Clathrin Mediated Endocytosis Pipeline

Basic Requirements

Basic understanding of [Jupyter Notebooks](https://www.codecademy.com/article/how-to-use-jupyter-notebooks)

Initial setup: Downloading the [repository](https://github.com/MatsulabUW/LLSM-CME-ANALYSIS/) from Github and setting up the virtual environment

* Install [conda](https://www.anaconda.com/download/success) and [git](https://git-scm.com/) if you don’t have them on your computer
* Navigate to a folder where the analysis routine will go, and [open a Terminal window](https://www.howtogeek.com/682770/how-to-open-the-terminal-on-a-mac/) in that folder
* In Terminal, type:
* git clone https://github.com/MatsulabUW/LLSM-CME-ANALYSIS.git
* conda create --name cme\_pipeline python==3.10
* conda activate cme\_pipeline
* cd LLSM-CME-ANALYSIS
* pip install -r requirements.txt

This will download and install the latest version of the analysis pipeline, at the current folder, and install the required python libraries.

Step 0: Converting your movie to a zarr file

This step is necessary if your movie isn’t already in zarr format.

* Navigate to the new “LLSM-CME-ANALYSIS” folder. Make a copy of the “movie data” folder, which is in the “Final” folder. (You will be using the original “movie data” folder.)
* Move your movie file to the Final => movie\_data => full\_movie folder. Copy the file name.
* In Terminal, navigate to the “LLSM-CME-ANALYSIS” folder. Type *jupyter notebook* to open jupyter notebook.
* Open Final => Data Preparation => full\_movie\_to\_zarr.ipynb
* Change the file name (input\_file\_name = ) to the name of your movie file. Click “Run all” in the “Cell” menu to run all the code in the cells. (Or type shift+enter to run through each cell sequentially.)

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 1: 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

1. In Jupyter notebook, open Final => Jupyter\_Notebooks => 01.Big\_data\_detection\_zarr\_parallel.ipynb

You’ll be setting parameters for detecting the spots. The key parameters are **channel\_to\_detect** and **threshold\_intensity**

* **channel\_to\_detect:** Which channel will be tracked? This should be the channel with the longest tracks (i.e. AP2). Options are channel 1, 2, or 3.
* **threshold\_intensity:** What intensity value distinguishes background from signal? Open up a frame of the movie in Fiji or napari (at the end of this notebook) and mouse over different pixels to figure out this threshold value.

Other parameters to optimize:

* **number of cores(n\_jobs)** based on how many CPUs and how much RAM is available on your machine. Try the default n\_jobs = -1, this means it uses all cores – 1. If it slows down your computer too much, figure out how many CPUs are on your computer, and pick a number that is 75-80% of the total number of CPUs
* **sigma\_estimations** are the expected radius of our spots and follow the convention [spread\_in\_z, spread\_in\_y, spread\_in\_x]
* **all\_frames:** When initially optimizing, set this to false and set **max\_frames** to 2, in order to run detection on only two time points. This is faster and will speed up diagnosing detection quality at the end of the notebook.

More details are in the notebook, and can be accessed by entering the following into a new cell:

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

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

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.

1. Per the notebook instructions, run the final cells to open up napari and compare the detections (in green) with the spots you see by eye (magenta).
2. If the detections don't line up well with the spots in the image:
   * make sure you are looking at the first time point
   * mouse over the spots in napari to get a sense for the intensity of the spots vs background - use the threshold distinguishing spots from background as spot\_intensity
   * vary the dist\_between\_spots: if the detections are at a higher density than the visible spots, increase the dist\_between\_spots. And vice versa, if you see spots at a higher density than detections, lower the dist\_between\_spots.
   * If the detections are missing larger or smaller spots you can try increasing or decreasing the sigma\_estimations.

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

Step 2: Filtering the Detected Spots

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

1. Open: Final => Jupyter\_Notebooks => 02.Filtering\_spots.ipynb

The filtering is performed on the basis of:

Size (sigma) estimates in z,y,x. Spots with very high values as compared to expected sigma values may be dropped.

Spots with very high intensities may also be dropped

Detection quality: The error between the initial guess of the spot location (from peaklocalmax) and size, compared to the value fitted by a Gaussian.

1. Use the “tolerance” parameters to adjust how strict this detection filtering step is.

* 1 = default. Which is the threshold for an outlier from [box and whisker plots](https://www.tableau.com/data-insights/reference-library/visual-analytics/charts/box-whisker#:~:text=Upper%20Whisker%3A%201.5*%20the%20IQR,individual%20points%20are%20considered%20outliers.&text=Lower%20Whisker%3A%201.5*%20the%20IQR,individual%20points%20are%20considered%20outliers.)
* <1 = more strict; i.e. filter out more detections at a lower threshold
* >1 = more tolerant; i.e. don’t filter out so many detections

The value is a multiplier of the initial limit. i.e. if the threshold for outliers is a sigma ( width) value of 5 pixels, a tolerance of 2 increases the threshold to 10 pixels.

1. At the end of the notebook, open the movie + detections in Napari and visually inspect the quality of the filters.

**The goal is for the green spots to circle true spots, and the red-ish spots to circle noise**

* + Toggle each red circle one at a time and assess if it captures noise. If it is capturing real spots, then increase the tolerance for that feature and re run the condition setting and visualization cells.
  + Look at the green circles. If the green circles are capturing noise, then increase the tolerance for a feature and see if it changes.
  + You can also use the histograms below to guide your choice: look for breaks between two distributions, or long tails

1. Repeat until you're satisfied with the quality of your detections!

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

* Tell us how the detection algorithm performed
* Improve tracking algorithm performance as invalid spots are dropped

Step 3: Tracking spots over time

This part of the pipeline uses Laptrack algorithm to connect detected spots over time.

1. In Jupyter notebook, open Final => Jupyter\_Notebooks => 03.cleaned\_spots\_c3\_tracking.ipynb
2. Try running the entire notebook, and then open the two napari windows to assess tracking quality.
   1. The second napari window opens up a smaller slice of the movie, to make it easier to distinguish tracks. You can adjust “min and max z to show” values to change the size of this slice.
3. Based on the tracking quality, adjust the tracking parameters:
   1. If a single track appears to be broken into multiple tracks you can increase the max\_distance and/or increase the gap size and rerun tracking
   2. If multiple tracks appear to be merged into a single track you can decrease the max\_distance and/or decrease the gap size and rerun tracking

Visualizing and assessing track quality is challenging! In general:

* **max\_distance** (in pixels) roughly correlates to how far two spots can be separated to be considered part of a track. Default = 3 pixels. Should be higher if your time step is longer, or spots move faster.
* **gap\_size** is the number of frames in which a detection can be absent. The track will fill in the gap. Default = 1 frame.
* **Gap\_closing\_cost\_tolerance** is how strict the penalty is for closing gaps. Lower = more penalty for having gaps

This notebook outputs a dataframe with track\_id assigned to each spot.

Step 4: Extracting intensities information of other Channels based on tracks formed from the primary channel

In this part of the pipeline tracks from the primary channel are used to calculate metrics in all the channels.

1. In jupyter notebook, open: Final => Jupyter\_Notebooks => 04.extracting\_alt\_channel\_intensities\_BD
2. Run through this entire notebook

The key metrics calculated are:

* Mean, maximum, and sum intensity using regions based on the coordinates and the sigma values . Sum (called voxel sum) subtracts the local background based on a radius outside the spot. The background size is adjusted with “background\_radius\_for\_voxel\_sum.”
* Amplitude value from Gaussian fit. This will re-center the spots based on a calculated peak intensity, and report the peak intensity from that fit. This allows for offset in xyz if any exists between different channels

These metrics can be either calculated with a variable radius (based on each sigma value) or fixed radius (default).

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

Note:

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

Step 5: Filtering Tracks

This step filters out tacks based on criteria like track length and intensity profile shape.

1. Open Final => Jupyter\_Notebooks => 05.filtering\_tracks.ipynb
2. Run through the notebook

The tracks are filtered based on the following criteria by default:

* Tracks with short length i.e 3 are dropped
* Tracks with clathrin, dynamin or actin peak in the first frame are dropped
* Tracks with clathrin, dynamin or actin peak in the last frame are dropped
* Tracks with clathrin, dynamin or actin peak in the first three frames are dropped
* Tracks starting at frame 1 or ending at the last frame (incomplete tracks)

In the end three different types of tracks are identified:

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

Tracks are assigned apical, basal or lateral part of the membrane based on the value of mean\_z

You could filter based on additional criteria based on the following list:

* Track\_length
* Track\_start (frame number)
* Track\_end (frame number)
* C3\_peak (intensity)
* C2\_peak(intensity)
* C1\_peak(intensity)
* C3\_peak\_frame
* C2\_peak\_frame
* C1\_peak\_frame
* Mean\_displacement
* Mean\_z
* Mean\_z\_displacement

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

Step 6: 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.

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

This part takes two inputs:

* The valid tracks obtained from the previous step after filtering
* The dataframe obtained from step 6

Both of these dataframes are used together for plotting purposes.

The types of plots in this notebook are

* Subplots (64 tracks) of ranges of different lengths grouped together and plotted
* 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.

Step 7: Visualizing the Pipeline performance in Dashboard

This part of the pipeline can be used to analyze the performance of all the previous steps.

The dashboard can be run from the following path:

Final => multipage\_dashboard => app.py

Code for the two pages can be found in

Final => multipage\_dashbpard => pages => home.py & demo\_page.py

This part takes three inputs

1. The main tracking dataframe which contains information on all of the tracks
2. The filtered tracking dataframe generated from the 05.filtering\_tracks.ipynb
3. The zarr file for the raw movie

The dashboard has the following features

1. Display raw projections of 3-D tracks to 2-D
2. Display intensity over time
3. Select different categories of tracks
4. Select different types of track like Channel 1 positive and Channel 2 positive
5. Display track stats for each selected track
6. Display track movement over time in 3-D space
7. The user can save tracks as good, bad or fine and also give remarks for each track to assist in improving parameters of the pipeline

**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