mycelyso Documentation

Release 1.0.0

Christian C. Sachs

Sep 04, 2019

Contents

See *mycelyso Readme* for information.

CHAPTER 1

Indices and tables

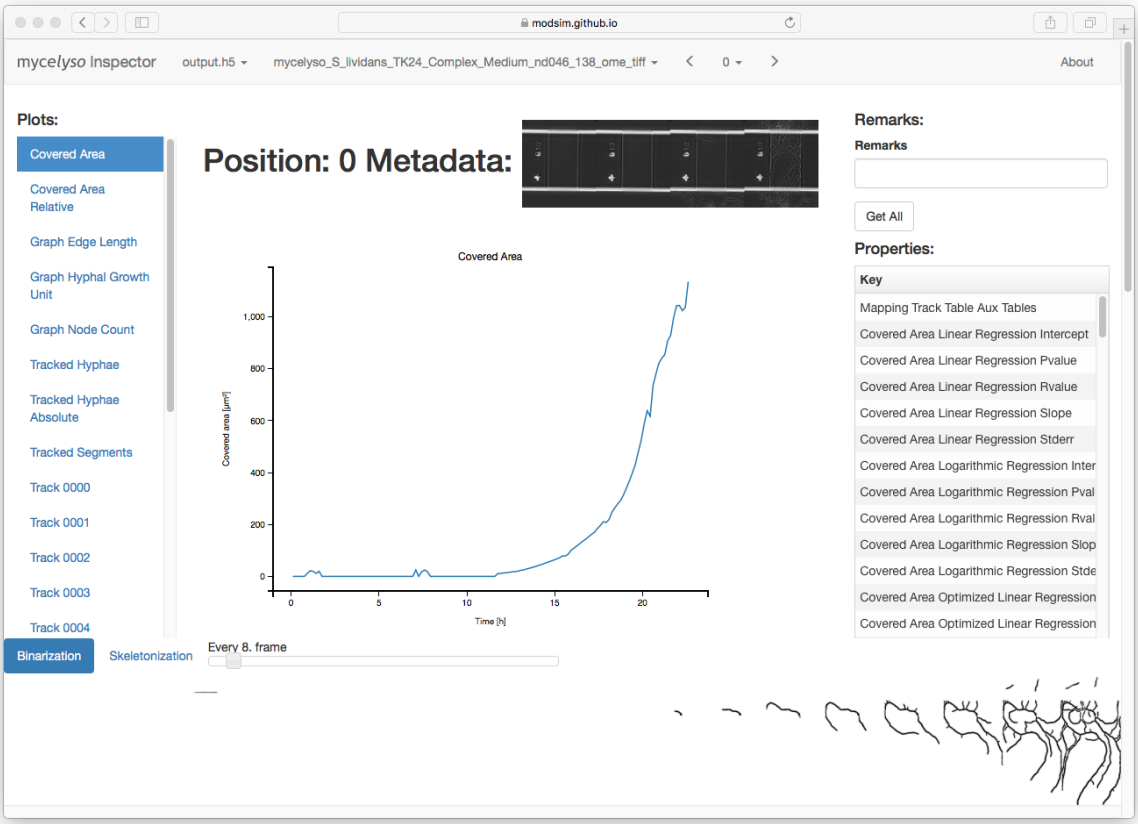
- genindex
- modindex
- search

Contents:

1.1 mycelyso Readme

1.1.1 Frontmatter

1.1.1.1 Screenshot



To quickly get a grasp what results can be generated with mycelyso, please take a look at the static demo page of mycelyso Inspector generated with the example dataset.

1.1.1.2 Installation and Analysis Tutorial Videos

These videos shows how to download and unpack *mycelyso* as well as to run a test analysis using the pre-packages Windows version of *mycelyso*.

1.1.1.3 Publication

When using *mycelyso* for scientific applications, please cite our publication:

> Sachs CC, Koepff J, Wiechert W, Grünberger A, Nöh K (2019) mycelyso – high-throughput analysis of Streptomyces mycelium live cell imaging data BMC Bioinformatics, volume 20, 452, doi: 10.1186/s12859-019-3004-1

It is available on the *BMC Bioinformatics* homepage at DOI: 10.1186/s12859-019-3004-1

1.1.1.4 Documentation

Documentation can be built using sphinx, but is available online as well at Read the Docs.

1.1.1.5 License

mycelyso is free/libre open source software under the 2-clause BSD License. See *License*

1.1.2 Getting mycelyso and Datasets

1.1.2.1 Example Datasets

You can find an example dataset deposited at zenodo DOI: 10.5281/zenodo.376281.

1.1.2.2 Ways to install mycelyso

1.1.3 Pre-Bundled Windows Application

If you don't have a Python 3 installation ready, and want to just run *mycelyso*, we you can download a pre-packaged version for 64-bit versions of Windows (mycelyso-win64.zip) from AppVeyor.

Please note, that, instead of `python -m mycelyso` or `python -m mycelyso_inspector`, the calls would then be `mycelyso.exe` or `mycelyso_inspector.exe`.

1.1.4 Docker

Please see the Docker section near the end.

1.1.5 Packages for the conda Package manager

While *mycelyso* is a pure Python package, it has some dependencies which are a bit more complex to build and might not be present in the PyPI (Python Package Index). Thankfully the conda Package manager / Anaconda environment provides all packages necessary in an easy to use manner. To use it, please download Anaconda (Miniconda could be downloaded as well, but as most packages included in Anaconda are needed anyways, it does hardly provide a size benefit).

You have to enable the necessary channels (we aim to add mycelyso to bioconda lateron):

```
> conda config --add channels conda-forge
> conda config --add channels bioconda
> conda config --add channels csachs

> conda install -y mycelyso mycelyso-inspector
```

Please note that this readme assumes you are starting with a fresh install of Anaconda/Miniconda. If you start with an existing installation, individual dependency packages might need to be updated.

1.1.6 Packages from PyPI (for advanced users)

If you have a working Python 3 installation and can eventually fix missing dependencies, you can as well use the PyPI version:

```
> pip install --user mycelyso mycelyso-inspector
```

1.1.7 Directly from github (for advanced users)

```
> pip install --user https://github.com/modsim/mycelyso/archive/master.zip␣
↪mycelyso-inspector
```

1.1.8 mycelyso Quickstart

mycelyso is packaged as a Python module, to run it, use the following syntax:

```
> python -m mycelyso
```

Which will produce the help screen:

```
mycelyso INFO
  MYCElium   anaLYsis __ SOftware
   ___   __ __________ / /_ _____ ___          Developed  2015 - 2019 by
 /  ' \/ // / __/ -_) / // (_-</ _ \ __
/_/_/_/\_, /\__/\__/_/\_, /___/\___/'  \.   Christian   C.  Sachs  at
      /___/           /___/               |
             \    `             __     ,''''   Modeling&Simulation Group
              \    `----._ _,'  `'   _/
               ---'       ''      `-'         Research  Centre  Juelich

                For more information visit: github.com/modsim/mycelyso

If you use this software in a publication, please cite our paper:

Sachs CC, Koepff J, Wiechert W, Grünberger A, Nöh K (2019)
mycelyso - high-throughput analysis of Streptomyces mycelium live cell imaging data
BMC Bioinformatics, volume 20, 452, doi: 10.1186/s12859-019-3004-1

usage: __main__.py [-h] [-m MODULES] [-n PROCESSES] [--prompt]
                   [-tp TIMEPOINTS] [-mp POSITIONS] [-t TUNABLE]
                   [--tunables-show] [--tunables-load TUNABLES_LOAD]
                   [--tunables-save TUNABLES_SAVE] [--meta META]
                   [--interactive] [--output OUTPUT]
                   input

positional arguments:
  input                 input file

optional arguments:
  -h, --help            show this help message and exit
  -m MODULES, --module MODULES
  -n PROCESSES, --processes PROCESSES
  --prompt, --prompt
  -tp TIMEPOINTS, --timepoints TIMEPOINTS
  -mp POSITIONS, --positions POSITIONS
  -t TUNABLE, --tunable TUNABLE
  --tunables-show
  --tunables-load TUNABLES_LOAD
  --tunables-save TUNABLES_SAVE
  --meta META, --meta META
  --interactive, --interactive
  --output OUTPUT, --output OUTPUT
```

To run an analysis, just pass the appropriate filename as a parameter. The desired timepoints can be selected via the `--timepoints` switch, and if the file contains multiple positions, they can be selected with `--positions`. Supported file formats are TIFF, OME-TIFF, Nikon ND2 and Zeiss CZI.

The analysis will use all cores present by default. While this is generally desirable, it might lead to consuming too much memory (as each parallel acting process needs a certain additionally amount of memory). If you notice that *mycelyso* takes up too much memory, try limiting the number of processes via `-n`. If you choose `-n 0`, the code will additionally not use the parallel subsystem (`multiprocessing`).

1.1.8.1 Running an analysis

To analyze the example dataset, run: (`-t BoxDetection=1` is used, as the spores were grown in rectangular growth chambers, which are to be detected. Otherwise, the software will use the whole image, or cropping values as set via `-t CropWidth=.../-t CropHeight=...`. If the data is pre-segmented (i.e. input is a binary image stack), choose `-t SkipBinarization=1`.

```
> python -m mycelyso S_lividans_TK24_Complex_Medium_nd046_138.ome.tiff -t␣
↪BoxDetection=1
```

Optionally, you can inspect the segmentation and produced graph on a per-frame basis before running a complete analysis, by adding the `--interactive` flag, in which case *mycelyso* will start an interactive viewer.

mycelyso stores all data compressed in HDF5 files, by default it will write a file called `output.h5` (can be changed with `--output`).

```
> ls -lh --time-style=+
total 1.3G
-rw-rw-r-- 1 sachs sachs 5.4M  output.h5
-rw-rw-r-- 1 sachs sachs 1.5G  S_lividans_TK24_Complex_Medium_nd046_138.ome.tiff
```

Multiple datasets/positions can be stored in the same file, it will only make problems if the same position is about to be analyzed twice. Binary masks/skeletons are stored in the HDF5 file, as well as GraphML representations of the tracking graphs. The HDF5 file can be investigated with standard HDF5 tools, tabular data is to be opened with pandas, as it is stored with its format.

1.1.8.2 Results visualization using mycelyso Inspector

However, since the raw data is only interesting if you want to perform custom analyses, it is much more straightforward to use the integrated visualization tool *mycelyso Inspector* as a helper to take a look at the results:

```
> python -m mycelyso_inspector
```

mycelyso Inspector will output the URL it is serving content at, and by default automatically open a browser window with it.

In *mycelyso Inspector*, you have various information displays: On the top, the HDF5 file / analyzed dataset / position can be selected. On the left, there is a list of graphs available. In the middle, there is the currently selected graph displayed. On the right, there is general information about the whole position (colony level statistics), below the main part is a table with information about individual tracks, and scrolled further down is the possibility to show individual graph tracking in 2D or a colony growth oversight in 3D. Sticky at the bottom is binarized or skeletonized timeline of the position.

The data to all graphs can be downloaded as tab separated text by pressing the right mouse button on a certain graph link (in the left menu) and choosing 'Save As'.

Information: Occasional warnings in the console about invalid values are due to missing/invalid data points, and are of no particular concern.

WARNING: *mycelyso Inspector* will serve results from all HDF5 (`.h5`) files found in the current directory via an embedded webserver. Furthermore as a research tool, no special focus was laid on security, as such, you are assumed to prevent unauthorized access to the tool if you choose to use an address accessible by third parties.

1.1.8.3 Setting calibration data for TIFF files

TIFF files provide no standard way to set temporal information per frame. To set these parameters manually, e.g. a pixel size of 0.09 µm/pixel and an acquisition interval of 600 s (10 min) use:

```
> python -m mycelyso "the_file.tif?calibration=0.09;interval=600"
```

1.1.8.4 Tunable Parameters

The analysis' internal workings are dependent upon some tunable parameters. All tunables are listed in the *tunables* documentation subpage. To check their current value, you can view them all using the `--tunables-show` command line option, which will as well print documentation. To set individual ones to a different values one can use `-t SomeTunable=NewValue`. Individual tunables are documented within the API documentation as well.

```
> python -m mycelyso --tunables-show
> python -m mycelyso -t SomeTunable=42
```

1.1.9 Docker

Docker a tool allowing for software to be run in pre-defined, encapsulated environments called containers. To run *mycelyso* via Docker, an image is used which is a self-contained Linux system with *mycelyso* installed, which can either be preloaded or will be downloaded on the fly.

Use the following commands to run mycelyso via Docker:

To analyze:

```
> docker run --tty --interactive --rm --volume `pwd`:/data --user `id -u` modsim/
↪mycelyso <parameters ...>
```

To run *mycelyso Inspector*:

```
> docker run --tty --interactive --rm --volume `pwd`:/data --user `id -u` --
↪publish 8888:8888 --entrypoint python modsim/mycelyso -m mycelyso_inspector
↪<parameters ...>
```

To run interactive mode (display on local X11, under Linux):

```
> docker run --tty --interactive --rm --volume `pwd`:/data --user `id -u` --env␣
↪DISPLAY=$DISPLAY --volume /tmp/.X11-unix:/tmp/.X11-unix modsim/mycelyso --
↪interactive <parameters ...>
```

General remarks: `--tty` is used to allocate a tty, necessary for interactive usage, like `--interactive` which connects to stdin/stdout. The `--rm` switch tells docker to remove the container (not image) again after use. As aforementioned, docker is containerized, i.e. unless explicitly stated, no communication with the outside is possible. Therefore via `--volume` the current working directory is mapped into the container.

1.1.10 Third Party Licenses

Note that this software contains the following portions from other authors, under the following licenses (all BSD-flavoured):

mycelyso/pilyso/imagestack/readers/external/czifile.py:

> **czifile.py by Christoph Gohlke, licensed BSD (see file head).** Copyright (c) 2013-2015, Christoph Gohlke, 2013-2015, The Regents of the University of California

1.2 License

1.2.1 The 2-clause BSD License

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

1.3 Example HDF5 Insights

This Jupyter Notebook should give a brief overview how to programmatically analyze the HDF5 files produced by *mycelyso*. Please note that you can always inspect these files with *mycelyso Inspector* as well, this tutorial should just give you a hint how to open these files if you might want to write your own analyses.

First, it is assumed that an `output.h5` is present in the current directory, with an analysis of the *example dataset* contained.

You can fetch the *example dataset* by running `get-dataseth.sh` or download it manually at https://zenodo.org/record/376281.

Afterwards, analyze it with:

```
> python -m mycelyso S_lividans_TK24_Complex_Medium_nd046_138.ome.tiff -t␣
↪BoxDetection=1
```

Afterwards, you will have an `output.h5` in the residing in the directory.

We will be using Pandas to read our data, while the non-tabular data could easily be read with any other HDF5 compatible tool, the tabular data is layed out in a chunked format particular to Pandas, and as such it is easiest to open it with Pandas.

First, some general setup ...

```
%matplotlib inline
%config InlineBackend.figure_formats=['svg']
import pandas
pandas.options.display.max_columns = None
import numpy as np
import networkx as nx
from networkx.readwrite import GraphMLReader

from matplotlib import pyplot, ticker
pyplot.rcParams.update({
    'figure.figsize': (10, 6), 'svg.fonttype': 'none',
    'font.sans-serif': 'Arial', 'font.family': 'sans-serif',
    'image.cmap': 'gray_r', 'image.interpolation': 'none'
})
```

1.3.1 Opening the HDF5 file

We will load the `output.h5` using `pandas.HDFStore` ...

```
store = pandas.HDFStore('output.h5', 'r')
store
```

```
<class 'pandas.io.pytables.HDFStore'>
File path: output.h5
/results/mycelyso_S_lividans_TK24_Complex_Medium_nd046_138_ome_tiff/pos_000000000_
↪t_Collected/result_table
↪                  frame        (shape->[1,208])
/results/mycelyso_S_lividans_TK24_Complex_Medium_nd046_138_ome_tiff/pos_000000000_
↪t_Collected/result_table_collected
↪                  frame        (shape->[136,27])
/results/mycelyso_S_lividans_TK24_Complex_Medium_nd046_138_ome_tiff/pos_000000000_
↪t_Collected/tables/_individual_track_table_aux_tables/track_table_aux_tables_
↪000000001             frame        (shape->[22,8])
/results/mycelyso_S_lividans_TK24_Complex_Medium_nd046_138_ome_tiff/pos_000000000_
↪t_Collected/tables/_individual_track_table_aux_tables/track_table_aux_tables_
↪000000002             frame        (shape->[29,8])
/results/mycelyso_S_lividans_TK24_Complex_Medium_nd046_138_ome_tiff/pos_000000000_
↪t_Collected/tables/_individual_track_table_aux_tables/track_table_aux_tables_
↪000000003             frame        (shape->[11,8])
/results/mycelyso_S_lividans_TK24_Complex_Medium_nd046_138_ome_tiff/pos_000000000_
↪t_Collected/tables/_individual_track_table_aux_tables/track_table_aux_tables_
↪000000004             frame        (shape->[23,8])
/results/mycelyso_S_lividans_TK24_Complex_Medium_nd046_138_ome_tiff/pos_000000000_
↪t_Collected/tables/_individual_track_table_aux_tables/track_table_aux_tables_
↪000000005             frame        (shape->[16,8])
/results/mycelyso_S_lividans_TK24_Complex_Medium_nd046_138_ome_tiff/pos_000000000_
↪t_Collected/tables/_individual_track_table_aux_tables/track_table_aux_tables_
↪000000006             frame        (shape->[14,8])
/results/mycelyso_S_lividans_TK24_Complex_Medium_nd046_138_ome_tiff/pos_000000000_
↪t_Collected/tables/_individual_track_table_aux_tables/track_table_aux_tables_
↪000000007             frame        (shape->[12,8])
/results/mycelyso_S_lividans_TK24_Complex_Medium_nd046_138_ome_tiff/pos_000000000_
↪t_Collected/tables/_individual_track_table_aux_tables/track_table_aux_tables_
↪000000008             frame        (shape->[9,8])
/results/mycelyso_S_lividans_TK24_Complex_Medium_nd046_138_ome_tiff/pos_000000000_
↪t_Collected/tables/_individual_track_table_aux_tables/track_table_aux_tables_
↪000000009             frame        (shape->[17,8])
/results/mycelyso_S_lividans_TK24_Complex_Medium_nd046_138_ome_tiff/pos_000000000_
↪t_Collected/tables/_individual_track_table_aux_tables/track_table_aux_tables_
↪000000010             frame        (shape->[11,8])
/results/mycelyso_S_lividans_TK24_Complex_Medium_nd046_138_ome_tiff/pos_000000000_
↪t_Collected/tables/_individual_track_table_aux_tables/track_table_aux_tables_
↪000000011             frame        (shape->[8,8])
/results/mycelyso_S_lividans_TK24_Complex_Medium_nd046_138_ome_tiff/pos_000000000_
↪t_Collected/tables/_individual_track_table_aux_tables/track_table_aux_tables_
↪000000012             frame        (shape->[7,8])
/results/mycelyso_S_lividans_TK24_Complex_Medium_nd046_138_ome_tiff/pos_000000000_
↪t_Collected/tables/_individual_track_table_aux_tables/track_table_aux_tables_
↪000000013             frame        (shape->[10,8])
/results/mycelyso_S_lividans_TK24_Complex_Medium_nd046_138_ome_tiff/pos_000000000_
↪t_Collected/tables/_individual_track_table_aux_tables/track_table_aux_tables_
↪000000014             frame        (shape->[5,8])
/results/mycelyso_S_lividans_TK24_Complex_Medium_nd046_138_ome_tiff/pos_000000000_
↪t_Collected/tables/_individual_track_table_aux_tables/track_table_aux_tables_
↪000000015             frame        (shape->[7,8])
/results/mycelyso_S_lividans_TK24_Complex_Medium_nd046_138_ome_tiff/pos_000000000_
↪t_Collected/tables/_individual_track_table_aux_tables/track_table_aux_tables_
↪000000016             frame        (shape->[5,8])
/results/mycelyso_S_lividans_TK24_Complex_Medium_nd046_138_ome_tiff/pos_000000000_
↪t_Collected/tables/_individual_track_table_aux_tables/track_table_aux_tables_
↪000000017             frame        (shape->[7,8])
```

(continues on next page)

(continued from previous page)

```
/results/mycelyso_S_lividans_TK24_Complex_Medium_nd046_138_ome_tiff/pos_000000000_
↪t_Collected/tables/_individual_track_table_aux_tables/track_table_aux_tables_
↪000000018            frame        (shape->[8,8])
/results/mycelyso_S_lividans_TK24_Complex_Medium_nd046_138_ome_tiff/pos_000000000_
↪t_Collected/tables/_individual_track_table_aux_tables/track_table_aux_tables_
↪000000019            frame        (shape->[8,8])
/results/mycelyso_S_lividans_TK24_Complex_Medium_nd046_138_ome_tiff/pos_000000000_
↪t_Collected/tables/_individual_track_table_aux_tables/track_table_aux_tables_
↪000000020            frame        (shape->[7,8])
/results/mycelyso_S_lividans_TK24_Complex_Medium_nd046_138_ome_tiff/pos_000000000_
↪t_Collected/tables/_mapping_track_table_aux_tables/track_table_aux_tables_
↪000000000               frame        (shape->[20,2])
/results/mycelyso_S_lividans_TK24_Complex_Medium_nd046_138_ome_tiff/pos_000000000_
↪t_Collected/tables/track_table/track_table_000000000
↪                 frame        (shape->[20,66])
```

Now let's dive a bit into the HDF5 file.

Remember that HDF5 stands for *Hierarchical* Data Format 5 . . .

```
root = store.get_node('/')

print("Root:")
print(repr(root))
print()
print("/results:")
print(repr(root.results))
```

```
Root:
/ (RootGroup) ''
  children := ['results' (Group)]

/results:
/results (Group) ''
  children := ['mycelyso_S_lividans_TK24_Complex_Medium_nd046_138_ome_tiff'
↪(Group)]
```

The key names are dependent on the on-disk path of the analyzed file. Assuming there is only one file analyzed with one position in the file, we pick the first . . .

```
for image_file in root.results:
    print(image_file)
    for position in image_file:
        print(position)
        break
```

```
/results/mycelyso_S_lividans_TK24_Complex_Medium_nd046_138_ome_tiff (Group) ''
/results/mycelyso_S_lividans_TK24_Complex_Medium_nd046_138_ome_tiff/pos_000000000_
↪t_Collected (Group) ''
```

We can now investigate what data is available for that particular position

There is e.g., (binary) data, there are images, and there are various tabular datasets

```
print("data")
print(position.data)
for node in position.data:
    print(node)

print()
```

(continues on next page)

(continued from previous page)

```
print("nodes")
print(position.images)
for node in position.images:
    print(node)

print()
```

```
data
/results/mycelyso_S_lividans_TK24_Complex_Medium_nd046_138_ome_tiff/pos_000000000_
↪t_Collected/data (Group) ''
/results/mycelyso_S_lividans_TK24_Complex_Medium_nd046_138_ome_tiff/pos_000000000_
↪t_Collected/data/banner (Group) ''
/results/mycelyso_S_lividans_TK24_Complex_Medium_nd046_138_ome_tiff/pos_000000000_
↪t_Collected/data/graphml (Group) ''
/results/mycelyso_S_lividans_TK24_Complex_Medium_nd046_138_ome_tiff/pos_000000000_
↪t_Collected/data/overall_graphml (Group) ''
/results/mycelyso_S_lividans_TK24_Complex_Medium_nd046_138_ome_tiff/pos_000000000_
↪t_Collected/data/tunables (Group) ''
/results/mycelyso_S_lividans_TK24_Complex_Medium_nd046_138_ome_tiff/pos_000000000_
↪t_Collected/data/version (Group) ''

nodes
/results/mycelyso_S_lividans_TK24_Complex_Medium_nd046_138_ome_tiff/pos_000000000_
↪t_Collected/images (Group) ''
/results/mycelyso_S_lividans_TK24_Complex_Medium_nd046_138_ome_tiff/pos_000000000_
↪t_Collected/images/binary (Group) ''
/results/mycelyso_S_lividans_TK24_Complex_Medium_nd046_138_ome_tiff/pos_000000000_
↪t_Collected/images/skeleton (Group) ''
```

1.3.2 Accessing Graph and Image Data

Let's for example start with pulling out an image from the file, and displaying it ...

```
binary_images = list(position.images.binary)
skeleton_images = list(position.images.skeleton)

n = 120

total = len(binary_images)
assert 0 <= n < total

print("Total count of images: %d" % (total,))

fig, (ax_l, ax_r) = pyplot.subplots(1, 2, sharey=True)

fig.suptitle('Images of Timepoint #%d:' % (n,))

ax_l.imshow(binary_images[n])
ax_l.set_title('Binary Image')

ax_r.imshow(skeleton_images[n])
ax_r.set_title('Skeleton')
```

```
Total count of images: 136
```

```
Text(0.5,1,'Skeleton')
```

Let's now take a look at the graph data present for the position, display it and overlay it onto the image data ...

```
# The graph structure is saved in GraphML
draw_parameters = dict(node_size=25, node_color='darkgray', linewidths=0, edge_
↪color='darkgray', with_labels=False)

#graphml_data = list([np.array(graphml).tobytes() for graphml in list(position.
↪data.graphml)])
graphml_data = list(position.data.graphml)

graph, = GraphMLReader()(string=np.array(graphml_data[n]).tobytes())

# the following draw function needs separate positions...
# each node has its position saved as attributes:

example_node_id = list(sorted(graph.node.keys()))[1]

print("Example node: %s: %r" % (example_node_id, graph.node[example_node_id],))

other_node_id = list(sorted(graph.adj[example_node_id].keys(), reverse=True))[0]

print("Some other node: %s" % (other_node_id,))


print("The distance between the two nodes is: %.2f px" % (graph.adj[example_node_
↪id][other_node_id]['weight']))

pyplot.title('Graph Representation of Timepoint #%d:' % (n,))

# first draw the graph,
pos = {n_id: (n['x'], n['y']) for n_id, n in graph.node.items()}
nx.draw_networkx(graph, pos=pos, **draw_parameters)

example_nodes = [graph.node[node_id] for node_id in [example_node_id, other_node_
↪id]]

# mark on top the two choosen sample nodes
pyplot.scatter([p['x'] for p in example_nodes], [p['y'] for p in example_nodes],␣
↪zorder=2)

# then show the corresponding binarized image
pyplot.imshow(binary_images[n])
```

```
Example node: 1: {'x': 543.0, 'y': 91.0}
Some other node: 4
The distance between the two nodes is: 192.05 px
```

```
<matplotlib.image.AxesImage at 0x7f89d9770128>
```

1.3.3 Accessing Tabular Data

In the next few cells we'll take a look at the tabular data stored in the HDF5 file.

There is for example the `result_table`, which contains compounded information about the whole position:

```
result_table = store[position.result_table._v_pathname]
result_table
```

Then there is the `result_table_collected`, which contains collected information about every single frame of the time series of one position:

```
result_table_collected = store[position.result_table_collected._v_pathname]
result_table_collected
```

The per-frame informations contain e.g. the graph length (i.e. the mycelium length), which can be plotted over time:

```
timepoint = result_table_collected.timepoint / (60*60)
length = result_table_collected.graph_edge_length

pyplot.title('Length over Time')

pyplot.xlabel('Time [h]')
pyplot.ylabel('Length [µm]')

pyplot.plot(timepoint, length)
```

```
[<matplotlib.lines.Line2D at 0x7f89d964edd8>]
```

Last but not least, we will look at mycelium level tracking data in the `track_table`. The `track_table` is a level deeper in the HDF5 structure, next to tables with individual tracks.

```
track_table = store[list(position.tables.track_table)[0]._v_pathname]
track_table
```

Let's find the longest track and try to visualize it:

```
track_table.sort_values(by=['count'], ascending=False, inplace=True)
particular_tracking_table = track_table.aux_table[0]  # the first

_mapping_track_table_aux_tables = store[list(position.tables._mapping_track_table_
↪aux_tables)[0]._v_pathname]

index = _mapping_track_table_aux_tables.query('_index == @particular_tracking_table
↪').individual_table

the_longest_track = store[getattr(position.tables._individual_track_table_aux_
↪tables, 'track_table_aux_tables_%09d' % (index,))._v_pathname]

the_longest_track
```

```
timepoint = the_longest_track.timepoint / (60*60)
length = the_longest_track.distance

pyplot.title('Length over Time')

pyplot.xlabel('Time [h]')
pyplot.ylabel('Length [µm]')

pyplot.plot(timepoint, length)
```

```
[<matplotlib.lines.Line2D at 0x7f89d9621470>]
```

Now all tracked hyphae:

```
pyplot.title('Length over Time')

pyplot.xlabel('Time [h]')
```

(continues on next page)

(continued from previous page)

```
pyplot.ylabel('Length [µm]')

for idx, row in track_table.iterrows():
    particular_tracking_table = int(row.aux_table)
    index = _mapping_track_table_aux_tables.query('_index == @particular_tracking_
↪table').individual_table
    track = store[getattr(position.tables._individual_track_table_aux_tables,
↪'track_table_aux_tables_%09d' % (index,))._v_pathname]

    timepoint = track.timepoint / (60*60)
    length = track.distance - track.distance.min()
    pyplot.plot(timepoint, length)

pyplot.xlim(0, None)
```

```
(0, 22.961576228743152)
```

1.4 Example Alternative Growth Fit

Please first see the other Jupyter Notebook, explaining the basics of accessing mycelyso's HDF5 files. Furthermore, this file assumes the `output.h5` described in the other notebook to be present in the current directory.

Within this notebook, we will fit the mycelium length data using a third-party library, croissance (DOI: 10.5281/zenodo.229905 by Lars Schöning (2017)). Please install the current version off github first:

```
pip install https://github.com/biosustain/croissance/archive/master.zip
```

First, some general setup ...

```
%matplotlib inline
%config InlineBackend.figure_formats=['svg']
import pandas
pandas.options.display.max_columns = None
import numpy as np
import warnings
import croissance
from croissance.figures import PDFWriter as CroissancePDFWriter
from matplotlib import pyplot

class OutputInstead:
    @classmethod
    def savefig(cls, fig):
        pyplot.gcf().set_size_inches(10, 12)
        pyplot.show()

# croissance's PDFWriter is supposed to write to a PDF
# but we want an inline figure, so we mock some bits
CroissancePDFWriter.doc = OutputInstead
CroissancePDFWriter._include_shifted_exponentials = False
def display_result(result, name="Mycelium Length"):
    return CroissancePDFWriter.write(CroissancePDFWriter, name, result)

warnings.simplefilter(action='ignore', category=FutureWarning)

pyplot.rcParams.update({
    'figure.figsize': (10, 6), 'svg.fonttype': 'none',
    'font.sans-serif': 'Arial', 'font.family': 'sans-serif',
```

(continues on next page)

(continued from previous page)

```
    'image.cmap': 'gray_r', 'image.interpolation': 'none'
})
```

1.4.1 Opening the HDF5 file

We will load the `output.h5` using `pandas.HDFStore` ...

```
store = pandas.HDFStore('output.h5', 'r')
root = store.get_node('/')
for image_file in root.results:
    print(image_file)
    for position in image_file:
        print(position)
        break
```

```
/results/mycelyso_S_lividans_TK24_Complex_Medium_nd046_138_ome_tiff (Group) ''
/results/mycelyso_S_lividans_TK24_Complex_Medium_nd046_138_ome_tiff/pos_000000000_
↪t_Collected (Group) ''
```

and load the first growth curve

```
result_table_collected = store[position.result_table_collected._v_pathname]
timepoint = result_table_collected.timepoint / (60*60)
length = result_table_collected.graph_edge_length

pyplot.title('Length over Time')

pyplot.xlabel('Time [h]')
pyplot.ylabel('Length [µm]')

pyplot.plot(timepoint, length)
```

```
[<matplotlib.lines.Line2D at 0x7fb7dd6e5160>]
```

Here, we will use the third party tool `croissance` to fit the data to an exponential growth model:

```
curve = pandas.Series(data=np.array(length), index=np.array(timepoint))

estimator = croissance.Estimator()
result = estimator.growth(curve)
# print(result)
print(result.growth_phases)
```

```
[GrowthPhase(start=9.928127173487246, end=22.594538504723342, slope=0.
↪36693539043981077, intercept=3.1588539520729086, n0=-25.525547240977755,␣
↪attributes={'SNR': 172.5009988033075, 'rank': 100.0})]
```

And furthermore use its plotting functionality to show the results:

```
print("Growth rate as determined by croissance µ=%.2f" % (result.growth_phases[0].
↪slope,))
display_result(result)
```

```
Growth rate as determined by croissance µ=0.37
```

1.5 mycelyso Developer Documentation

1.5.1 Subpackages

1.5.1.1 mycelyso.tunables

Module contents

This file contains all the tunables available in mycelyso.

You can set them via `-t Name=value` on the command line.

class mycelyso.tunables.**BorderArtifactRemovalBorderSize**
Bases: tunable.tunable.Tunable

Remove structures, whose centroid lies within that distance [µm] of a border

default = 10.0

value = 10.0

class mycelyso.tunables.**BoxDetection**
Bases: tunable.tunable.Tunable

Whether to run the rectangular microfluidic growth structure detection as ROI detection

default = False

value = False

class mycelyso.tunables.**CleanUpGaussianSigma**
Bases: tunable.tunable.Tunable

Clean up step: Sigma [µm] used for Gaussian filter

default = 0.075

value = 0.075

class mycelyso.tunables.**CleanUpGaussianThreshold**
Bases: tunable.tunable.Tunable

Clean up step: Threshold used after Gaussian filter (values range from 0 to 1)

default = 0.5

value = 0.5

class mycelyso.tunables.**CleanUpHoleFillSize**
Bases: tunable.tunable.Tunable

Clean up step: Maximum size of holes [μm^2] which will be filled

default = 1.0

value = 1.0

class mycelyso.tunables.**CropHeight**
Bases: tunable.tunable.Tunable

Crop value (vertical) of the image [pixels]

default = 0

value = 0

class mycelyso.tunables.**CropWidth**
Bases: tunable.tunable.Tunable

Crop value (horizontal) of the image [pixels]

default = 0

value = 0

class mycelyso.tunables.**NodeEndpointMergeRadius**
Bases: tunable.tunable.Tunable

Radius in which endpoints are going to be merged [µm]

default = 0.5

value = 0.5

class mycelyso.tunables.**NodeJunctionMergeRadius**
Bases: tunable.tunable.Tunable

Radius in which junctions are going to be merged [µm]

default = 0.5

value = 0.5

class mycelyso.tunables.**NodeLookupCutoffRadius**
Bases: tunable.tunable.Tunable

Radius at which nodes will be ignored if they are further away [µm]

default = 2.5

value = 2.5

class mycelyso.tunables.**NodeLookupRadius**
Bases: tunable.tunable.Tunable

Radius in which nodes will be searched for found pixel structures [µm]

default = 0.5

value = 0.5

class mycelyso.tunables.**NodeTrackingEndpointShiftRadius**
Bases: tunable.tunable.Tunable

Maximum search radius for endpoints [$\mu m \cdot h^{1}$]

default = 100.0

value = 100.0

class mycelyso.tunables.**NodeTrackingJunctionShiftRadius**
Bases: tunable.tunable.Tunable

Maximum search radius for junctions [$\mu m \cdot h^{1}$]

default = 5.0

value = 5.0

class mycelyso.tunables.**RemoveSmallStructuresSize**
Bases: tunable.tunable.Tunable

Remove structures up to this size [μm^2]

default = 10.0

value = 10.0

class mycelyso.tunables.**SkipBinarization**
Bases: tunable.tunable.Tunable

Whether to directly use the input image as binary mask. Use in case external binarization is desired.

default = False

value = False

class mycelyso.tunables.**StoreImage**
Bases: tunable.tunable.Tunable

Whether to store images in the resulting HDF5. This leads to a potentially much larger output file.

default = False

value = False

class mycelyso.tunables.**ThresholdingParameters**
Bases: tunable.tunable.Tunable

Parameters for the used binarization method, passed as key1:value1,key2:value2,... string

default = ''

value = ''

class mycelyso.tunables.**ThresholdingTechnique**
Bases: tunable.tunable.Tunable

Binarization method to use, for available methods see documentation (mycelyso.processing.binarization)

default = 'experimental_thresholding'

classmethod test(*value*)

value = 'experimental_thresholding'

class mycelyso.tunables.**TrackingMaximumCoverage**
Bases: tunable.tunable.Tunable

Tracking, maximum covered area ratio at which tracking is still performed

default = 0.2

value = 0.2

class mycelyso.tunables.**TrackingMaximumRelativeShrinkage**
Bases: tunable.tunable.Tunable

Tracking, maximal relative shrinkage

default = 0.2

value = 0.2

class mycelyso.tunables.**TrackingMaximumTipElongationRate**
Bases: tunable.tunable.Tunable

Tracking, maximum tip elongation rate [$\mu m \cdot h^1$]

default = 100.0

value = 100.0

class mycelyso.tunables.**TrackingMinimalGrownLength**
Bases: tunable.tunable.Tunable

Tracking, minimal hyphae gained length in track filter [µm]

default = 5.0

value = 5.0

class mycelyso.tunables.**TrackingMinimalMaximumLength**
Bases: tunable.tunable.Tunable

Tracking, minimal hyphae end length in track filter [µm]

default = 10.0

value = 10.0

class mycelyso.tunables.**TrackingMinimumTipElongationRate**
Bases: tunable.tunable.Tunable

Tracking, minimum tip elongation rate [µm·h[1]]

default = -0.0

value = -0.0

class mycelyso.tunables.**TrackingMinimumTrackedPointCount**
Bases: tunable.tunable.Tunable

Tracking, minimal time steps in track filter [#]

default = 5

value = 5

1.5.1.2 mycelyso.highlevel package

Submodules

mycelyso.highlevel.nodeframe module

The nodeframe module contains the NodeFrame class, a representation of the graph of one time lapse frame.

class mycelyso.highlevel.nodeframe.**NodeFrame**(*pf*)
Bases: object

Node frame is a representation of an image stack frame on the graph/node level, it populates its values from a PixelFrame passed.

cleanup_adjacency()
Cleans up the adjacency matrix after alterations on the node level have been performed.

Returns

cycles
Detects whether a cycle exists in the graph.

Returns

generate_derived_data()
Generates derived data from the current adjacency matrix.

Derived data are shortest paths, as well as connected components.

Returns

get_connected_nodes(*some_node*)
Get all nodes which are (somehow) connected to node some_node.

Parameters **some_node** –

Returns

get_networkx_graph(*with_z=0*, *return_positions=False*)
Convert the adjacency matrix based internal graph representation to a networkx graph representation.

Positions are additionally set based upon the pixel coordinate based positions of the nodes.

Parameters

- **with_z** – Whether z values should be set based upon the timepoint the nodes appear on
- **return_positions** – Whether positions should be returned jointly with the graph

Returns

get_path(*start_node*, *end_node*)
Walks from start_node to end_node in the graph and returns the list of nodes (including both).

Parameters

- **start_node** –
- **end_node** –

Returns

is_endpoint(*i*)
Returns whether node i is an endpoint.

Parameters **i** –

Returns

is_junction(*i*)
Returns whether node i is a junction.

Parameters **i** –

Returns

prepare_graph(*pf*)
Prepares the graph from the data stored in the PixelFrame pf. :param pf: PixelFrame :return:

track(*successor*)
Tracks nodes on this frame to nodes on a successor frame.

Parameters **successor** –

Returns

mycelyso.highlevel.pipeline module

The pipeline module contains the mycelyso-Pipeline, assembled from various functions.

class mycelyso.highlevel.pipeline.**Mycelyso**
Bases: mycelyso.pilyso.application.application.App

The Mycelyso App, implementing a pilyso App.

arguments(*argparser*)

handle_args()

interactive_run()

options()

class mycelyso.highlevel.pipeline.**MycelysoPipeline**(*args*)
Bases: mycelyso.pilyso.pipeline.pipeline.PipelineExecutionContext

The MycelysoPipeline, defining the pipeline (with slight alterations based upon arguments passed via command line).

mycelyso.highlevel.pixelframe module

The pixelframe module contains the PixelFrame class, a representation of one time lapse frame at the binary mask/pixel level.

class mycelyso.highlevel.pixelframe.**PixelFrame**(*image*, *timepoint=0.0*, *calibration=1.0*)
Bases: object

A PixelFrame represents the pixel graph of one (skeletonized) image stack frame.

create_graph()
Creates the graph from the pixel skeleton.

Returns

mycelyso.highlevel.steps module

The steps module contains most of the individual, albeit mycelyso-specific processing steps.

mycelyso.highlevel.steps.**binarize**(*image*, *binary=None*)
Binarizes the input image using the experimental thresholding technique.

Parameters

- **image** –
- **binary** –

Returns

mycelyso.highlevel.steps.**clean_up**(*calibration*, *binary*)
Cleans up the image by removing holes smaller than the configured size.

Parameters

- **calibration** –
- **binary** –

Returns

```
>>> clean_up(0.1, np.array([[ True,  True,  True],
...                         [ True, False,  True],
...                         [ True,  True,  True]]))
array([[ True,  True,  True],
       [ True,  True,  True],
       [ True,  True,  True]])
```

mycelyso.highlevel.steps.**convert_to_nodes**(*skeleton*, *timepoint*, *calibration*, *pixel_frame=None*, *node_frame=None*)
Passes the input skeleton into a PixelFrame and instantiates a NodeFrame based upon that.

Parameters

- **skeleton** –
- **timepoint** –
- **calibration** –
- **pixel_frame** –
- **node_frame** –

Returns

mycelyso.highlevel.steps.**generate_graphml**(*node_frame*, *result*)
Generates a GraphML representation of a particular frame.

Parameters

- **node_frame** –
- **result** –

Returns

mycelyso.highlevel.steps.**generate_overall_graphml**(*collected*, *result*)
Generates a GraphML representation of the whole graph of one image stack.

Parameters

- **collected** –
- **result** –

Returns

mycelyso.highlevel.steps.**graph_statistics**(*node_frame*, *result=None*)

Adds some information about the graph to the results.

Parameters

- **node_frame** –
- **result** –

Returns

```
>>> pf = PixelFrame(np.array([[0, 0, 0],
...                           [1, 1, 1],
...                           [0, 0, 0]]), calibration=15.0)
>>> sorted(graph_statistics(NodeFrame(pf)).items())
[('graph_edge_count', 1.0), ('graph_edge_length', 30.0), ('graph_endpoint_count
↪', 2), ('graph_junction_count', 0), ('graph_node_count', 2)]
```

mycelyso.highlevel.steps.**image_statistics**(*image*, *calibration*, *result=None*)

Adds some numeric image parameters (i.e. size) to the results.

Parameters

- **image** –
- **calibration** –
- **result** –

Returns

```
>>> sorted(image_statistics(np.array([[0, 0, 0],
...                                   [0, 0, 0],
...                                   [0, 0, 0]]), calibration=15.0).items())
[('area', 2025.0), ('area_pixel', 9), ('input_height', 45.0), ('input_height_
↪pixel', 3), ('input_width', 45.0), ('input_width_pixel', 3)]
```

mycelyso.highlevel.steps.**individual_tracking**(*collected*, *tracked_fragments=None*, *tracked_fragments_fates=None*)

After correspondence has been established by NodeFrame#track, reconstructs growing paths over time.

Parameters

- **collected** –
- **tracked_fragments** –
- **tracked_fragments_fates** –

Returns

mycelyso.highlevel.steps.**prepare_position_regressions**(*collected*, *result*)

Prepares some regressions over parameters collected per position over time.

Parameters

- **collected** –
- **result** –

Returns

mycelyso.highlevel.steps.**prepare_tracked_fragments**(*collected*, *tracked_fragments*, *tracked_fragments_fates*, *track_table=None*, *track_table_aux_tables=None*)

Filters and converts tracked growing segments to result datasets.

Parameters

- **collected** –
- **tracked_fragments** –
- **tracked_fragments_fates** –
- **track_table** –
- **track_table_aux_tables** –

Returns

mycelyso.highlevel.steps.**qimshow**(*image*, *cmap='gray'*)

Debug function, quickly shows the passed image via matplotlibs imshow-facilities.

Parameters

- **image** –
- **cmap** –

Returns

mycelyso.highlevel.steps.**quantify_binary**(*binary*, *calibration*, *result=None*)

Adds some information about the binary image (i.e. covered ratio, area ...) to the results.

Parameters

- **binary** –
- **calibration** –
- **result** –

Returns

```
>>> sorted(quantify_binary(np.array([[0, 0, 0],
...                                  [1, 1, 1],
...                                  [0, 0, 0]]), calibration=15.0).items())
[('covered_area', 675.0), ('covered_area_pixel', 3), ('covered_ratio', 0.
↪3333333333333333)]
```

mycelyso.highlevel.steps.**remove_border_artifacts**(*calibration*, *binary*)

Removes structures, which are most likely artifacts because their centroid lies near the border.

Parameters

- **calibration** –
- **binary** –

Returns

```
>>> remove_border_artifacts(0.1, np.array([[ False, False, False],
...                                        [ False, False,  True],
...                                        [ False, False,  True]]))
array([[False, False, False],
       [False, False, False],
       [False, False, False]])
```

mycelyso.highlevel.steps.**remove_small_structures**(*calibration*, *binary*)

Cleans up the image by removing structures smaller than the configured size.

Parameters

- **calibration** –
- **binary** –

Returns

```
>>> remove_small_structures(0.1, np.array([[ False, False, False],
...                                        [ False, False,  True],
...                                        [ False, False,  True]]))
array([[False, False, False],
       [False, False, False],
       [False, False, False]])
```

mycelyso.highlevel.steps.**set_empty_crops**(*image*, *crop_t=None*, *crop_b=None*, *crop_l=None*, *crop_r=None*)

Defines crop parameters based upon image size, effectively not cropping at all.

Parameters

- **image** –
- **crop_t** –
- **crop_b** –
- **crop_l** –
- **crop_r** –

Returns

mycelyso.highlevel.steps.**skeletonize**(*binary*, *skeleton=None*)

Skeletonizes the image using scikit-image's skeletonize function.

Parameters

- **binary** –
- **skeleton** –

Returns

```
>>> skeletonize(np.array([[0, 0, 1, 1],
...                       [0, 0, 1, 1],
...                       [0, 0, 1, 1],
...                       [0, 0, 1, 1]]))
array([[False, False, False, False],
       [False, False,  True, False],
       [False, False,  True, False],
       [False, False, False, False]])
```

mycelyso.highlevel.steps.**skip_if_image_is_below_size**(*min_height=4*, *min_width=4*)

Raises a Skip exception if the image size falls below the set image size.

Parameters

- **min_height** –
- **min_width** –

Returns

```
>>> skip_if_image_is_below_size(32, 32)(np.zeros((16,16)), Meta(0, 0))
Traceback (most recent call last):
  ...
mycelyso.pilyso.pipeline.executor.Skip: Meta(pos=0, t=<class 'mycelyso.pilyso.
↪pipeline.executor.Collected'>)
```

mycelyso.highlevel.steps.**track_multipoint**(*collected*)
Initiates tracking between consecutive NodeFrames.

Parameters **collected** –

Returns

Module contents

The highlevel package contains highlevel functionality of mycelyso.

1.5.1.3 mycelyso.misc package

Submodules

mycelyso.misc.graphml module

The graphml module contains output routines to output GraphML structured data from internal graph representations.

mycelyso.misc.graphml.**to_graphml_string**(*g*)
Converts a networkx graph to a GraphML representation.

Parameters **g** – graph

Returns graphml string

mycelyso.misc.graphml.**to_graphml_writer**(*g*)
Takes a networkx graph and returns a GraphMLWriter containing the graph.

Parameters **g** – graph

Returns GraphMLWriter instance

mycelyso.misc.graphml.**write_graphml**(*g*, *name*)
Writes a networkx graph in GraphML format to a file.

Parameters

- **g** – graph
- **name** – filename

Returns

mycelyso.misc.regression module

The regression modules contains some helpers to perform linear fits on data with non-linear begins or ends.

mycelyso.misc.regression.**find_linear_window**(*x*, *y*, *begin=nan*, *end=nan*, *window=0.1*, *condition=('rvalue', 'gt', 0.95)*, *return_begin_end=False*, *return_nan_if_impossible=True*)
Tries to find a continuous window in x/y which (mostly) follows a linear relation subject to condition.

If window is a float, it is seen as relative length of the input lists. Linear regressions will be performed on each window, then the windows will be filtered by the condition (eg that they have a rvalue better than 0.95). Then the range between the first and the last window to follow these conditions will be used to perform the overall regression.

See also `scipy.stats.linregress()`

Parameters

- **x** – Input data, independent value
- **y** – Input data, dependent value
- **begin** –
- **end** –
- **window** – Window, either
- **condition** – Condition to check, a tuple of three. The first must be a key of a linear regression result object, the second either 'gt' or 'lt', and the third the value to compare.
- **return_begin_end** – If true, return the found range as well
- **return_nan_if_impossible** – If True, return NaN if no suitable region was found, otherwise throws RuntimeError

Returns

mycelyso.misc.regression.**prepare_optimized_regression**(*x*, *y*)

First finds an optimal window using *find_linear_window()*, than performs a linear regression.

Parameters

- **x** – independent variable
- **y** – dependent variable

Returns

```
>>> x = np.linspace(1, 100, 100)
>>> y = x * 5 + 10
>>> y[0:10] = 0  # break our nice linear curve
>>> prepare_optimized_regression(x, y)
OrderedDict([('slope', 5.0), ('intercept', 10.0), ('rvalue', 0.
↪9999999999999999), ('pvalue', 0.0), ('stderr', 7.942345602646859e-09), (
↪'begin_index', 10), ('end_index', 100), ('begin', 11.0), ('end', 100.0)])
```

mycelyso.misc.util module

mycelyso.misc.util.**calculate_length**(*points*, *times=1*, *w=5*)

Calculates the length of a path.

Paths sampled from pixel grids may contain notable measuring error, if euclidean distances are calculated naively. This method uses an adapted approach from [Cornelisse1984], by repeatedly smoothing the coordinates with a moving average filter before calculating the euclidean distance.

Parameters

- **points** – Input points, a numpy array (X, 2)
- **times** – Times smoothing should be applied
- **w** – window width of the moving average filter

Returns Length of the input path

```
>>> calculate_length(np.array([[1.0, 1.0],
...                            [5.0, 5.0]]))
5.656854249492381
```

mycelyso.misc.util.**clean_by_radius**(*points*, *radius=15.0*)

Bins points by radius and returns only one per radius, removing duplicates.

Parameters

- **points** – Input points

- **radius** – Radius

Returns Filtered points

```
>>> clean_by_radius(np.array([[1.0, 1.0],
...                           [1.1, 1.1],
...                           [9.0, 9.0]]), radius=1.5)
array([[1., 1.],
       [9., 9.]])
```

mycelyso.misc.util.**pairwise**(*iterable*)
s -> (s0,s1), (s1,s2), (s2, s3), ...

Module contents

The misc package contains various helper functions.

1.5.1.4 mycelyso.processing package

Submodules

mycelyso.processing.binarization module

The binarization module contains the binarization routine used to segment phase contrast images of mycelium networks into foreground and background.

mycelyso.processing.binarization.**bataineh**(*image*, *mask=None*, *window_size=15*, *return_threshold=False*, ***kwargs*)
Thresholding method as developed by [Bataineh2011a].

Parameters

- **image** – Input image
- **mask** – Possible mask denoting a ROI
- **window_size** – Window size
- **return_threshold** – Whether to return a binarization, or the actual threshold values
- **kwargs** – For compatibility

Returns

mycelyso.processing.binarization.**experimental_thresholding**(*image*, *mask=None*, *window_size=15*, *gaussian_sigma=3.0*, *shift=0.2*, *target=-0.5*, *quotient=1.2*, *return_threshold=False*, ***kwargs*)
A novel thresholding method basing upon the shape index as defined by [Koenderink1992], and [Bataineh2011] automatic adaptive thresholding. The method is due to be explained in detail in the future.

Parameters

- **image** – Input image

- **mask** – Possible mask denoting a ROI
- **window_size** – Window size
- **gaussian_sigma** – Sigma of the Gaussian used for smoothing
- **shift** – Shift parameter
- **target** – Target shape index parameter
- **quotient** – Quotient parameter
- **return_threshold** – Whether to return a binarization, or the actual threshold values
- **kwargs** – For compatibility

Returns

mycelyso.processing.binarization.**feng**(*image*, *mask=None*, *window_size=15*, *window_size2=30*, *a1=0.12*, *gamma=2*, *k1=0.25*, *k2=0.04*, *return_threshold=False*, ***kwargs*)

Thresholding method as developed by [Feng2004].

Parameters

- **image** – Input image
- **mask** – Possible mask denoting a ROI
- **window_size** – Window size
- **window_size2** – Second window size
- **a1** – a1 value
- **gamma** – gamma value
- **k1** – k1 value
- **k2** – k2 value
- **return_threshold** – Whether to return a binarization, or the actual threshold values
- **kwargs** – For compatibility

Returns

mycelyso.processing.binarization.**mean_and_std**(*image*, *window_size=15*)

Helper function returning mean and average images sped up using integral images / summed area tables.

Parameters

- **image** – Input image
- **window_size** – Window size

Returns tuple (mean, std)

mycelyso.processing.binarization.**nick**(*image*, *window_size=15*, *k=-0.1*, *return_threshold=False*, ***kwargs*)

Thresholding method as developed by [Khurshid2009].

Parameters

- **image** – Input image
- **window_size** – Window size
- **k** – k value
- **return_threshold** – Whether to return a binarization, or the actual threshold values

- **kwargs** – For compatibility

Returns

mycelyso.processing.binarization.**normalize**(*image*)
Normalizes an image to the range 0-1

Parameters **image** –

Returns

mycelyso.processing.binarization.**phansalkar**(*image*, *window_size=15*, *k=0.25*, *r=0.5*, *p=2.0*, *q=10.0*, *return_threshold=False*, ***kwargs*)
Thresholding method as developed by [Phansalkar2011].

Parameters

- **image** – Input image
- **window_size** – Window size
- **k** – k value
- **r** – r value
- **p** – p value
- **q** – q value
- **return_threshold** – Whether to return a binarization, or the actual threshold values
- **kwargs** – For compatibility

Returns

mycelyso.processing.binarization.**sauvola**(*image*, *window_size=15*, *k=0.5*, *r=128*, *return_threshold=False*, ***kwargs*)
Thresholding method as developed by [Sauvola1997].

Parameters

- **image** – Input image
- **window_size** – Window size
- **k** – k value
- **r** – r value
- **return_threshold** – Whether to return a binarization, or the actual threshold values
- **kwargs** – For compatibility

Returns

mycelyso.processing.binarization.**wolf**(*image*, *mask=None*, *window_size=15*, *a=0.5*, *return_threshold=False*, ***kwargs*)
Thresholding method as developed by [Wolf2004].

Parameters

- **image** – Input image
- **mask** – Possible mask denoting a ROI
- **window_size** – Window size
- **a** – a value
- **return_threshold** – Whether to return a binarization, or the actual threshold values

- **kwargs** – For compatibility

Returns

mycelyso.processing.pixelgraphs module

The pixelgraphs module contains various functions to work with skeleton images, and treating the paths of the skeleton as graphs, which can be walked along.

mycelyso.processing.pixelgraphs.**get_all_neighbor_nums**(*num*)
Return all set neighbor bits in num.

Parameters **num** – Neighborhood representation.

Returns Array of values

mycelyso.processing.pixelgraphs.**get_all_neighbors**(*num*)
Return positions for all set neighbor bits in num

Parameters **num** – Neighborhood representation

Returns Array of shifts

mycelyso.processing.pixelgraphs.**get_connectivity_map**(*binary*)
Returns a 'connectivity map', where each value represents the count of neighbors at a position.

Parameters **binary** – Binary input image

Returns

mycelyso.processing.pixelgraphs.**get_inverse_neighbor_shift**(*num*)
Get the shift corresponding to the inverse direction represented by num.

Parameters **num** – Neighborhood bit

Returns Shift (r, c)

mycelyso.processing.pixelgraphs.**get_neighborhood_map**(*binary*)
Returns a 'neighborhood map', where each value binary encodes the connections at a point.

Parameters **binary** – Binary input image

Returns

mycelyso.processing.pixelgraphs.**get_next_neighbor**(*num*)
Returns the coordinates represented by a numeric neighbor bit.

Parameters **num** – Neighbor bit

Returns Shift (r, c)

mycelyso.processing.pixelgraphs.**is_edge**(*connectivity*)
Returns True if connectivity corresponds an edge (is two).

Parameters **connectivity** – Scalar or matrix

Returns Boolean or matrix of boolean

mycelyso.processing.pixelgraphs.**is_end**(*connectivity*)
Returns True if connectivity corresponds to an endpoint (is one).

Parameters **connectivity** – Scalar or matrix

Returns Boolean or matrix of boolean

mycelyso.processing.pixelgraphs.**is_junction**(*connectivity*)
Returns True if connectivity corresponds a junction (is greater than two).

Parameters **connectivity** – Scalar or matrix

Returns Boolean or matrix of boolean

mycelyso.processing.pixelgraphs.**where2d**(*image*)
numpy.where for 2D matrices.

Parameters **image** – Input images

Returns Coordinate list where image is non-zero

```
>>> where2d(np.array([[ 0, 0, 0],
...                   [ 0, 1, 1],
...                   [ 0, 0, 0]]))
array([[1, 1],
       [1, 2]])
```

Module contents

The processing submodule contains various functions and management classes concerned with image processing of hyphae network images.

1.5.2 Module contents

mycelyso, the MYCElium anaLYsis SOftware

Bibliography

[Cornelisse1984] Cornelisse and van den Berg (1984) Journal of Microscopy 10.1111/j.1365-2818.1984.tb00544.x

[Bataineh2011a] Bataineh et al. (2011) Pattern Recognit. Lett. DOI: 10.1016/j.patrec.2011.08.001

[Koenderink1992] Koenderink and van Doorn (1992) Image Vision Comput. DOI: 10.1016/0262-8856(92)90076-F

[Bataineh2011] Bataineh et al. (2011) Pattern Recognit. Lett. DOI: 10.1016/j.patrec.2011.08.001

[Feng2004] Fend & Tan (2004) IEICE Electronics Express DOI: 10.1587/elex.1.501

[Khurshid2009] Khurshid et al. (2009) Proc. SPIE DOI: 10.1117/12.805827

[Phansalkar2011] Phansalkar et al. (2011) Proc. ICCSP DOI: 10.1109/ICCSP.2011.5739305

[Sauvola1997] Sauvola et al. (1997) Proc. Doc. Anal. Recog. DOI: 10.1109/ICDAR.1997.619831

[Wolf2004] Wolf & Jolion (2004) Form. Pattern Anal. & App. DOI: 10.1007/s10044-003-0197-7

Python Module Index

m

Index

P

Q

R

S

T

V

W