Geometric Computing for Biomedicine: Final Project Report

Lisa Liao and Emmory Stump

How to Use

Clone this repository

- Be sure to have access to the following frameworks
  - Tkinter
  - PIL
  - OpenCV
  - Sklearn
  - Scipy
  - NumPy
  - TiffCapture
- Run "python GUI.py" to load our GUI
- Press "Load Data" and select a dataset to load
- Select BOTH ends of microtubule of interest
  - You will see that this has been done when a line of best fit appears
- Press "Play" to start tracking the Microtubule
- Read the section below “GUI Use” for a more detailed explanation of how to use our application.

Features Accomplished

1. Allow a user to upload a video
2. Ability to select a microtubule which the user would like to track by clicking on both ends of the microtubule
3. Segment the selected microtubule in each frame of the video and dynamically track its length
4. Plot length of microtubule against time

5. If the algorithm gets lost (possibly at the intersection of microtubules) allow the user to reselect endpoints of the microtubule - should only need for hard problems
6. All of the above features should be displayed in a GUI

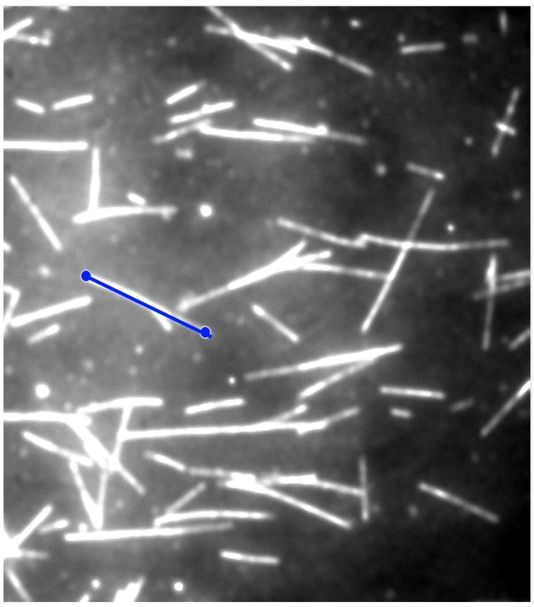
Application Details

GUI Development

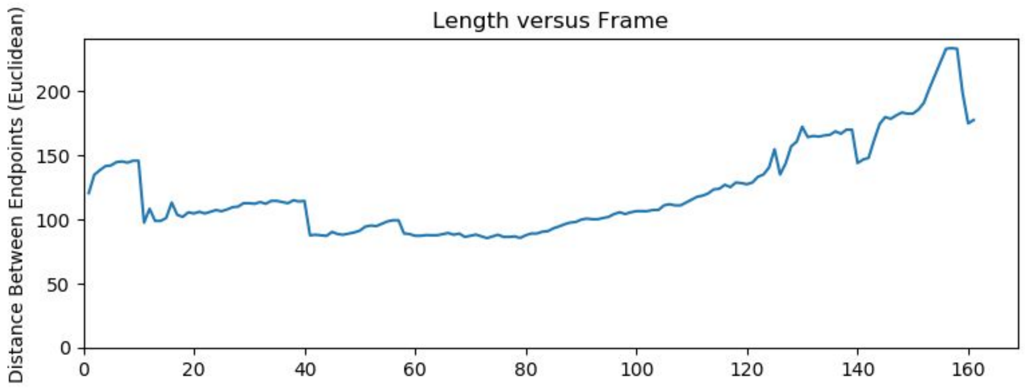
For our application, we used the Tkinter library to create our GUI. The GUI contains a number of buttons, including a button to load data, reset the video, play the video, pause the video, proceed frame by frame, and to reselect the microtubule's endpoints.

GUI Use

To begin using our application, select "Load Data" and then choose the data which you would like to view. Once you have loaded your data, select both ends of the microtubule of interest. You will know that you have successfully selected both ends when a line of best fit appears.


Selected Microtubule with line of best fit

This line of best fit should roughly approximate the location of your chosen microtubule. After you have selected both ends of your microtubule, press “Play” to begin tracking. After all frames have been analyzed, a graph will pop up depicting how the microtubule’s length changes from frame to frame.



Microtubule length over all frames

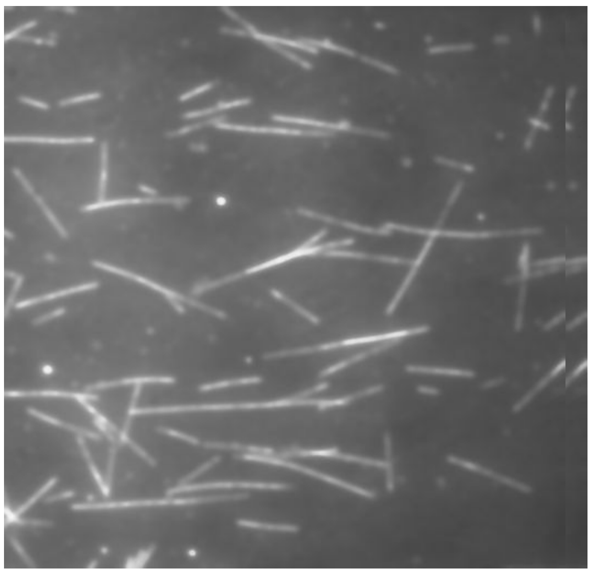
We also provide other functionalities which the user can use within the GUI. One such feature is to reset the video. This resets the video to the first frame, where users must reselect the microtubule they would like to track. Another feature is the ability to pause the video. Additionally, we provide the user the ability to iterate through the video frame by frame by using the “Next Frame” button. You should pause the video before using this function. Lastly, the user can reselect endpoints in the middle of the video by clicking the “Reselect Endpoints” button. Before clicking the “Reselect Endpoints” button, please ensure that the video is **playing**. This functionality will clear the current endpoints from the screen, and the user must reselect the endpoints of their microtubule and then press “Play”.

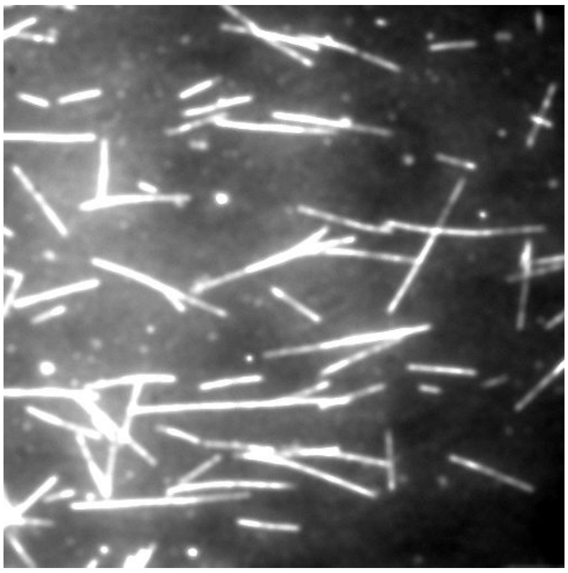
Core Algorithm

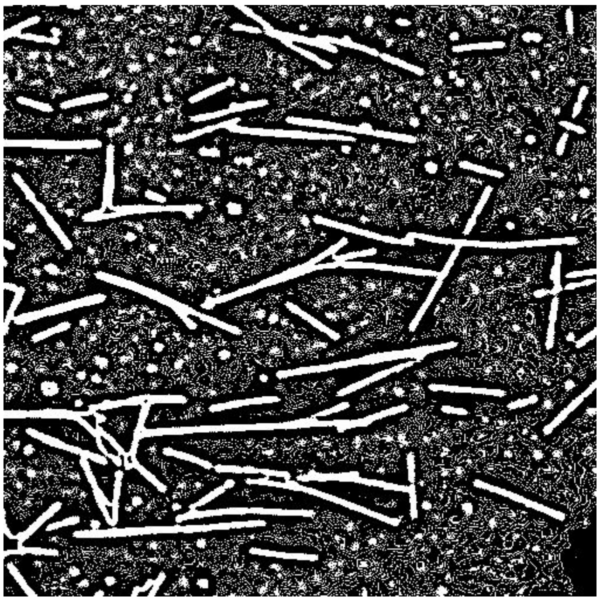
Our core algorithm can be split into roughly three processes: preprocessing the frame, finding the connected component, and updating the endpoints.

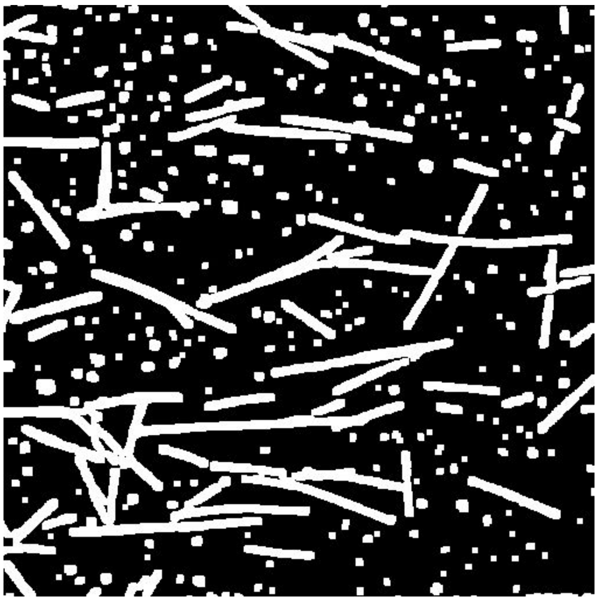
Preprocess Frame

For each frame in our tiff array, we perform a number of preprocessing steps. To begin, we first normalize our image so that it contains values only between 0 and 255. After this, we apply a bilateral filter to blur the frame. We chose to use a bilateral filter for several reasons: Firstly, it blurs the frame to remove some troublesome background noise. Secondly, bilateral filtering in particular retains the edges of the microtubules better than a Gaussian kernel, which will help us in the thresholding step. Following this, we enhance the frame's contrast and sharpness. This allows for better distinguishing between the microtubule objects and the background. Next, we threshold our frame using OpenCV's adaptive threshold method. We chose this method because we could not find a global threshold value that was effective on all microtubules within a given frame. After we run the adaptive thresholding on the frame, we once again blur the frame using a bilateral filter. This once again helps to remove any remaining background noise following our thresholding. Finally, we run an assortment of morphological operators in the frame in order to further distinguish individual microtubules from each other.


Bilateral filtered

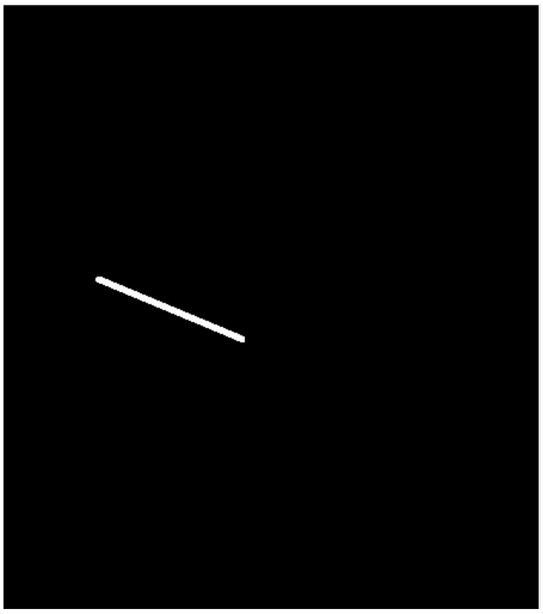

Enhanced contrast and sharpness


Adaptive threshold applied


Morphological operations applied

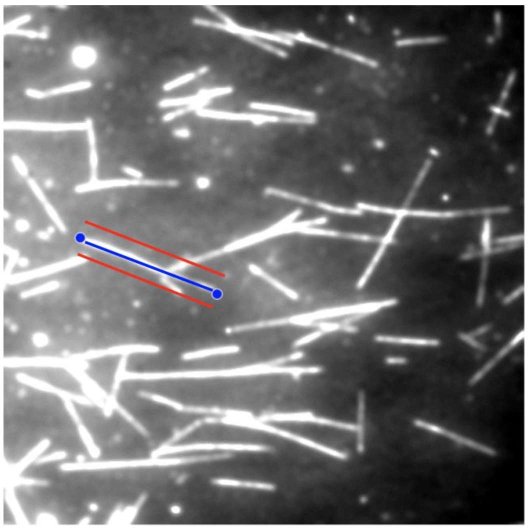
Find Connected Component

In order to track our microtubule throughout the series of frames, we rely heavily on the endpoints, slope, and b-intercept from the previous frame. With our preprocessed frame, we find the connected components within the frame using OpenCV's connected component function. This returns our frame, but the pixel values are now replaced by component labels. To find the user-selected microtubule in the frame, we start by taking the previous iteration's endpoints, slope, and intercept, and project an array of points between these endpoints and along the slope. Then, taking this array of coordinates where we believe the microtubule is located, we search in a 2x2 neighborhood surrounding each pixel, and find the maximum component number in each of these ranges. This is to distinguish the component label from the background label. Next, we find the most frequently occurring label number, and take this to be the microtubule's component label for this frame. Using this component label, we set everything within our original frame that is not part of this component to be the background background (value 0).


Filtered connected component

Find Endpoints

Before we finally find the endpoints of our microtubule for the current frame, we once again perform some processing on our newly segmented frame. Because of the overlapping nature of many of the microtubules, simply retrieving the connected component label and setting everything else to the background is not sufficient to isolate our tracked microtubule. Additionally, because we update our slope and intercept using Linear Regression, we want to ensure that our updated values will not be skewed by the presence of other microtubules. Therefore, in order to further isolate our microtubule, we create an intercept threshold that only keeps white pixels within a neighborhood along our slope. While we overall characterize the microtubule as linear, this threshold also allows for any slight curvature that might be present in the microtubule. As an example, we would keep the white pixels between the red lines in the image below. This is under the assumption that the slope of a microtubule does not change drastically over time. This assumption is far from perfect, but it works well in localizing our search to a smaller area, and keeps the endpoints from switching to another microtubule in an entirely different direction. Finally, we update the slope and intercept with Linear Regression, and find the endpoints of the microtubule. This is done by calculating the mean coordinates of the remaining white pixels within the frame, and finding the coordinates of the 2 pixels that are farthest away from the mean..


Region where white pixels are kept

Future Work and Improvements

One point of improvement in our application is reducing the jumpiness in our endpoints between frames. Though we do a lot of preprocessing in order to isolate our microtubule of interest, sometimes other microtubules with points roughly along the same slope and within the x and y padding we designate will be designated as endpoints, even though they are not part of the microtubule of interest. We would like to be able to know when the endpoints have jumped to a different microtubule, and either change the endpoint prediction, or prompt the user to reselect the microtubule's endpoints.

Sources

https://stackoverflow.com/questions/31667070/max-distance-between-2-points-in-a-data-set-and-identifying-the-points

https://solarianprogrammer.com/2018/04/21/python-opencv-show-video-tkinter-window/