# ethoscope Documentation

***Release 1.0***

**Quentin Geissmann**

**Jul 30, 2017**

# Contents

CHAPTER 1

# Module contents

Ethoscope is a platform developed at Gilestro lab. It provide an integrated set of tools to acquire behavioural data on small animals and build feed back modules to interact with them in real time. A description of the whole system is available on its website The documentation herein describes the ethoscope python package, which is the core of the device software. It is intended for programmers who want to contribute to development, and assumes familiarity with `python` programming language.

The first purpose of the package is to provide biologists with a modular API to acquire videos, track animals in real time, feed back to deliver stimuli upon specific triggers, annotate video frames with tracking information and save data in a consistent format (database). In addition, is implements a webserver that can run a a daemon and performs actions upon POST requests.

## Installation

Probably you want to work on a virtual environment. Then you want to install OpenCV (which is an external library – i.e. not ip pip). Afterwards, you can clone the repository (the branch `dev` being the development version) and run:

```
` cd src pip install -e .[dev] `
```

## Core API

This diagram represents the core of the API in UML:

The classes prefixed with `Base` are abstract, and several derived classes are already implemented for most of them, but more can be done in the prospect of achieving modularity.

## Local tracking example

Since the API is modular, it can be used to simply perform of line tracking from a video file. Here is a very simple example. If you want to run this code yourself, you can download the test video.

```
>>> # We import all the bricks from ethoscope package
>>> from ethoscope.core.monitor import Monitor
>>> from ethoscope.trackers.adaptive_bg_tracker import AdaptiveBGModel
>>> from ethoscope.utils.io import SQLiteResultWriter
>>> from ethoscope.hardware.input.cameras import MovieVirtualCamera
>>> from ethoscope.drawers.drawers import DefaultDrawer
>>>
>>> # You can also load other types of ROI builder. This one is for 20 tubes (two␣
↪columns of ten rows)
>>> from ethoscope.roi_builders.target_roi_builder import␣
↪SleepMonitorWithTargetROIBuilder
>>>
>>> # change these three variables according to how you name your input/output files
>>> INPUT_VIDEO = "test_video.mp4"
>>> OUTPUT_VIDEO = "/tmp/my_output.avi"
>>> OUTPUT_DB = "/tmp/results.db"
>>>
>>> # We use a video input file as if it was a "camera"
>>> cam = MovieVirtualCamera(INPUT_VIDEO)
>>>
>>> # here, we generate ROIs automatically from the targets in the images
>>> roi_builder = SleepMonitorWithTargetROIBuilder()
>>> rois = roi_builder.build(cam)
>>> # Then, we go back to the first frame of the video
>>> cam.restart()
>>>
>>> # we use a drawer to show inferred position for each animal, display frames and␣
↪save them as a video
>>> drawer = DefaultDrawer(OUTPUT_VIDEO, draw_frames = True)
>>> # We build our monitor
>>> monitor = Monitor(cam, AdaptiveBGModel, rois)
>>>
>>> # Now everything ius ready, we run the monitor with a result writer and a drawer
>>> with SQLiteResultWriter(OUTPUT_DB, rois) as rw:
>>>     monitor.run(rw,drawer)
```

## *Post hock* data analysis

A large amount of data can be generated thanks to ethoscope. In order to render the analysis (visualisation, summaries, statistics ...) straightforward and flexible, we developed an `R` package named rethomics.

CHAPTER 2

# Submodules

## ethoscope.core package

### Module contents

This module is the core of the *ethoscope*. It defines the building bricks at the basis of the package.

Overview:

- *Monitor* is the most important class. It glues together all the other elements of the package in order to perform (video tracking, interacting , data writing and drawing).
- *TrackingUnit* are internally used by monitor. They forces to conceptually treat each ROI independently.
- *ROI* formalise and facilitates the use of Region Of Interests.
- *variables* are custom types of variables that result from tracking and interacting.
- *DataPoint* stores efficiently Variables.

### ethoscope.core.monitor module

**class** ethoscope.core.monitor.**Monitor**(*camera*, *tracker_class*, *rois=None*, *stimulators=None*, **args*, ***kwargs*)

Bases: object

Class to orchestrate the tracking of multiple objects. It performs, in order, the following actions:

•Requesting raw frames (delegated to BaseCamera)

•Cutting frame portions according to the ROI layout (delegated to *TrackingUnit*).

•Detecting animals and computing their positions and other variables (delegated to *BaseTracker*).

•Using computed variables to interact physically (i.e. feed-back) with the animals (delegated to *BaseStimulator*).

•Drawing results on a frame, optionally saving video (delegated to *BaseDrawer*).

•Saving the result of tracking in a database (delegated to ResultWriter).

**Parameters**

- **camera** (BaseCamera) – a camera object responsible of acquiring frames and associated time stamps
- **tracker_class** (*class*) – The algorithm that will be used for tracking. It must inherit from *BaseTracker*
- **rois** (list(*ROI*)) – A list of region of interest.
- **stimulators** (list(BaseInteractor) – The class that will be used to analyse the position of the object and interact with the system/hardware.
- **args** – additional arguments passed to the tracking algorithm
- **kwargs** – additional keyword arguments passed to the tracking algorithm

**last_frame_idx**

**Returns** The number of the last acquired frame.

**Return type** int

**last_positions**

**Returns** The last positions (and other recorded variables) of all detected animals

**Return type** dict

**last_time_stamp**

**Returns** The time, in seconds, since monitoring started running. It will be 0 if the monitor is not running yet.

**Return type** float

**run**(*result_writer=None*, *drawer=None*)
Runs the monitor indefinitely.

**Parameters**

- **result_writer** (ResultWriter) – A result writer used to control how data are saved. *None* means no results will be saved.
- **drawer** (*BaseDrawer*) – A drawer to plot the data on frames, display frames and/or save videos. *None* means none of the aforementioned actions will performed.

**stop**()
Interrupts the *run* method. This is meant to be called by another thread to stop monitoring externally.

## ethoscope.core.tracking_unit module

**class** ethoscope.core.tracking_unit.**TrackingUnit**(*tracking_class*, *roi*, *stimulator=None*, **args*, ***kwargs*)

Bases: object

Class instantiating a tracker(*BaseTracker*), and linking it with an individual ROI(ROI) and stimulator(*BaseStimulator*). Typically, several *TrackingUnit* objects are built internally by a Monitor(*Monitor*).

**Parameters**

- **tracker_class** (*class*) – The algorithm that will be used for tracking. It must inherit from *BaseTracker*
- **roi** (*ROI*.) – A region of interest.
- **stimulator** (*BaseStimulator*.) – an object used to physically interact with the detected animal.
- **args** – additional arguments passed to the tracking algorithm.
- **kwargs** – additional keyword arguments passed to the tracking algorithm.

**get_last_positions** (*absolute=False*)
The last position of the animal monitored by this *TrackingUnit*

**Parameters** **absolute** – Whether the position should be relative to the top left corner of the raw frame (*true*), or to the top left of the used ROI (*false*).

**Returns** A container with the last variable recorded for this roi.

**Return type** *DataPoint*

**roi**

**Returns** A reference to the roi used by this *TrackingUnit*

**Return type** *ROI*

**stimulator**

**Returns** A reference to the stimulator used by this *TrackingUnit*

**Return type** *BaseStimulator*

**track** (*t*, *img*)
Uses the whole frame acquired, along with its time stamp to infer position of the animal. Also runs the stimulator object.

**Parameters**

- **t** (*int*) – the time stamp associated to the provided frame (in ms).
- **img** (ndarray) – the entire frame to analyse

**Returns** The resulting data point

**Return type** *DataPoint*

## ethoscope.core.roi module

**class** ethoscope.core.roi.**ROI** (*polygon*, *idx*, *value=None*, *orientation=None*, *regions=None*)
Bases: object

Class to define a region of interest(ROI). Internally, ROIs are single polygons. At the moment, they cannot have any holes. The polygon defining the ROI is used to draw a mask to exclude off-target pixels (so cross-ROI info).

**Parameters**

- **polygon** (ndarray) – An array of points
- **idx** (*int*) – the index of this ROI
- **value** – an optional value to be save for this ROI (e.g. to define left and right side)
- **orientation** – Optional orientation Not implemented yet
- **regions** – Optional sub-regions within the ROI. Not implemented yet

**apply**(*img*)
Cut an image where the ROI is defined.

**Parameters** **img** (ndarray) – An image. Typically either one or three channels *uint8*.

**Returns** a tuple containing the resulting cropped image and the associated mask (both have the same dimension).

**Return type** (ndarray, ndarray)

**bounding_rect**()

**get_feature_dict**()

**Returns** A dictionary of freatures for this roi. It containes the folowing fields:

- "x"
- "y"
- "w"
- "h"
- "value"
- "idx"

**Return type** dict

**idx**

**Returns** The index of this ROI

**Return type** int

**longest_axis**

**Returns** the value of the longest axis (w or h)

**Return type** float

**mask**()

**Returns** The mask as a single chanel, *uint8* image.

**Return type** ndarray

**offset**

**Returns** the x,y offset of the ROI compared to the frame it was build on.

**Return type** (int,int)

**polygon**

**Returns** the internal polygon defining the ROI.

**Return type** ndarray

**rectangle**

**Returns** The upright bounding rectangle to the ROI formatted (x,y,w,h). Where x and y are to coordinates of the top left corner

**Return type** (int,int,int,int)

**set_value**(*new_val*)

**Parameters** **new_val** – assign a nex value to a ROI

**value**

**Returns** the value of a ROI

## ethoscope.core.variables module

**class** ethoscope.core.variables.**BaseBoolVariable**
Bases: *ethoscope.core.variables.BaseIntVariable*

Abstract type encoding boolean values. Internally stored as int as bool type cannot be derived.

**functional_type = 'bool'**

**sql_data_type = 'BOOLEAN'**

**class** ethoscope.core.variables.**BaseDistanceIntVar**
Bases: *ethoscope.core.variables.BaseIntVariable*

Abstract type encoding variables representing distances.

**functional_type = 'distance'**

**class** ethoscope.core.variables.**BaseIntVariable**
Bases: int

Template class for defining arbitrary variable types. Each class derived from this one should at least define the three following attributes:

- *sql_data_type*, The MySQL data type. This allows to use minimal space to save data points.
- *header_name*, The name of this variable. this will be used as the column name in the result table, so it must be unique.
- *functional_type*, A keyword defining what type of variable this is. For instance "distance", "angle" or "proba". this allow specific post-processing per functional type.

**functional_type = None**

**header_name = None**

**sql_data_type = 'SMALLINT'**

**class** ethoscope.core.variables.**BaseRelativeVariable**
Bases: *ethoscope.core.variables.BaseDistanceIntVar*

Abstract type encoding distance variables that can be expressed relatively to an origin. They converted to absolute using information form the ROI.

**to_absolute**(*roi*)
Converts a positional variable from a relative (to the top left of a ROI) to an absolute (e.i. top left of the parent image).

**Parameters** **roi** (ROI.) – a region of interest

**Returns** A new variable

**Return type** BaseRelativeVariable

**class** ethoscope.core.variables.**HeightVariable**
Bases: *ethoscope.core.variables.BaseDistanceIntVar*

Type storing the height of a detected object.

**header_name = 'h'**

**class** ethoscope.core.variables.**IsInferredVariable**
Bases: *ethoscope.core.variables.BaseBoolVariable*

Type encoding whether a data point is inferred (from past values) or observed; 0 or 1, respectively.

**header_name = 'is_inferred'**

**class** ethoscope.core.variables.**Label**
Bases: *ethoscope.core.variables.BaseIntVariable*

Type encoding a discrete label when several objects, in the same ROI, are detected.

**functional_type = 'label'**

**header_name = 'label'**

**class** ethoscope.core.variables.**PhiVariable**
Bases: *ethoscope.core.variables.BaseIntVariable*

Type encoding the angle of a detected object, in degrees.

**functional_type = 'angle'**

**header_name = 'phi'**

**class** ethoscope.core.variables.**WidthVariable**
Bases: *ethoscope.core.variables.BaseDistanceIntVar*

Type storing the width of a detected object.

**header_name = 'w'**

**class** ethoscope.core.variables.**XPosVariable**
Bases: *ethoscope.core.variables.BaseRelativeVariable*

Type storing the X position of a detected object.

**header_name = 'x'**

**class** ethoscope.core.variables.**XYDistance**
Bases: *ethoscope.core.variables.BaseIntVariable*

Type storing distance moved between two consecutive observations. Log10 x 1000 is used so that floating point distance is stored as an int.

**functional_type = 'relative_distance_1e6'**

**header_name = 'xy_dist_log10x1000'**

**class** ethoscope.core.variables.**YPosVariable**
Bases: *ethoscope.core.variables.BaseRelativeVariable*

Type storing the Y position of a detected object.

**header_name = 'y'**

**class** ethoscope.core.variables.**mLogLik**
Bases: *ethoscope.core.variables.BaseIntVariable*

Type representing a log likelihood. It should be multiplied by 1000 to be stored as an int.

**functional_type = 'proba'**

**header_name = 'mlog_L_x1000'**

### ethoscope.core.data_point module

**class** ethoscope.core.data_point.**DataPoint**(*data*)
Bases: collections.OrderedDict

A container to store variables. It derived from OrderedDict. Variables are accessible by header name, which is an individual identifier of a variable type (see *BaseIntVariable*):

```
>>> from ethoscope.core.variables import DataPoint, XPosVariable, YPosVariable,␣
↪HeightVariable
>>> y = YPosVariable(18)
>>> x = XPosVariable(32)
>>> data = DataPoint([x,y])
>>> print data["x"]
>>> h = HeightVariable(3)
>>> data.append(h)
>>> print data
```

**Parameters** **data** (list(*BaseIntVariable*)) – a list of data points

**append**(*item*)
Add a new variable in the *DataPoint* The order is preserved.

**Parameters**

- **item** – A variable to be added.
- **item** – *BaseIntVariable*

**Returns**

**copy**()
Deep copy a data point. Copying using the = operator will simply create an alias to a *DataPoint* object (i.e. allow modification of the original object).

**Returns** a copy of this object

**Return type** *DataPoint*

## ethoscope.roi_builder package

### Module contents

This module define modular *ROI builders*. These objects take an image (or camera stream) and use them to construct a list of BaseROIBuilder is defined. This will typically be done when new arenas (hardware components) is defined.

### ethoscope.roi_builders.roi_builders module

**class** ethoscope.roi_builders.roi_builders.**BaseROIBuilder**
Bases: ethoscope.utils.description.DescribedObject

Template to design ROIBuilders. Subclasses must implement a _rois_from_img method.

**build**(*input*)
Uses an input (image or camera) to build ROIs. When a camera is used, several frames are acquired and averaged to build a reference image.

**Parameters** **input** (BaseCamera or ndarray) – Either a camera object, or an image.

**Returns** list(*ROI*)

**class** ethoscope.roi_builders.roi_builders.**DefaultROIBuilder**
Bases: *ethoscope.roi_builders.roi_builders.BaseROIBuilder*

The default ROI builder. It simply defines the entire image as a unique ROI.

Template to design ROIBuilders. Subclasses must implement a `_rois_from_img` method.

## ethoscope.roi_builders.img_roi_builder module

**class** ethoscope.roi_builders.img_roi_builder.**ImgMaskROIBuilder**(*mask_path*)
Bases: *ethoscope.roi_builders.roi_builders.BaseROIBuilder*

Class to build rois from greyscale image file. Each continuous region is used as a ROI. The greyscale value inside the ROI determines it's value.

IMAGE HERE

## ethoscope.roi_builders.target_roi_builder module

**class** ethoscope.roi_builders.target_roi_builder.**HD12TubesRoiBuilder**
Bases: *ethoscope.roi_builders.target_roi_builder.TargetGridROIBuilder*

Class to build ROIs for a twelve columns, one row for the HD tracking arena (see here)

**class** ethoscope.roi_builders.target_roi_builder.**OlfactionAssayROIBuilder**
Bases: *ethoscope.roi_builders.target_roi_builder.TargetGridROIBuilder*

Class to build ROIs for a one-column, ten-rows (see here)

**class** ethoscope.roi_builders.target_roi_builder.**SleepMonitorWithTargetROIBuilder**
Bases: *ethoscope.roi_builders.target_roi_builder.TargetGridROIBuilder*

Class to build ROIs for a two-columns, ten-rows for the sleep monitor (see here).

**class** ethoscope.roi_builders.target_roi_builder.**TargetGridROIBuilder**(*n_rows=1*, *n_cols=1*, *top_margin=0*, *bottom_margin=0*, *left_margin=0*, *right_margin=0*, *horizontal_fill=0.9*, *vertical_fill=0.9*)
Bases: *ethoscope.roi_builders.roi_builders.BaseROIBuilder*

This roi builder uses three black circles drawn on the arena (targets) to align a grid layout:

IMAGE HERE

**Parameters**

- **n_rows** (*int*) – The number of rows in the grid.
- **n_cols** (*int*) – The number of columns.

- **top_margin** (*float*) – The vertical distance between the middle of the top ROIs and the middle of the top target
- **bottom_margin** (*float*) – same as top_margin, but for the bottom.
- **left_margin** (*float*) – same as top_margin, but for the left side.
- **right_margin** (*float*) – same as top_margin, but for the right side.
- **horizontal_fill** (*float*) – The proportion of the grid space user by the roi, horizontally (between 0 and 1).
- **vertical_fill** (*float*) – same as vertical_fill, but horizontally.

# ethoscope.trackers package

## Module contents

## ethoscope.trackers.trackers module

**class** ethoscope.trackers.trackers.**BaseTracker**(*roi*, *data=None*)
Bases: ethoscope.utils.description.DescribedObject

Template class for video trackers. A video tracker locate animal in a ROI. Derived class must implement the _find_position method.

**Parameters**

- **roi** (ROI) – The Region Of Interest the the tracker will use to locate the animal.
- **data** – An optional data set. For instance, it can be used for pre-trained algorithms

**Returns**

**last_time_point**

**Returns** The last time point that the tracker used. This is updated even when position is inferred/no animal is found

**Return type** int

**positions**

**Returns** The last few positions found by the tracker. Positions are kept for a certain duration defined by the _max_history_length attribute.

**Return type** deque

**times**

**Returns** The last few time points corresponding to *positions*.

**Return type** deque

**track**(*t*, *img*)
Locate the animal in a image, at a given time.

**Parameters**

- **t** (*int*) – time in ms
- **img** (ndarray) – the whole frame.

**Returns** The position of the animal at time t

**Return type** *DataPoint*

**xy_pos**(*i*)

**exception** ethoscope.trackers.trackers.**NoPositionError**
Bases: exceptions.Exception

Used to abort tracking. When it is raised within the _find_position method, data is inferred from previous position.

## ethoscope.trackers.single_roi_tracker module

**class** ethoscope.trackers.single_roi_tracker.**AdaptiveBGModelOneObject**(*roi*, *data=None*)
Bases: *ethoscope.trackers.trackers.BaseTracker*

## ethoscope.trackers.adaptive_bg_tracker module

**class** ethoscope.trackers.adaptive_bg_tracker.**AdaptiveBGModel**(*roi*, *data=None*)
Bases: *ethoscope.trackers.trackers.BaseTracker*

An adaptive background subtraction model to find position of one animal in one roi.

TODO more description here :param roi: :param data: :return:

**fg_model = <ethoscope.trackers.adaptive_bg_tracker.ObjectModel object>**

**class** ethoscope.trackers.adaptive_bg_tracker.**BackgroundModel**(*max_half_life=100000.0*, *min_half_life=1000.0*, *increment=1.2*)
Bases: object

A class to model background. It uses a dynamic running average and support arbitrary and heterogeneous frame rates

**bg_img**

**decrease_learning_rate**()

**increase_learning_rate**()

**update**(*img_t*, *t*, *fg_mask=None*)

**class** ethoscope.trackers.adaptive_bg_tracker.**ObjectModel**(*history_length=1000*)
Bases: object

A class to model, update and predict foreground object (i.e. tracked animal).

**compute_features**(*img*, *contour*)

**distance**(*features*, *time*)

**features_header**

**is_ready**

**update**(*img*, *contour*, *time*)

# ethoscope.interactors package

## Module contents

## ethoscope.stimulators.stimulators module

**class** ethoscope.stimulators.stimulators.**BaseStimulator**(*hardware_connection*, *date_range=''*)

Bases: ethoscope.utils.description.DescribedObject

Template class to interact with the tracked animal in a real-time feedback loop. Derived classes must have an attribute _hardwareInterfaceClass defining the class of the BaseInterface object (not on object) that instances will share with one another. In addition, they must implement a _decide() method.

**Parameters**

- **hardware_connection** (BaseInterface) – The hardware interface to use.
- **date_range** (*str*) – the start and stop date/time for the stimulator. Format described here

**apply**()

Apply this stimulator. This method will:

1.check _tracker exists

2.decide (_decide) whether to interact

3.if 2. pass the interaction arguments to the hardware interface

**Returns** whether a stimulator has worked, and a result dictionary

**bind_tracker**(*tracker*)

Link a tracker to this interactor

**Parameters** **tracker** (*BaseTracker*) – a tracker object.

**class** ethoscope.stimulators.stimulators.**DefaultStimulator**(*hardware_connection*, *date_range=''*)

Bases: *ethoscope.stimulators.stimulators.BaseStimulator*

Default interactor. Simply never interacts

Template class to interact with the tracked animal in a real-time feedback loop. Derived classes must have an attribute _hardwareInterfaceClass defining the class of the BaseInterface object (not on object) that instances will share with one another. In addition, they must implement a _decide() method.

**Parameters**

- **hardware_connection** (BaseInterface) – The hardware interface to use.
- **date_range** (*str*) – the start and stop date/time for the stimulator. Format described here

**class** ethoscope.stimulators.stimulators.**HasInteractedVariable**

Bases: *ethoscope.core.variables.BaseIntVariable*

**Custom variable to save whether the stimulator has sent instruction to its hardware interface. 0 means** no interaction. Any positive integer describes a different interaction.

**functional_type = 'interaction'**

**header_name = 'has_interacted'**

## ethoscope.stimulators.sleep_depriver_stimulators module

any new class added here need to be added to web_utils/control_thread.py too

**class** ethoscope.stimulators.sleep_depriver_stimulators.**ExperimentalSleepDepStimulator**(*hardware_ve- loc- ity_thresh date_rang*

Bases: *ethoscope.stimulators.sleep_depriver_stimulators.SleepDepStimulator*

A stimulator to control a sleep depriver module. This is an experimental version where each channel has a different inactivity_time_threshold.

**Parameters**

- **hardware_connection** (SleepDepriverInterface) – the sleep depriver module hardware interface
- **velocity_threshold** (*float*) –

**Returns**

**bind_tracker**(*tracker*)

**class** ethoscope.stimulators.sleep_depriver_stimulators.**IsMovingStimulator**(*hardware_connection=Non ve- loc- ity_threshold=0.006, date_range='', **kwargs*)

Bases: *ethoscope.stimulators.stimulators.BaseStimulator*

class implementing an stimulator that decides whether an animal has moved though does nothing accordingly. :param hardware_connection: a default hardware interface object :param velocity_threshold: Up to which velocity an animal is considered to be immobile :type velocity_threshold: float

**class** ethoscope.stimulators.sleep_depriver_stimulators.**MiddleCrossingStimulator**(*hardware_connect p=1.0, date_range=''*)

Bases: *ethoscope.stimulators.stimulators.BaseStimulator*

**Parameters**

- **hardware_connection** (SleepDepriverInterface) – the sleep depriver module hardware interface
- **p** (*float*) – the probability of disturbing the animal when a beam cross happens

**Returns**

**class** ethoscope.stimulators.sleep_depriver_stimulators.**OptomotorSleepDepriver**(*hardware_connection ve- loc- ity_threshold=0.006, min_inactive_time=12 pulse_duration=1000 stim- u- lus_type=2, date_range=''*)

Bases: *ethoscope.stimulators.sleep_depriver_stimulators.SleepDepStimulator*

**class** ethoscope.stimulators.sleep_depriver_stimulators.**SleepDepStimulator**(*hardware_connection*, *velocity_threshold=0.006*, *min_inactive_time=120*, *date_range=''*)

Bases: *ethoscope.stimulators.sleep_depriver_stimulators.IsMovingStimulator*

A stimulator to control a sleep depriver module.

**Parameters**

- **hardware_connection** (SleepDepriverInterface) – the sleep depriver module hardware interface
- **velocity_threshold** (*float*) –
- **min_inactive_time** (*float*) – the minimal time without motion after which an animal should be disturbed (in seconds)

**Returns**

**class** ethoscope.stimulators.sleep_depriver_stimulators.**SleepDepStimulatorCR**(*hardware_connection*, *velocity_threshold=0.006*, *min_inactive_time=120*, *date_range=''*)

Bases: *ethoscope.stimulators.sleep_depriver_stimulators.SleepDepStimulator*

A stimulator to control a sleep depriver module.

**Parameters**

- **hardware_connection** (SleepDepriverInterface) – the sleep depriver module hardware interface
- **velocity_threshold** (*float*) –
- **min_inactive_time** (*float*) – the minimal time without motion after which an animal should be disturbed (in seconds)

**Returns**

## ethoscope.stimulators.odour_stimulators module

**class** ethoscope.stimulators.odour_stimulators.**DynamicOdourDeliverer**(*hardware_connection*, *date_range=''*)

Bases: *ethoscope.stimulators.odour_stimulators.HasChangedSideStimulator*

A stimulator to control a sleep depriver module

**Parameters** **hardware_connection** () – the sleep depriver module hardware interface

**Returns**

**class** ethoscope.stimulators.odour_stimulators.**DynamicOdourSleepDepriver**(*hardware_connection*, *velocity_threshold=0.003*, *min_inactive_time=120*, *stimulus_duration=5*, *date_range=''*)

Bases: *ethoscope.stimulators.sleep_depriver_stimulators.SleepDepStimulator*

A stimulator to control an odour sleep depriver module.

**Parameters**

- **hardware_connection** (SleepDepriverInterface) – the sleep depriver module hardware interface
- **velocity_threshold** (*float*) – The minimal velocity that counts as movement.
- **stimulus_duration** (*float*) – how long the odour delivery takes place for
- **min_inactive_time** (*float*) – the minimal time without motion after which an animal should be disturbed (in seconds)

**Returns**

**class** ethoscope.stimulators.odour_stimulators.**HasChangedSideStimulator**(*hardware_connection=None*, *middle_line=0.5*)

Bases: *ethoscope.stimulators.stimulators.BaseStimulator*

class implementing a stimulator that decides whether an animal has change side in its ROI. :param hardware_connection: a default hardware interface object :param middle_line: the x position defining the line to be crossed (from 0 to 1, relative to ROI) :type middle_line: float

**class** ethoscope.stimulators.odour_stimulators.**MiddleCrossingOdourStimulator**(*hardware_connection*, *p=1.0*, *refractory_period=300*, *stimulus_duration=5*, *date_range=''*)

Bases: *ethoscope.stimulators.sleep_depriver_stimulators.MiddleCrossingStimulator*

# ethoscope.drawers package

## Submodules

## ethoscope.drawers.drawers module

**class** ethoscope.drawers.drawers.**BaseDrawer**(*video_out=None*, *draw_frames=True*, *video_out_fourcc='DIVX'*, *video_out_fps=2*)

Bases: object

A template class to annotate and save the processed frames. It can also save the annotated frames in a video file and/or display them in a new window. The `_annotate_frame()` abstract method defines how frames are annotated.

**Parameters**

- **video_out** (*str*) – The path to the output file (.avi)
- **draw_frames** (*bool*) – Whether frames should be displayed on the screen (a new window will be created).
- **video_out_fourcc** (*str*) – When setting `video_out`, this defines the codec used to save the output video (see fourcc)
- **video_out_fps** (*float*) – When setting `video_out`, this defines the output fps. typically, the same as the input fps.

**draw**(*img*, *positions*, *tracking_units*)
Draw results on a frame.

**Parameters**

- **img** (ndarray) – the frame that was just processed
- **positions** (list(*DataPoint*)) – a list of positions resulting from analysis of the frame by a tracker
- **tracking_units** (list(*TrackingUnit*)) – the tracking units corresponding to the positions

**Returns**

**last_drawn_frame**

**class** ethoscope.drawers.drawers.**DefaultDrawer**(*video_out=None*, *draw_frames=False*)
Bases: *ethoscope.drawers.drawers.BaseDrawer*

The default drawer. It draws ellipses on the detected objects and polygons around ROIs. When an "interaction" see `BaseInteractor` happens within a ROI, the ellipse is red, blue otherwise.

**Parameters**

- **video_out** (*str*) – The path to the output file (.avi)
- **draw_frames** (*bool*) – Whether frames should be displayed on the screen (a new window will be created).

**class** ethoscope.drawers.drawers.**NullDrawer**
Bases: *ethoscope.drawers.drawers.BaseDrawer*

A drawer that does nothing (no video writing, no annotation, no display on the screen).

**Returns**

## Module contents

CHAPTER 3

# Indices and tables

- genindex
- modindex
- search

# Python Module Index

## e

# Index

## F



## G



## H



## I



## L



## M