pyFIM Documentation

Release 0.2

Philipp Schlegel

Apr 25, 2018

Contents

Release 0.2

Date Apr 25, 2018

pyFIM is a Python 3 package for analysis of FIMTrack data. It extracts parameters from .csv files produced by FIMTrack, performs additional analyses and facilitates comparison of experiments.

FIMTrack is an object tracker developed by Risse et al. (University of Muenster, Germany). From their Github repository:

> "FIMTrack is a larval tracking program to acquire locomotion trajectories and conformation information of Drosophila melanogaster larvae. It is optimized for FIM images. FIM is an acronym for FTIR-based Imaging Method, whereby FTIR is the short form for Frustrated Total Internal Reflection."

CHAPTER 1

Core Features

- import of .csv files
- extraction of FIMTrack parameters
- built-in additional high-level analyses
- easy handling and comparison of experiments

CHAPTER 2

Contribute

Source Code: https://github.com/schlegelp/pyfim

Issue Tracker: https://github.com/schlegelp/pyfim/issues

CHAPTER 3

Support

If you are having issues, drop me a message: pms70[AT]cam[DOT]ac[DOT]uk

CHAPTER 4

License

pyFIM is licensed under the GNU GPL v3+ license

CHAPTER 5

Acknowledgments

Big thanks to Dimitri Berh, Benjamin Risse, Nils Otto and Christian Klämbt for sharing their MatLab code.

CHAPTER 6

FIMTrack References

Risse B, Berh D, Otto N, Klämbt C, Jiang X. FIMTrack: An open source tracking and locomotion analysis software for small animals. PLoS Computational Biology. 2017;13(5):e1005530. doi:10.1371/journal.pcbi.1005530.

Risse B, Otto N, Berh D, Jiang X, Klämbt C. FIM Imaging and FIMtrack: Two New Tools Allowing High-throughput and Cost Effective Locomotion Analysis. Journal of Visualized Experiments: JoVE. 2014;(94):52207. doi:10.3791/52207.

Risse B, Thomas S, Otto N, et al. FIM, a Novel FTIR-Based Imaging Method for High Throughput Locomotion Analysis. PLoS ONE. 2013;8(1):e53963. doi:10.1371/journal.pone.0053963.

CHAPTER 7

Table of Contents

7.1 Install

7.1.1 Requirements

pyFIM requires Python 3.3 or higher.

Please make sure you have all these dependencies installed. They are all available via PIP.

- Pandas >= 0.21.0
- Numpy >= 1.13.3
- PeakUtils >= 1.1.0
- tqdm >= 4.15.0

Note: If you are on Windows, it is probably easiest to install a scientific Python distribution such as Anaconda, Enthought Canopy, Python(x,y), WinPython, or Pyzo. If you use one of these Python distribution, please refer to their online documentation.

7.1.2 Installation

pyFIM is not listed in the Python Packaging Index but you can install the current version directly from Github using:

```
pip install git+git://github.com/schlegelp/pyfim@master
```

See here how to get PIP.

Depending on your default Python version you may have to specify that you want pyFIM to be installed for Python 3:

```
pip3 install git+git://github.com/schlegelp/pyfim@master
```

7.1.3 Installing from source

Alternatively, you can install pyFIM from source:

1. Download the source (tar.gz file) from

   https://github.com/schlegelp/pyfim/tree/master/dist

2. Unpack and change directory to the source directory

   (the one with setup.py).

3. Run `python setup.py install` to build and install

7.2 Introduction

This section will teach you the basics of how to use pyFIM.

7.2.1 Experiments and Collections

Everython in pyFIM is done by two basic classes: *Experiment* and *Collection*.

pyfim.Experiment extracts data from .csv files, does analyses and helps you access each parameter. The idea is that you divide data from e.g. different genotypes into an Experiment each.

As soon as you initialize an Experiment, data is extracted, processed and additional analyses are run. Data clean up involves:

- removal of objects with too few data points
- filling of gaps in thresholded parameters
- conversion from pixel to mm/mm^2 (optional)
- remove frames at the beginning or end of the tracks

You can fine tune how this and the analyses are done by changing the defaults in *config.py*. Please note that changes to the *config.py* will only take effect if you restart your Python session. On the fly, you can change the defaults by e.g.

```
>>> pyfim.defaults['PIXEL_PER_MM'] = 300
```

See the Configuration section for details.

pyfim.Collection keep track of your Experiments. Their job is to generate data tables from attached Experiments collapsing data into means per larva.

Both these classes generate pandas DataFrames for the data and facilitate juggling it. I highly recommend getting familiar with pandas:

- pandas tutorials
- pandas visualization

7.2.2 Learning by doing

Let's start off with a simple case: exploring a single *Experiment*.

```
>>> import pyfim
>>> import matplotlib.pyplot as plt
>>> # Initialise an experiment using a single CSV file
>>> exp = pyfim.Experiment('/experiments/genotype1/exp1.csv')
... INFO  : Data clean-up dropped 51 objects and 0 frames (pyfim)
```

As you see, 51 objects were dropped during import. That's because, by default, object tracks have to have at least 500 frames - if not they are dropped.

Next, get a summary and available parameters:

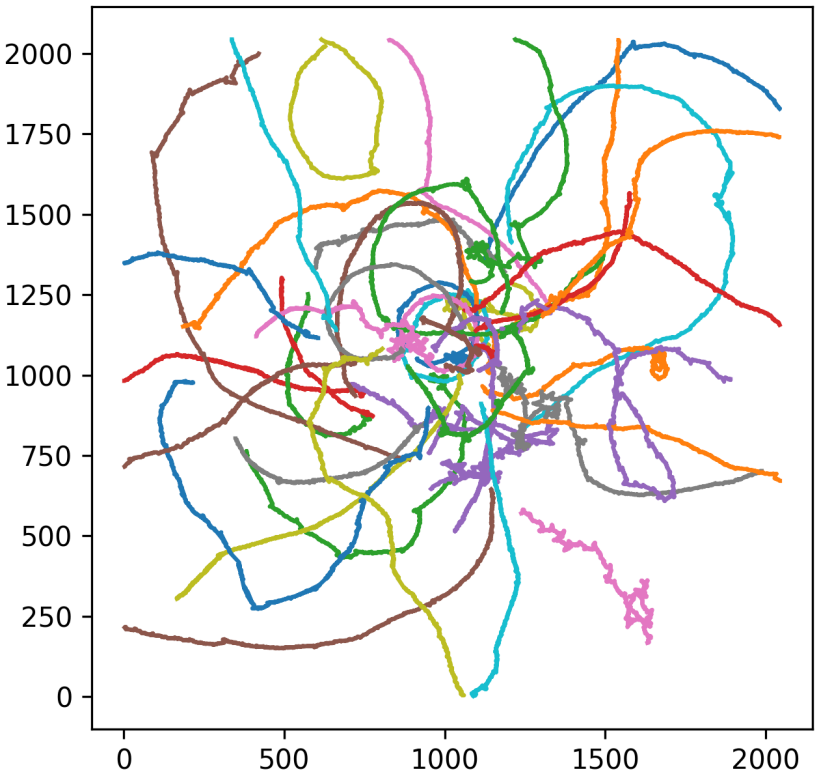
```
>>> print( exp )
... <class 'pyfim.core.Experiment'> with: 48 objects; 1800 frames.
... Available parameters: acc_dst, acceleration, area, bending,
... bending_strength, dst_to_origin, go_phase, head_bends, head_x, head_y,
... is_coiled, is_well_oriented, left_bended, mom_dst, mom_x, mom_y,
... mov_direction, pause_turns, perimeter, peristalsis_efficiency,
... peristalsis_frequency, radius_1, radius_2, radius_3, right_bended,
... spine_length, spinepoint_1_x, spinepoint_1_y, spinepoint_2_x,
... spinepoint_2_y, spinepoint_3_x, spinepoint_3_y, stops, tail_x, tail_y,
... velocity
```

Access to all these data tables is always the same:

```
>>> exp.acc_dst
...    object_1  object_100  object_101  object_102  object_103 \
... 0   0.00000     0.00000     0.00000     0.00000     0.00000
... 1   2.23607     0.00000     2.00000     1.00000     1.00000
... 2   3.65028     1.00000     3.41421     1.00000     3.23607
... 3   3.65028     2.00000     3.41421     2.41421     4.23607
... 4   4.65028     3.41421     4.41421     3.82843     4.23607
... ...
```

Let's do some plotting: traces over time

```
>>> ax = exp.plot_tracks()
>>> plt.show()
```



Access data tables. Please note that some data tables are 2 dimensional (e.g. velocity) while others are 1 dimensional (e.g. pause_turns)

```
>>> velocity = exp.velocity
>>> pause_turns = exp.pause_turns
```

Get the mean over all objects tracked

```
>>> mean_velocity = exp.mean('velocity')
```

Alternatively (for 2 dimensional data tables)

```
>>> mean_velocity = exp.velocity.mean(axis=0)
```

The second way also lets you get other metrics

```
>>> max_velocity = exp.velocity.max(axis=0)
```

Get all means over all parameters

```
>>> all_means = exp.mean()
```

We can also access data by objects:

```
>>> # Get a list of all tracked objects
>>> exp.objects
```

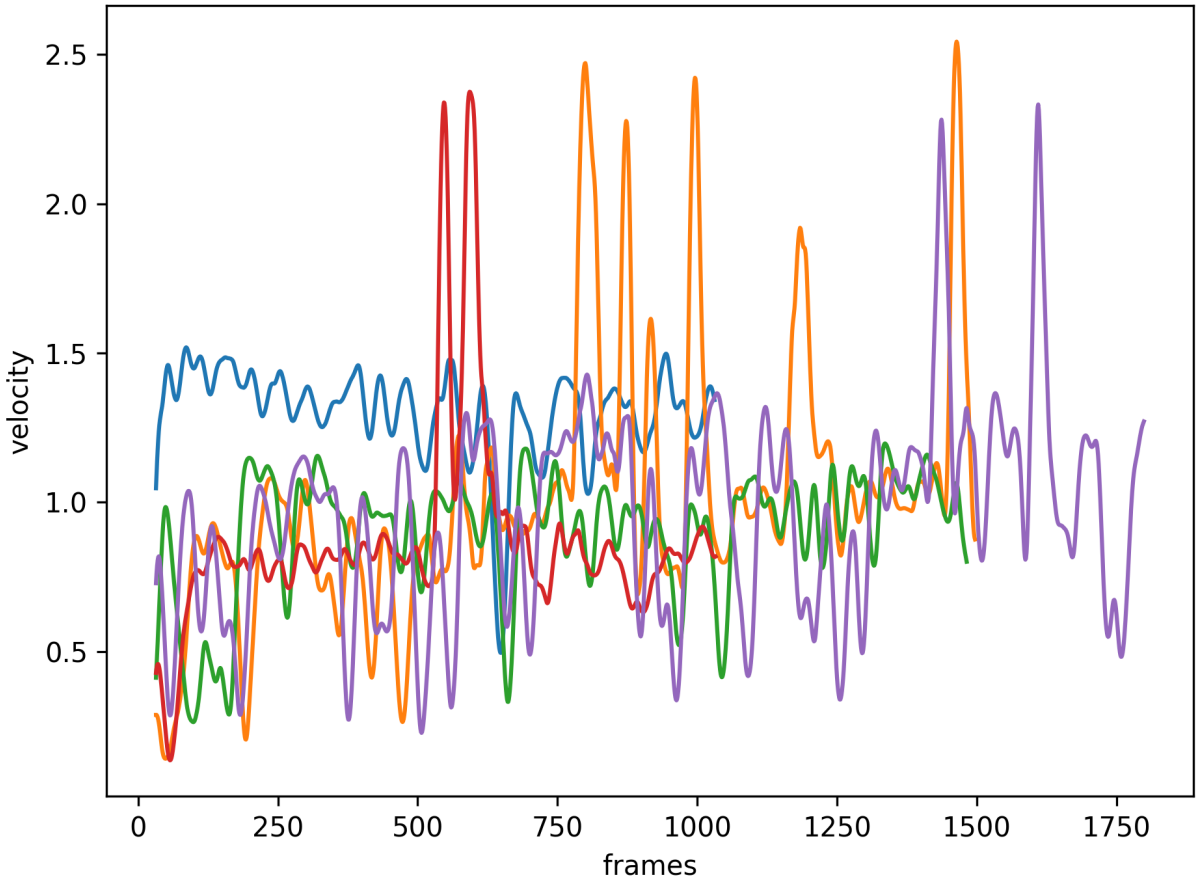
```
... ['object_1',
... 'object_100',
... 'object_101',
... 'object_102',
... 'object_103',
... ...
```

Access all parameters for a single object:

```
>>> obj1_data = exp['object_1']
```

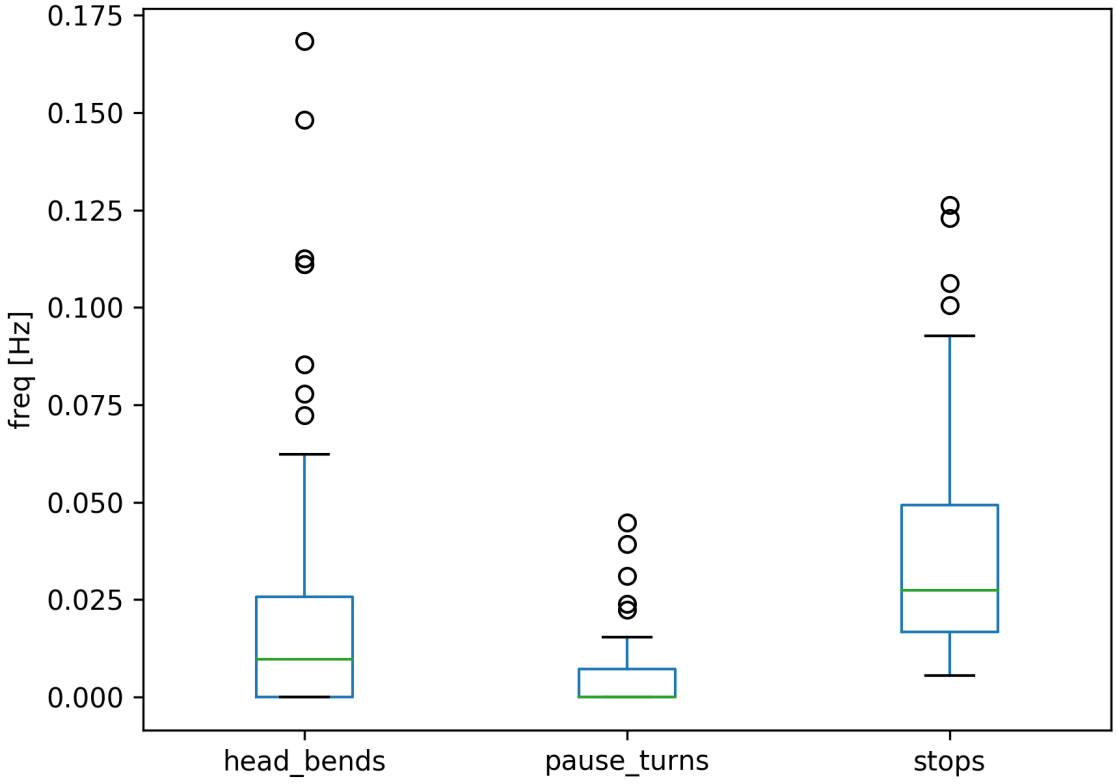
Plot velocity for the first 5 objects

```
>>> vel = exp.velocity.iloc[:,:5]
>>> # Smooth over 20 frames
>>> vel = vel.rolling(window=20).mean()
>>> # Plot over time
>>> ax = vel.plot(legend=False)
>>> ax.set_xlabel('frames')
>>> ax.set_ylabel('velocity')
>>> plt.show()
```



Plot some frequency parameters over all objects

```
>>> param_to_plot = ['head_bends','pause_turns','stops']
>>> ax = exp.mean().loc[param_to_plot].T.plot(kind='box')
>>> ax.set_ylabel('freq [Hz]')
>>> plt.show()
```



Next, lets have a look at *Collections*:

```
>>> import pyfim
>>> import matplotlib.pyplot as plt
```

```
>>> # Initialize Experiments from CSV files in two folders
>>> exp1_folder = '/experiments/genotype1'
>>> exp2_folder = '/experiments/genotype2'
>>> exp1 = pyfim.Experiment(exp1_folder)
>>> exp2 = pyfim.Experiment(exp2_folder)
```

Initialise a Collection and add the Experiments

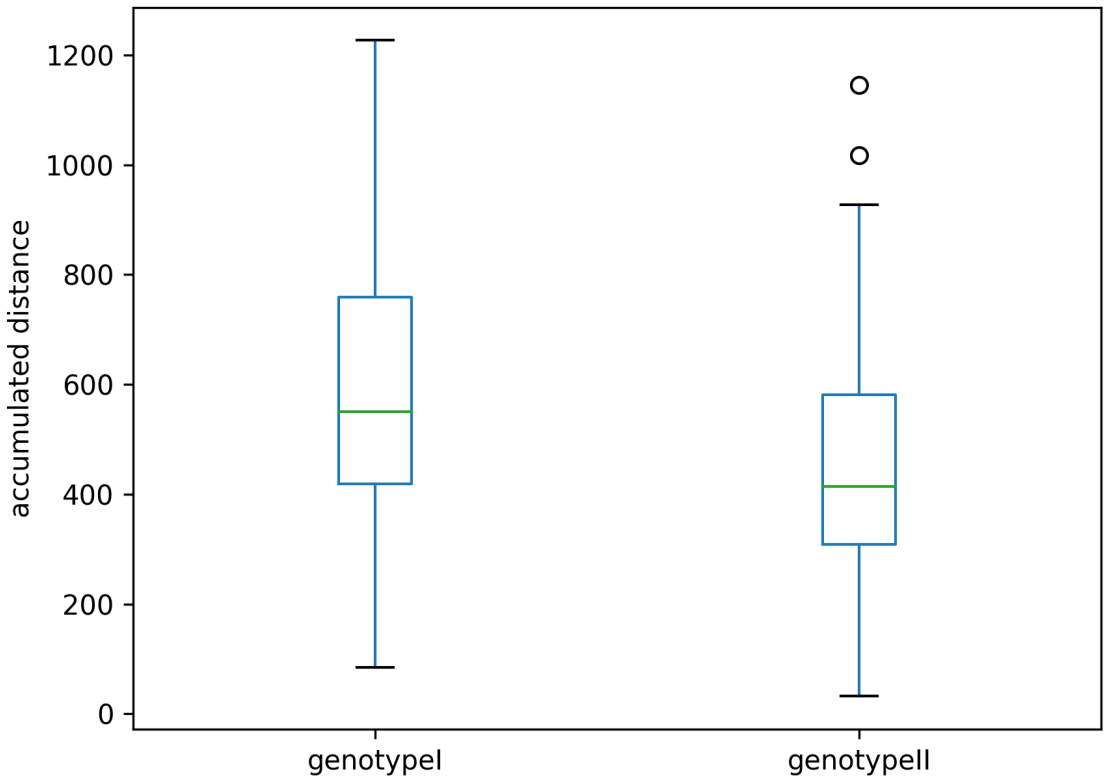
```
>>> coll = pyfim.Collection()
>>> coll.add_data(exp1, label='genotypeI')
>>> coll.add_data(exp2, label='genotypeII')
```

Get a summary of the Collection

```
>>> coll
... <class 'pyfim.core.Collection'> with 3 experiments:
...           name  n_objects  n_frames
... 0   genotypeI         46      1800
... 1   genotypeI         46      1800
... 2  genotypeII         47      1800
... Available parameters: tail_x, mom_dst, acc_dst, is_well_oriented, spinepoint_3_y,␣
↪spine_length, right_bended, spinepoint_1_x, radius_2, peristalsis_frequency, radius_
↪1, acceleration, spinepoint_1_y, area, head_bends, spinepoint_2_y, mom_y, go_phase,␣
↪peristalsis_efficiency, bending_strength, spinepoint_2_x, tail_y, spinepoint_3_x,␣
↪velocity, perimeter, pause_turns, head_x, mov_direction, left_bended, dst_to_origin,
↪ bending, head_y, is_coiled, radius_3, mom_x, stops
```

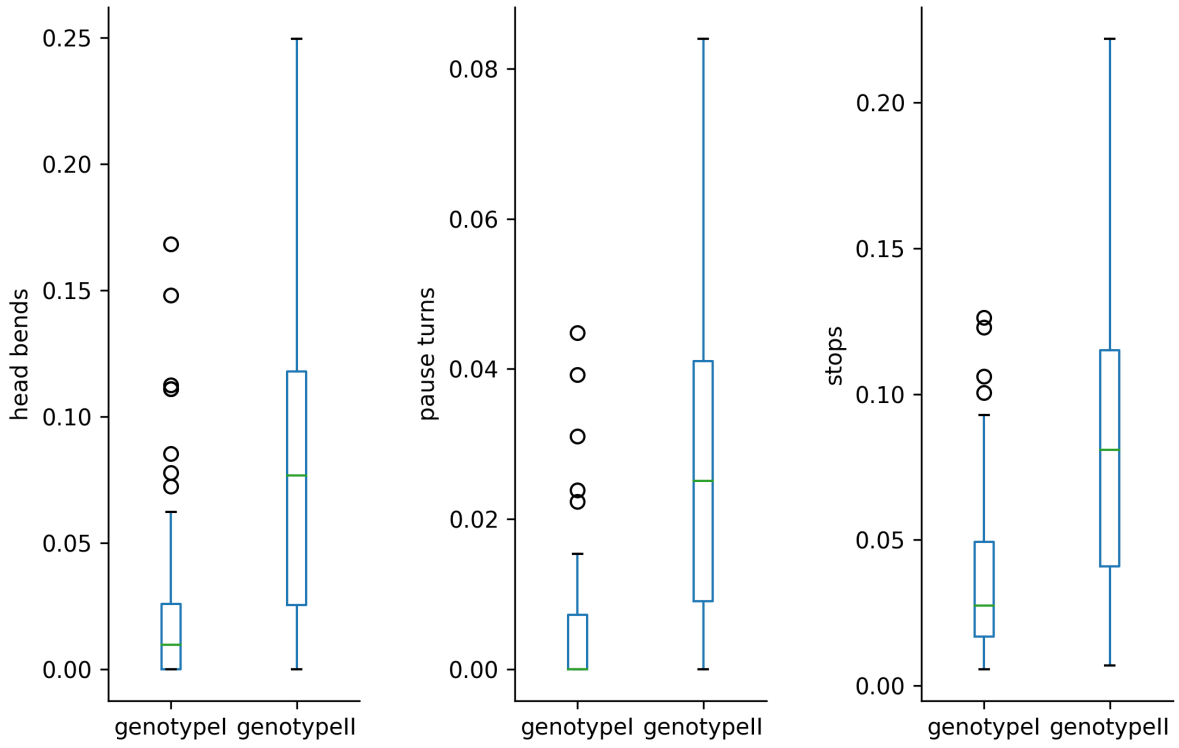
Get and plot a single parameter

```
>>> mean_acc_dst = coll.acc_dst
>>> ax = mean_acc_dst.plot(kind='box')
>>> ax.set_ylabel('accumulated distance')
>>> plt.show()
```



Collections have a built-in plotting function that lets you plot multiple parameters as boxplots

```
>>> ax = coll.plot(['head_bends','pause_turns','stops'])
>>> plt.show()
```



7.2.3 A special case: Two-Choice Experiments

In two-choice experiments objects can be split into two groups based on some parameter. Classically, you would have setup in which half the arena has different conditions than the other. For example: light vs dark or hot vs cold. For this kind of experiment you might want to:

1. Quantify how objects distribute by calculating a preference index (PI)
2. Look at individual parameters separated by which side they are on.

For this, you can use the *TwoChoiceExperiment*. This class inherits from *Experiment* - so it can do all of the stuff the base class can plus some additional stuff like calculating a PI.

An example:

```
>>> # Set split to be made along x-axis (default)
>>> pyfim.defaults['TC_PARAM'] = 'mom_x'
>>> # Set where to make the split in pixel or mm
>>> pyfim.defaults['TC_BOUNDARY'] = 1000
>>> # Create two-choice experiment
>>> tc_exp = pyfim.TwoChoiceExperiment( '/2choice-experiments/1/' )
>>> # Get preference index (PI)
>>> tc_exp.preference_index
... 0.8712
>>> # Plot PI over time
>>> tc_exp.PI_over_time.plot()
>>> # Compare other parameters between left and right side of the experiment
>>> comp = tc_exp.split_data()
>>> comp.velocity.plot()
```

Reference

Experiment(f[, keep_raw, include_subfolders])	Class that holds raw data for a set of data.
Collection()	Collection of experiments.
TwoChoiceExperiment(f[, keep_raw, ...])	Variation of *Experiment* base class that performs additional analyses.

pyfim.Experiment

class pyfim.**Experiment**(*f*, *keep_raw=False*, *include_subfolders=False*)

Class that holds raw data for a set of data.

Parameters

- **f** (*{filename, folder, file object}*) –

  Provide either:

  – a CSV file name

  – a CSV file object

  – single folder

  – list of the above

  Lists of files will be merged and objects (columns) will be renumbered.

- **keep_raw** (*bool, optional*) – If False, will discard raw data after extraction to save memory.
- **include_subfolders** (*bool, optional*) – If True and folder is provided, will also search subfolders for .csv files.

Examples

```
>>> # Generate an experiment from all csv files in one folder
>>> folder = 'users/downloads/genotype1'
>>> exp = pyfim.Experiment( folder )
>>> # See available analysis
>>> exp.parameters
... ['acc_dst', 'acceleration', 'area', 'bending',...
>>> # Access data
>>> exp.dst_to_origin.head()
...    object_1  object_13  object_15  object_18  object_19      ... 0   0.00000 ␣
↪  0.00000        NaN        NaN        NaN
... 1   2.23607    0.00000        NaN        NaN        NaN
... 2   3.60555    1.41421        NaN        NaN        NaN
... 3   3.60555    2.82843        NaN        NaN        NaN
... 4   4.47214    4.24264        0.0        NaN        NaN
>>> # Plot data individual objects over time
>>> ax = exp.dst_to_origin.plot()
>>> plt.show()
>>> # Get mean of all values
>>> exp.mean()
```

__init__(*f*, *keep_raw=False*, *include_subfolders=False*)

Methods

__init__(f[, keep_raw, include_subfolders])	
analyze(p)	Returns analysis for given parameter.
clean_data()	Cleans up the data.
extract_data()	Extracts parameters from .csv file.
mean([p])	Return mean of given parameter over given parameter.
plot_tracks([obj, ax])	Plots traces of tracked objects.
sanity_check()	Does a sanity check of attached data.

pyfim.Collection

class pyfim.**Collection**

Collection of experiments. This allows you to easily collect and plot data from multi experiments.

Examples

```
>>> # Initialise two experiments from CSVs in a folder
>>> exp1 = pyfim.Experiment( 'users/data/genotype1' )
>>> exp2 = pyfim.Experiment( 'users/data/genotype2' )
>>> # Initialise collection and add data
>>> c = pyfim.Collection()
>>> c.add_data( exp1, 'Genotype I')
>>> c.add_data( exp2, 'Genotype II')
>>> # Get a summary
>>> c
... <class 'pyfim.core.Collection'> with 2 experiments:
...    name         n_objects  n_frames
... 0  Genotype I          47      1800
... 1  Genotype II         46      1800
... Available parameters: mom_y, perimeter, peristalsis_frequency,
... radius_3, pause_turns, spinepoint_2_x, acc_dst, ...
>>> # Access data
>>> c.peristalsis_frequency
>>> # Plot as boxplot
>>> ax = c.peristalsis_frequency.plot(kind='box')
>>> plt.show()
```

__init__()

Methods

__init__()	
add_data(x[, label, keep_raw])	Add data (e.g.
extract_data()	Get the mean over all parameters.
plot([param])	Plots a set of parameters from this pyFIM Collection.
summary()	Gives a summary of the data in this analysis.

pyfim.TwoChoiceExperiment

class pyfim.**TwoChoiceExperiment**(*f*, *keep_raw=False*, *include_subfolders=False*)

Variation of *Experiment* base class that performs additional analyses.

__init__(*f*, *keep_raw=False*, *include_subfolders=False*)

Methods

__init__(f[, keep_raw, include_subfolders])	
analyze(p)	Returns analysis for given parameter.
clean_data()	Cleans up the data.
extract_data()	Extracts parameters from .csv file.
mean([p])	Return mean of given parameter over given parameter.
plot_tracks([obj, ax])	Plots traces of tracked objects.
sanity_check()	Does a sanity check of attached data.
split_data()	Split data into experiment and control.
two_choice_analyses()	Performs additional two-choice analyses.

7.3 Built-in Analyses

This section gives you an overview of the additional analyses that are performed when you initialize an experiment. They are based on MatLab code kindly shared by Dimitri Berh (Klaembt lab, University of Muenster, Germany):

7.3.1 Documentation

Base analyses

stops(exp)	Calculates frequency of stops [Hz] for each object.
stop_duration(exp)	Calculates mean duration of a stop.
pause_turns(exp)	Calculates the frequency of pause-turns [Hz] for each object.
bending_strength(exp[, during])	Calculates the median (!) bending strength for each object.
head_bends(exp)	Calculates the head bend frequency [Hz] for each object.
peristalsis_efficiency(exp)	Calculates the peristalsis efficiency for each object.
peristalsis_frequency(exp)	Calculates the peristalsis frequency [Hz] for each object.
binary_phases(x[, mode, min_len])	Low-level function: Extracts phases from binary indicators such as “go_phase” or “is_coiled”.

pyfim.analysis.stops

pyfim.analysis.**stops**(*exp*)

Calculates frequency of stops [Hz] for each object. This analysis is based on MatLab code by Dimitri Berh (University of Muenster, Germany).

Notes

This function counts the phases in which *go_phase* is zero. You can finetune this behaviour by adjusting the following parameter in the config file:

- *MIN_STOP_PHASE*: minimum number of frames for a stop-phase

Parameters **exp** (pyfim.Experiment) – Experiment holding the raw data.

Returns **Stop frequency [Hz]**

Return type pandas.DataFrame

pyfim.analysis.stop_duration

pyfim.analysis.**stop_duration**(*exp*)

Calculates mean duration of a stop. This analysis is based on MatLab code by Dimitri Berh (University of Muenster, Germany).

Notes

This function measures the average length of phases in which *go_phase* is zero. You can finetune this behaviour by adjusting the following parameter in the config file:

- *MIN_STOP_PHASE*: minimum number of frames for a stop

Parameters **exp** (pyfim.Experiment) – Experiment holding the raw data.

Returns **Mean stop duration [Frames]**

Return type pandas.DataFrame

pyfim.analysis.pause_turns

pyfim.analysis.**pause_turns**(*exp*)

Calculates the frequency of pause-turns [Hz] for each object. This analysis is based on MatLab code by Dimitri Berh (University of Muenster, Germany).

Parameters **exp** (pyfim.Experiment) – Experiment holding the raw data.

Notes

This function counts the number of pause-turns by (1) finding pauses and (2) determining if the movement direction before and after the pause differs sufficiently. You can finetune this behaviour by changing the following parameters in the config file:

- *MIN_STOP_TIME*: minimum number of frames for a pause to be counted as one.
- *MIN_GO_TIME*: minimum frames before and after the pause to be counted as pause-turn.
- *TURN_ANGLE_THRESHOLD*: minimum angular difference in movement direction before and after the pause.

Returns **Pause-Turn frequency [Hz]**

Return type pandas.DataFrame

pyfim.analysis.bending_strength

pyfim.analysis.**bending_strength**(*exp*, *during=None*)

Calculates the median (!) bending strength for each object. This analysis is based on MatLab code by Dimitri Berh (University of Muenster, Germany).

Parameters

- **exp** (pyfim.Experiment) – Experiment holding the raw data.
- **during** (*{'stop','go', None}, optional*) – Use to restrict to stop or go-phases.

Notes

This function determines the bending strength by (1) taking all bending angles, (2) thresholding them and (3) getting the median bending angle. You can finetune this behaviour using the following parameter in the config file:

- *BENDING_ANGLE_THRESHOLD_FOR_BENDING_STRENGTH*: minimum bending angle

Returns **Median bending strengths [angle]** – Returns NaN if no bends.

Return type pandas.DataFrame

pyfim.analysis.head_bends

pyfim.analysis.**head_bends**(*exp*)

Calculates the head bend frequency [Hz] for each object. This analysis is based on MatLab code by Dimitri Berh (University of Muenster, Germany).

Parameters **exp** (pyfim.Experiment) – Experiment holding the raw data.

Notes

This function determines the number of head bends by (1) taking all bending angles, (2) thresholding them and (3) counting the number of bending phases of a given minimum length. You can finetune this behaviour using the following parameters in the config file:

- *BENDING_ANGLE_THRESHOLD*: minimum bending angle
- *MIN_BENDED_PHASE*: minimum consecutive number of frames above angle threshold

Returns **Mean head bending frequencies [Hz]**

Return type pandas.DataFrame

pyfim.analysis.peristalsis_efficiency

pyfim.analysis.**peristalsis_efficiency**(*exp*)

Calculates the peristalsis efficiency for each object. The unit is depending on the input data: [pixel/peristalsis] or [mm/peristalsis]. This analysis is based on MatLab code by Dimitri Berh (University of Muenster, Germany).

Parameters **exp** (pyfim.Experiment) – Experiment holding the raw data.

Notes

This function determines the number of peristalses by performing peak detection of the object's area in its go-phases. The efficieny is the distance (in pixel or mm) per peristalsis. You can finetune this behaviour using the following parameters in the config file:

- *MIN_GO_PHASE*: minimum length of the go phases
- *MIN_PEAK_DIST*: minimal distance in frames between peristalses

Returns **Mean peristalsis efficiency [Hz]**

Return type pandas.DataFrame

pyfim.analysis.peristalsis_frequency

pyfim.analysis.**peristalsis_frequency**(*exp*)

Calculates the peristalsis frequency [Hz] for each object. This analysis is based on MatLab code by Dimitri Berh (University of Muenster, Germany).

Parameters **exp** (pyfim.Experiment) – Experiment holding the raw data.

Notes

This function determines the number of peristalses by performing peak detection of the object's area in its go-phases. You can finetune this behaviour using the following parameters in the config file:

- *MIN_GO_PHASE*: minimum length of the go phases
- *MIN_PEAK_DIST*: minimal distance in frames between peristalses

Returns **Mean peristalsis frequencies [Hz]**

Return type pandas.DataFrame

pyfim.analysis.binary_phases

pyfim.analysis.**binary_phases**(*x*, *mode='ON'*, *min_len=1*)

Low-level function: Extracts phases from binary indicators such as "go_phase" or "is_coiled".

Parameters

- **x** (*(list, np.ndarray, pd.Series)*) – Must be consist of True/False or 0/1. E.g. [0,0,0,1,1,1,0,1,1]
- **mode** (*{'ON','OFF','ALL'}, optional*) –

  Phases to return. For above example:

  - 'ON', will return [(3,6),(7,9)]
  - 'OFF', will return [(0,3),(6,7)]
  - 'ALL' will return [(0,3),(3,6),(6,7),(7,9)]

- **min_len** (*int, optional*) –

Returns

Return type Indices of phases

Two-choice analyses

preference_index(exp)	Calculates the preference index (PI) for a two choice experiment:
PI_over_time(exp)	Calculates the preference index (PI) for a two choice experiment over time:

pyfim.analysis.preference_index

pyfim.analysis.**preference_index**(*exp*)

Calculates the preference index (PI) for a two choice experiment:

PI = (exp-control)/(exp+control)

with *exp* and *control* being the number of objects on the experimental and the control side, respectively.

Based on code by Sebastian Hueckesfeld (University of Bonn, Germany).

Notes

This function counts the number of objects in rolling windows of 10s on either side of a boundary. You can finetune this behaviour by adjusting the following parameters in the config file:

- *TC_PARAM*: parameter used to split data (e.g. "mom_x" for split along x-axis)
- *TC_BOUNDARY*: boundary between control and experiment
- *TC_CONTROL_SIDE*: defines which side is the control
- *TC_COUNT_WINDOW*: rolling window (in frames) over which to count max objects
- *TC_SMOOTHING_WINDOW* : rolling window (in frames) over which to smooth PI
- *TC_CUT_HEAD*: set to ignore the first X frames for PI calculation. Can be fraction (e.g. 0.75) of total frames.
- *TC_CUT_TAIL*: set to ignore the last X frames for PI calculation. Can be fraction (e.g. 0.1) of total frames.

Parameters **exp** (pyfim.Experiment) – Experiment holding the raw data.

Returns **PI**

Return type float

pyfim.analysis.PI_over_time

pyfim.analysis.**PI_over_time**(*exp*)

Calculates the preference index (PI) for a two choice experiment over time:

PI = (exp-control)/(exp+control)

with *exp* and *control* being the number of objects on the experimental and the control side, respectively.

Based on code by Sebastian Hueckesfeld (University of Bonn, Germany).

Notes

This function counts the number of objects in rolling windows of 10s on either side of a boundary. You can finetune this behaviour by adjusting the following parameters in the config file:

- *TC_PARAM*: parameter used to split data (e.g. "mom_x" for split along x-axis)
- *TC_BOUNDARY*: boundary between control and experiment
- *TC_CONTROL_SIDE*: defines which side is the control
- *TC_COUNT_WINDOW*: rolling window (in frames) over which to count max objects
- *TC_SMOOTHING_WINDOW* : rolling window (in frames) over which to smooth PI

Parameters **exp** (pyfim.Experiment) – Experiment holding the raw data.

Returns **PI over time**

Return type pandas.DataFrame

7.4 Configuration

When you initialize a *Experiment*, raw data is extracted from the .csv(s) and cleaned-up. Then, additional analyses are performed. You can fine tune the clean up and the analyses by changing default parameters.

Upon importing pyfim, defaults are loaded from *config.py* in the pyFIM directory. You can either change the defaults in the file which will affect all subsequent sessions (persistent, does not work on-the-fly!) or change the defaults in the current session (temporary, only for this session).

7.4.1 Making lasting changes

Open a Python session, import pyFIM and get it's location:

```
>>> import pyfim
>>> pyfim.__file__
... '/Library/Frameworks/Python.framework/Versions/3.5/lib/python3.5/site-packages/
↪pyfim/__init__.py'
```

Next, navigate to the pyFIM directory, open *config.py* and make your changes.

7.4.2 Making temporary changes

You can change defaults for the current session.

```
>>> import pyfim
>>> # Defaults are stored as dictionary
>>> pyfim.defaults
... {'AREA_PARAMS': ['area'],
... 'BENDING_ANGLE_THRESHOLD': 45,
... 'BENDING_ANGLE_THRESHOLD_FOR_BENDING_STRENGTH': 20,
```

```
... 'CUT_TABLE_HEAD': False, ...
>>> # Change some parameter
>>> pyfim.defaults['MIN_STOP_TIME'] = 10
```

7.4.3 What is what

The *config.py* is well documented and superseeds this document but here is a list of relevant parameters:

Table 7.7: Config parameters

Function	Variable	Desciption
Import	*FILE_FORMAT*	File format to search for
Import	*DELIMITER*	Delimiter in CSV file
Import	*PIXEL2MM*	If True pixel coords are converted to mm or mm^2
Import	*PIXEL_PER_MM*	Adjust this according to your setup
Import	*SPATIAL_PARAMS*	List parameters that can be converted to mm
Import	*AREA_PARAMS*	List parameters that can be converted to mm^2
Import	*FPS*	Frames per second
Import	*CUT_TABLE_HEAD*	Remove first N Frames
Import	*CUT_TABLE_TAIL*	Remove last N Frames
Import	*REMOVE_NANS*	Remove objects without any values
Import	*MIN_TRACK_LENGTH*	Minimum track length in frames
Import	*FILL_GAPS*	Fill sub-threshold gaps within thresholded columns: [0 1 1 0 0 1 1] -> [0 1 1 1 1 1 1]
Import	*MAX_GAP_SIZE*	Max gap size
Import	*THRESHOLDED_PARAMS*	Parameters to fill gaps for
Head bends	*BENDING_ANGLE_THRESHOLD*	Minimum angle to be counted as bend
Head bends	*MIN_BENDED_PHASE*	Minimum consecutive frames spend bent
Stops	*MIN_STOP_PHASE*	Minimum number of frames for a stop
Peristalses	*MIN_PEAK_DIST*	Minimum frames between peristalses
Pause-turns	*MIN_STOP_TIME*	Minimum length of pause in frames
Pause-turns	*MIN_GO_TIME*	Minimum frames of go phase before and after pause
Pause-turns	*TURN_ANGLE_THRESHOLD*	Minimum anglular difference in movement direction before vs after pause
Pause-turns	*DIRECTION_SMOOTHING*	Direction will be smoother over X frames
Bend strength	*BENDING_ANGLE_THRESHOLD_FOR_BENDING_STRENGTH*	Minimum angle for bending strength
Two-Choice	*TC_PARAM*	Parameter used to split data (e.g. “mom_x” for split along x-axis)
Two-Choice	*TC_BOUNDARY*	Boundary between control and experiment
Two-Choice	*TC_CONTROL_SIDE*	Defines which side is the control
PreferenceIndex	*TC_COUNT_WINDOW*	Rolling window over which to count objects on either side
PreferenceIndex	*TC_SMOOTHING_WINDOW*	Rolling window over which to smooth preference index (PI)
PreferenceIndex	*TC_CUT_HEAD*	Ignore the first X frames for PI calculation
PreferenceIndex	*TC_CUT_TAIL*	Ignore the last X frames for PI calculation

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Index

Symbols

B

C

E

H

P

S

T