Clathrin Mediated Endocytosis Pipeline

Basic Requirements

Basic understanding of:

1. Python
2. Jupyter Notebooks

Backend programming language

1. Python

3-D viewer

1. Napari

Step wise guide

Step 1: Downloading the repository from Github and setting up the virtual environment

GitHub Repo Link: <https://github.com/Mdanishnadeem/Image-Analysis-Tracking>

1. Go to the link above and download the repository in your working directory
2. Look at the readme file on the link and follow the instructions to set up the virtual environment
3. Once the virtual environment is setup, activate it

* git clone [git@github.com:Mdanishnadeem/Image-Analysis-Tracking.git](mailto:git@github.com:Mdanishnadeem/Image-Analysis-Tracking.git)
* conda create --name cme\_pipeline python==3.10
* conda activate cme\_pipeline
* pip install -r requirements.txt

Step 2: Converting the 3-Channel CME movie to a single zarr file before passing into next steps (optional)

The code for this step can be found in the following directory

Final => Data Preparation => full\_movie\_to\_zarr.ipynb

The purpose of doing this is:

1. To reduce the storage size of the movies as the LLSM movies can become very large
2. Zarr format also facilitates parallel processing, which is very useful throughout the pipeline
3. Specific time frame and channel can be accessed real time without keeping everything in memory thus keeping the pipeline efficient and less resource intensive

Note:

Works best when the initial movie is in a tiff format

Step 3: Performing Spot detection on the Primary Channel

In this step spot detection is performed on one channel (primary channel/clathrin channel).

The **input** for this notebook is a zarr file

The code for this step can be found in:

Final => Jupyter\_Notebooks => 01.Big\_data\_detection\_zarr\_parallel.ipynb

This part of the code builds up on [Pylattice](https://github.com/pylattice/pyLattice) making a couple of important additions:

1. Allows for segmentation of all frames in one go
2. Allows for parallel processing significantly reducing the run time

In this part the user must set up the Detector Object as per requirement. More details on the Class and its functions can be found by adding a question mark before the class or any of its functions, for example:

*?Detector (this would show the docustring/ meaning of each parameter)*

*??Detector(this would show the source code)*

The **output** from this notebook is a combined dataframe of detections on all channels (it returns the gaussian estimates of the centers(z,y,x) and the spreads of the spots along with frame numbers

Key points to look out for

1. Set the **number of cores(n\_jobs)** to use in accordance with the RAM available on your machine. As more cores are used more RAM is needed (by default n\_jobs = -1, this means it uses all cores – 1)
2. **channel\_to\_detect** parameter in the Detector object follows the convention (1 for channel 1, 2 for channel 2 and so on)
3. **sigma\_estimations** are the expected radius of our spots and follow the convention [spread\_in\_z, spread\_in\_y, spread\_in\_x]
4. In the function **run\_parallel\_frame\_processing** if you just want perform detection on a subset of frames set **max\_frames** to the number you want. If you want to perform detection on all of the frames set **all\_frames = True**. In this case the function ignores the value assigned to **max\_frames** and performs detections on all frames

An ideal approach over here would be to detect for a few frames and validate your parameters before going on to process all the channels.

Step 4: Filtering the Detected Spots

In this step the user can filter for incorrect detections done in the previous step. The user can also view spots in 3-D and 2-D in Napari and adjust cutoffs accordingly.

The code for this step can be found in:

Final => Jupyter\_Notebooks => 02.Filtering\_spots.ipynb

The filtering is performed on the basis of:

1. Sigma estimates provided in the previous step for each z,y,x dimension. Spots with very high or low sigma values as compared to expected sigma values may be dropped.
2. Spots with very high amplitudes may also be dropped

The techniques used for dropping points are

1. Box and whisker plots
2. Histograms

After cutoffs are determined with the help of the above techniques, they can be validated in Napari and adjusted accordingly.

User can view cleaned and dropped spots in Napari and finally output a dataframe with cleaned spots only.

This step serves as an important part in the pipeline as it will

1. Tell us how the detection algorithm performed
2. Improve tracking algorithm performance as invalid spots are dropped

Step 5: Tracking spots over time

This part of the pipeline uses Laptrack algorithm to connect detected spots over time. It is pretty straightforward and just needs a few parameters to be adjusted.

The code for this step can be found in:

Final => Jupyter\_Notebooks => 03.cleaned\_spots\_c3\_tracking.ipynb

The input for this notebook is the cleaned spots dataframe created in the previous step.

This notebook outputs a dataframe with track\_id and tree\_id assigned to each spot. In our case tree\_id is not that relevant so it can be ignored. Whereas track\_id is important.

Step 6: Extracting intensities information of other Channels based on tracks formed from the primary channel

In this part of the pipeline the tracks formed from the primary channel are associated with the other two channels.

The code for this step can be found in:

Final => Jupyter\_Notebooks => 04.extracting\_alt\_channel\_intensities\_BD

The input for this notebook is

1. The output from the previous step where tracking has been performed.
2. Moreover, the zarr file which has raw data for all three channels

You have to create an object Extractor with certain parameters to avoid passing same parameters again and again.

The key features of this part are:

1. Mean value of intensities for the other two channels could be found using the coordinates and the sigma values (or fixed radius can be provided) of the spots in our primary channel
2. Maximum value of intensities for the other two channels could be found using the coordinates and the sigma values (or fixed radius can be provided) of the spots in our primary channel
3. Peak pixels coordinates for each spot could be identified for all the channels
4. The functions allow for offset if any exists between different channels
5. Voxel sum can be calculated around the peak coordinates. The voxel sum can be either calculated with a variable radius (sigma values) or fixed radius. Moreover, the voxel also takes into account the local background to get the true intensity
6. Gaussian fitting can be done on the peak coordinates of the other two channels. This can give an additional parameter in future for filtering tracks with very high or low sigma values

The output of this step is a dataframe which has intensity values and coordinates for the other two channels as well

Note:

1. The convention for coordinates/radius/sigma values in any part of the notebook is [z,y,x]
2. The convention for channel number in any part of the notebook is, 1 for channel 1, 2 for channel 2 and so on

Step 7: Filtering Tracks

Once information for each track across all three channels is obtained now tracks could be filtered to keep only valid tracks.

The code for this step can be found in:

Final => Jupyter\_Notebooks => 05.filtering\_tracks.ipynb

The input for this part of the pipeline is the dataframe outputted from the previous step having all relevant information for each track.

You have to create an object called track to read in the dataframe and find different features of each track from that dataframe. The features include:

1. Track\_id
2. Track\_length
3. Track\_start (frame number)
4. Track\_end (frame number)
5. C3\_peak (intensity)
6. C2\_peak(intensity)
7. C1\_peak(intensity)
8. C3\_peak\_frame
9. C2\_peak\_frame
10. C1\_peak\_frame
11. Mean\_displacement
12. Mean\_z
13. Mean\_z\_displacement

These features will assist in filtering out valid tracks.

The tracks are filtered on the basis of the following criteria (cutoffs could be adjusted):

1. Tracks with short length i.e 3 are dropped
2. Tracks are identified to be either channel 2 or channel 1 positive or both and tracks which are not positive in either of the channel are dropped
3. Tracks with clathrin, dynamin or actin peak in the first frame are dropped
4. Tracks with clathrin, dynamin or actin peak in the last frame are dropped
5. Tracks with clathrin, dynamin or actin peak in the first three frames are dropped
6. Tracks are assigned apical, basal or lateral part of the membrane based on the value of mean\_z

In the end three different types of tracks are identified:

1. Clathirn and Dynamin positive
2. Clathrin and Actin positive
3. Clathrin, Dynamin and Actin positive

The output of this notebook is a dataframe which has the above information (types of track and the identified features) for each track.

Step 8: Plotting Intensity over time

In this part of the pipeline intensity over time plots are developed to give user an idea of how individual tracks are behaving and how they behave on average.

The code for this step can be found in:

Final => Jupyter\_Notebooks => 06.intensity\_time\_plots.ipynb

This part takes two inputs:

1. The valid tracks obtained from the previous step after filtering
2. The dataframe obtained from step 6

Both of these dataframes are used together for plotting purposes.

The types of plots in this notebook are

1. Subplots (64 tracks) of ranges of different lengths grouped together and plotted
2. Averaged plots of either two or three channels aligned by the peak frame of the last channel

This notebook majorly gives an idea about how tracks behave when different proteins are involved.

**Optional Parts:**

A tutorial to the relevant functions of Napari is included and could be found in:

Final => Napari Tutorial => Tutorial.ipynb

Source Code:

All of the source code is could be found in:

Final => src