3D multicolor tracking 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.)

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.

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 as [spread\_in\_z, spread\_in\_y, spread\_in\_x]
* **dist\_between\_spots**: this distance divided by 2 is the minimum distance that should exist between spots in pixels. For example if you set this to 10 then all spots within 5 pixels of the center of your spot will be dropped. (to understand which spot is dropped you can refer to the source code in the Final/src/gaussian\_fitting.py file)
* **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 will speed up diagnosing detection quality at the end of the notebook.

An ideal approach is to initially detect 2 frames and validate your parameters before going on to process all the frames.

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

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

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

*Note:* if n\_jobs = -1 slows your computer too much, you can change the number to fewer CPUs (google to figure out how many CPUs your machine has)

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), in 2D and 3D.

A screenshot of a computer

Description automatically generated A purple and white background

Description automatically generated

1. If the detections don't line up well with the spots in the image:
   * 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.

Step 2: Filtering Detected Spots

In this step you filter out low-quality detections.

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 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. It’s 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 5 pixels, a tolerance of 2 increases the threshold to 10 pixels.

You can use the histograms below to guide your choice: look for breaks between two distributions, or long tails. In general it’s better to accept more spots at this step, and filter out tracks in the next notebook

1. At the end of the notebook, open the movie + detections in Napari and visually inspect the quality of the filters in 3D.

**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.

A close-up of a grey surface with green and red dots

Description automatically generated

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 low-quality 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 at the end 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: Extract intensities for other channels based on tracks from the primary channel

Here tracks from the primary channel are used to calculate metrics in all of 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 of regions around the track coordinates. Size of region is set with a radius value in “radii\_extractor” as [z, y, x].
* “Voxel\_sum\_adjusted” is background-subtracted sum intensity, subtracting the local background based on a “shell” of pixels around the region of interest, of thickness determined by “background\_radius\_for\_voxel\_sum.”

A diagram of a cube with lines and circles

Description automatically generated

These metrics can be either calculated with a fixed radius (default) or variable radius based on Gaussian fitting.

If you see an offset in XY between the channels of your movie, you can optionally correct for that here. There are a few ways to correct for an offset between two channels:

* Follow the Protocol/Channel registration page in Roam, or
* In the middle of the notebook, manually set “offset\_x\_ch2vs3” and “offset\_y\_ch2vs3” based on the offset you observe in Fiji or Napari between channel 2 and channel 3. or:
* The end of the notebook will report the average offset between channels 2 and 3 (location of brightest pixel within the region), which you can use to manually set the offset values above.

Step 5: Filtering Tracks

This step filters out tracks 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 are dropped, determined by “threshold\_length” (in frames)
* Tracks with an intensity peak in the first time points are dropped, determined by “peak\_cutoff” (in frames)
* Tracks starting at frame 1 or ending at the last frame (incomplete tracks)

Each track is assessed whether there is signal for each channel, based on the the threshold values that you set.

In the end three different types of tracks are identified:

1. Channel 3 + 2 positive
2. Channel 3 + 1 positive
3. All three channels positive

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

You could also add filters based on additional criteria in 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

Step 6: Plotting Intensity over time

Here we plot intensity over time for tracks, individually and in lifetime cohorts (tracks of similar lifetime).

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

Parameters to set:

* Background\_channel\_x: intensity value that counts as background for each channel (will be subtracted from most intensity measurements)
* Framerate\_msec: time between frames, in milliseconds
* Protein names (to label plots)
* Value\_to\_plot: this intensity value will be plotted, calculated from regions around tracked coordinates. Options:
  + voxel\_sum\_adjusted (default): background-subtracted sum intensity
  + voxel\_sum: sum intensity
  + peak\_mean: mean intensity
  + peak\_max: max intensity
* region\_to\_plot: which region (apical, basal, lateral, etc) to plot (default: all)
* Channels\_to\_plot: choose which tracks to plot: positive for three channels (default) versus two specific channels, etc.

The types of plots in this notebook are

* Subplots of individual tracks (optionally grouped by lifetime)
* Averaged plots aligned by the peak frame of one channel (default: channel 2) and split into lifetime cohorts

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