openglider Documentation

Release 0.01

booya

Apr 11, 2017

Contents

CHAPTER 1

Installation

The fastest way is to use pip2 if installed on your system:

```
git clone https://github.com/hiaselhans/OpenGlider.git
cd OpenGlider
pip2 install -e .
```

this will install a linked version of openglider on your system. If you pull from the repository the installation will be up to date. Also you might want to install vtk or freecad from your systems package manager.

Manual way is as follows:

install all dependencies first:

- ezodf(2)
- dxfwrite
- scipy
- (svgwrite)
- (vtk)

clone the repo:

```
git clone https://github.com/hiaselhans/OpenGlider.git
```

and install using setup.py:

```
cd OpenGlider
python2 setup.py install
```

(developers choice):

```
python2 setup.py develop
```

Contents:

Getting Started

Running Tests

To get familiar, run and take a look at the unittests.

Run all unittests (including fancy visual ones) using:

```
./testall.py -a
```

from the main directory.

Interactive Shell

Openglider is intended to be used as a module in scripts. Best practice is to use ipython notebook or normal python console:: .. code-block:: bash

    python

or .. code-block:: bash

    ipython notebook

Next step is to create a glider, import a geometry file and modify:

```
>>>glider=openglider.Glider.import_geometry("tests/demokite.ods")
>>>for rib in glider.ribs:
...     rib.aoa_relative += 3
...
>>>
```

Then, show the glider:

```
>>>import openglider.graphics as graphics
>>>polygons, points = glider.return_polygons(midribs=4)
>>>graphics.Graphics(map(graphics.Polygon, polygons), points)
```

Export obj file for openfoam, and also json for future needs:

```
>>>glider.export_3d('/tmp/teil.obj')
>>>import openglider.jsonify
>>>with open('/tmp/myglider.json', 'w') as myfile:
...     openglider.jsonify.dump(glider, myfile)
```

Which is to import the whole glider at a later point:

```
>>>import openglider.jsonify
>>>with open('/tmp/myglider.json') as myfile:
...     openglider.jsonify.load(myfile)['data']
```

If you are not yet familiar with python, here is some places to start:

- codeacademy
- dive into python

Project Structure

OpenGlider has grown towards a set of tools:

- airfoil: a class for easy airfoil-manipulations (map x_values, set nr. of coordinates, normalize,...)

- freecad: freecad workbench as a possible gui
- **glider: Classes related to building paragliders:**
  - Glider
  - Rib
  - Cell
  - ...
- graphics: a wrapper to simulate mathematica-graphics with the use of vtk
- gui: several qt-widgets
- input: matplotlib inputs of splines, shape, aoa,...
- jsonify: store OpenGlider objects in json format, load them and migrate between versions
- **lines: A class for LineSets which could be on paragliders, kites,...** Line-geometry is calculated as a linear-equation-system and sag is added
- plots: functions to create plots ready to be sent to factories
- **utils:**
  - **cache: a Cache-class to be inherited and a decorator to be applied on class-functions.** This adds a cache to calculus-intensive functions
  - bezier: a bezier curve implementation
- vector: 2D- and 3D-vector operations and Objects (Polyline, Polygon)

Airfoil

Airfoils are considered to follow the '.dat' convention, which means they are represented by a list of (x, y) vectors, starting from upper-back via the nose towards the lower end. For convenience, profilepoints can be called for x-values in the range (-1,1) whereas <0 significates a point on the upper surface, 0==nose, >0 -> lower surface

Glider

A glider consits of cells, which themselves consist of ribs, miniribs,.. It can also contain a LineSet In order to create a glider you have to create ribs first, then create cells from the ribs. Openglider defines ballooning per rib.

class `openglider.glider.`**`Glider`**(*cells=None*, *lineset=None*)

`apply_mean_ribs`(*num_mean=8*)
Calculate Mean ribs :param num_mean: :return:

`copy_complete`()
Returns a mirrored and combined copy of the glider, ready for export/view

`get_point`(*y=0*, *x=-1*)
Get a point on the glider :param y: span-wise argument (0, cell_no) :param x: chord-wise argument (-1, 1) :return: point

`get_spanwise`(*x=None*)
Return a list of points for a x_value

`return_ribs`(*num=0*, *ballooning=True*)
Get a list of rib-curves :param num: number of midribs per cell :param ballooning: calculate ballooned cells :return: nested list of ribs [[[x,y,z],p2,p3...],rib2,rib3,..]

`shape_flattened`
Projected Shape of the glider (as it would lie on the ground - flattened)

shape_simple
Simple (rectangular) shape representation for spline inputs

Develop

There is a lot to do, including:

- Creating good code
- Improving the existing code
- Writing documentation
- Run and write unittests
- Create civil airships using needle and cloth

Code Conventions

There is not much to say about code conventions in python, but:

- Use python-3 compatible Language:

```
print("use print as a function")
```

- Best practice: http://www.python.org/dev/peps/pep-0008/
- Write unittests for everything

Class Reference

Contents:

openglider package

Subpackages

openglider.airfoil package

Module contents

openglider.airfoil.**get_x_value**(*x_value_list*, *x*)
Get position of x in a list of x_values zb get_x_value([1,2,3],1.5)=0.5

openglider.glider package

Subpackages

openglider.glider.in_out package

Submodules

openglider.glider.in_out.Excel module

openglider.glider.in_out.export_3d module

openglider.glider.in_out.export_3d.**export_apame**(*glider*, *path=''*, *midribs=0*, *numpoints=None*, **other*)

openglider.glider.in_out.export_3d.**export_dxf**(*glider*, *path=''*, *midribs=0*, *numpoints=None*, **other*)

openglider.glider.in_out.export_3d.**export_json**(*glider*, *path*, *numpoints*, *midribs=0*, *wake_panels=1*, *wake_length=0.2*, **other*)

export json geometry file for panelmethod calculation

openglider.glider.in_out.export_3d.**export_obj**(*glider*, *path*, *midribs=0*, *numpoints=None*, *floatnum=6*, *copy=True*)

openglider.glider.in_out.export_3d.**paraBEM_Panels**(*glider*, *midribs=0*, *profile_numpoints=None*, *num_average=0*, *symmetric=False*, *distribution=None*)

return the vertices, panels and the trailing edge of a glider, as paraBEM objects.

midribs: midribs of a cell spanwise. if num_average is greater then 0 ballooning will be disables

profile_numpoints: coordinates of every rib, choordwise num_average: steps to average a cell profile symmetric: set to True if a symmetric result is expected (this will

reduce evaluation time)

openglider.glider.in_out.import_3d module

openglider.glider.in_out.import_3d.**import_json**(*filename*)

openglider.glider.in_out.import_geometry module

openglider.glider.in_out.import_geometry.**get_lower_aufhaengepunkte**(*data*)

openglider.glider.in_out.import_geometry.**import_ods**(*filename*, *glider*)

openglider.glider.in_out.import_geometry.**import_xls**()

openglider.glider.in_out.import_geometry.**merge**(*factor*, *container*)

openglider.glider.in_out.import_geometry.**read_elements**(*sheet*, *keyword*, *len_data=2*)

Return rib/cell_no for the element + data

openglider.glider.in_out.import_geometry.**tolist_lines**(*sheet*, *attachment_points_lower*, *attachment_points_upper*)

openglider.glider.in_out.import_geometry.**transpose_columns**(*sheet=<ezodf.table.Table object>*, *columnswidth=2*)

Module contents

Submodules

openglider.glider.ballooning module

class openglider.glider.ballooning.**ArcSinc**

interpolate(*numpoints*)

numpoints

class openglider.glider.ballooning.**Ballooning**(*f_upper*, *f_lower*)
Bases: object

amount_integral

amount_maximal

arcsinc = <openglider.glider.ballooning.ArcSinc instance>

copy()

mapx(*xvals*)

classmethod phi(**baloon*)
Return the angle of the piece of cake. b/l=R*phi/(R*Sin(phi)) -> Phi=arsinc(l/b)

scale(*factor*)

class openglider.glider.ballooning.**BallooningBezier**(*upper=None*, *lower=None*, *name='ballooning'*)
Bases: *openglider.glider.ballooning.Ballooning*

apply_splines()

controlpoints

numpoints

points

scale(*factor*)

openglider.glider.cell_elements module

openglider.glider.cells module

openglider.glider.glider module

class openglider.glider.glider.**Glider**(*cells=None*, *lineset=None*)
Bases: object

apply_mean_ribs(*num_mean=8*)
Calculate Mean ribs :param num_mean: :return:

arc

area

aspect_ratio

attachment_points

cell_naming_scheme = 'c{cell_no}'

close_rib(*rib=-1*)

copy()

copy_complete()
Returns a mirrored and combined copy of the glider, ready for export/view

export_3d (*path=''*, **args*, ***kwargs*)

get_mesh (*midribs=0*)

get_mesh_hull (*num_midribs=0*, *ballooning=True*)

get_mesh_panels (*num_midribs=0*)

get_midrib (*y=0*)

get_panel_groups ()

get_point (*y=0*, *x=-1*)
Get a point on the glider :param y: span-wise argument (0, cell_no) :param x: chord-wise argument (-1, 1) :return: point

get_spanwise (*x=None*)
Return a list of points for a x_value

glide

has_center_cell

classmethod import_geometry (*path*, *filetype=None*)

mirror (*cutmidrib=True*)

profile_numpoints

profile_x_values

projected_area

rename_parts ()

return_average_ribs (*num=0*, *num_average=8*)

return_ribs (*num=0*, *ballooning=True*)
Get a list of rib-curves :param num: number of midribs per cell :param ballooning: calculate ballooned cells :return: nested list of ribs [[[x,y,z],p2,p3...],rib2,rib3,..]

rib_naming_scheme = 'r{rib_no}'

ribs

scale (*faktor*)

shape_flattened
Projected Shape of the glider (as it would lie on the ground - flattened)

shape_simple
Simple (rectangular) shape representation for spline inputs

span

openglider.glider.rib_elements module

openglider.glider.ribs module

Module contents

openglider.graphics package

Submodules

openglider.graphics.Functions module

Module contents

openglider.input package

Module contents

openglider.plots package

Submodules

openglider.plots.cuts module

class openglider.plots.cuts.**CutResult**(*curve*, *index_left*, *index_right*)

class openglider.plots.cuts.**DesignCut**(*amount*, *num_folds=1*)
Bases: object

apply(*inner_lists*, *outer_left*, *outer_right*)

class openglider.plots.cuts.**FoldedCut**(*amount*, *num_folds=2*)
Bases: *openglider.plots.cuts.DesignCut*

apply(*inner_lists*, *outer_left*, *outer_right*)

class openglider.plots.cuts.**ParallelCut**(*amount*, *num_folds=1*)
Bases: *openglider.plots.cuts.DesignCut*

Cut to continue in a parrallel way (trailing-edge)

apply(*inner_lists*, *outer_left*, *outer_right*)

class openglider.plots.cuts.**SimpleCut**(*amount*, *num_folds=1*)
Bases: *openglider.plots.cuts.DesignCut*

apply(*inner_lists*, *outer_left*, *outer_right*)

openglider.plots.marks module

class openglider.plots.marks.**Arrow**(*left=True*, *scale=0.8*, *name=None*)
Bases: object

class openglider.plots.marks.**Cross**(*rotation=0*, *offset=0*, *name=None*)
Bases: *openglider.plots.marks.Line*

class openglider.plots.marks.**Dot**(**positions*)
Bases: object

class openglider.plots.marks.**Inside**(*func*)
Bases: openglider.plots.marks._Modify

Modify Mark to be on the other side (inside) |x| <- old | | | **|x <- new l1|**

l2

class openglider.plots.marks.**Line**(*rotation=0*, *offset=0*, *name=None*)
Bases: object

class openglider.plots.marks.**OnLine**(*func*)
Bases: openglider.plots.marks._Modify

Modify Mark to sit centered on p2 rather than in between |x| <- old | | | x <- new | |

class openglider.plots.marks.**Polygon**(*edges=3*, *scale=0.8*, *name=None*)
Bases: object

class openglider.plots.marks.**Rotate**(*func*, *rotation*, *center=True*)
Bases: openglider.plots.marks._Modify

class openglider.plots.marks.**Triangle**(*scale=0.8*)
Bases: *openglider.plots.marks.Polygon*

openglider.plots.projection module

Module contents

class openglider.plots.**Patterns**(*glider2d*, *config=None*)
Bases: object

class **DefaultConf**(*dct=None*)
Bases: openglider.plots.glider.config.OtherPatternConfig

Patterns.**unwrap**(*outdir*, *glider=None*)

openglider.utils package

Submodules

openglider.utils.bezier module

openglider.utils.cached_property module

Module contents

class openglider.utils.**Config**(*dct=None*)
Bases: object

get(*key*)

openglider.utils.**consistent_value**(*elements*, *attribute*)

class openglider.utils.**dualmethod**(*func*)
Bases: object

A Decorator to have a combined class-/instancemethod

>>>class a: ... @dualmethod ... def test(this): ... return this ... >>>a.test() <class '__main__.a'> >>>a().test() <__main__.a object at 0x7f133b5f7198> >>>

an instance-check could be:

is_instance = not type(this) is type

openglider.utils.**linspace**(*start*, *stop*, *count*)

openglider.utils.**sign**(*val*)

openglider.vector package

Module contents

openglider.vector.**arrtype**(*arg*)
return type of a vector list: 2d-point (1), list of 2d-points (2), 3d-point (3), list of 3d-points (4)

openglider.vector.**depth**(*arg*)

class openglider.vector.**mirror_func**(*direction=None*)

Submodules

openglider.console module

Module contents

openglider.**load**(*filename*)

openglider.**save**(*data*, *filename*, *add_meta=True*)

Indices and tables

- genindex
- modindex
- search

Indices and tables

- genindex
- modindex
- search

Python Module Index

o

Index