorized version of self

Returns `pseudoinverse_vector` ((N,) ndarray) – The pseudoinverse of the vector provided

**set_target** *(new_target)*
Update this object so that it attempts to recreate the `new_target`.

Parameters `new_target` (PointCloud) – The new target that this object should try and re-generate.

**has_true_inverse**
Whether the transform has true inverse.

Type `bool`

**n_dims**
The number of dimensions that the transform supports.

Type `int`

**n_dims_output**
The output of the data from the transform.

None if the output of the transform is not dimension specific.

Type `int` or `None`

**n_parameters**
The total number of parameters.

Type `int`

**n_points**
The number of points on the `target`.

Type `int`

**target**
The current `menpo.shape.PointCloud` that this object produces.

Type `menpo.shape.PointCloud`

---

**LinearOrthoMDTransform**

```python
class menpofit.transform.LinearOrthoMDTransform(model, sparse_instance)
Bases: OrthoPDM, Transform
```

A transform that couples an alignment transform to a statistical model together with a global similarity transform, such that the weights of the transform are fully specified by both the weights of statistical model and the weights of the similarity transform. The model is assumed to generate an instance which is then transformed by the similarity transform; the result defines the target landmarks of the transform. If no source is provided, the mean of the model is defined as the source landmarks of the transform.

This transform (in contrast to the `GlobalMDTransform`) additionally orthonormalises both the global and the model basis against each other, ensuring that orthogonality and normalization is enforced across the unified bases.

---
This transform (in contrast to the OrthoMDTransform) should be used with linear statistical models of dense shapes.

**Parameters**

- **model** ([menpo.model.LinearModel]) – A linear statistical shape model.
- **sparse_instance** ([menpo.shape.PointCloud]) – The source landmarks of the transform.

**apply** *(x, batch_size=None, **kwargs)*

Applies this transform to x.

If x is Transformable, x will be handed this transform object to transform itself non-destructively (a transformed copy of the object will be returned).

If not, x is assumed to be an ndarray. The transformation will be non-destructive, returning the transformed version.

Any kwargs will be passed to the specific transform _apply() method.

**Parameters**

- **x** (Transformable or (n_points, n_dims) ndarray) – The array or object to be transformed.
- **batch_size** (int, optional) – If not None, this determines how many items from the numpy array will be passed through the transform at a time. This is useful for operations that require large intermediate matrices to be computed.
- **kwargs** (dict) – Passed through to _apply().

**Returns** transformed (type(x)) – The transformed object or array

**apply_inplace** *(args, **kwargs)*

Deprecated as public supported API, use the non-mutating apply() instead.

For internal performance-specific uses, see _apply_inplace().

**as_vector** (**kwargs)**

Returns a flattened representation of the object as a single vector.

**Returns** vector ((N,) ndarray) – The core representation of the object, flattened into a single vector. Note that this is always a view back on to the original object, but is not writable.

**compose_after** *(transform)*

Returns a TransformChain that represents this transform composed after the given transform:

```
c = a.compose_after(b)
c.apply(p) == a.apply(b.apply(p))
```

a and b are left unchanged.

This corresponds to the usual mathematical formalism for the compose operator, o.

**Parameters** transform (Transform) – Transform to be applied before self

**Returns** transform (TransformChain) – The resulting transform chain.

**compose_before** *(transform)*

Returns a TransformChain that represents this transform composed before the given transform:

```
c = a.compose_before(b)
c.apply(p) == b.apply(a.apply(p))
```

a and b are left unchanged.
**Parameters**

- **transform** *(Transform)* – Transform to be applied after self

**Returns**

- **transform** *(TransformChain)* – The resulting transform chain.

**copy()**

Generate an efficient copy of this object.

Note that Numpy arrays and other **Copyable** objects on self will be deeply copied. Dictionaries and sets will be shallow copied, and everything else will be assigned (no copy will be made).

Classes that store state other than numpy arrays and immutable types should overwrite this method to ensure all state is copied.

**Returns**

- **type(self)** – A copy of this object

**d_dp(.)**

The derivative with respect to the parametrisation changes evaluated at points.

**Parameters**

- **points** *(n_points, n_dims) ndarray* – The spatial points at which the derivative should be evaluated.

**Returns**

- **d_dp** *(n_points, n_parameters, n_dims) ndarray* – The Jacobian with respect to the parametrisation.

**from_vector(vector)**

Build a new instance of the object from it’s vectorized state.

self is used to fill out the missing state required to rebuild a full object from it’s standardized flattened state. This is the default implementation, which is which is a **deepcopy** of the object followed by a call to **from_vector_inplace()**. This method can be overridden for a performance benefit if desired.

**Parameters**

- **vector** *(n_parameters,) ndarray* – Flattened representation of the object.

**Returns**

- **object** *(type(self))* – An new instance of this class.

**from_vector_inplace(vector)**

Deprecated. Use the non-mutating API, **from_vector**.

For internal usage in performance-sensitive spots, see **from_vector_inplace()**

**Parameters**

- **vector** *(n_parameters,) ndarray* – Flattened representation of this object

**has_nan_values()**

Tests if the vectorized form of the object contains nan values or not. This is particularly useful for objects with unknown values that have been mapped to nan values.

**Returns**

- **has_nan_values** *(bool)* – If the vectorized object contains nan values.

**increment(shapes, n_shapes=None, forgetting_factor=1.0, max_n_components=None, verbose=False)**

Update the eigenvectors, eigenvalues and mean vector of this model by performing incremental PCA on the given samples.

**Parameters**

- **shapes** *(list of menpo.shape.PointCloud)* – List of new shapes to update the model from.

- **n_shapes** *(int or None, optional)* – If int, then shapes must be an iterator that yields n_shapes. If None, then shapes has to be a list (so we know how large the data matrix needs to be).

- **forgetting_factor** *(0.0, 1.0) float, optional)* – Forgetting factor that weights the relative contribution of new samples vs old samples. If 1.0, all samples are weighted
equally and, hence, the results is the exact same as performing batch PCA on the concatenated list of old and new simples. If <1.0, more emphasis is put on the new samples. See [1] for details.

• **max_n_components** *(int or None, optional)* – The maximum number of components that the model will keep. If None, then all the components will be kept.

• **verbose** *(bool, optional)* – If True, then information about the progress will be printed.

---

**References**

**set_target** *(target)*

Update this object so that it attempts to recreate the new_target.

Parameters **new_target** *(menpo.shape.PointCloud)* – The new target that this object should try and regenerate.

**property dense_target**

The current dense menpo.shape.PointCloud that this object produces.

Type menpo.shape.PointCloud

**property global_parameters**

The parameters for the global transform.

Type *(n_global_parameters,) ndarray*

**property n_active_components**

The number of components currently in use on this model.

Type *int*

**property n_dims**

The number of dimensions of the spatial instance of the model

Type *int*

**property n_dims_output**

The output of the data from the transform.

None if the output of the transform is not dimension specific.

Type *int or None*

**property n_global_parameters**

The number of parameters in the global_transform

Type *int*

**property n_landmarks**

The number of sparse landmarks.

Type *int*

**property n_parameters**

The length of the vector that this object produces.

Type *int*

**property n_points**

The number of points on the target.

Type *int*
property n_weights
    The number of parameters in the linear model.
    Type int

property sparse_target
    The current sparse menpo.shape.PointCloud that this object produces.
    Type menpo.shape.PointCloud

property target
    The current menpo.shape.PointCloud that this object produces.
    Type menpo.shape.PointCloud

property weights
    The weights of the model.
    Type (n_weights,) ndarray

Homogeneous Transforms

DifferentiableAffine

class menpofit.transform.DifferentiableAffine (h_matrix, copy=True, skip_checks=False)
    Bases: Affine, DP, DX

    Base class for an affine transformation that can compute its own derivative with respect to spatial changes, as well as its parametrisation.

apply (x, batch_size=None, **kwargs)
    Applies this transform to x.

    If x is Transformable, x will be handed this transform object to transform itself non-destructively (a transformed copy of the object will be returned).

    If not, x is assumed to be an ndarray. The transformation will be non-destructive, returning the transformed version.

    Any kwargs will be passed to the specific transform _apply() method.

Parameters

• x (Transformable or (n_points, n_dims) ndarray) – The array or object to be transformed.

• batch_size (int, optional) – If not None, this determines how many items from the numpy array will be passed through the transform at a time. This is useful for operations that require large intermediate matrices to be computed.

• kwargs (dict) – Passed through to _apply().

Returns transformed (type(x)) – The transformed object or array

apply_inplace (*args, **kwargs)
    Deprecated as public supported API, use the non-mutating apply() instead.

    For internal performance-specific uses, see _apply_inplace().

as_vector (**kwargs)
    Returns a flattened representation of the object as a single vector.
Returns vector \( ((N,) \text{ndarray}) \) – The core representation of the object, flattened into a single vector. Note that this is always a view back on to the original object, but is not writable.

**compose_after** (transform)

A Transform that represents this transform composed after the given transform:

```python
a = a.compose_after(b)
c.apply(p) == a.apply(b.apply(p))
```

\( a \) and \( b \) are left unchanged.

This corresponds to the usual mathematical formalism for the compose operator, \( \circ \).

An attempt is made to perform native composition, but will fall back to a TransformChain as a last resort. See composes_with for a description of how the mode of composition is decided.

**Parameters**

- **transform** (Transform) – Transform to be applied before **self**

**Returns**

- **transform** (Transform or TransformChain) – If the composition was native, a single new Transform will be returned. If not, a TransformChain is returned instead.

**compose_after_from_vector_inplace** (vector)

Specialised inplace composition with a vector. This should be overridden to provide specific cases of composition whereby the current state of the transform can be derived purely from the provided vector.

**Parameters**

- **vector** ((n_parameters,) \text{ndarray}) – Vector to update the transform state with.

**compose_after_inplace** (transform)

Update **self** so that it represents this transform composed after the given transform:

```python
a = a.copy()
a.compose_after_inplace(b)
a.apply(p) == a_orig.apply(b.apply(p))
```

\( a \) is permanently altered to be the result of the composition. \( b \) is left unchanged.

**Parameters**

- **transform** (composes_inplace_with) – Transform to be applied before **self**

**Raises**

- **ValueError** – If **transform** isn’t an instance of composes_inplace_with

**compose_before** (transform)

A Transform that represents this transform composed before the given transform:

```python
a = a.copy()
a.compose_before(b)
c.apply(p) == b.apply(a.apply(p))
```

\( a \) and \( b \) are left unchanged.

An attempt is made to perform native composition, but will fall back to a TransformChain as a last resort. See composes_with for a description of how the mode of composition is decided.

**Parameters**

- **transform** (Transform) – Transform to be applied after **self**

**Returns**

- **transform** (Transform or TransformChain) – If the composition was native, a single new Transform will be returned. If not, a TransformChain is returned instead.

**compose_before_inplace** (transform)

Update **self** so that it represents this transform composed before the given transform:
```python
a_orig = a.copy()
a.compose_before_inplace(b)
a.apply(p) == b.apply(a_orig.apply(p))
```

a is permanently altered to be the result of the composition. b is left unchanged.

**Parameters**

- **transform** (*composes_inplace_with*) – Transform to be applied after self

**Raises**

- **ValueError** – If transform isn’t an instance of composes_inplace_with

**copy** ()

Generate an efficient copy of this object.

Note that Numpy arrays and other Copyable objects on self will be deeply copied. Dictionaries and sets will be shallow copied, and everything else will be assigned (no copy will be made).

Classes that store state other than numpy arrays and immutable types should overwrite this method to ensure all state is copied.

**Returns**

- **type(self)** – A copy of this object

**d_dp** (*points*)

The derivative with respect to the parametrisation changes evaluated at points.

**Parameters**

- **points** (*ndarray*) – The spatial points at which the derivative should be evaluated.

**Returns**

- **d_dp** (*ndarray*) – The Jacobian with respect to the parametrisation.

\[d_{dp}[i, j, k]\]

is the scalar differential change that the \(k\)'th dimension of the \(i\)'th point experiences due to a first order change in the \(j\)'th scalar in the parametrisation vector.

**d_dx** (*points*)

The first order derivative with respect to spatial changes evaluated at points.

**Parameters**

- **points** (*ndarray*) – The spatial points at which the derivative should be evaluated.

**Returns**

- **d_dx** (*ndarray*) – The Jacobian wrt spatial changes.

\[d_{dx}[i, j, k]\]

is the scalar differential change that the \(j\)'th dimension of the \(i\)'th point experiences due to a first order change in the \(k\)'th dimension.

It may be the case that the Jacobian is constant across space - in this case axis zero may have length 1 to allow for broadcasting.

**decompose** ()

Decompose this transform into discrete Affine Transforms.

Useful for understanding the effect of a complex composite transform.

**Returns**

- **transforms** (*list of DiscreteAffine*) – Equivalent to this affine transform, such that

\[
\text{reduce}(\lambda x, y: x.chain(y), \text{self}.decompose()) == \text{self}
\]
from_vector(vector)
   Build a new instance of the object from its vectorized state.
   
   self is used to fill out the missing state required to rebuild a full object from it’s standardized flattened state. This is the default implementation, which is a deepcopy of the object followed by a call to from_vector_inplace(). This method can be overridden for a performance benefit if desired.
   
   Parameters vector((n_parameters,), ndarray) – Flattened representation of the object.
   
   Returns transform (Homogeneous) – An new instance of this class.

from_vector_inplace(vector)
   Deprecated. Use the non-mutating API, from_vector.
   
   For internal usage in performance-sensitive spots, see _from_vector_inplace()
   
   Parameters vector((n_parameters,), ndarray) – Flattened representation of this object

has_nan_values()
   Tests if the vectorized form of the object contains nan values or not. This is particularly useful for objects with unknown values that have been mapped to nan values.
   
   Returns has_nan_values (bool) – If the vectorized object contains nan values.

classmethod init_from_2d_shear(phi, psi, degrees=True)
   Convenience constructor for 2D shear transformations about the origin.
   
   Parameters
   • phi (float) – The angle of shearing in the X direction.
   • psi (float) – The angle of shearing in the Y direction.
   • degrees (bool, optional) – If True phi and psi are interpreted as degrees. If False, phi and psi are interpreted as radians.
   
   Returns shear_transform (Affine) – A 2D shear transform.

classmethod init_identity(n_dims)
   Creates an identity matrix Affine transform.
   
   Parameters n_dims (int) – The number of dimensions.
   
   Returns identity (Affine) – The identity matrix transform.

pseudoinverse()
   The pseudoinverse of the transform - that is, the transform that results from swapping source and target, or more formally, negating the transforms parameters. If the transform has a true inverse this is returned instead.
   
   Type Homogeneous

pseudoinverse_vector(vector)
   The vectorized pseudoinverse of a provided vector instance. Syntactic sugar for:

   ```python
   self.from_vector(vector).pseudoinverse().as_vector()
   ```
   
   Can be much faster than the explicit call as object creation can be entirely avoided in some cases.
   
   Parameters vector((n_parameters,), ndarray) – A vectorized version of self
   
   Returns pseudoinverse_vector ((n_parameters,), ndarray) – The pseudoinverse of the vector provided
set_h_matrix(value, copy=True, skip_checks=False)
    Deprecated
    Deprecated - do not use this method - you are better off just creating a new transform!
    Updates h_matrix, optionally performing sanity checks.
    Note that it won’t always be possible to manually specify the h_matrix through this method, specifically
    if changing the h_matrix could change the nature of the transform. See h_matrix_is_mutable for
    how you can discover if the h_matrix is allowed to be set for a given class.

    Parameters
    • value (ndarray) – The new homogeneous matrix to set.
    • copy (bool, optional) – If False, do not copy the h_matrix. Useful for performance.
    • skip_checks (bool, optional) – If True, skip checking. Useful for performance.

    Raises NotImplementedError – If h_matrix_is_mutable returns False.

property composes_inplace_with
    Affine can swallow composition with any other Affine.

property composes_with
    Any Homogeneous can compose with any other Homogeneous.

property h_matrix
    The homogeneous matrix defining this transform.
    Type (n_dims + 1, n_dims + 1) ndarray

property h_matrix_is_mutable
    Deprecated True iff set_h_matrix() is permitted on this type of transform.
    If this returns False calls to set_h_matrix() will raise a Not ImplementedError.
    Type bool

property has_true_inverse
    The pseudoinverse is an exact inverse.
    Type True

property linear_component
    The linear component of this affine transform.
    Type (n_dims, n_dims) ndarray

property n_dims
    The dimensionality of the data the transform operates on.
    Type int

property n_dims_output
    The output of the data from the transform.
    Type int

property n_parameters
    n_dims * (n_dims + 1) parameters - every element of the matrix but the homogeneous part.
    Type int

Examples
    2D Affine: 6 parameters:
3D Affine: 12 parameters:

\[
\begin{cases}
[p_1, p_3, p_5] \\
[p_2, p_4, p_6]
\end{cases}
\]

property translation_component
The translation component of this affine transform.

Type (n_dims,) ndarray

DifferentiableSimilarity

class menpofit.transform.DifferentiableSimilarity(h_matrix, copy=True, skip_checks=False)

Bases: Similarity, DX

Base class for a similarity transformation that can compute its own derivative with respect to spatial changes, as well as its parametrisation.

apply (x, batch_size=None, **kwargs)
Applies this transform to x.

If x is Transformable, x will be handed this transform object to transform itself non-destructively (a transformed copy of the object will be returned).

If not, x is assumed to be an ndarray. The transformation will be non-destructive, returning the transformed version.

Any kwargs will be passed to the specific transform _apply() method.

Parameters

• x (Transformable or (n_points, n_dims) ndarray) – The array or object to be transformed.

• batch_size (int, optional) – If not None, this determines how many items from the numpy array will be passed through the transform at a time. This is useful for operations that require large intermediate matrices to be computed.

• kwargs (dict) – Passed through to _apply().

Returns transformed (type (x)) – The transformed object or array

apply_inplace(*args, **kwargs)
Deprecated as public supported API, use the non-mutating apply() instead.

For internal performance-specific uses, see _apply_inplace().

as_vector (**kwargs)
Returns a flattened representation of the object as a single vector.

Returns vector ((N,) ndarray) – The core representation of the object, flattened into a single vector. Note that this is always a view back on to the original object, but is not writable.

compose_after (transform)
A Transform that represents this transform composed after the given transform:
\[ c = a.compose_after(b) \]
\[ c.apply(p) == a.apply(b.apply(p)) \]

a and b are left unchanged.

This corresponds to the usual mathematical formalism for the compose operator, \( o \).

An attempt is made to perform native composition, but will fall back to a TransformChain as a last resort. See \( \text{composes_with} \) for a description of how the mode of composition is decided.

**Parameters**

- **transform** (`Transform`) – Transform to be applied before `self`

**Returns**

- **transform** (`Transform` or `TransformChain`) – If the composition was native, a single new `Transform` will be returned. If not, a `TransformChain` is returned instead.

**compose_after_from_vector_inplace** (`vector`)

Specialised inplace composition with a vector. This should be overridden to provide specific cases of composition whereby the current state of the transform can be derived purely from the provided vector.

**Parameters**

- **vector** `((n_parameters,) ndarray)` – Vector to update the transform state with.

**compose_after_inplace** (`transform`)

Update `self` so that it represents this transform composed after the given transform:

\[ a_orig = a.copy() \]
\[ a.compose_after_inplace(b) \]
\[ a.apply(p) == a_orig.apply(b.apply(p)) \]

a is permanently altered to be the result of the composition. b is left unchanged.

**Parameters**

- **transform** (`composes_inplace_with`) – Transform to be applied before `self`

**Raises**

- **ValueError** – If transform isn’t an instance of `composes_inplace_with`

**compose_before** (`transform`)

A `Transform` that represents this transform composed before the given transform:

\[ c = a.compose_before(b) \]
\[ c.apply(p) == b.apply(a.apply(p)) \]

a and b are left unchanged.

An attempt is made to perform native composition, but will fall back to a TransformChain as a last resort. See \( \text{composes_with} \) for a description of how the mode of composition is decided.

**Parameters**

- **transform** (`Transform`) – Transform to be applied after `self`

**Returns**

- **transform** (`Transform` or `TransformChain`) – If the composition was native, a single new `Transform` will be returned. If not, a `TransformChain` is returned instead.

**compose_before_inplace** (`transform`)

Update `self` so that it represents this transform composed before the given transform:

\[ a_orig = a.copy() \]
\[ a.compose_before_inplace(b) \]
\[ a.apply(p) == b.apply(a_orig.apply(p)) \]

a is permanently altered to be the result of the composition. b is left unchanged.
Parameters `transform` *(composes_inplace_with)* – Transform to be applied after `self`

Raises `ValueError` – If `transform` isn’t an instance of `composes_inplace_with`

`copy()`
Generate an efficient copy of this object.

Note that Numpy arrays and other `Copyable` objects on `self` will be deeply copied. Dictionaries and sets will be shallow copied, and everything else will be assigned (no copy will be made).

Classes that store state other than numpy arrays and immutable types should overwrite this method to ensure all state is copied.

Returns `type(self)` – A copy of this object

`d_dp(points)`
The derivative with respect to the parametrisation changes evaluated at points.

Parameters `points` *(n_points, n_dims) ndarray* – The spatial points at which the derivative should be evaluated.

Returns `d_dp((n_points, n_parameters, n_dims) ndarray)` – The Jacobian with respect to the parametrisation.

`d_dp[i, j, k]` is the scalar differential change that the k’th dimension of the i’th point experiences due to a first order change in the j’th scalar in the parametrisation vector.

`d_dx(points)`
The first order derivative with respect to spatial changes evaluated at points.

Parameters `points` *(n_points, n_dims) ndarray* – The spatial points at which the derivative should be evaluated.

Returns `d_dx((n_points, n_dims, n_dims) ndarray)` – The Jacobian wrt spatial changes.

`d_dx[i, j, k]` is the scalar differential change that the j’th dimension of the i’th point experiences due to a first order change in the k’th dimension.

It may be the case that the Jacobian is constant across space - in this case axis zero may have length 1 to allow for broadcasting.

`decompose()`
Decompose this transform into discrete Affine Transforms.

Useful for understanding the effect of a complex composite transform.

Returns `transforms` *(list of DiscreteAffine)* – Equivalent to this affine transform, such that

```python
reduce(lambda x, y: x.chain(y), self.decompose()) == self
```

`from_vector(vector)`
Build a new instance of the object from its vectorized state.

`self` is used to fill out the missing state required to rebuild a full object from it’s standardized flattened state. This is the default implementation, which is a `deepcopy` of the object followed by a call to `from_vector_inplace()`. This method can be overridden for a performance benefit if desired.

Parameters `vector` *(n_parameters,) ndarray* – Flattened representation of the object.
Returns **transform** (Homogeneous) – An new instance of this class.

**from_vector_inplace** *(vector)*
Depreciated. Use the non-mutating API, `from_vector`.
For internal usage in performance-sensitive spots, see `_from_vector_inplace()`

**Parameters**

- **vector** *((n_parameters,) ndarray)* – Flattened representation of this object

**has_nan_values** *
Tests if the vectorized form of the object contains `nan` values or not. This is particularly useful for objects with unknown values that have been mapped to `nan` values.

**Returns** **has纳米_values** *(bool)* – If the vectorized object contains `nan` values.

**classmethod init_from_2d_shear** *(phi, psi, degrees=True)*
Convenience constructor for 2D shear transformations about the origin.

**Parameters**

- **phi** *(float)* – The angle of shearing in the X direction.
- **psi** *(float)* – The angle of shearing in the Y direction.
- **degrees** *(bool, optional)* – If `True` phi and psi are interpreted as degrees. If `False`, phi and psi are interpreted as radians.

**Returns** **shear_transform** *(Affine)* – A 2D shear transform.

**classmethod init_identity** *(n_dims)*
Creates an identity transform.

**Parameters**

- **n_dims** *(int)* – The number of dimensions.

**Returns** **identity** *(Similarity)* – The identity matrix transform.

**pseudoinverse** *
The pseudoinverse of the transform - that is, the transform that results from swapping source and target, or more formally, negating the transforms parameters. If the transform has a true inverse this is returned instead.

**Type** Homogeneous

**pseudoinverse_vector** *(vector)*
The vectorized pseudoinverse of a provided vector instance. Syntactic sugar for:

```python
self.from_vector(vector).pseudoinverse().as_vector()
```

Can be much faster than the explicit call as object creation can be entirely avoided in some cases.

**Parameters**

- **vector** *((n_parameters,) ndarray)* – A vectorized version of self

**Returns** **pseudoinverse_vector** *((n_parameters,) ndarray)* – The pseudoinverse of the vector provided

**set_h_matrix** *(value, copy=True, skip_checks=False)*
Deprecated Depreciated - do not use this method - you are better off just creating a new transform!

Updates h_matrix, optionally performing sanity checks.

Note that it won’t always be possible to manually specify the h_matrix through this method, specifically if changing the h_matrix could change the nature of the transform. See `h_matrix_is_mutable` for how you can discover if the h_matrix is allowed to be set for a given class.

**Parameters**
property **composes_inplace_with**

Affine can swallow composition with any other Affine.

property **composes_with**

Any Homogeneous can compose with any other Homogeneous.

property **h_matrix**

The homogeneous matrix defining this transform.

Type `(n_dims + 1, n_dims + 1) ndarray`

property **h_matrix_is_mutable**

Deprecated True iff `set_h_matrix()` is permitted on this type of transform.

If this returns False calls to `set_h_matrix()` will raise a `NotImplementedError`.

Type `bool`

property **has_true_inverse**

The pseudoinverse is an exact inverse.

Type `True`

property **linear_component**

The linear component of this affine transform.

Type `(n_dims, n_dims) ndarray`

property **n_dims**

The dimensionality of the data the transform operates on.

Type `int`

property **n_dims_output**

The output of the data from the transform.

Type `int`

property **n_parameters**

Number of parameters of Similarity

2D Similarity - 4 parameters

```
[(1 + a), -b, tx]
[b, (1 + a), ty]
```

3D Similarity: Currently not supported

Returns `n_parameters (int)` – The transform parameters

Raises `DimensionalityError, NotImplementedeError` – Only 2D transforms are supported.

property **translation_component**

The translation component of this affine transform.

Type `(n_dims,) ndarray`
DifferentiableAlignmentSimilarity

```python
class menpofit.transform.DifferentiableAlignmentSimilarity(source, target, rotation=True, allow_mirror=False)
```

**Bases:** AlignmentSimilarity, DP, DX

Base class that constructs a similarity transformation that is the optimal transform to align the *source* to the *target*. It can compute its own derivative with respect to spatial changes, as well as its parametrisation.

**aligned_source()**
The result of applying `self` to `source`

**alignment_error()**
The Frobenius Norm of the difference between the target and the aligned source.

**apply(x, batch_size=None, **kwargs)**
Applies this transform to `x`.

- `x` is `Transformable`, `x` will be handed this transform object to transform itself non-destructively (a transformed copy of the object will be returned).
- If not, `x` is assumed to be an `ndarray`. The transformation will be non-destructive, returning the transformed version.

Any `kwargs` will be passed to the specific transform `_apply()` method.

**Parameters**

- `x` (**Transformable** or `(n_points, n_dims) ndarray`) – The array or object to be transformed.
- `batch_size` (**int**, optional) – If not `None`, this determines how many items from the numpy array will be passed through the transform at a time. This is useful for operations that require large intermediate matrices to be computed.
- `kwargs` (**dict**) – Passed through to `_apply()`.

**Returns** `transformed` (**type(x)**) – The transformed object or array

**apply_inplace(*args, **kwargs)**
Deprecated as public supported API, use the non-mutating `apply()` instead.

For internal performance-specific uses, see `_apply_inplace()`.

**as_non_alignment()**
Returns the non-alignment version of the transform.

**as_vector(**kwargs)**
Returns a flattened representation of the object as a single vector.

**Returns** `vector` (**(N,) ndarray**) – The core representation of the object, flattened into a single vector. Note that this is always a view back on to the original object, but is not writable.

**compose_after(transform)**
A `Transform` that represents `this` transform composed **after** the given transform:
c = a.compose_after(b)
c.apply(p) == a.apply(b.apply(p))

a and b are left unchanged.

This corresponds to the usual mathematical formalism for the compose operator, \( \circ \).

An attempt is made to perform native composition, but will fall back to a TransformChain as a last resort. See composes_with for a description of how the mode of composition is decided.

**Parameters**

**transform** (*Transform*) – Transform to be applied before self

**Returns**

**transform** (*Transform* or *TransformChain*) – If the composition was native, a single new Transform will be returned. If not, a TransformChain is returned instead.

**compose_after_from_vector_inplace** (*vector*)

Specialised inplace composition with a vector. This should be overridden to provide specific cases of composition whereby the current state of the transform can be derived purely from the provided vector.

**Parameters**

**vector** (*((n_parameters,), ndarray]*) – Vector to update the transform state with.

**compose_after_inplace** (*transform*)

Update self so that it represents this transform composed after the given transform:

a = a.copy()
a.compose_after_inplace(b)
a.apply(p) == a_orig.apply(b.apply(p))

a is permanently altered to be the result of the composition. b is left unchanged.

**Parameters**

**transform** (*composes_inplace_with*) – Transform to be applied before self

**Raises**

**ValueError** – If transform isn’t an instance of composes_inplace_with

**compose_before** (*transform*)

A Transform that represents this transform composed before the given transform:

\[
c = a.compose_before(b)
c.apply(p) == b.apply(a.apply(p))
\]

a and b are left unchanged.

An attempt is made to perform native composition, but will fall back to a TransformChain as a last resort. See composes_with for a description of how the mode of composition is decided.

**Parameters**

**transform** (*Transform*) – Transform to be applied after self

**Returns**

**transform** (*Transform* or *TransformChain*) – If the composition was native, a single new Transform will be returned. If not, a TransformChain is returned instead.

**compose_before_inplace** (*transform*)

Update self so that it represents this transform composed before the given transform:

a = a.copy()
a.compose_before_inplace(b)
a.apply(p) == b.apply(a_orig.apply(p))

a is permanently altered to be the result of the composition. b is left unchanged.
Parameters **transform** (*composes_inplace_with*) - Transform to be applied after
self

Raises **ValueError** - If **transform** isn’t an instance of *composes_inplace_with*

copy() - Generate an efficient copy of this `HomogFamilyAlignment`.

Returns **new_transform** (type(self)) - A copy of this object

d\_dp(points) - The derivative with respect to the parametrisation changes evaluated at points.

Parameters **points** ((n\_points, n\_dims) `ndarray`) - The spatial points at which the
derivative should be evaluated.

Returns

\[ \frac{d}{dp} ((n\_points, n\_parameters, n\_dims) `ndarray`) - The Jacobian with respect to the parametrisation.

\[ \frac{d}{dp}[i, j, k] \text{ is the scalar differential change that the } k\text{'th dimension of the } i\text{'th point experiences due to a first order change in the } j\text{'th scalar in the parametrisation vector.}

d\_dx(points) - The first order derivative with respect to spatial changes evaluated at points.

Parameters **points** ((n\_points, n\_dims) `ndarray`) - The spatial points at which the
derivative should be evaluated.

Returns

\[ \frac{d}{dx} ((n\_points, n\_dims, n\_dims) `ndarray`) - The Jacobian wrt spatial changes.

\[ \frac{d}{dx}[i, j, k] \text{ is the scalar differential change that the } j\text{'th dimension of the } i\text{'th point experiences due to a first order change in the } k\text{'th dimension.}

It may be the case that the Jacobian is constant across space - in this case axis zero may have
length 1 to allow for broadcasting.

decompose() - Decompose this transform into discrete Affine Transforms.

Useful for understanding the effect of a complex composite transform.

Returns

**transforms** (list of DiscreteAffine) - Equivalent to this affine transform, such that

\[ \text{reduce(lambda } x, y: x.chain(y), self.decompose()) == self \]

from\_vector(vector) - Build a new instance of the object from its vectorized state.

\[ \text{self is used to fill out the missing state required to rebuild a full object from it’s standardized flattened state. This is the default implementation, which is a deepcopy of the object followed by a call to from\_vector\_inplace(). This method can be overridden for a performance benefit if desired.} \]

Parameters **vector** ((n\_parameters,) `ndarray`) - Flattened representation of the object.

Returns **transform** (Homogeneous) - An new instance of this class.

from\_vector\_inplace(vector) - Deprecated. Use the non-mutating API, from\_vector.

For internal usage in performance-sensitive spots, see from\_vector\_inplace()
Parameters **vector**((n_parameters,) ndarray) – Flattened representation of this object

**has_nan_values**()
Tests if the vectorized form of the object contains nan values or not. This is particularly useful for objects with unknown values that have been mapped to nan values.

Returns **has_nan_values** (bool) – If the vectorized object contains nan values.

**classmethod init_from_2d_shear** (phi, psi, degrees=True)
Convenience constructor for 2D shear transformations about the origin.

Parameters

- **phi** (float) – The angle of shearing in the X direction.
- **psi** (float) – The angle of shearing in the Y direction.
- **degrees** (bool, optional) – If True phi and psi are interpreted as degrees. If False, phi and psi are interpreted as radians.

Returns **shear_transform** (Affine) – A 2D shear transform.

**classmethod init_identity**(n_dims)
Creates an identity transform.

Parameters **n_dims** (int) – The number of dimensions.

Returns **identity** (Similarity) – The identity matrix transform.

**pseudoinverse**()
The pseudoinverse of the transform - that is, the transform that results from swapping source and target, or more formally, negating the transforms parameters. If the transform has a true inverse this is returned instead.

Returns **transform** (type(self)) – The inverse of this transform.

**pseudoinverse_vector** (vector)
The vectorized pseudoinverse of a provided vector instance. Syntactic sugar for:

```python
self.from_vector(vector).pseudoinverse().as_vector()
```

Can be much faster than the explicit call as object creation can be entirely avoided in some cases.

Parameters **vector**((n_parameters,) ndarray) – A vectorized version of self

Returns **pseudoinverse_vector**((n_parameters,) ndarray) – The pseudoinverse of the vector provided

**set_h_matrix** (value, copy=True, skip_checks=False)
Deprecated Deprecated - do not use this method - you are better off just creating a new transform!

Updates h_matrix, optionally performing sanity checks.

Note that it won’t always be possible to manually specify the h_matrix through this method, specifically if changing the h_matrix could change the nature of the transform. See h_matrix_is_mutable for how you can discover if the h_matrix is allowed to be set for a given class.

Parameters

- **value** (ndarray) – The new homogeneous matrix to set.
- **copy** (bool, optional) – If False, do not copy the h_matrix. Useful for performance.
- **skip_checks** (bool, optional) – If True, skip checking. Useful for performance.

Raises **NotImplementedError** – If h_matrix_is_mutable returns False.
**set_target** *(new_target)*

Update this object so that it attempts to recreate the `new_target`.

**Parameters**

`new_target` *(PointCloud)* – The new target that this object should try and regenerate.

**property composes_inplace_with**

Affine can swallow composition with any other Affine.

**property composes_with**

Any Homogeneous can compose with any other Homogeneous.

**property h_matrix**

The homogeneous matrix defining this transform.

**Type** *(n_dims + 1, n_dims + 1) ndarray*

**property h_matrix_is_mutable**

Deprecated True iff `set_h_matrix()` is permitted on this type of transform.

If this returns False calls to `set_h_matrix()` will raise a `NotImplementedError`.

**Type** *bool*

**property has_true_inverse**

The pseudoinverse is an exact inverse.

**Type** *True*

**property linear_component**

The linear component of this affine transform.

**Type** *(n_dims, n_dims) ndarray*

**property n_dims**

The number of dimensions of the target.

**Type** *int*

**property n_dims_output**

The output of the data from the transform.

**Type** *int*

**property n_parameters**

Number of parameters of Similarity

2D Similarity - 4 parameters

\[
\begin{bmatrix}
(l + a), & -b, & tx \\
[b, & (1 + a), & ty]
\end{bmatrix}
\]

3D Similarity: Currently not supported

**Returns**

`n_parameters` *(int)* – The transform parameters

**Raises** `DimensionalityError`, `NotImplementedError` – Only 2D transforms are supported.

**property n_points**

The number of points on the target.

**Type** *int*
**property source**
The source :class:`PointCloud` that is used in the alignment.
The source is not mutable.

Type :class:`PointCloud`

**property target**
The current :class:`PointCloud` that this object produces.
To change the target, use :meth:`set_target()`.

Type :class:`PointCloud`

**property translation_component**
The translation component of this affine transform.

Type :class:`(n_dims,) ndarray`

### DifferentiableAlignmentAffine

```python
class menpo.transform.DifferentiableAlignmentAffine(source, target)
Bases: AlignmentAffine, DP, DX
```

Base class that constructs an affine transformation that is the optimal transform to align the `source` to the `target`.
It can compute its own derivative with respect to spatial changes, as well as its parametrisation.

**aligned_source**
The result of applying `self` to `source`

Type :class:`PointCloud`

**alignment_error**
The Frobenius Norm of the difference between the target and the aligned source.

Type :class:`float`

**apply**
Applies this transform to `x`.

If `x` is :class:`Transformable`, `x` will be handed this transform object to transform itself non-destructively (a transformed copy of the object will be returned).

If not, `x` is assumed to be an :class:`ndarray`. The transformation will be non-destructive, returning the transformed version.

Any `kwargs` will be passed to the specific transform `_apply()` method.

**Parameters**

- **x** (:class:`Transformable` or :class:`(n_points, n_dims) ndarray`) – The array or object to be transformed.

- **batch_size** (`int`, optional) – If not `None`, this determines how many items from the numpy array will be passed through the transform at a time. This is useful for operations that require large intermediate matrices to be computed.

- **kwargs** (`dict`) – Passed through to `_apply()`.

**Returns**

- **transformed** (`type(x)`) – The transformed object or array

**apply_inplace**(*args, **kwargs*)

Deprecated as public supported API, use the non-mutating :meth:`apply()` instead.
For internal performance-specific uses, see _apply_inplace().

as_non_alignment()  
Returns the non-alignment version of the transform.

    Type DifferentiableAffine

as_vector(**kwargs)  
Returns a flattened representation of the object as a single vector.

    Returns vector ((N,) ndarray) – The core representation of the object, flattened into a single vector. Note that this is always a view back on to the original object, but is not writable.

compose_after(transform)  
A Transform that represents this transform composed after the given transform:

    c = a.compose_after(b)
    c.apply(p) == a.apply(b.apply(p))

a and b are left unchanged.

This corresponds to the usual mathematical formalism for the compose operator, o.

An attempt is made to perform native composition, but will fall back to a TransformChain as a last resort. See composes_with for a description of how the mode of composition is decided.

    Parameters transform (Transform) – Transform to be applied before self

    Returns transform (Transform or TransformChain) – If the composition was native, a single new Transform will be returned. If not, a TransformChain is returned instead.

compose_after_from_vector_inplace(vector)  
Specialised inplace composition with a vector. This should be overridden to provide specific cases of composition whereby the current state of the transform can be derived purely from the provided vector.

    Parameters vector ((n_parameters,) ndarray) – Vector to update the transform state with.

compose_after_inplace(transform)  
Update self so that it represents this transform composed after the given transform:

    a_orig = a.copy()
    a.compose_after_inplace(b)
    a.apply(p) == a_orig.apply(b.apply(p))

a is permanently altered to be the result of the composition. b is left unchanged.

    Parameters transform (composes_inplace_with) – Transform to be applied before self

    Raises ValueError – If transform isn’t an instance of composes_inplace_with

compose_before(transform)  
A Transform that represents this transform composed before the given transform:

    c = a.compose_before(b)
    c.apply(p) == b.apply(a.apply(p))

a and b are left unchanged.

An attempt is made to perform native composition, but will fall back to a TransformChain as a last resort. See composes_with for a description of how the mode of composition is decided.

    Parameters transform (Transform) – Transform to be applied after self
Returns **transform** *(Transform or TransformChain)* – If the composition was native, a single new Transform will be returned. If not, a TransformChain is returned instead.

**compose_before_inplace** *(transform)*

Update self so that it represents this transform composed before the given transform:

```python
a_orig = a.copy()
a.compose_before_inplace(b)
a.apply(p) == b.apply(a_orig.apply(p))
```

a is permanently altered to be the result of the composition. b is left unchanged.

**Parameters** **transform** *(composes_inplace_with)* – Transform to be applied after self

**Raises** ValueError – If transform isn’t an instance of composes_inplace_with

**copy** ()

Generate an efficient copy of this HomogFamilyAlignment.

**Returns** **new_transform** *(type(self))* – A copy of this object

**d_dp** *(points)*

The derivative with respect to the parametrisation changes evaluated at points.

**Parameters** **points** *((n_points, n_dims) ndarray)* – The spatial points at which the derivative should be evaluated.

**Returns**

**d_dp** *((n_points, n_parameters, n_dims) ndarray)* – The Jacobian with respect to the parametrisation.

**d_dp[i, j, k]** is the scalar differential change that the k’th dimension of the i’th point experiences due to a first order change in the j’th scalar in the parametrisation vector.

**d_dx** *(points)*

The first order derivative with respect to spatial changes evaluated at points.

**Parameters** **points** *((n_points, n_dims) ndarray)* – The spatial points at which the derivative should be evaluated.

**Returns**

**d_dx** *((n_points, n_dims, n_dims) ndarray)* – The Jacobian wrt spatial changes.

**d_dx[i, j, k]** is the scalar differential change that the j’th dimension of the i’th point experiences due to a first order change in the k’th dimension.

It may be the case that the Jacobian is constant across space - in this case axis zero may have length 1 to allow for broadcasting.

**decompose** ()

Decompose this transform into discrete Affine Transforms.

Useful for understanding the effect of a complex composite transform.

**Returns**

**transforms** *(list of DiscreteAffine)* – Equivalent to this affine transform, such that

```python
reduce(lambda x, y: x.chain(y), self.decompose()) == self
```
from_vector(vector)
Build a new instance of the object from its vectorized state.

`self` is used to fill out the missing state required to rebuild a full object from it's standardized flattened state. This is the default implementation, which is a `deepcopy` of the object followed by a call to `from_vector_inplace()`. This method can be overridden for a performance benefit if desired.

Parameters
- **vector** *(n_parameters,) ndarray* – Flattened representation of the object.

Returns
- **transform** *(Homogeneous)* – An new instance of this class.

from_vector_inplace(vector)
Deprecated. Use the non-mutating API, `from_vector`.

For internal usage in performance-sensitive spots, see `_from_vector_inplace()`.

Parameters
- **vector** *(n_parameters,) ndarray* – Flattened representation of this object

has_nan_values()
Tests if the vectorized form of the object contains `nan` values or not. This is particularly useful for objects with unknown values that have been mapped to `nan` values.

Returns
- **has_nan_values** *(bool)* – If the vectorized object contains `nan` values.

classmethod init_from_2d_shear(phi, psi, degrees=True)
Convenience constructor for 2D shear transformations about the origin.

Parameters
- **phi** *(float)* – The angle of shearing in the X direction.
- **psi** *(float)* – The angle of shearing in the Y direction.
- **degrees** *(bool, optional)* – If `True`, phi and psi are interpreted as degrees. If `False`, phi and psi are interpreted as radians.

Returns
- **shear_transform** *(Affine)* – A 2D shear transform.

classmethod init_identity(n_dims)
Creates an identity matrix Affine transform.

Parameters
- **n_dims** *(int)* – The number of dimensions.

Returns
- **identity** *(Affine)* – The identity matrix transform.

pseudoinverse()
The pseudoinverse of the transform - that is, the transform that results from swapping source and target, or more formally, negating the transforms parameters. If the transform has a true inverse this is returned instead.

Returns
- **transform** *(type(self))* – The inverse of this transform.

pseudoinverse_vector(vector)
The vectorized pseudoinverse of a provided vector instance. Syntactic sugar for:

```python
self.from_vector(vector).pseudoinverse().as_vector()
```

Can be much faster than the explicit call as object creation can be entirely avoided in some cases.

Parameters
- **vector** *(n_parameters,) ndarray* – A vectorized version of `self`

Returns
- **pseudoinverse_vector** *(n_parameters,) ndarray* – The pseudoinverse of the vector provided
set_h_matrix(value, copy=True, skip_checks=False)

Deprecated - do not use this method - you are better off just creating a new transform!

Updates h_matrix, optionally performing sanity checks.

Note that it won’t always be possible to manually specify the h_matrix through this method, specifically if changing the h_matrix could change the nature of the transform. See h_matrix_is_mutable for how you can discover if the h_matrix is allowed to be set for a given class.

Parameters

- value (ndarray) – The new homogeneous matrix to set.
- copy (bool, optional) – If False, do not copy the h_matrix. Useful for performance.
- skip_checks (bool, optional) – If True, skip checking. Useful for performance.

Raises Not Implemented Error – If h_matrix_is_mutable returns False.

set_target(new_target)

Update this object so that it attempts to recreate the new_target.

Parameters new_target (PointCloud) – The new target that this object should try and re-generate.

property composes_inplace_with

Affine can swallow composition with any other Affine.

property composes_with

Any Homogeneous can compose with any other Homogeneous.

property h_matrix

The homogeneous matrix defining this transform.

Type (n_dims + 1, n_dims + 1) ndarray

property h_matrix_is_mutable

Deprecated True iff set_h_matrix() is permitted on this type of transform.

If this returns False calls to set_h_matrix() will raise a Not Implemented Error.

Type bool

property has_true_inverse

The pseudoinverse is an exact inverse.

Type True

property linear_component

The linear component of this affine transform.

Type (n_dims, n_dims) ndarray

property n_dims

The number of dimensions of the target.

Type int

property n_dims_output

The output of the data from the transform.

Type int

property n_parameters

n_dims * (n_dims + 1) parameters - every element of the matrix but the homogeneous part.

Type int
Examples

2D Affine: 6 parameters:

\[
[p_1, p_3, p_5] \\
[p_2, p_4, p_6]
\]

3D Affine: 12 parameters:

\[
[p_1, p_4, p_7, p_{10}] \\
[p_2, p_5, p_8, p_{11}] \\
[p_3, p_6, p_9, p_{12}]
\]

**property n_points**

The number of points on the target.

Type `int`

**property source**

The source `PointCloud` that is used in the alignment.

The source is not mutable.

Type `PointCloud`

**property target**

The current `PointCloud` that this object produces.

To change the target, use `set_target()`.

Type `PointCloud`

**property translation_component**

The translation component of this affine transform.

Type `(n_dims,) ndarray`

Alignments

**DifferentiablePiecewiseAffine**

class `menpofit.transform.DifferentiablePiecewiseAffine(source, target)`

Bases: `CachedPWA, DL, DX`

A differentiable Piecewise Affine Transformation.

This is composed of a number of triangles defined by a set of source and target vertices. These vertices are related by a common triangle list. No limitations on the nature of the triangle list are imposed. Points can then be mapped via barycentric coordinates from the source to the target space. Trying to map points that are not contained by any source triangle throws a `TriangleContainmentError`, which contains diagnostic information.

The transform can compute its own derivative with respect to spatial changes, as well as anchor landmark changes.

**aligned_source()**

The result of applying self to source

Type `PointCloud`
alignment_error()

The Frobenius Norm of the difference between the target and the aligned source.

Type  float

apply(x, batch_size=None, **kwargs)

Applies this transform to x.

If x is Transformable, x will be handed this transform object to transform itself non-destructively (a transformed copy of the object will be returned).

If not, x is assumed to be an ndarray. The transformation will be non-destructive, returning the transformed version.

Any kwargs will be passed to the specific transform _apply() method.

Parameters
• x (Transformable or (n_points, n_dims) ndarray) – The array or object to be transformed.
• batch_size (int, optional) – If not None, this determines how many items from the numpay array will be passed through the transform at a time. This is useful for operations that require large intermediate matrices to be computed.
• kwargs (dict) – Passed through to _apply().

Returns transformed (type(x)) – The transformed object or array

apply_inplace(*args, **kwargs)

Deprecated as public supported API, use the non-mutating apply() instead.

For internal performance-specific uses, see _apply_inplace().

compose_after(transform)

Returns a TransformChain that represents this transform composed after the given transform:

\[
\begin{align*}
c &= a.compose_after(b) \\
c.apply(p) &= a.apply(b.apply(p))
\end{align*}
\]

a and b are left unchanged.

This corresponds to the usual mathematical formalism for the compose operator, \( o \).

Parameters transform (Transform) – Transform to be applied before self

Returns transform (TransformChain) – The resulting transform chain.

compose_before(transform)

Returns a TransformChain that represents this transform composed before the given transform:

\[
\begin{align*}
c &= a.compose_before(b) \\
c.apply(p) &= b.apply(a.apply(p))
\end{align*}
\]

a and b are left unchanged.

Parameters transform (Transform) – Transform to be applied after self

Returns transform (TransformChain) – The resulting transform chain.

copy()

Generate an efficient copy of this object.

Note that Numpy arrays and other Copyable objects on self will be deeply copied. Dictionaries and sets will be shallow copied, and everything else will be assigned (no copy will be made).
Classes that store state other than numpy arrays and immutable types should overwrite this method to ensure all state is copied.

**Returns**
`type(self) – A copy of this object`

**d_dl (points)**
The derivative of the warp with respect to spatial changes in anchor landmark points or centres, evaluated at points.

**Parameters**
`points ((n_points, n_dims) ndarray) – The spatial points at which the derivative should be evaluated.``

**Returns**
`d_dl ((n_points, n_centres, n_dims) ndarray) – The Jacobian wrt landmark changes.``

\[ d_{dl}[i, k, m] \] is the scalar differential change that the any dimension of the \( i \)’th point experiences due to a first order change in the \( m \)’th dimension of the \( k \)’th landmark point.

Note that at present this assumes that the change in every dimension is equal.

**d_dx (points)**
The first order derivative of the warp with respect to spatial changes evaluated at points.

**Parameters**
`points ((n_points, n_dims) ndarray) – The spatial points at which the derivative should be evaluated.``

**Returns**
`d_dx ((n_points, n_dims, n_dims) ndarray) – The Jacobian wrt spatial changes.``

\[ d_{dx}[i, j, k] \] is the scalar differential change that the \( j \)’th dimension of the \( i \)’th point experiences due to a first order change in the \( k \)’th dimension.

It may be the case that the Jacobian is constant across space - in this case axis zero may have length 1 to allow for broadcasting.

:raises TriangleContainmentError:: If any point is outside any triangle of this PWA.

**index_alpha_beta (points)**
Finds for each input point the index of its bounding triangle and the alpha and beta value for that point in the triangle. Note this means that the following statements will always be true:

\[
\begin{align*}
\text{alpha + beta} & \leq 1 \\
\text{alpha} & \geq 0 \\
\text{beta} & \geq 0
\end{align*}
\]

for each triangle result.

Trying to map a point that does not exist in a triangle throws a `TriangleContainmentError`.

**Parameters**
`points ((K, 2) ndarray) – Points to test.``

**Returns**
- `tri_index ((L,) ndarray) – Triangle index for each of the points, assigning each point to it’s containing triangle.``
- `alpha ((L,) ndarray) – Alpha for containing triangle of each point.``
- `beta ((L,) ndarray) – Beta for containing triangle of each point.``

:raises TriangleContainmentError:: All points must be contained in a source triangle. Check `error:points_outside_source_domain` to handle this case.
The pseudoinverse of the transform - that is, the transform that results from swapping source and target, or more formally, negating the transforms parameters. If the transform has a true inverse this is returned instead.

```python
def pseudoinverse()

```

**set_target** *(new_target)*
Update this object so that it attempts to recreate the new_target.

Parameters **new_target** *(PointCloud)* – The new target that this object should try and regenerate.

```python
set_target(new_target)
```

**has_true_inverse**
The inverse is true.

```python
property has_true_inverse

```

**n_dims**
The number of dimensions of the target.

```python
property n_dims

```

**n_dims_output**
The output of the data from the transform.

```python
property n_dims_output

```

**n_points**
The number of points on the target.

```python
property n_points

```

**n_tris**
The number of triangles in the triangle list.

```python
property n_tris

```

**source**
The source PointCloud that is used in the alignment.

```python
property source

```

**target**
The current PointCloud that this object produces.
To change the target, use *set_target()*.

```python
property target

```

**trilist**
The triangle list.

```python
property trilist

```

**DifferentiableThinPlateSplines**

```python
class menpofit.transform.DifferentiableThinPlateSplines(source, target, kernel=None)

Bases: ThinPlateSplines, DL, DX

```
The Thin Plate Splines (TPS) alignment between 2D source and target landmarks. The transform can compute its own derivative with respect to spatial changes, as well as anchor landmark changes.

Parameters

- **source** *(\(N, 2\) ndarray)* – The source points to apply the tps from
- **target** *(\(N, 2\) ndarray)* – The target points to apply the tps to
- **kernel** *(class or None, optional)* – The differentiable kernel to apply. Possible options are `DifferentiableR2LogRBF` and `DifferentiableR2LogR2RBF`. If None, then `DifferentiableR2LogR2RBF` is used.

**aligned_source()**

The result of applying `self` to `source`

Type `PointCloud`

**alignment_error()**

The Frobenius Norm of the difference between the target and the aligned source.

Type `float`

**apply** *(x, batch_size=None, **kwargs)*

Applies this transform to `x`.

If `x` is `Transformable`, `x` will be handed this transform object to transform itself non-destructively (a transformed copy of the object will be returned).

If not, `x` is assumed to be an `ndarray`. The transformation will be non-destructive, returning the transformed version.

Any `kwargs` will be passed to the specific transform `_apply()` method.

Parameters

- **x** *(Transformable or (n_points, n_dims) ndarray)* – The array or object to be transformed.
- **batch_size** *(int, optional)* – If not None, this determines how many items from the numpy array will be passed through the transform at a time. This is useful for operations that require large intermediate matrices to be computed.
- **kwargs** *(dict)* – Passed through to `_apply()`.

Returns transformed *(type(x))* – The transformed object or array

**apply_inplace** *(*args, **kwargs)*

Deprecated as public supported API, use the non-mutating `apply()` instead.

For internal performance-specific uses, see `_apply_inplace()`.

**compose_after** *(transform)*

Returns a `TransformChain` that represents this transform composed after the given transform:

```python
  a = a.compose_after(b)
  c.apply(p) == a.apply(b.apply(p))
```

a and b are left unchanged.

This corresponds to the usual mathematical formalism for the compose operator, o.

Parameters **transform** *(Transform)* – Transform to be applied before self

Returns transform *(TransformChain)* – The resulting transform chain.
**compose_before**(*transform*)

Returns a [TransformChain](#) that represents this transform composed before the given transform:

```python
    c = a.compose_before(b)
c.apply(p) == b.apply(a.apply(p))
```

a and b are left unchanged.

**Parameters**

- **transform** ([Transform](#)) – Transform to be applied after self

**Returns**

- **transform** ([TransformChain](#)) – The resulting transform chain.

**copy**()

Generate an efficient copy of this object.

Note that Numpy arrays and other [Copyable](#) objects on self will be deeply copied. Dictionaries and sets will be shallow copied, and everything else will be assigned (no copy will be made).

Classes that store state other than numpay arrays and immutable types should overwrite this method to ensure all state is copied.

**Returns**

- **type(self)** – A copy of this object

**d_dl**(*points*)

Calculates the Jacobian of the TPS warp wrt to the source landmarks assuming that the target is equal to the source. This is a special case of the Jacobian wrt to the source landmarks that is used in AAMs to weight the relative importance of each pixel in the reference frame wrt to each one of the source landmarks.

\[
\begin{align*}
    dW_{dl} &= dOmega_{dl} * k(points) \\
    &= T * d\_L**-1\_dl * k(points) \\
    &= T * -L**-1 dL_{dl} L**-1 * k(points) \\
\end{align*}
\]

**Parameters**

- **points** ([((n_points, n_dims)] ndarray) – The spatial points at which the derivative should be evaluated.

**Returns**

- **dW/dl** ([((n_points, n_params, n_dims)] ndarray) – The Jacobian of the transform wrt to the source landmarks evaluated at the previous points and assuming that the target is equal to the source.

**d_dx**(*points*)

The first order derivative of this TPS warp wrt spatial changes evaluated at points.

**Parameters**

- **points** ([((n_points, n_dims)] ndarray) – The spatial points at which the derivative should be evaluated.

**Returns**

- **d_dx** ([((n_points, n_dims, n_dims)] ndarray) – The Jacobian wrt spatial changes.

\[d\_dx[i, j, k] is the scalar differential change that the j'th dimension of the i'th point experiences due to a first order change in the k'th dimension.\]

It may be the case that the Jacobian is constant across space - in this case axis zero may have length 1 to allow for broadcasting.

**pseudoinverse**()

The pseudoinverse of the transform - that is, the transform that results from swapping source and target, or more formally, negating the transforms parameters. If the transform has a true inverse this is returned instead.

**Type**

- **type(self)**
**set_target** (*new_target*)
Update this object so that it attempts to recreate the *new_target*.

**Parameters**
new_target (*PointCloud*) – The new target that this object should try and re-generate.

**property has_true_inverse**

False

**Type**
type

**property n_dims**
The number of dimensions of the *target*.

**Type**
int

**property n_dims_output**
The output of the data from the transform.

None if the output of the transform is not dimension specific.

**Type**
int or None

**property n_points**
The number of points on the *target*.

**Type**
int

**property source**
The source *PointCloud* that is used in the alignment.

The source is not mutable.

**Type**
PointCloud

**property target**
The current *PointCloud* that this object produces.

To change the target, use *set_target()*.

**Type**
PointCloud

RBF

**DifferentiableR2LogR2RBF**

```python
class menpo.fit.transform.DifferentiableR2LogR2RBF(c)

Bases: R2LogR2RBF, DL

The $r^2 \log r^2$ basis function.

The derivative of this function is $2r(\log r^2 + 1)$, where $r = \|x - c\|$.

It can compute its own derivative with respect to landmark changes.
```

**apply** (*x*, *batch_size=None*, **kwargs)
Applies this transform to *x*.

If *x* is *Transformable*, *x* will be handed this transform object to transform itself non-destructively (a transformed copy of the object will be returned).

If not, *x* is assumed to be an *ndarray*. The transformation will be non-destructive, returning the transformed version.
Any kwargs will be passed to the specific transform _apply() method.

**Parameters**

- **x** *(Transformable or (n_points, n_dims) ndarray)* – The array or object to be transformed.
- **batch_size** *(int, optional)* – If not None, this determines how many items from the numpy array will be passed through the transform at a time. This is useful for operations that require large intermediate matrices to be computed.
- **kwargs** *(dict)* – Passed through to _apply().

Returns **transformed** *(type(x))* – The transformed object or array

**apply_inplace** (*args, **kwargs)*

Deprecated as public supported API, use the non-mutating apply() instead.

For internal performance-specific uses, see _apply_inplace().

**compose_after** *(transform)*

Returns a *TransformChain* that represents this transform composed after the given transform:

```python
    def compose_after(self, transform):
        c = self.compose_after(transform)
        c.apply(p) == a.apply(b.apply(p))
```

**compose_before** *(transform)*

Returns a *TransformChain* that represents this transform composed before the given transform:

```python
    def compose_before(self, transform):
        c = self.compose_before(transform)
        c.apply(p) == b.apply(a.apply(p))
```

**copy** ()

Generate an efficient copy of this object.

Note that Numpy arrays and other Copyable objects on self will be deeply copied. Dictionaries and sets will be shallow copied, and everything else will be assigned (no copy will be made).

Classes that store state other than numpy arrays and immutable types should overwrite this method to ensure all state is copied.

Returns **transform** *(TransformChain)* – The resulting transform chain.

**d_dl** *(points)*

Apply the derivative of the basis function wrt the centres and the points given by points. Let points be x, then

\[2(x - c)^T (\log r_{x,l} + 1) = 2(x - c)^T (2\log r_{x,l} + 1)\]

where \(r_{x,l} = \|x - c\|\).

**Parameters** **points** *(n_points, n_dims) ndarray* – The spatial points at which the derivative should be evaluated.
Returns \( \text{d}_dl ((n\_points, n\_centres, n\_dims) \text{ndarray}) \) – The jacobian tensor representing the first order derivative of the radius from each centre wrt the centre’s position, evaluated at each point.

**property n\_centres**
The number of centres.
Type int

**property n\_dims**
The RBF can only be applied on points with the same dimensionality as the centres.
Type int

**property n\_dims\_output**
The result of the transform has a dimension (weight) for every centre.
Type int

**DifferentiableR2LogRRBF**

**class** menpofit.transform.DifferentiableR2LogRRBF(c)
Bases: R2LogRRBF, DL
Calculates the \( r^2 \log r \) basis function.
The derivative of this function is \( r(1 + 2 \log r) \), where \( r = \|x - c\| \).
It can compute its own derivative with respect to landmark changes.

**apply** (x, batch_size=None, **kwargs)
Applies this transform to \( x \).
If \( x \) is Transformable, \( x \) will be handed this transform object to transform itself non-destructively (a transformed copy of the object will be returned).
If not, \( x \) is assumed to be an ndarray. The transformation will be non-destructive, returning the transformed version.
Any kwargs will be passed to the specific transform _apply() method.

**Parameters**

- **x** (Transformable or \( (n\_points, n\_dims) \text{ndarray} \)) – The array or object to be transformed.
- **batch_size** (int, optional) – If not None, this determines how many items from the numpy array will be passed through the transform at a time. This is useful for operations that require large intermediate matrices to be computed.
- **kwargs** (dict) – Passed through to _apply().

**Returns** transformed (type (x)) – The transformed object or array

**apply\_inplace** (*args, **kwargs)
Deprecated as public supported API, use the non-mutating apply() instead.
For internal performance-specific uses, see _apply\_inplace().

**compose\_after** (transform)
Returns a TransformChain that represents this transform composed after the given transform:
\[ c = a.compose_after(b) \]
\[ c.apply(p) == a.apply(b.apply(p)) \]

\[ a \text{ and } b \text{ are left unchanged.} \]

This corresponds to the usual mathematical formalism for the compose operator, \( o \).

**Parameters**

- **transform (Transform)** – Transform to be applied before self

**Returns**

- **transform (TransformChain)** – The resulting transform chain.

\[ \text{compose.before(transform)} \]

Returns a TransformChain that represents this transform composed before the given transform:

\[ c = a.compose_before(b) \]
\[ c.apply(p) == b.apply(a.apply(p)) \]

\[ a \text{ and } b \text{ are left unchanged.} \]

**Parameters**

- **transform (Transform)** – Transform to be applied after self

**Returns**

- **transform (TransformChain)** – The resulting transform chain.

\[ copy() \]

Generate an efficient copy of this object.

Note that Numpy arrays and other Copyable objects on self will be deeply copied. Dictionaries and sets will be shallow copied, and everything else will be assigned (no copy will be made).

Classes that store state other than numpy arrays and immutable types should overwrite this method to ensure all state is copied.

**Returns**

- **type(self)** – A copy of this object

\[ d_{dl}(points) \]

The derivative of the basis function wrt the coordinate system evaluated at points. Let points be \( x \), then \( (x - c)^T(1 + 2 \log r_{x,t}) \), where \( r_{x,t} = \|x - c\| \).

**Parameters**

- **points ((n_points, n_dims) ndarray)** – The spatial points at which the derivative should be evaluated.

**Returns**

- **d_{dl} ((n_points, n_centres, n_dims) ndarray)** – The Jacobian wrt landmark changes.

**property n_centres**

The number of centres.

Type int

**property n_dims**

The RBF can only be applied on points with the same dimensionality as the centres.

Type int

**property n_dims_output**

The result of the transform has a dimension (weight) for every centre.

Type int

### 2.2.11 menpofit.visualize

Print Utilities
print_progress

menpo.visualize.print_progress(iterable, prefix="", n_items=None, offset=0, show_bar=True, show_count=True, show_eta=True, end_with_newline=True, verbose=True)

Print the remaining time needed to compute over an iterable.

To use, wrap an existing iterable with this function before processing in a for loop (see example).

The estimate of the remaining time is based on a moving average of the last 100 items completed in the loop.

This method is identical to menpo.visualize.print_progress, but adds a verbose flag which allows the printing to be skipped if necessary.

Parameters

- **iterable** (iterable) – An iterable that will be processed. The iterable is passed through by this function, with the time taken for each complete iteration logged.
- **prefix** (str, optional) – If provided a string that will be prepended to the progress report at each level.
- **n_items** (int, optional) – Allows for iterator to be a generator whose length will be assumed to be n_items. If not provided, then iterator needs to be Sizable.
- **offset** (int, optional) – Useful in combination with n_items - report back the progress as if offset items have already been handled. n_items will be left unchanged.
- **show_bar** (bool, optional) – If False, The progress bar (e.g. [========= ]) will be hidden.
- **show_count** (bool, optional) – If False, The item count (e.g. (4/25)) will be hidden.
- **show_eta** (bool, optional) – If False, The estimated time to finish (e.g. - 00:00:03 remaining) will be hidden.
- **end_with_newline** (bool, optional) – If False, there will be no new line added at the end of the dynamic printing. This means the next print statement will overwrite the dynamic report presented here. Useful if you want to follow up a print_progress with a second print_progress, where the second overwrites the first on the same line.
- **verbose** (bool, optional) – Printing is performed only if set to True.

Raises ValueError – offset provided without n_items

Examples

This for loop:

```python
from time import sleep
for i in print_progress(range(100)):
    sleep(1)
```

prints a progress report of the form:

```
[========= ] 70% (7/10) - 00:00:03 remaining
```
Errors Visualization

\texttt{statistics_table}

\begin{verbatim}
menpofit.visualize.statistics_table(errors, method_names, auc_max_error,
    auc_error_step, auc_min_error=0.0, stats_types=None,
    stats_names=None, sort_by=None, precision=4)
\end{verbatim}

Function that generates a table with statistical measures on the fitting results of various methods using pandas. It supports multiple types of statistical measures.

Note that the returned object is a pandas table which can be further converted to Latex tabular or simply a string. See the examples for more details.

Parameters

- **errors** (list of list of float) – A list that contains lists of float with the errors per method.
- **method_names** (list of str) – The list with the names that will appear for each method. Note that it must have the same length as \texttt{errors}.
- **auc_max_error** (float) – The maximum error value for computing the area under the curve.
- **auc_error_step** (float) – The sampling step of the error bins for computing the area under the curve.
- **auc_min_error** (float, optional) – The minimum error value for computing the area under the curve.
- **stats_types** (list of str or None, optional) – The types of statistical measures to compute. Possible options are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mean</td>
<td>The mean value of the errors.</td>
</tr>
<tr>
<td>std</td>
<td>The standard deviation of the errors.</td>
</tr>
<tr>
<td>median</td>
<td>The median value of the errors.</td>
</tr>
<tr>
<td>mad</td>
<td>The median absolute deviation of the errors.</td>
</tr>
<tr>
<td>max</td>
<td>The max value of the errors.</td>
</tr>
<tr>
<td>auc</td>
<td>The area under the curve based on the CED of the errors.</td>
</tr>
<tr>
<td>fr</td>
<td>The failure rate (percentage of images that failed).</td>
</tr>
</tbody>
</table>

If None, then all of them will be used with the above order.

- **stats_names** (list of str, optional) – The list with the names that will appear for each statistical measure type selected in \texttt{stats_types}. Note that it must have the same length as \texttt{stats_types}.
- **sort_by** (str or None, optional) – The column to use for sorting the methods. If None, then no sorting is performed and the methods will appear in the provided order of \texttt{method_names}. Possible options are:
<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mean</td>
<td>The mean value of the errors.</td>
</tr>
<tr>
<td>std</td>
<td>The standard deviation of the errors.</td>
</tr>
<tr>
<td>median</td>
<td>The median value of the errors.</td>
</tr>
<tr>
<td>mad</td>
<td>The median absolute deviation of the errors.</td>
</tr>
<tr>
<td>max</td>
<td>The max value of the errors.</td>
</tr>
<tr>
<td>auc</td>
<td>The area under the curve based on the CED of the errors.</td>
</tr>
<tr>
<td>fr</td>
<td>The failure rate (percentage of images that failed).</td>
</tr>
</tbody>
</table>

- **precision** *(int, optional)* – The precision of the reported values, i.e. the number of decimals.

**Raises**

- **ValueError** – stat_type must be selected from [mean, std, median, mad, max, auc, fr]
- **ValueError** – sort_by must be selected from [mean, std, median, mad, max, auc, fr]
- **ValueError** – stats_types and stats_names must have the same length

**Returns**

- **table** *(pandas.DataFrame)* – The pandas table. It can be further converted to various format, such as Latex tabular or str.

**Examples**

Let us create some errors for 3 methods sampled from Normal distributions with different mean and standard deviations:

```python
import numpy as np
from menpofit.visualize import statistics_table

method_names = ['Method_1', 'Method_2', 'Method_3']
errors = [list(np.random.normal(0.07, 0.02, 400)),
          list(np.random.normal(0.06, 0.03, 400)),
          list(np.random.normal(0.08, 0.04, 400))]

tab = statistics_table(errors, method_names, auc_max_error=0.1,
                        auc_error_step=0.001, sort_by='auc')

print(tab.to_string())
print(tab.to_latex())
```

Pandas offers excellent functionalities. For example, the table can be converted to an str as:

```python
print(tab.to_string())
```

or to a Latex tabular as:

```python
print(tab.to_latex())
```
plot_cumulative_error_distribution

menpofit.visualize.plot_cumulative_error_distribution(errors, error_range=None, figure_id=None, new_figure=False, title='Cumulative Error Distribution', x_label='Normalized Point-to-Point Error', y_label='Images Proportion', legend_entries=None, render_lines=True, line_colour=None, line_style='-', line_width=2, render_markers=True, marker_style='s', marker_size=7, marker_face_colour='w', marker_edge_colour=None, marker_edge_width=2, render_legend=True, legend_title=None, legend_font_name='sans-serif', legend_font_style='normal', legend_font_size=10, legend_font_weight='normal', legend_marker_scale=1.0, legend_location=2, legend_bbox_to_anchor=(1.05, 1.0), legend_border_axes_pad=1.0, legend_n_columns=1, legend_horizontal_spacing=1.0, legend_vertical_spacing=1.0, legend_border=True, legend_border_padding=0.5, legend_shadow=False, legend_rounded_corners=False, render_axes=True, axes_font_name='sans-serif', axes_font_size=10, axes_font_style='normal', axes_font_weight='normal', axes_x_limits=None, axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None, figure_size=(7, 7), render_grid=True, grid_line_style='–', grid_line_width=0.5)

Plot the cumulative error distribution (CED) of the provided fitting errors.

Parameters
• **errors** *(list of lists)* – A list with lists of fitting errors. A separate CED curve will be rendered for each errors list.

• **error_range** *(list of float with length 3, optional)* – Specifies the horizontal axis range, i.e.

```python
error_range[0] = min_error
def error_range[1] = max_error
def error_range[2] = error_step
```

If None, then 'error_range = [0., 0.101, 0.005]'.

• **figure_id** *(object, optional)* – The id of the figure to be used.

• **new_figure** *(bool, optional)* – If True, a new figure is created.

• **title** *(str, optional)* – The figure’s title.

• **x_label** *(str, optional)* – The label of the horizontal axis.

• **y_label** *(str, optional)* – The label of the vertical axis.

• **legend_entries** *(list of ‘str or None, optional)* – If list of str, it must have the same length as errors list and each str will be used to name each curve. If None, the CED curves will be named as ‘Curve %d’.

• **render_lines** *(bool or list of bool, optional)* – If True, the line will be rendered. If bool, this value will be used for all curves. If list, a value must be specified for each fitting errors curve, thus it must have the same length as errors.

• **line_colour** *(colour or list of colour or None, optional)* – The colour of the lines. If not a list, this value will be used for all curves. If list, a value must be specified for each curve, thus it must have the same length as y_axis. If None, the colours will be linearly sampled from jet colormap. Example colour options are

```python
('r', 'g', 'b', 'c', 'm', 'k', 'w')
or
(3, ) ndarray
```

• **line_style** *({'-', '--', '-.', ':' } or list of those, optional)* – The style of the lines. If not a list, this value will be used for all curves. If list, a value must be specified for each curve, thus it must have the same length as errors.

• **line_width** *(float or list of float, optional)* – The width of the lines. If float, this value will be used for all curves. If list, a value must be specified for each curve, thus it must have the same length as errors.

• **render_markers** *(bool or list of bool, optional)* – If True, the markers will be rendered. If bool, this value will be used for all curves. If list, a value must be specified for each curve, thus it must have the same length as errors.

• **marker_style** *(marker or list of markers, optional)* – The style of the markers. If not a list, this value will be used for all curves. If list, a value must be specified for each curve, thus it must have the same length as errors. Example marker options

```python
('.', ',', 'o', 'v', '^', '<', '>', '+', 'x', 'D', 'd', 's',
'p', '*', 'h', 'H', '1', '2', '3', '4', '8')
```

• **marker_size** *(int or list of int, optional)* – The size of the markers in points. If int, this value will be used for all curves. If list, a value must be specified for each curve, thus it must have the same length as errors.
• **marker_face_colour** *(colour or list of colour or None, optional)* – The face (filling) colour of the markers. If not a list, this value will be used for all curves. If list, a value must be specified for each curve, thus it must have the same length as `errors`. If None, the colours will be linearly sampled from jet colormap. Example *colour* options are

```python
{'r', 'g', 'b', 'c', 'm', 'k', 'w'}
or
(3, ) ndarray
```

• **marker_edge_colour** *(colour or list of colour or None, optional)* – The edge colour of the markers. If not a list, this value will be used for all curves. If list, a value must be specified for each curve, thus it must have the same length as `errors`. If None, the colours will be linearly sampled from jet colormap. Example *colour* options are

```python
{'r', 'g', 'b', 'c', 'm', 'k', 'w'}
or
(3, ) ndarray
```

• **marker_edge_width** *(float or list of float, optional)* – The width of the markers’ edge. If float, this value will be used for all curves. If list, a value must be specified for each curve, thus it must have the same length as `errors`.

• **render_legend** *(bool, optional)* – If True, the legend will be rendered.

• **legend_title** *(str, optional)* – The title of the legend.

• **legend_font_name** *(See below, optional)* – The font of the legend. Example options

```python
{'serif', 'sans-serif', 'cursive', 'fantasy', 'monospace'}
```

• **legend_font_style** *(See below, optional)* – The font style of the legend.

• **legend_font_size** *(int, optional)* – The font size of the legend.

• **legend_font_weight** *(See below, optional)* – The font weight of the legend. Example options

```python
{'ultralight', 'light', 'normal', 'regular', 'book', 'medium',
'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy',
'extra bold', 'black'}
```

• **legend_marker_scale** *(float, optional)* – The relative size of the legend markers with respect to the original

• **legend_location** *(int, optional)* – The location of the legend. The predefined values are:
MenpoFit Documentation, Release 0.6.1

| 'best'                  | 0 |
| 'upper right'           | 1 |
| 'upper left'            | 2 |
| 'lower left'            | 3 |
| 'lower right'           | 4 |
| 'right'                 | 5 |
| 'center left'           | 6 |
| 'center right'          | 7 |
| 'lower center'          | 8 |
| 'upper center'          | 9 |
| 'center'                | 10 |

- **legend_bbox_to_anchor** ((float, float), optional) – The bbox that the legend will be anchored.
- **legend_border_axes_pad** (float, optional) – The pad between the axes and legend border.
- **legend_n_columns** (int, optional) – The number of the legend’s columns.
- **legend_horizontal_spacing** (float, optional) – The spacing between the columns.
- **legend_vertical_spacing** (float, optional) – The vertical space between the legend entries.
- **legend_border** (bool, optional) – If True, a frame will be drawn around the legend.
- **legend_border_padding** (float, optional) – The fractional whitespace inside the legend border.
- **legend_shadow** (bool, optional) – If True, a shadow will be drawn behind legend.
- **legend_rounded_corners** (bool, optional) – If True, the frame’s corners will be rounded (fancybox).
- **render_axes** (bool, optional) – If True, the axes will be rendered.
- **axes_font_name** *(See below, optional)* – The font of the axes. Example options

```python
{'serif', 'sans-serif', 'cursive', 'fantasy', 'monospace'}
```

- **axes_font_size** (int, optional) – The font size of the axes.
- **axes_font_style** (["normal", 'italic', 'oblique'], optional) – The font style of the axes.
- **axes_font_weight** *(See below, optional)* – The font weight of the axes. Example options

```python
{"ultralight", 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'}
```

- **axes_x_limits** (float or (float, float) or None, optional) – The limits of the x axis. If float, then it sets padding on the right and left of the graph as a percentage of the curves’ width. If tuple or list, then it defines the axis limits. If None, then the limits are set to (0., error_range[1]).
• `axes_y_limits (float or (float, float) or None, optional) – The limits of the y axis. If float, then it sets padding on the top and bottom of the graph as a percentage of the curves' height. If tuple or list, then it defines the axis limits. If None, then the limits are set to (0., 1.).

• `axes_x_ticks (list or tuple or None, optional) – The ticks of the x axis.

• `axes_y_ticks (list or tuple or None, optional) – The ticks of the y axis.

• `figure_size ((float, float) or None, optional) – The size of the figure in inches.

• `render_grid (bool, optional) – If True, the grid will be rendered.

• `grid_line_style ({'--', '-.-', '-.', ':'}, optional) – The style of the grid lines.

• `grid_line_width (float, optional) – The width of the grid lines.

Raises `ValueError – legend_entries list has different length than errors list

Returns `viewer (menpo.visualize.GraphPlotter) – The viewer object.`
INDEX

A

AAM (class in menpofit.aam.base), 11
aam() (menpofit.aam.LucasKanadeAAMFitter property), 32
AAMAlgorithmResult (class in menpofit.aam.result), 72
AAMResult (class in menpofit.aam.result), 54
ActiveShapeModel (class in menpofit.clm), 156
align_shape_with_bounding_box() (in module menpofit.fitter), 321
align_shapes() (in module menpofit.builder), 295
aligned_source() (menpofit.transform.DifferentiableAlignmentAffine method), 438
aligned_source() (menpofit.transform.DifferentiableAlignmentSimilarity method), 433
aligned_source() (menpofit.transform.DifferentiablePiecewiseAffine method), 443
aligned_source() (menpofit.transform.DifferentiableThinPlateSplines method), 447
alignment_error() (menpofit.transform.DifferentiableAlignmentAffine method), 438
alignment_error() (menpofit.transform.DifferentiableAlignmentSimilarity method), 433
alignment_error() (menpofit.transform.DifferentiablePiecewiseAffine method), 443
alignment_error() (menpofit.transform.DifferentiableThinPlateSplines method), 447
AlternatingForwardCompositional (class in menpofit.aam), 36
AlternatingInverseCompositional (class in menpofit.aam), 37
AlternatingRegularisedLandmarkMeanShift (class in menpofit.unified_aam_clm), 169
appearance_costs() (menpofit.aps.result.APSAlgorithmResult property), 133
appearance_costs() (menpofit.aps.result.APSResult property), 115
appearance_model() (menpofit.aam.AlternatingForwardCompositional property), 36
appearance_model() (menpofit.aam.AlternatingInverseCompositional property), 37
appearance_model() (menpofit.aam.AppearanceWeightsGaussNewton property), 47
appearance_model() (menpofit.aam.AppearanceWeightsNewton property), 46
appearance_model() (menpofit.aam.MeanTemplateGaussNewton property), 51
appearance_model() (menpofit.aam.MeanTemplateNewton property), 49
appearance_model() (menpofit.aam.ModifiedAlternatingForwardCompositional property), 38
appearance_model() (menpofit.aam.ModifiedAlternatingInverseCompositional property), 39
appearance_model() (menpofit.aam.ProjectOutForwardCompositional property), 40
appearance_model() (menpofit.aam.ProjectOutGaussNewton property), 54
appearance_model() (menpofit.aam.ProjectOutInverseCompositional property), 41
appearance_model() (menpofit.aam.ProjectOutNewton property), 52
appearance_model() (menpofit.aam.SimultaneousForwardCompositional property), 41
appearance_model() (menpofit.aam.SimultaneousInverseCompositional property), 42
appearance_model() (menpofit.aam.WibergForwardCompositional property), 43
appearance_model() (menpofit.aam.WibergInverseCompositional property), 44
appearance_model() (menpofit.aps.Forward property), 99
appearance_model() (menpofit.aps.Inverse property), 98
appearance_parameters() (menpofit.aam.result.AAMAlgorithmResult property), 88
appearance_parameters() (menpofit.aam.result.AAMResult property), 70
appearance_parameters() (menpofit.unified_aam_clm.result.UnifiedAAMCLMAlgorithmResult property), 204
appearance_parameters() (menpofit.unified_aam_clm.result.UnifiedAAMCLMResult property), 187
appearance_reconstructions() (menpofit.aam.base.AAM method), 12
appearance_reconstructions() (menpofit.aam.LinearAAM method), 20
appearance_reconstructions() (menpofit.aam.LinearMaskedAAM method), 23
appearance_reconstructions() (menpofit.aam.LucasKanadeAAMFitter method), 30
appearance_reconstructions() (menpofit.aam.MaskedAAM method), 16
appearance_reconstructions() (menpofit.aam.PatchAAM method), 27
appearance_reconstructions() (menpofit.unified_aam_clm.base.UnifiedAAMCLM method), 163
appearance_reconstructions() (menpofit.unified_aam_clm.UnifiedAAMCLMFitter method), 167
AppearanceWeightsGaussNewton (class in menpofit.aam), 46
AppearanceWeightsNewton (class in menpofit.aam), 44
apply() (menpofit.transform.DifferentiableAffine method), 423
apply() (menpofit.transform.DifferentiableAlignmentAffine method), 438
apply() (menpofit.transform.DifferentiableAlignmentSimilarity method), 433
apply() (menpofit.transform.DifferentiablePiecewiseAffine method), 444
apply() (menpofit.transform.DifferentiableR2LogR2RBF method), 449
apply() (menpofit.transform.DifferentiableR2LogRRBF method), 451
apply() (menpofit.transform.DifferentiableSimilarity method), 428
apply() (menpofit.transform.LinearOrthoMDTransform method), 420
apply() (menpofit.transform.LinearR2LogR2RBF method), 450
apply() (menpofit.transform.LinearR2LogRRBF method), 451
apply() (menpofit.transform.LinearThinPlateSplines method), 447
apply() (menpofit.transform.LinearOrthoMDTransform method), 420
apply() (menpofit.transform.OrthoMDTransform method), 417
apply_inplace() (menpofit.transform.DifferentiableAffine method), 423
apply_inplace() (menpofit.transform.DifferentiableAlignmentAffine method), 438
apply_inplace() (menpofit.transform.DifferentiableAlignmentSimilarity method), 433
apply_inplace() (menpofit.transform.DifferentiablePiecewiseAffine method), 444
apply_inplace() (menpofit.transform.DifferentiableR2LogR2RBF method), 449
apply_inplace() (menpofit.transform.DifferentiableR2LogRRBF method), 451
apply_inplace() (menpofit.transform.DifferentiableSimilarity method), 428
apply_inplace() (menpofit.transform.LinearOrthoMDTransform method), 420
apply_inplace() (menpofit.transform.LinearThinPlateSplines method), 447
apply_inplace() (menpofit.transform.LinearOrthoMDTransform method), 420
apply_inplace() (menpofit.transform.OrthoMDTransform method), 417
aps() (menpofit.aps.GaussNewtonAPSFitter property), 97
APSAlgorithmResult (class in menpofit.aps.result), 116
area_under_curve_and_failure_rate() (in module menpofit.error), 312
as_non_alignment() (menpofit.transform.DifferentiableAlignmentAffine method), 438
as_non_alignment() (menpofit.transform.DifferentiableAlignmentSimilarity method), 433
as_non_alignment() (menpofit.transform.DifferentiablePiecewiseAffine method), 444

compose_after() (menpofit.transform.DifferentiableSimilarity method), 428
compose_after() (menpofit.transform.DifferentiableThinPlateSplines method), 447
compose_after() (menpofit.transform.LinearOrthoMDTransform method), 420
compose_after() (menpofit.transform.OrthoMDTransform method), 417
compose_after_from_vector_inplace() (menpofit.transform.DifferentiableAffine method), 424
compose_after_from_vector_inplace() (menpofit.transform.DifferentiableAlignmentAffine method), 439
compose_after_from_vector_inplace() (menpofit.transform.DifferentiableAlignmentSimilarity method), 434
compose_after_from_vector_inplace() (menpofit.transform.DifferentiableSimilarity method), 429
compose_after_from_vector_inplace() (menpofit.transform.OrthoMDTransform method), 417
compose_before() (menpofit.transform.DifferentiableAffine method), 424
compose_before() (menpofit.transform.DifferentiableAlignmentAffine method), 439
compose_before() (menpofit.transform.DifferentiableAlignmentSimilarity method), 434
compose_before() (menpofit.transform.DifferentiableSimilarity method), 429
compose_before_inplace() (menpofit.transform.DifferentiableAffine property), 427
compose_before_inplace() (menpofit.transform.DifferentiableAlignmentAffine property), 442
compose_before_inplace() (menpofit.transform.DifferentiableAlignmentSimilarity property), 437
compose_before_inplace() (menpofit.transform.DifferentiableSimilarity property), 432
compute_cumulative_error() (in module menpofit.error), 311
compute_features() (in module menpofit.builder), 450
compute_reference_shape() (in module menpofit.builder), 297
compute_statistical_measures() (in module menpoft.error), 312
copy() (menpoft.modelinstance.GlobalPDM method), 338
copy() (menpoft.modelinstance.GlobalSimilarityModel method), 334

copy() (menpoft.modelinstance.ModelInstance method), 332

copy() (menpoft.modelinstance.OrthoPDM method), 340
copy() (menpoft.modelinstance.PDM method), 335
copy() (menpoft.transform.DifferentiableAffine method), 425
copy() (menpoft.transform.DifferentiableAlignmentAffine method), 440
copy() (menpoft.transform.DifferentiableAlignmentSimilarity method), 435
copy() (menpoft.transform.DifferentiablePiecewiseAffine method), 444
copy() (menpoft.transform.DifferentiableR2LogR2RBF method), 450
copy() (menpoft.transform.DifferentiableR2LogRRBF method), 452
copy() (menpoft.transform.DifferentiableSimilarity method), 430
copy() (menpoft.transform.DifferentiableThinPlateSplines method), 448
copy() (menpoft.transform.LinearOrthoMDTransform method), 421
copy() (menpoft.transform.OrthoMDTransform method), 418

CorrelationFilterExpertEnsemble (class in menpoft.clm), 158

cost_closure() (menpoft.lk.ECC method), 218
cost_closure() (menpoft.lk.FourierSSD method), 216
cost_closure() (menpoft.lk.GradientCorrelation method), 220
cost_closure() (menpoft.lk.GradientImages method), 219
cost_closure() (menpoft.lk.SSD method), 215

costs() (menpoft.aam.result.AAMAlgorithmResult property), 88

costs() (menpoft.aam.result.AAMResult property), 70

costs() (menpoft.aps.result.APSAlgorithmResult property), 133

costs() (menpoft.aps.result.APSResult property), 115
costs() (menpoft.lk.result.LucasKanadeAlgorithmResult property), 255
costs() (menpoft.lk.result.LucasKanadeResult property), 238

costs() (menpoft.result.MultiScaleNonParametricIterativeResult property), 398
costs() (menpoft.result.MultiScaleParametricIterativeResult property), 415
costs() (menpoft.result.NonParametricIterativeResult property), 364
costs() (menpoft.result.ParametricIterativeResult property), 381
costs() (menpoft.unified_aam_clm.result.UnifiedAAMCLMAlgorithmResult property), 204
costs() (menpoft.unified_aam_clm.result.UnifiedAAMCLMResult property), 187

D
d_d1() (menpoft.transform.DifferentiableDL method), 304
d_d1() (menpoft.transform.DifferentiablePiecewiseAffine method), 445
d_d1() (menpoft.transform.DifferentiableR2LogR2RBF method), 450
d_d1() (menpoft.transform.DifferentiableR2LogRRBF method), 452
d_dp() (menpoft.transform.DifferentiableThinPlateSplines method), 448
d_dp() (menpoft.transform.DifferentiableThinPlateSplines method), 448
d_dp() (menpoft.modelinstance.GlobalPDM method), 338
d_dp() (menpoft.modelinstance.GlobalSimilarityModel method), 334
d_dp() (menpoft.modelinstance.ModelInstance method), 332
d_dp() (menpoft.modelinstance.OrthoPDM method), 340
d_dp() (menpoft.modelinstance.PDM method), 336
d_dp() (menpoft.transform.DifferentiableAffine method), 425
d_dp() (menpoft.transform.DifferentiableAlignmentAffine method), 440
d_dp() (menpoft.transform.DifferentiableAlignmentSimilarity method), 435
d_dp() (menpoft.transform.DifferentiableAlignmentAffine method), 440
d_dp() (menpoft.transform.LinearOrthoMDTransform method), 430
d_dp() (menpoft.transform.LinearOrthoMDTransform method), 421
d_dp() (menpoft.transform.OrthoMDTransform method), 418
d_dx() (menpoft.transform.DifferentiableDX method), 305
d_dx() (menpoft.transform.DifferentiableAffine method), 425
d_dx() (menpoft.transform.DifferentiableAlignmentAffine method), 440
d_dx() (menpoft.transform.DifferentiableAlignmentSimilarity method), 435
d_dx() (menpoft.transform.DifferentiablePiecewiseAffine method), 445

Index
MenpoFit Documentation, Release 0.6.1

- `d_dx()` (`menpofit.transform.DifferentiableSimilarity` method), 430
- `d_dx()` (`menpofit.transform.DifferentiableThinPlateSplines` method), 448
- `decompose()` (`menpofit.transform.DifferentiableAffine` method), 425
- `decompose()` (`menpofit.transform.DifferentiableAlignmentAffine` method), 440
- `decompose()` (`menpofit.transform.DifferentiableAlignmentSimilarity` method), 435
- `decompose()` (`menpofit.transform.DifferentiableSimilarity` method), 430
- `deformation_costs()` (`menpofit.aps.result.APSAlgorithmResult` property), 133
- `deformation_costs()` (`menpofit.aps.result.APSResult` property), 115
- `deformation_model()` (`menpofit.aps.Forward` property), 99
- `deformation_model()` (`menpofit.aps.Inverse` property), 98
- `dense_target()` (`menpofit.transform.LinearOrthoMDTransform` property), 422
- `densify_shapes()` (in module `menpofit.builder`), 297
- `DifferentiableAffine` (class in `menpofit.transform`), 423
- `DifferentiableAlignmentAffine` (class in `menpofit.transform`), 438
- `DifferentiableAlignmentSimilarity` (class in `menpofit.transform`), 428
- `DifferentiablePiecewiseAffine` (class in `menpofit.transform`), 433
- `DifferentiableR2LogR2BF` (class in `menpofit.transform`), 449
- `DifferentiableR2LogRRBF` (class in `menpofit.transform`), 451
- `DifferentiableSimilarity` (class in `menpofit.transform`), 428
- `DifferentiableThinPlateSplines` (class in `menpofit.transform`), 446
- `displacements()` (`menpofit.aam.result.AAMAlgorithmResult` method), 72
- `displacements()` (`menpofit.aam.result.AAMResult` method), 55
- `displacements()` (`menpofit.aps.result.APSAlgorithmResult` method), 117
- `displacements()` (`menpofit.aps.result.APSResult` method), 100
- `displacements()` (`menpofit.lk.result.LucasKanadeAlgorithmResult` method), 239
- `displacements()` (`menpofit.lk.result.LucasKanadeResult` method), 222
- `displacements()` (`menpofit.aps.result.APSAlgorithmResult` method), 117
- `displacements()` (`menpofit.aps.result.APSResult` method), 100
- `displacements()` (`menpofit.unified_aam_clm.result.UnifiedAAMCLMAlgorithmResult` method), 189
- `displacements()` (`menpofit.unified_aam_clm.result.UnifiedAAMCLMResult` method), 171
- `displacements_stats()` (`menpofit.aam.result.AAMAlgorithmResult` method), 72
- `displacements_stats()` (`menpofit.aam.result.AAMResult` method), 55
- `displacements_stats()` (`menpofit.aps.result.APSAlgorithmResult` method), 117
- `displacements_stats()` (`menpofit.aps.result.APSResult` method), 100
- `displacements_stats()` (`menpofit.lk.result.LucasKanadeAlgorithmResult` method), 239
- `displacements_stats()` (`menpofit.lk.result.LucasKanadeResult` method), 222
- `displacements_stats()` (`menpofit.result.MultiScaleNonParametricIterativeResult` method), 382
- `displacements_stats()` (`menpofit.result.MultiScaleParametricIterativeResult` method), 399
- `displacements_stats()` (`menpofit.result.NonParametricIterativeResult` method), 349
- `displacements_stats()` (`menpofit.result.ParametricIterativeResult` method), 365
- `displacements_stats()` (`menpofit.result.PartialNonParametricIterativeResult` method), 191

466 Index
MenpoFit Documentation, Release 0.6.1

pofit.unified_aam_clm.result.UnifiedAAMCLMResult

method, 172
distance_two_indices() (in module menpofit.error), 306
DL (class in menpofit.differentiable), 304
DlibERT (class in menpofit.dlib), 206
DlibWrapper (class in menpofit.dlib), 208
DP (class in menpofit.differentiable), 304
DX (class in menpofit.differentiable), 305

E
ECC (class in menpofit.lk), 217
eerrors() (menpofit.aam.result.AAMAlgorithmResult method), 72
eerrors() (menpofit.aam.result.AAMResult method), 55
eerrors() (menpofit.aps.result.APSAlgorithmResult method), 117
eerrors() (menpofit.aps.result.APSResult method), 100
eerrors() (menpofit.lk.result.LucasKanadeAlgorithmResult method), 240
eerrors() (menpofit.lk.result.LucasKanadeResult method), 223
eerrors() (menpofit.result.MultiScaleNonParametricIterativeResult method), 383
eerrors() (menpofit.result.MultiScaleParametricIterativeResult method), 400
eerrors() (menpofit.result.NonParametricIterativeResult method), 349
eerrors() (menpofit.result.ParametricIterativeResult method), 366
eerrors() (menpofit.result.Result method), 342
eerrors() (menpofit.unified_aam_clm.result.UnifiedAAMCLMAlgorithmResult method), 189
eerrors() (menpofit.unified_aam_clm.result.UnifiedAAMCLMResult method), 172
eeuclidean_bb_normalised_error() (in module menpofit.error), 309
eeuclidean_distance_indexed_normalised_error() (in module menpofit.error), 311
eeuclidean_distance_normalised_error() (in module menpofit.error), 310
eeuclidean_error() (in module menpofit.error), 309
extract_patches() (in module menpofit.builder), 297

F
final_error() (menpofit.aam.result.AAMAlgorithmResult method), 73
final_error() (menpofit.aam.result.AAMResult method), 55
final_error() (menpofit.aps.result.APSAlgorithmResult method), 117
final_error() (menpofit.aps.result.APSResult method), 100
final_error() (menpofit.lk.result.LucasKanadeAlgorithmResult method), 240
final_error() (menpofit.lk.result.LucasKanadeResult method), 223
final_error() (menpofit.result.MultiScaleNonParametricIterativeResult method), 383
final_error() (menpofit.result.MultiScaleParametricIterativeResult method), 400
final_error() (menpofit.result.NonParametricIterativeResult method), 349
final_error() (menpofit.result.ParametricIterativeResult method), 366
final_error() (menpofit.result.Result method), 342
final_error() (menpofit.unified_aam_clm.result.UnifiedAAMCLMAlgorithmResult method), 189
final_error() (menpofit.unified_aam_clm.result.UnifiedAAMCLMResult method), 172
final_shape() (menpofit.aam.result.AAMAlgorithmResult property), 88
final_shape() (menpofit.aam.result.AAMResult property), 71
final_shape() (menpofit.aps.result.APSAlgorithmResult property), 133
final_shape() (menpofit.aps.result.APSResult property), 115
final_shape() (menpofit.lk.result.LucasKanadeAlgorithmResult property), 255
final_shape() (menpofit.lk.result.LucasKanadeResult property), 238
final_shape() (menpofit.result.MultiScaleNonParametricIterativeResult property), 398
final_shape() (menpofit.result.MultiScaleParametricIterativeResult property), 415
final_shape() (menpofit.result.NonParametricIterativeResult property), 364
final_shape() (menpofit.result.ParametricIterativeResult property), 342

Index

467
MenpoFit Documentation, Release 0.6.1

final_shape() (menpofit.result.Result property), 348

fit_from_bb() (menpofit.unified_aam_clm.result.UnifiedAAMCLMAlgorithm property), 155

fit_from_shape() (menpofit.dlib.DlibERT method), 208

fit_from_shape() (menpofit.dlib.DlibWrapper method), 209

fit_from_shape() (menpofit.fitter.MultiScaleNonParametricFitter method), 318

fit_from_shape() (menpofit.fitter.MultiScaleParametricFitter method), 319

fit_from_shape() (menpofit.unified_aam_clm.UnifiedAAMCLMResult property), 187

fit_from_shape() (menpofit.unified_aam_clm.UnifiedAAMCLMFitter method), 167

fit_from_shape() (menpofit.unified_aam_clm.UnifiedAAMCLMResult property), 204

frequency_filter_images() (menpofit.clm.CorrelationFilterExpertEnsemble property), 160

from_vector() (menpofit.modelinstance.GlobalSimilarityModel method), 334

from_vector() (menpofit.modelinstance.ModelInstance method), 332

from_vector() (menpofit.modelinstance.OrthoPDM method), 340

from_vector() (menpofit.modelinstance.PDM method), 336

from_vector() (menpofit.transform.DifferentiableAffine method), 425

from_vector() (menpofit.transform.DifferentiableAlignmentAffine method), 440

from_vector() (menpofit.template.MatlabPSD method), 168

Forward (class in menpofit.aps), 98

ForwardAdditive (class in menpofit.atm), 149

ForwardCompositional (class in menpofit.atm), 149

ForwardCompositional (class in menpofit.lk), 212

FourierSSD (class in menpofit.lk), 216

frequency_filter_images() (menpofit.clm.CorrelationFilterExpertEnsemble property), 160

from_vector() (menpofit.modelinstance.GlobalPDM method), 338

from_vector() (menpofit.modelinstance.GlobalSimilarityModel method), 334

from_vector() (menpofit.modelinstance.ModelInstance method), 332

from_vector() (menpofit.modelinstance.OrthoPDM method), 340

from_vector() (menpofit.modelinstance.PDM method), 336

from_vector() (menpofit.transform.DifferentiableAffine method), 425

from_vector() (menpofit.transform.DifferentiableAlignmentAffine method), 440

468 Index
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>from_vector()</td>
<td>(menpo.fit.transform.DifferentiableAlignmentSimilarity method)</td>
<td>433</td>
</tr>
<tr>
<td>from_vector()</td>
<td>(menpo.fit.transform.DifferentiableSimilarity method)</td>
<td>430</td>
</tr>
<tr>
<td>from_vector()</td>
<td>(menpo.fit.transform.LinearOrthoMDTransform method)</td>
<td>421</td>
</tr>
<tr>
<td>from_vector()</td>
<td>(menpo.fit.transform.OrthoMDTransform method)</td>
<td>418</td>
</tr>
<tr>
<td>from_vector_inplace()</td>
<td>(menpo.fit.modelinstance.GlobalPDM method)</td>
<td>338</td>
</tr>
<tr>
<td>from_vector_inplace()</td>
<td>(menpo.fit.modelinstance.GlobalSimilarityModel method)</td>
<td>334</td>
</tr>
<tr>
<td>from_vector_inplace()</td>
<td>(menpo.fit.modelinstance.ModelInstance method)</td>
<td>333</td>
</tr>
<tr>
<td>from_vector_inplace()</td>
<td>(menpo.fit.modelinstance.OrthoPDM method)</td>
<td>340</td>
</tr>
<tr>
<td>from_vector_inplace()</td>
<td>(menpo.fit.modelinstance.PDM method)</td>
<td>336</td>
</tr>
<tr>
<td>from_vector_inplace()</td>
<td>(menpo.fit.transform.DifferentiableAffine method)</td>
<td>426</td>
</tr>
<tr>
<td>from_vector_inplace()</td>
<td>(menpo.fit.transform.DifferentiableAlignmentAffine method)</td>
<td>441</td>
</tr>
<tr>
<td>from_vector_inplace()</td>
<td>(menpo.fit.transform.DifferentiableAlignmentSimilarity method)</td>
<td>435</td>
</tr>
<tr>
<td>from_vector_inplace()</td>
<td>(menpo.fit.transform.DifferentiableSimilarity method)</td>
<td>431</td>
</tr>
<tr>
<td>from_vector_inplace()</td>
<td>(menpo.fit.transform.LinearOrthoMDTransform method)</td>
<td>421</td>
</tr>
<tr>
<td>from_vector_inplace()</td>
<td>(menpo.fit.transform.OrthoMDTransform method)</td>
<td>418</td>
</tr>
<tr>
<td>FullyParametricMeanTemplateNewton</td>
<td>(class in menpo.fit.sdm)</td>
<td>291</td>
</tr>
<tr>
<td>FullyParametricProjectOutGaussNewton</td>
<td>(class in menpo.fit.sdm)</td>
<td>289</td>
</tr>
<tr>
<td>FullyParametricProjectOutNewton</td>
<td>(class in menpo.fit.sdm)</td>
<td>287</td>
</tr>
<tr>
<td>FullyParametricProjectOutOPP</td>
<td>(class in menpo.fit.sdm)</td>
<td>294</td>
</tr>
<tr>
<td>FullyParametricWeightsNewton</td>
<td>(class in menpo.fit.sdm)</td>
<td>292</td>
</tr>
<tr>
<td>GaussNewtonAPSFitter</td>
<td>(class in menpo.fit.aps)</td>
<td>95</td>
</tr>
<tr>
<td>generate_perturbations_from_gt()</td>
<td>(in module menpo.fit.fitter)</td>
<td>321</td>
</tr>
<tr>
<td>GenerativeAPS</td>
<td>(class in menpo.fit.aps)</td>
<td>90</td>
</tr>
<tr>
<td>global_parameters()</td>
<td>(menpo.fit.modelinstance.GlobalPDM property)</td>
<td>339</td>
</tr>
<tr>
<td>global_parameters()</td>
<td>(menpo.fit.modelinstance.OrthoPDM property)</td>
<td>341</td>
</tr>
<tr>
<td>GlobalPDM</td>
<td>(class in menpo.fit.modelinstance)</td>
<td>337</td>
</tr>
<tr>
<td>GlobalSimilarityModel</td>
<td>(class in menpo.fit.modelinstance)</td>
<td>334</td>
</tr>
<tr>
<td>gradient()</td>
<td>(menpo.fit.lk.ECC class method)</td>
<td>218</td>
</tr>
<tr>
<td>gradient()</td>
<td>(menpo.fit.lk.FourierSSD class method)</td>
<td>216</td>
</tr>
<tr>
<td>gradient()</td>
<td>(menpo.fit.lk.GradientCorrelation class method)</td>
<td>220</td>
</tr>
<tr>
<td>gradient()</td>
<td>(menpo.fit.lk.GradientImages class method)</td>
<td>219</td>
</tr>
<tr>
<td>gradient()</td>
<td>(menpo.fit.lk.GradientDescentCLMFitter class method)</td>
<td>154</td>
</tr>
<tr>
<td>GradientImages</td>
<td>(class in menpo.fit.lk)</td>
<td>219</td>
</tr>
<tr>
<td>gt_shape()</td>
<td>(menpo.fit.aam.result.AAMAlgorithmResult property)</td>
<td>88</td>
</tr>
<tr>
<td>gt_shape()</td>
<td>(menpo.fit.aam.result.AAMResult property)</td>
<td>71</td>
</tr>
<tr>
<td>gt_shape()</td>
<td>(menpo.fit.aps.result.APSAlgorithmResult property)</td>
<td>133</td>
</tr>
<tr>
<td>gt_shape()</td>
<td>(menpo.fit.aps.result.APSResult property)</td>
<td>115</td>
</tr>
<tr>
<td>gt_shape()</td>
<td>(menpo.fit.lk.result.LucasKanadeAlgorithmResult property)</td>
<td>255</td>
</tr>
<tr>
<td>gt_shape()</td>
<td>(menpo.fit.lk.result.LucasKanadeResult property)</td>
<td>238</td>
</tr>
<tr>
<td>gt_shape()</td>
<td>(menpo.fit.result.MultiScaleNonParametricIterativeResult property)</td>
<td>398</td>
</tr>
<tr>
<td>gt_shape()</td>
<td>(menpo.fit.result.MultiScaleParametricIterativeResult property)</td>
<td>415</td>
</tr>
<tr>
<td>gt_shape()</td>
<td>(menpo.fit.result.NonParametricIterativeResult property)</td>
<td>364</td>
</tr>
<tr>
<td>gt_shape()</td>
<td>(menpo.fit.result.ParametricIterativeResult property)</td>
<td>381</td>
</tr>
<tr>
<td>gt_shape()</td>
<td>(menpo.fit.result.Result property)</td>
<td>348</td>
</tr>
<tr>
<td>gt_shape()</td>
<td>(menpo.fit.unified_aam_clm.result.UnifiedAAMCLMAccuracyResult property)</td>
<td>205</td>
</tr>
<tr>
<td>gt_shape()</td>
<td>(menpo.fit.unified_aam_clm.result.UnifiedAAMCLMAlgorithmResult property)</td>
<td>187</td>
</tr>
<tr>
<td>Function</td>
<td>Class</td>
<td>Page</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>--------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td><code>h_matrix()</code></td>
<td><code>menpofit.transform.DifferentiableAffine</code></td>
<td>427</td>
</tr>
<tr>
<td><code>h_matrix()</code></td>
<td><code>menpofit.transform.DifferentiableAlignmentAffine</code></td>
<td>442</td>
</tr>
<tr>
<td><code>h_matrix()</code></td>
<td><code>menpofit.transform.DifferentiableAlignmentSimilarity</code></td>
<td>437</td>
</tr>
<tr>
<td><code>h_matrix()</code></td>
<td><code>menpofit.transform.DifferentiableSimilarity</code></td>
<td>432</td>
</tr>
<tr>
<td><code>h_matrix_is_mutable()</code></td>
<td><code>menpofit.transform.DifferentiableAffine</code></td>
<td>427</td>
</tr>
<tr>
<td><code>h_matrix_is_mutable()</code></td>
<td><code>menpofit.transform.DifferentiableAlignmentAffine</code></td>
<td>442</td>
</tr>
<tr>
<td><code>h_matrix_is_mutable()</code></td>
<td><code>menpofit.transform.DifferentiableAlignmentSimilarity</code></td>
<td>437</td>
</tr>
<tr>
<td><code>h_matrix_is_mutable()</code></td>
<td><code>menpofit.transform.DifferentiableSimilarity</code></td>
<td>432</td>
</tr>
<tr>
<td><code>has_nan_values()</code></td>
<td><code>menpofit.modelinstance.GlobalPDM</code></td>
<td>338</td>
</tr>
<tr>
<td><code>has_nan_values()</code></td>
<td><code>menpofit.modelinstance.GlobalSimilarityModel</code></td>
<td>334</td>
</tr>
<tr>
<td><code>has_nan_values()</code></td>
<td><code>menpofit.modelinstance.ModelInstance</code></td>
<td>333</td>
</tr>
<tr>
<td><code>has_nan_values()</code></td>
<td><code>menpofit.modelinstance.OrthoPDM</code></td>
<td>340</td>
</tr>
<tr>
<td><code>has_nan_values()</code></td>
<td><code>menpofit.modelinstance.PDM</code></td>
<td>336</td>
</tr>
<tr>
<td><code>has_nan_values()</code></td>
<td><code>menpofit.transform.DifferentiableAffine</code></td>
<td>426</td>
</tr>
<tr>
<td><code>has_nan_values()</code></td>
<td><code>menpofit.transform.DifferentiableAlignmentAffine</code></td>
<td>441</td>
</tr>
<tr>
<td><code>has_nan_values()</code></td>
<td><code>menpofit.transform.DifferentiableAlignmentSimilarity</code></td>
<td>436</td>
</tr>
<tr>
<td><code>has_nan_values()</code></td>
<td><code>menpofit.transform.DifferentiableSimilarity</code></td>
<td>431</td>
</tr>
<tr>
<td><code>has_nan_values()</code></td>
<td><code>menpofit.transform.LinearOrthoMDTransform</code></td>
<td>421</td>
</tr>
<tr>
<td><code>has_nan_values()</code></td>
<td><code>menpofit.transform.OrthoMDTransform</code></td>
<td>418</td>
</tr>
<tr>
<td><code>has_true_inverse()</code></td>
<td><code>menpofit.transform.DifferentiableAffine</code></td>
<td>427</td>
</tr>
<tr>
<td><code>has_true_inverse()</code></td>
<td><code>menpofit.transform.DifferentiableAlignmentAffine</code></td>
<td>442</td>
</tr>
<tr>
<td><code>has_true_inverse()</code></td>
<td><code>menpofit.transform.DifferentiableAlignmentSimilarity</code></td>
<td>437</td>
</tr>
<tr>
<td><code>has_true_inverse()</code></td>
<td><code>menpofit.transform.DifferentiableSimilarity</code></td>
<td>432</td>
</tr>
<tr>
<td><code>has_true_inverse()</code></td>
<td><code>menpofit.transform.DifferentiablePiecewiseAffine</code></td>
<td>446</td>
</tr>
<tr>
<td><code>has_true_inverse()</code></td>
<td><code>menpofit.transform.DifferentiableThinPlateSplines</code></td>
<td>449</td>
</tr>
<tr>
<td><code>has_true_inverse()</code></td>
<td><code>menpofit.transform.OrthoMDTransform</code></td>
<td>419</td>
</tr>
<tr>
<td><code>hessian()</code></td>
<td><code>menpofit.lk.ECC</code></td>
<td>218</td>
</tr>
<tr>
<td><code>hessian()</code></td>
<td><code>menpofit.lk.FourierSSD</code></td>
<td>216</td>
</tr>
<tr>
<td><code>hessian()</code></td>
<td><code>menpofit.lk.GradientCorrelation</code></td>
<td>221</td>
</tr>
<tr>
<td><code>hessian()</code></td>
<td><code>menpofit.lk.GradientImages</code></td>
<td>219</td>
</tr>
<tr>
<td><code>hessian()</code></td>
<td><code>menpofit.lk.SSD</code></td>
<td>215</td>
</tr>
<tr>
<td><code>holistic_features()</code></td>
<td><code>menpofit.aam.LucasKanadeAAMFitter</code></td>
<td>32</td>
</tr>
<tr>
<td><code>holistic_features()</code></td>
<td><code>menpofit.aam.SupervisedDescentAAMFitter</code></td>
<td>35</td>
</tr>
<tr>
<td><code>holistic_features()</code></td>
<td><code>menpofit.aps.GaussNewtonAPSFitter</code></td>
<td>97</td>
</tr>
<tr>
<td><code>holistic_features()</code></td>
<td><code>menpofit.atm.LucasKanadeATMFitter</code></td>
<td>149</td>
</tr>
<tr>
<td><code>holistic_features()</code></td>
<td><code>menpofit.clm.GradientDescentCLMFitter</code></td>
<td>155</td>
</tr>
<tr>
<td><code>holistic_features()</code></td>
<td><code>menpofit.dlib.DlibERT</code></td>
<td>208</td>
</tr>
<tr>
<td><code>holistic_features()</code></td>
<td><code>menpofit.fitter.MultiScaleNonParametricFitter</code></td>
<td>318</td>
</tr>
<tr>
<td><code>holistic_features()</code></td>
<td><code>menpofit.fitter.MultiScaleParametricFitter</code></td>
<td>320</td>
</tr>
<tr>
<td><code>holistic_features()</code></td>
<td><code>menpofit.lk.LucasKanadeFitter</code></td>
<td>211</td>
</tr>
<tr>
<td><code>holistic_features()</code></td>
<td><code>menpofit.sdm.RegularizedSDM</code></td>
<td>264</td>
</tr>
</tbody>
</table>

holistic_features() (menpo.unified_aam_clm.UnifiedAAMCLMFitter property), 169

homogeneous_parameters() (menpo.lk.result.LucasKanadeAlgorithmResult property), 255

homogeneous_parameters() (menpo.lk.result.LucasKanadeResult property), 238

increment() (menpo.aam.PatchAAM method), 27
increment() (menpo.aam.ProjectOutGaussNewton method), 53
increment() (menpo.aam.ProjectOutNewton method), 51
increment() (menpo.aam.SupervisedDescentAAMFitter method), 35
increment() (menpo.ap.s.GenerativeAPS method), 91
increment() (menpo.atm.base.ATM method), 135
increment() (menpo.atm.LinearATM method), 140
increment() (menpo.atm.LinearMaskedATM method), 143
increment() (menpo.atm.MaskedATM method), 138
increment() (menpo.atm.PatchATM method), 146
increment() (menpo.clm.CL method), 152
increment() (menpo.clm.CorrelationFilterExpertEnsemble method), 158
increment() (menpo.clm.IncrementalCorrelationFilterThinWrapper method), 161
increment() (menpo.math.IRLRegression method), 327
increment() (menpo.math.IRLRegression method), 326
increment() (menpo.math.OPPRegression method), 328
increment() (menpo.math.OPTRegression method), 328
increment() (menpo.math.PCRRegression method), 327
increment() (menpo.model.instance.GlobalPDM method), 338
increment() (menpo.model.instance.OrthoPDM method), 340
increment() (menpo.model.instance.PDM method), 336
increment() (menpo.sdm.FullyParametricMeanTemplateNewton method), 291
increment() (menpo.sdm.FullyParametricProjectOutGaussNewton method), 289
increment() (menpo.sdm.FullyParametricProjectOutNewton method), 288
increment() (menpo.sdm.FullyParametricProjectOutOPP method), 294
increment() (menpo.sdm.FullyParametricWeightsNewton method), 293
increment() (menpo.sdm.NonParametricGaussNewton method), 266
increment() (menpo.sdm.NonParametricNewton method), 265
increment() (menpo.sdm.NonParametricOPPRegression method), 270
increment() (menpo.sdm.NonParametricOptimalRegression method), 270

MenpoFit Documentation, Release 0.6.1
MenpoFit Documentation, Release 0.6.1

increment() (menpofit.sdm.NonParametricPCRRegression method), 269

increment() (menpofit.sdm.ParametricAppearanceMeanTemplateGaussNewton method), 284

increment() (menpofit.sdm.ParametricAppearanceMeanTemplateGaussNewton method), 431

increment() (menpofit.sdm.ParametricAppearanceMeanTemplateGaussNewton method), 282

increment() (menpofit.sdm.ParametricAppearanceMeanTemplateGaussNewton method), 281

increment() (menpofit.sdm.ParametricAppearanceWeightsGaussNewton method), 286

increment() (menpofit.sdm.ParametricAppearanceWeightsNewton method), 285

increment() (menpofit.sdm.ParametricShapeGaussNewton method), 274

increment() (menpofit.sdm.ParametricShapeNewton method), 272

increment() (menpofit.sdm.ParametricShapeOPPRegression method), 278

increment() (menpofit.sdm.ParametricShapeOPPRegression method), 277

increment() (menpofit.sdm.ParametricShapeOPPRegression method), 275

increment() (menpofit.sdm.RegularizedSDM method), 263

increment() (menpofit.sdm.SupervisedDescentFitter method), 259

increment() (menpofit.transform.LinearOrthoMDTransform method), 421

IncrementalCorrelationFilterThinWrapper (class in menpofit.clm), 160

index_alpha_beta() (menpofit.transform.DifferentiablePiecewiseAffine method), 445

init_from_2d_shear() (menpofit.transform.DifferentiableAffine method), 426

init_from_2d_shear() (menpofit.transform.DifferentiableAffine method), 426

init_from_2d_shear() (menpofit.transform.DifferentiableAlignmentAffine method), 441

init_from_2d_shear() (menpofit.transform.DifferentiableAlignmentAffine method), 441

init_from_2d_shear() (menpofit.transform.DifferentiableAlignmentAffine method), 436

init_from_2d_shear() (menpofit.transform.DifferentiableAlignmentAffine method), 436

init_identity() (menpofit.transform.DifferentiableAlignmentSimilarity class method), 436

init_identity() (menpofit.transform.DifferentiableAlignmentSimilarity class method), 436

initial_error() (menpofit.aam.result.AAMAlgorithmResult method), 73

initial_error() (menpofit.aam.result.AAMResult method), 56

initial_error() (menpofit.aps.result.APSAlgorithmResult method), 118

initial_error() (menpofit.aps.result.APSResult method), 100

initial_error() (menpofit.lk.result.LucasKanadeAlgorithmResult method), 240

initial_error() (menpofit.lk.result.LucasKanadeResult method), 223

initial_error() (menpofit.result.MultiScaleNonParametricIterativeResult method), 383

initial_error() (menpofit.result.MultiScaleParametricIterativeResult method), 400

initial_error() (menpofit.result.NonParametricIterativeResult method), 349

initial_error() (menpofit.result.ParametricIterativeResult method), 366

initial_error() (menpofit.result.Result method), 342

initial_shape() (menpofit.unified_aam_clm.result.UnifiedAAMCLMAlgorithmResult method), 189

initial_shape() (menpofit.unified_aam_clm.result.UnifiedAAMCLMAlgorithmResult method), 172

initial_shape() (menpofit.unified_aam_clm.result.UnifiedAAMCLMResult method), 172

initial_shape() (menpofit.unified_aam_clm.result.UnifiedAAMCLMResult method), 189

initial_shape() (menpofit.aam.result.AAMAlgorithmResult property), 88

initial_shape() (menpofit.aam.result.AAMResult property), 71

initial_shape() (menpofit.aps.result.APSAlgorithmResult property), 133

initial_shape() (menpofit.aps.result.APSResult property), 116

initial_shape() (menpofit.lk.result.LucasKanadeAlgorithmResult property), 237
LucasKanadeAlgorithmResult (class in menpofit.lk.result), 239
LucasKanadeATMFitter (class in menpofit.atm), 147
LucasKanadeFitter (class in menpofit.lk), 209
LucasKanadeResult (class in menpofit.lk.result), 222

M
mad() (in module menpofit.error), 312
MaskedAAM (class in menpofit.aam), 15
MaskedATM (class in menpofit.atm), 137
mccf() (in module menpofit.math), 329
mean_pupil_49_error() (in module menpofit.error), 315
mean_pupil_68_error() (in module menpofit.error), 314
MeanTemplateGaussNewton (class in menpofit.aam), 49
MeanTemplateNewton (class in menpofit.aam), 48
MenpoFitBuilderWarning (class in menpofit.builder), 299
MenpoFitModelBuilderWarning (class in menpofit.builder), 299
ModelInstance (class in menpofit.modelinstance), 332
ModifiedAlternatingForwardCompositional (class in menpofit.aam), 38
ModifiedAlternatingInverseCompositional (class in menpofit.aam), 38
mosse() (in module menpofit.math), 330
MultiScaleNonParametricFitter (class in menpofit.fitter), 317
MultiScaleNonParametricIterativeResult (class in menpofit.result), 382
MultiScaleParametricFitter (class in menpofit.fitter), 319
MultiScaleParametricIterativeResult (class in menpofit.result), 399

N
n_active_components() (menpofit.modelinstance.GlobalizationD property), 339
n_active_components() (menpofit.modelinstance.OrthoD property), 341
n_active_components() (menpofit.modelinstance.PDMD property), 337
n_active_components() (menpofit.transform.LinearOrthoMDTransform property), 422
n_centres() (menpofit.transform.DifferentiableR2LogRRBF property), 451
n_dims() (menpofit.modelinstance.GlobalizationD property), 339
n_dims() (menpofit.modelinstance.GlobalizationSimilarityModel property), 334
n_dims() (menpofit.modelinstance.ModelInstance property), 333
n_dims() (menpofit.modelinstance.OrthoD property), 341
n_dims() (menpofit.modelinstance.PD property), 337
n_dims() (menpofit.transform.DifferentiableAffine property), 427
n_dims() (menpofit.transform.DifferentiableAlignmentAffine property), 442
n_dims() (menpofit.transform.DifferentiableAlignmentSimilarity property), 437
n_dims() (menpofit.transform.DifferentiablePiecewiseAffine property), 446
n_dims() (menpofit.transform.DifferentiableR2LogRRBF property), 451
n_dims() (menpofit.transform.DifferentiableR2LogRRBF property), 452
n_dims() (menpofit.transform.DifferentiableSimilarity property), 432
n_dims() (menpofit.transform.DifferentiableThinPlateSplines property), 449
n_dims() (menpofit.transform.LinearOrthoMDTransform property), 422
n_dims() (menpofit.transform.MiscTransform property), 419
n_dims_output() (menpofit.transform.DifferentiableAffine property), 427
n_dims_output() (menpofit.transform.DifferentiableAlignmentAffine property), 442
n_dims_output() (menpofit.transform.DifferentiableAlignmentSimilarity property), 437
n_dims_output() (menpofit.transform.DifferentiablePiecewiseAffine property), 446
n_dims_output() (menpofit.transform.DifferentiableR2LogRRBF property), 451
n_dims_output() (menpofit.transform.DifferentiableR2LogRRBF property), 452
n_dims_output() (menpofit.transform.DifferentiableSimilarity property), 432
n_dims_output() (menpofit.transform.DifferentiableThinPlateSplines property), 449
null
outer_eye_corner_51_euclidean_error() (in module menpofit.error), 316
outer_eye_corner_68_euclidean_error() (in module menpofit.error), 315

P

padded_size() (menpofit.clm.CorrelationFilterExpertEnsemble property), 160
ParametricAppearanceMeanTemplateGuassNewton (class in menpofit.sdm), 283
ParametricAppearanceMeanTemplateNewton (class in menpofit.sdm), 282
ParametricAppearanceProjectOutGuassNewton (class in menpofit.sdm), 281
ParametricAppearanceProjectOutNewton (class in menpofit.sdm), 279
ParametricAppearanceWeightsGuassNewton (class in menpofit.sdm), 286
ParametricAppearanceWeightsNewton (class in menpofit.sdm), 285
ParametricIterativeResult (class in menpofit.result), 365
ParametricShapeGuassNewton (class in menpofit.sdm), 273
ParametricShapeNewton (class in menpofit.sdm), 272
ParametricShapeOPPRegression (class in menpofit.sdm), 278
ParametricShapeOptimalRegression (class in menpofit.sdm), 276
ParametricShapePCRRegression (class in menpofit.sdm), 275
PatchAAM (class in menpofit.aam), 26
PatchATM (class in menpofit.atm), 145
PCRRegression (class in menpofit.math), 327
PDM (class in menpofit.modelinstance), 335
PickleWrappedFitter (class in menpofit.io), 323
plot_costs() (menpofit.aam.result.AAMAlgorithmResult method), 73
plot_costs() (menpofit.aam.result.AAMResult method), 56
plot_costs() (menpofit.aps.result.APSAlgorithmResult method), 118
plot_costs() (menpofit.aps.result.APSResult method), 101
plot Costs() (menpofit.lk.result.LucasKanadeAlgorithmResult method), 240
plot_costs() (menpofit.lk.result.LucasKanadeResult method), 223
plot_costs() (menpofit.result.MultiScaleNonParametricIterativeResult method), 383
plot_costs() (menpofit.result.MultiScaleParametricIterativeResult method), 400
plot_costs() (menpofit.result.NonParametricIterativeResult method), 350
plot_costs() (menpofit.result.ParametricIterativeResult method), 366
plot_costs() (menpofit.unified_aam_clm.result.UnifiedAAMCLMAlgorithmResult method), 190
plot_costs() (menpofit.unified_aam_clm.result.UnifiedAAMCMResult method), 172
plot_cumulative_error_distribution() (in module menpofit.visualize), 456
plot_displacements() (menpofit.aam.result.AMAAlgorithmResult method), 75
plot_displacements() (menpofit.aam.result.AAMResult method), 57
plot_displacements() (menpofit.aps.result.APSAlgorithmResult method), 119
plot_displacements() (menpofit.aps.result.APSResult method), 102
plot_displacements() (menpofit.lk.result.LucasKanadeAlgorithmResult method), 242
plot_displacements() (menpofit.lk.result.LucasKanadeResult method), 225
plot_displacements() (menpofit.result.MultiScaleNonParametricIterativeResult method), 385
plot_displacements() (menpofit.result.MultiScaleParametricIterativeResult method), 402
plot_displacements() (menpofit.result.NonParametricIterativeResult method), 351
plot_displacements() (menpofit.result.ParametricIterativeResult method), 368
plot_displacements() (menpofit.unified_aam_clm.result.UnifiedAAMCLMAlgorithmResult method), 191
plot_displacements() (menpofit.unified_aam_clm.result.UnifiedAAMCMResult method), 174
plot_errors() (menpofit.aam.result.AAMAlgorithmResult method), 76
plot_errors() (menpofit.aam.result.AAMResult method), 59
plot_errors() (menpofit.aps.result.APSAlgorithmResult method), 121
plot_errors() (menpofit.aps.result.APSResult method), 104
plot_errors() (menpofit.lk.result.LucasKanadeAlgorithmResult method), 244
plot_errors() (menpofit.lk.result.LucasKanadeResult method), 226
plot_errors() (menpofit.result.MultiScaleNonParametricIterativeResult method), 386
plot_errors() (menpofit.result.MultiScaleParametricIterativeResult method), 403
plot_errors() (menpofit.result.NonParametricIterativeResult method), 353
plot_errors() (menpofit.result.ParametricIterativeResult method), 369
plot_errors() (menpofit.unified_aam_clm.result.UnifiedAAMCLMAlgotithmResult method), 193
plot_errors() (menpofit.unified_aam_clm.result.UnifiedAAMCLMResult method), 176
predict() (menpofit.math.IIRLRegression method), 327
predict() (menpofit.math.IRLRegression method), 326
predict() (menpofit.math.OPPRegression method), 329
predict() (menpofit.math.OptimalLinearRegression method), 328
predict() (menpofit.math.PCRRegression method), 327
predict_probability() (menpofit.clm.CorrelationFilterExpertEnsemble method), 159
predict_response() (menpofit.clm.CorrelationFilterExpertEnsemble method), 159
print_progress() (in module menpofit.visualize), 453
project() (menpofit.aam.AppearanceWeightsGaussNewton method), 47
project() (menpofit.aam.AppearanceWeightsNewton method), 45
project_out() (menpofit.aam.ProjectOutForwardCompositional method), 39
project_out() (menpofit.aam.ProjectOutGaussNewton method), 53
project_out() (menpofit.aam.ProjectOutInverseCompositional method), 40
project_out() (menpofit.aam.ProjectOutNewton method), 52
ProjectOutForwardCompositional (class in menpofit), 39
ProjectOutGaussNewton (class in menpofit.aam), 53
ProjectOutInverseCompositional (class in menpofit), 40
ProjectOutNewton (class in menpofit.aam), 51
ProjectOutRegularisedLandmarkMeanShift (class in menpofit.unified_aam_clm), 170
pseudoinverse() (menpofit.transform.DifferentiableAffine method), 426
pseudoinverse() (menpofit.transform.DifferentiableAlignmentAffine method), 441
pseudoinverse() (menpofit.transform.DifferentiableAlignmentSimilarity method), 436
pseudoinverse() (menpofit.transform.DifferentiablePiecewiseAffine method), 445
pseudoinverse() (menpofit.transform.DifferentiableAffine method), 426
pseudoinverse() (menpofit.transform.DifferentiableSimilarity method), 431
pseudoinverse() (menpofit.transform.DifferentiableThinPlateSplines method), 448
pseudoinverse() (menpofit.transform.OrthoMDTransform method), 418
reconstructed_initial_error() (menpofit.unified_aam_clm.result.UnifiedAAMCLMAlgorithmResult method), 195
reconstructed_initial_error() (menpofit.unified_aam_clm.result.UnifiedAAMCLMResult method), 177
reconstructed_initial_shape() (menpofit.aam.result.AAMAlgorithmResult property), 88
reconstructed_initial_shape() (menpofit.aam.result.AAMResult property), 134
reconstructed_initial_shape() (menpofit.lk.result.LucasKanadeAlgorithmResult property), 256
reconstructed_initial_shape() (menpofit.lk.result.LucasKanadeResult property), 239
reconstructed_initial_shape() (menpofit.result.MultiScaleParametricIterativeResult property), 416
reconstructed_initial_shapes() (menpofit.unified_aam_clm.result.UnifiedAAMCLMAlgorithmResult property), 205
reconstructed_initial_shapes() (menpofit.unified_aam_clm.result.UnifiedAAMCLMResult property), 195
reconstructed_initial_shapes() (menpofit.unified_aam_clm.result.UnifiedAAMCLMResult property), 177
reconstructed_initial_shapec() (menpofit.aam.base.AAM method), 13
reconstructed_initial_shapec() (menpofit.aam.LinearAAM method), 21
reconstructed_initial_shapec() (menpofit.aam.LinearMaskedAAM method), 24
reconstructed_initial_shapec() (menpofit.aam.MaskedAAM method), 17
reconstructed_initial_shapec() (menpofit.aam.PatchAAM method), 28
reconstructed_initial_shapec() (menpofit.atm.base.ATM method), 136
reconstructed_initial_shapec() (menpofit.atm.LinearATM method), 141
reconstructed_initial_shapec() (menpofit.atm.LinearMaskedATM method), 144
reconstructed_initial_shapec() (menpofit.atm.MaskedATM method), 139
reconstructed_initial_shapec() (menpofit.atm.PatchATM method), 146
reconstructed_initial_shapec() (menpofit.unified_aam_clm.base.UnifiedAAMCLM method), 164
reconstructed_initial_error() (menpofit.aam.result.AAMAlgorithmResult method), 78
reconstructed_initial_error() (menpofit.aam.result.AAMResult method), 61
reconstructed_initial_error() (menpofit.aps.result.APSAlgorithmResult method), 123
reconstructed_initial_error() (menpofit.aps.result.APSResult method), 105
reconstructed_initial_error() (menpofit.lk.result.LucasKanadeAlgorithmResult method), 245
reconstructed_initial_error() (menpofit.lk.result.LucasKanadeResult method), 228
reconstructed_initial_error() (menpofit.result.MultiScaleParametricIterativeResult method), 405
reconstructed_initial_error() (menpofit.result.ParametricIterativeResult method), 371
reconstructed_initial_shape() (menpofit.aam.LucasKanadeAAMFitter property), 32
reconstructed_initial_shape() (menpofit.aam.SupervisedDescentAAMFitter property), 36
reconstructed_initial_shape() (menpofit.aps.GaussNewtonAPSFitter property), 97
reconstructed_initial_shape() (menpofit.atm.LucasKanadeATMFitter property), 149
reconstructed_initial_shape() (menpofit.clm.GradientDescentCLMFitter property), 156
reconstructed_initial_shape() (menpofit.fitter.MultiScaleNonParametricFitter property), 208
reconstructed_initial_shape() (menpofit.fitter.MultiScaleNonParametricFitter property), 318

random_instance() (menpofit.aam.base.AAM method), 17
random_instance() (menpofit.aam.LinearAAM method), 21
random_instance() (menpofit.aam.LinearMaskedAAM method), 24
random_instance() (menpofit.aam.MaskedAAM method), 17
random_instance() (menpofit.aam.PatchAAM method), 28
random_instance() (menpofit.aps.GenerativeAPS method), 92
random_instance() (menpofit.atm.base.ATM method), 136
random_instance() (menpofit.atm.LinearATM method), 141
random_instance() (menpofit.atm.LinearMaskedATM method), 144
random_instance() (menpofit.atm.MaskedATM method), 139
random_instance() (menpofit.atm.PatchATM method), 146
random_instance() (menpofit.unified_aam_clm.base.UnifiedAAMCLM method), 164

reconstructed_initial_error() (menpofit.aps.result.APSAlgorithmResult method), 123
reconstructed_initial_error() (menpofit.aps.result.APSResult method), 105
reconstructed_initial_error() (menpofit.lk.result.LucasKanadeAlgorithmResult method), 245
reconstructed_initial_error() (menpofit.lk.result.LucasKanadeResult method), 228
reconstructed_initial_error() (menpofit.result.MultiScaleParametricIterativeResult method), 405
reconstructed_initial_error() (menpofit.result.ParametricIterativeResult method), 371

reconstructed_initial_shape() (menpofit.aam.LucasKanadeAAMFitter property), 32
reconstructed_initial_shape() (menpofit.aam.SupervisedDescentAAMFitter property), 36
reconstructed_initial_shape() (menpofit.aps.GaussNewtonAPSFitter property), 97
reconstructed_initial_shape() (menpofit.atm.LucasKanadeATMFitter property), 149
reconstructed_initial_shape() (menpofit.clm.GradientDescentCLMFitter property), 156
reconstructed_initial_shape() (menpofit.fitter.MultiScaleNonParametricFitter property), 208
reconstructed_initial_shape() (menpofit.fitter.MultiScaleNonParametricFitter property), 318

reconstructed_initial_shape() (menpofit.aam.base.AAM method), 13
reconstructed_initial_shape() (menpofit.aam.LinearAAM method), 21
reconstructed_initial_shape() (menpofit.aam.LinearMaskedAAM method), 24
random_instance() (menpofit.aam.base.AAM method), 13
random_instance() (menpofit.aam.LinearAAM method), 21
random_instance() (menpofit.aam.LinearMaskedAAM method), 24
random_instance() (menpofit.aam.MaskedAAM method), 17
random_instance() (menpofit.aam.PatchAAM method), 28
random_instance() (menpofit.aps.GenerativeAPS method), 92
random_instance() (menpofit.atm.base.ATM method), 136
random_instance() (menpofit.atm.LinearATM method), 141
random_instance() (menpofit.atm.LinearMaskedATM method), 144
random_instance() (menpofit.atm.MaskedATM method), 139
random_instance() (menpofit.atm.PatchATM method), 146
random_instance() (menpofit.unified_aam_clm.base.UnifiedAAMCLM method), 164

reconstructed_initial_error() (menpofit.aps.result.APSAlgorithmResult method), 123
reconstructed_initial_error() (menpofit.aps.result.APSResult method), 105
reconstructed_initial_error() (menpofit.lk.result.LucasKanadeAlgorithmResult method), 245
reconstructed_initial_error() (menpofit.lk.result.LucasKanadeResult method), 228
reconstructed_initial_error() (menpofit.result.MultiScaleParametricIterativeResult method), 405
reconstructed_initial_error() (menpofit.result.ParametricIterativeResult method), 371

reconstructed_initial_shape() (menpofit.aam.LucasKanadeAAMFitter property), 32
reconstructed_initial_shape() (menpofit.aam.SupervisedDescentAAMFitter property), 36
reconstructed_initial_shape() (menpofit.aps.GaussNewtonAPSFitter property), 97
reconstructed_initial_shape() (menpofit.atm.LucasKanadeATMFitter property), 149
reconstructed_initial_shape() (menpofit.clm.GradientDescentCLMFitter property), 156
reconstructed_initial_shape() (menpofit.fitter.MultiScaleNonParametricFitter property), 208
reconstructed_initial_shape() (menpofit.fitter.MultiScaleNonParametricFitter property), 318
reference_shape() (menpofit.builder.MultiScaleParametricFitter property), 320
reference_shape() (menpofit.fitter.MultiScaleParametricFitter property), run() (menpofit.aam.SimultaneousInverseCompositional method), 41
reference_shape() (menpofit.fitter.MultiScaleParametricFitter property), run() (menpofit.aam.WibergForwardCompositional method), 42
reference_shape() (menpofit.fitter.MultiScaleParametricFitter property), run() (menpofit.aam.WibergInverseCompositional method), 43
reference_shape() (menpofit.sdm.RegularizedSDM property), 264
reference_shape() (menpofit.sdm.SupervisedDescentFitter property), 260
reference_shape() (menpofit.unified_aam_clm.UnifiedAAMCLMFitter property), 169
RegularisedLandmarkMeanShift (class in menpofit.clm), 157
RegularizedSDM (class in menpofit.sdm), 261
rescale_images_to_reference_shape() (in module menpofit.builder), 298
response_covariance() (menpofit.unified_aam_clm.UnifiedAAMCLMFitter property), 169
Result (class in menpofit.result), 342
root_mean_square_bb_normalised_error() (in module menpofit.error), 307
root_mean_square_distance_indexed_normalised_error() (in module menpofit.error), 308
root_mean_square_distance_normalised_error() (in module menpofit.error), 308
root_mean_square_error() (in module menpofit.error), 307
run() (menpofit.aam.AlternatingForwardCompositional method), 36
run() (menpofit.aam.AlternatingInverseCompositional method), 37
run() (menpofit.aam.AppearanceWeightsGaussNewton method), 47
run() (menpofit.aam.AppearanceWeightsNewton method), 45
run() (menpofit.aam.MeanTemplateGaussNewton method), 50
run() (menpofit.aam.MeanTemplateNewton method), 48
run() (menpofit.aam.ModifiedAlternatingForwardCompositional method), 38
run() (menpofit.aam.ModifiedAlternatingInverseCompositional method), 38
run() (menpofit.aam.ProjectOutForwardCompositional method), 39
run() (menpofit.aam.ProjectOutGaussNewtont method), 54
run() (menpofit.aam.ProjectOutInverseCompositional method), 40
run() (menpofit.aam.ProjectOutNewton method), 52
run() (menpofit.aam.SimultaneousForwardCompositional method), run() (menpofit.aam.SimultaneousInverseCompositional method), 41
run() (menpofit.aam.WibergForwardCompositional method), 42
run() (menpofit.aam.WibergInverseCompositional method), 43
run() (menpofit.aps.Forward method), 98
run() (menpofit.aps.Inverse method), 97
run() (menpofit.atm.ForwardCompositional method), 149
run() (menpofit.atm.InverseCompositional method), 150
run() (menpofit.clm.ActiveShapeModel method), 156
run() (menpofit.clm.RegularisedLandmarkMeanShift method), 157
run() (menpofit.lk.ForwardAdditive method), 212
run() (menpofit.lk.ForwardCompositional method), 213
run() (menpofit.lk.InverseCompositional method), 214
run() (menpofit.sdm.FullyParametricMeanTemplateNewton method), 291
run() (menpofit.sdm.FullyParametricProjectOutGaussNewton method), 290
run() (menpofit.sdm.FullyParametricProjectOutNewton method), 288
run() (menpofit.sdm.FullyParametricProjectOutOPP method), 295
run() (menpofit.sdm.FullyParametricWeightsNewton method), 293
run() (menpofit.sdm.NonParametricGaussNewton method), 266
run() (menpofit.sdm.NonParametricNewton method), 265
run() (menpofit.sdm.NonParametricOPPRegression method), 271
run() (menpofit.sdm.NonParametricOptimalRegression method), 269
run() (menpofit.sdm.NonParametricPCRRRegression method), 268
run() (menpofit.sdm.ParametricAppearanceMeanTemplateGaussNewton method), 284
run() (menpofit.sdm.ParametricAppearanceMeanTemplateNewton method), 282
run() (menpofit.sdm.ParametricAppearanceProjectOutGaussNewton method), 281
run() (menpofit.sdm.ParametricAppearanceProjectOutNewton method), 280
run() (menpofit.sdm.ParametricAppearanceWeightsGaussNewton method), 286
run() (menpofit.sdm.ParametricAppearanceWeightsNewton method), 285
run() (menpofit.sdm.ParametricShapeGaussNewton method), 274
### MenpoFit Documentation, Release 0.6.1

#### 272

- `run()` (MenpoFit.sdm.ParametricShapeOPPRegression method), 278
- `run()` (MenpoFit.sdm.ParametricShapeOptimalRegression method), 277
- `run()` (MenpoFit.sdm.ParametricShapePCRRegression method), 275
- `run()` (MenpoFit.unified_aam_clm.AlternatingRegularisedLandmarkMeanShift method), 169
- `run()` (MenpoFit.unified_aam_clm.ProjectOutRegularisedLandmarkMeanShift method), 170

#### S

- `scale_images()` (in module MenpoFit.builder), 298
- `scales()` (MenpoFit.aam.LucasKanadeAAMFitter property), 32
- `scales()` (MenpoFit.aam.SupervisedDescentAAMFitter property), 36
- `scales()` (MenpoFit.aps.GaussNewtonAPSFitter property), 97
- `scales()` (MenpoFit.atm.LucasKanadeATMFitter property), 149
- `scales()` (MenpoFit.clm.GradientDescentCLMFitter property), 156
- `scales()` (MenpoFit.dlib.DlibERT property), 208
- `scales()` (MenpoFit.fitter.MultiScaleNonParametricFitter property), 318
- `scales()` (MenpoFit.fitter.MultiScaleParametricFitter property), 320
- `scales()` (MenpoFit.lk.LucasKanadeFitter property), 211
- `scales()` (MenpoFit.sdmm.RegularizedSDM property), 264
- `scales()` (MenpoFit.sdmm.SupervisedDescentFitter property), 260
- `scales()` (MenpoFit.unified_aam_clm.UnifiedAAMCLMFitter property), 169
- `SDM()` (in module MenpoFit.sdmm), 260
- `search_shape()` (MenpoFit.clm.CorrelationFilterExpertEnsemble property), 160
- `set_h_matrix()` (MenpoFit.transform.DifferentiableAffine method), 426
- `set_h_matrix()` (MenpoFit.transform.DifferentiableAlignmentAffine method), 441
- `set_h_matrix()` (MenpoFit.transform.DifferentiableAlignmentSimilarity method), 436
- `set_h_matrix()` (MenpoFit.transform.DifferentiableSimilarity method), 431
- `set_models_components()` (in module MenpoFit.checks), 303
- `set_target()` (MenpoFit.modelinstance.GlobalPDM method), 339
- `set_target()` (MenpoFit.modelinstance.GlobalSimilarityModel method), 334
- `set_target()` (MenpoFit.modelinstance.ModelInstance method), 333
- `set_target()` (MenpoFit.modelinstance.OrthoPDM method), 341
- `set_target()` (MenpoFit.modelinstance.PDM method), 336
- `set_target()` (MenpoFit.transform.DifferentiableAlignmentAffine method), 442
- `set_target()` (MenpoFit.transform.DifferentiableAlignmentSimilarity method), 436
- `set_target()` (MenpoFit.transform.DifferentiablePiecewiseAffine method), 446
- `set_target()` (MenpoFit.transform.DifferentiableThinPlateSplines method), 448
- `set_target()` (MenpoFit.transform.LinearOrthoMDTransform method), 422
- `set_target()` (MenpoFit.transform.OrthoMDTransform method), 419
- `shape_instance()` (MenpoFit.clm.CLM method), 153
- `shape_instance()` (MenpoFit.unified_aam_clm.base.UnifiedAAMCLM method), 164
- `shape_parameters()` (MenpoFit.aam.result.AAMAlgorithmResult property), 89
- `shape_parameters()` (MenpoFit.aam.result.AAMResult property), 71
- `shape_parameters()` (MenpoFit.aps.result.APSAlgorithmResult property), 134
- `shape_parameters()` (MenpoFit.aps.result.APSSolutionResult property), 116
- `shape_parameters()` (MenpoFit.lk.result.LucasKanadeAlgorithmResult property), 256
- `shape_parameters()` (MenpoFit.lk.result.LucasKanadeResult property), 239
- `shape_parameters()` (MenpoFit...
MenpoFit Documentation, Release 0.6.1

pofit.result.MultiScaleParametricIterativeResult

shape_parameters() (menpofit.result.ParametricIterativeResult property), 381

shape_parameters() (menpofit.unified_aam_clm.result.UnifiedAAMCLMAlgorithmResult property), 205

shape_parameters() (menpofit.unified_aam_clm.result.UnifiedAAMCLMResult property), 188

shapes() (menpofit.aam.result.AAMAlgorithmResult property), 89

shapes() (menpofit.aam.result.AAMResult property), 71

shapes() (menpofit.aps.result.APSAlgorithmResult property), 134

shapes() (menpofit.aps.result.APSResult property), 116

shapes() (menpofit.lk.result.LucasKanadeAlgorithmResult property), 256

shapes() (menpofit.lk.result.LucasKanadeResult property), 239

shapes() (menpofit.result.MultiScaleNonParametricIterativeResult property), 399

shapes() (menpofit.result.MultiScaleParametricIterativeResult property), 416

shapes() (menpofit.result.NonParametricIterativeResult property), 365

shapes() (menpofit.result.ParametricIterativeResult property), 382

shapes() (menpofit.unified_aam_clm.result.UnifiedAAMCLMAlgorithmResult property), 205

shapes() (menpofit.unified_aam_clm.result.UnifiedAAMCLMResult property), 188

similarity_2d_instance_model() (in module menpofit.modelinstance), 333

SimultaneousForwardCompositional (class in menpofit.aam), 41

SimultaneousInverseCompositional (class in menpofit.aam), 42

source() (menpofit.transform.DifferentiableAlignmentAffine property), 443

source() (menpofit.transform.DifferentiableAlignmentSimilarity property), 437

source() (menpofit.transform.DifferentiablePiecewiseAffine property), 446

source() (menpofit.transform.DifferentiableThinPlateSplines property), 449

sparse_target() (menpofit.transform.LinearOrthoMDTransform property), 423

spatial_filter_images() (menpofit.clm.CorrelationFilterExpertEnsemble method), 218

SSD (class in menpofit.lk), 215

statistics_table() (in module menpofit.visualize), 454

steepest_descent_images() (menpofit.lk.ECC method), 218

steepest_descent_images() (menpofit.lk.FourierSSD method), 217

steepest_descent_images() (menpofit.lk.GradientCorrelation method), 221

steepest_descent_images() (menpofit.lk.GradientImages method), 220

steepest_descent_images() (menpofit.lk.SSD method), 215

steepest_descent_update() (menpofit.lk.ECC method), 218

steepest_descent_update() (menpofit.lk.FourierSSD method), 217

steepest_descent_update() (menpofit.lk.GradientCorrelation method), 221

steepest_descent_update() (menpofit.lk.GradientImages method), 220

steepest_descent_update() (menpofit.lk.SSD method), 216

SupervisedDescentAAMFitter (class in menpofit.aam), 33

SupervisedDescentFitter (class in menpofit.sdm), 256

T

target() (menpofit.modelinstance.GlobalPDM property), 337

target() (menpofit.modelinstance.GlobalSimilarityModel property), 335

target() (menpofit.modelinstance.ModelInstance property), 333

target() (menpofit.modelinstance.OrthoPDM property), 342

target() (menpofit.modelinstance.PDM property), 337

target() (menpofit.transform.DifferentiableAlignmentAffine property), 443

target() (menpofit.transform.DifferentiableAlignmentSimilarity property), 438

target() (menpofit.transform.DifferentiablePiecewiseAffine property), 446

target() (menpofit.transform.DifferentiableThinPlateSplines property), 449

target() (menpofit.transform.LinearOrthoMDTransform property), 423

target() (menpofit.transform.OrthoMDTransform property), 419
template() (menpofit.aam.AlternatingInverseCompositional property), 37
template() (menpofit.aam.ModifiedAlternatingForwardCompositional property), 38
template() (menpofit.aam.ModifiedAlternatingInverseCompositional property), 39
template() (menpofit.aam.ProjectOutForwardCompositional property), 40
template() (menpofit.aam.ProjectOutInverseCompositional property), 41
template() (menpofit.aam.SimultaneousForwardCompositional property), 42
template() (menpofit.aam.SimultaneousInverseCompositional property), 42
template() (menpofit.aam.WibergForwardCompositional property), 43
template() (menpofit.aam.WibergInverseCompositional property), 44
to_result() (menpofit.aam.result.AAMAlgorithmResult method), 78
to_result() (menpofit.aam.result.AAMResult method), 61
to_result() (menpofit.aps.result.APSAlgorithmResult method), 123
to_result() (menpofit.aps.result.APSResult method), 106
to_result() (menpofit.lk.result.LucasKanadeAlgorithmResult method), 245
to_result() (menpofit.lk.result.LucasKanadeResult method), 228
to_result() (menpofit.result.MultiScaleNonParametricIterativeResult method), 388
to_result() (menpofit.result.MultiScaleParametricIterativeResult method), 405
to_result() (menpofit.result.NonParametricIterativeResult method), 354
to_result() (menpofit.result.ParametricIterativeResult method), 371
to_result() (menpofit.unified_aam.clm.result.UnifiedAAMCLMAlgorithmResult method), 195
to_result() (menpofit.unified_aam.clm.result.UnifiedAAMCLMResult method), 177
train() (menpofit.aam.AppearanceWeightsGaussNewton method), 47
train() (menpofit.aam.AppearanceWeightsNewton method), 45
train() (menpofit.aam.MeanTemplateGaussNewton method), 50
train() (menpofit.aam.MeanTemplateNewton method), 49
train() (menpofit.aam.ProjectOutGaussNewton method), 52
train() (menpofit.clm.IncrementalCorrelationFilterThinWrapper method), 161
train() (menpofit.math.IIRLRegression method), 327
train() (menpofit.math.IRLRegression method), 326
train() (menpofit.math.OPTRegression method), 329
train() (menpofit.math.OPTRegression method), 328
train() (menpofit.math.PCRRegression method), 327
train() (menpofit.math.PCRRegression method), 326
train() (menpofit.math.PCRRegression method), 292
train() (menpofit.sdm.FullyParametricMeanTemplateNewton method), 290
train() (menpofit.sdm.FullyParametricProjectOutGaussNewton method), 290
train() (menpofit.sdm.FullyParametricProjectOutNewton method), 288
train() (menpofit.sdm.FullyParametricProjectOutOPP method), 295
train() (menpofit.sdm.FullyParametricWeightsNewton method), 293
train() (menpofit.sdm.NonParametricGaussNewton method), 267
train() (menpofit.sdm.NonParametricNewton method), 265
train() (menpofit.sdm.NonParametricOPPRegression method), 271
train() (menpofit.sdm.NonParametricOptimalRegression method), 270
train() (menpofit.sdm.NonParametricPCRRegression method), 268
train() (menpofit.sdm.ParametricAppearanceMeanTemplateGuassNewton method), 284
train() (menpofit.sdm.ParametricAppearanceMeanTemplateNewton method), 284
train() (menpofit.sdm.ParametricAppearanceProjectOutGuassNewton method), 282
train() (menpofit.sdm.ParametricAppearanceProjectOutNewton method), 280
train() (menpofit.sdm.ParametricAppearanceWeightsGuassNewton method), 287
train() (menpofit.sdm.ParametricAppearanceWeightsNewton method), 287
train() (menpofit.sdm.ParametricShapeGaussNewton method), 273
train() (menpofit.sdm.ParametricShapeOPPRegression method), 279
train() (menpofit.sdm.ParametricShapeOptimalRegression method), 277
train() (menpofit.sdm.ParametricShapePCRRegression method), 275
<table>
<thead>
<tr>
<th>Function/Method</th>
<th>Module/Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>view_widget()</td>
<td>menpofit.aps.result.APSAlgorithmResult method, 70</td>
</tr>
<tr>
<td>view_widget()</td>
<td>menpofit.aps.result.APSResult method, 115</td>
</tr>
<tr>
<td>view_widget()</td>
<td>menpofit.lk.result.LucasKanadeAlgorithmResult method, 255</td>
</tr>
<tr>
<td>view_widget()</td>
<td>menpofit.lk.result.LucasKanadeResult method, 238</td>
</tr>
<tr>
<td>view_widget()</td>
<td>menpofit.result.MultiScaleNonParametricIterativeResult method, 398</td>
</tr>
<tr>
<td>view_widget()</td>
<td>menpofit.result.MultiScaleParametricIterativeResult method, 415</td>
</tr>
<tr>
<td>view_widget()</td>
<td>menpofit.result.NonParametricIterativeResult method, 364</td>
</tr>
<tr>
<td>view_widget()</td>
<td>menpofit.result.ParametricIterativeResult method, 381</td>
</tr>
<tr>
<td>view_widget()</td>
<td>menpofit.unified_aam_clm.result.UnifiedAAMCLMFitter method, 168</td>
</tr>
<tr>
<td>view_widget()</td>
<td>menpofit.unified_aam_clm.result.UnifiedAAMCLMResult method, 187</td>
</tr>
<tr>
<td>weights()</td>
<td>menpofit.modelinstance.GlobalPDM property, 339</td>
</tr>
<tr>
<td>weights()</td>
<td>menpofit.modelinstance.GlobalSimilarityModel property, 335</td>
</tr>
<tr>
<td>weights()</td>
<td>menpofit.modelinstance.ModelInstance property, 333</td>
</tr>
<tr>
<td>weights()</td>
<td>menpofit.modelinstance.OrthoPDM property, 342</td>
</tr>
<tr>
<td>warp_images()</td>
<td>menpofit.lk.LucasKanadeFitter method, 211</td>
</tr>
<tr>
<td>warp_images()</td>
<td>menpofit.unified_aam_clm.UnifiedAAMCLMFitter method, 168</td>
</tr>
<tr>
<td>warped_images()</td>
<td>menpofit.lk.LucasKanadeFitter method, 211</td>
</tr>
<tr>
<td>warped_images()</td>
<td>menpofit.unified_aam_clm.UnifiedAAMCLMFitter method, 168</td>
</tr>
<tr>
<td>warped_images()</td>
<td>menpofit.builder.MenpoFitBuilderWarning method, 299</td>
</tr>
<tr>
<td>warped_images()</td>
<td>menpofit.modelinstance.PDM property, 337</td>
</tr>
<tr>
<td>warped_images()</td>
<td>menpofit.transform.LinearOrthoMDTransform property, 423</td>
</tr>
<tr>
<td>warped_images()</td>
<td>WibergForwardCompositional (class in menpofit.aam), 43</td>
</tr>
<tr>
<td>warped_images()</td>
<td>WibergInverseCompositional (class in menpofit.aam), 43</td>
</tr>
<tr>
<td>warped_images()</td>
<td>with_traceback()</td>
</tr>
</tbody>
</table>