02740 Project #3

Microscope characterization & curvilinear feature detection

Alan Shteyman, Luigi Leung, Yiming Xin, Yuxian Ruan

B. Microscope characterization

B.1 Image data

The image sequence was downloaded

B.2 Characterizing fluorescence image background noise

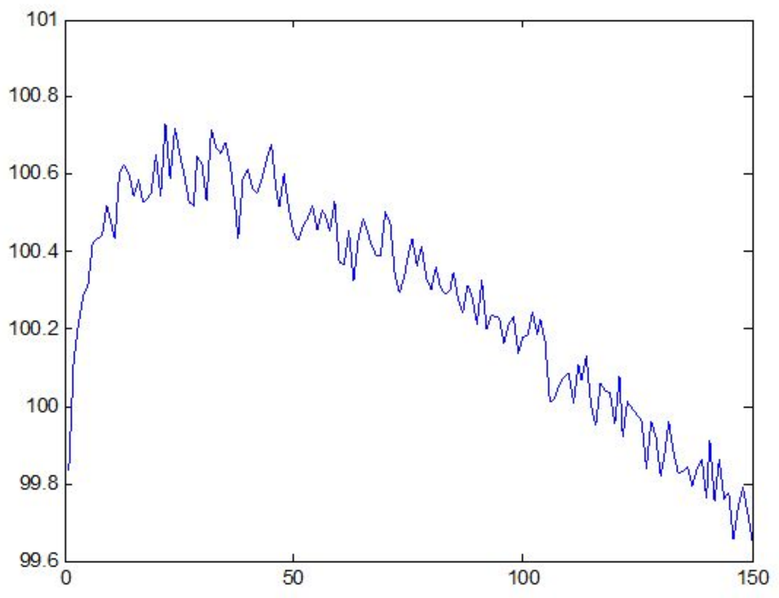
B.2.1

Using the kstest in matlab, we took a cropped background of each image (the same region was taken using the same set of pixel coordinates). Each of the cropped images, according to the kstest, had pixels that came from a normal intensity distribution, at the 0.05 confidence level. Therefore the intensity of the background pixels could be described as coming from a normal distribution and therefore is not white noise.

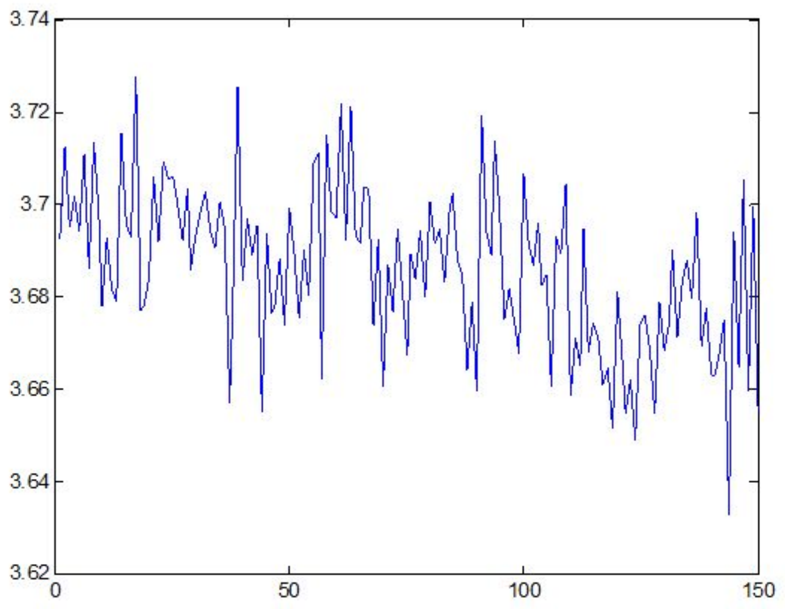
B.2.2

The mean and standard deviation (below the mean plot) of the intensity values for a cropped background region for each image was computed and plotted. As can be seen below both plots trend downward and change over time.

Mean over time



Standard Deviation over Time



B.2.3

The first image was used to determine if the distribution of background intensities is constant. Five different samples of the background were compared via their mean and standard deviation and found to be significantly different according to the kstest2 in matlab. Therefore at the .05 significance level for each pairwise test, the background distribution of intensity values differs across the image.

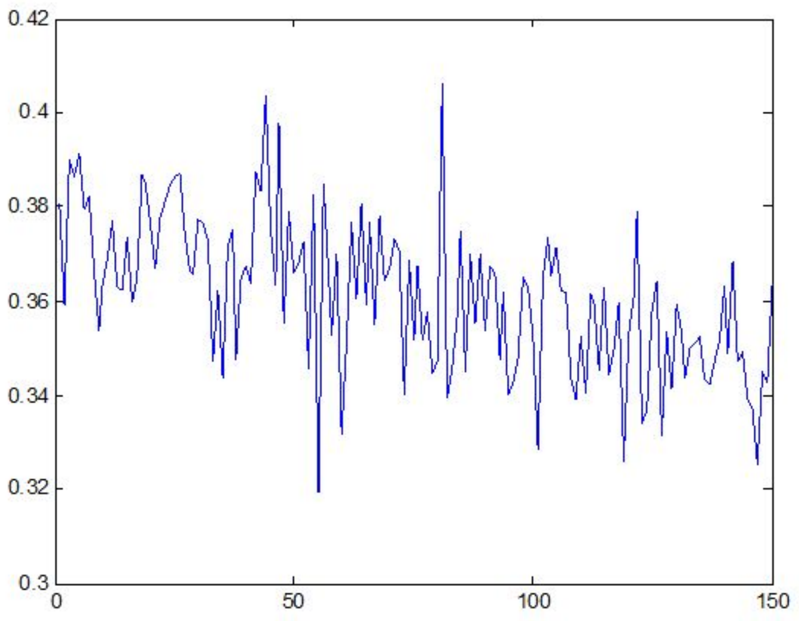
Then images 2 and 3 from project 1 were tested to see if the background intensity followed a normal distribution. Each channel of the image was analyzed separately if necessary. All test show that the background intensities of these images belong to a normal distribution. It might be reasonable to extrapolate that background intensities in biological images, always follow a normal distribution.

B.3 Characterizing illumination uniformity

Illumination Uniformity was described by a metric produced from the following algorithm. A large background region (with no imaged object) was selected from an image. Then that region was subdivided into several sub-regions. The pixels intensities of the sub-regions were compared with kstest2, with 0 meaning these regions likely have the same background distribution and 1 meaning that the regions likely have different background distributions at the 0.05 significance level. All pair-wise comparisons are summed and divided by the number of pairwise comparisons done. This resulting value is the Illumination Difference of an image. The Illumination Uniformity is 1-Illumination Difference.

The assumption that the background is of the normal type was extrapolated from the above analysis in B.2.3. The Illumination Uniformity (using 10 by 10 sub-regions) of the image sequence were calculated and plotted below. The same number of sub-regions was used for each image as analysis showed that the number of sub-regions affected the value for the Illumination Uniformity, so keeping number of sub regions constant allowed pairwise comparisons of the Illumination Uniformity for different images. This metric can be calculated in O(number of pixels^2) per image, which may make this metric an unreasonable for either large images, large image sequences or both, though modifying the algorithm may improve performance.

Illumination Uniformity vs. Time



B.4 Microscope pixel calibration

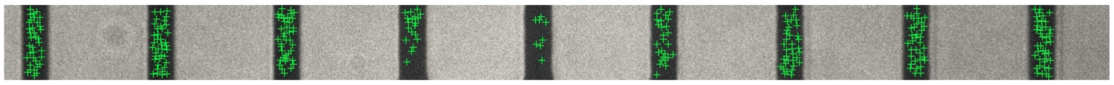
B.4.1 manual calibration

We choose to crop an area with width customized by user (each division is 0.01mm) and repeat this process for several times (set by user) to calculate the average of pixel size. For example, for 10X calibration image, if we choose to crop an area with width of 10 divisions for 5 times, the average of pixel size we get is 641nm, and the standard deviation is 4nm. Apparently, the more times you choose to crop the area and the higher the magnification is, the more accurate the pixel size would be obtained.

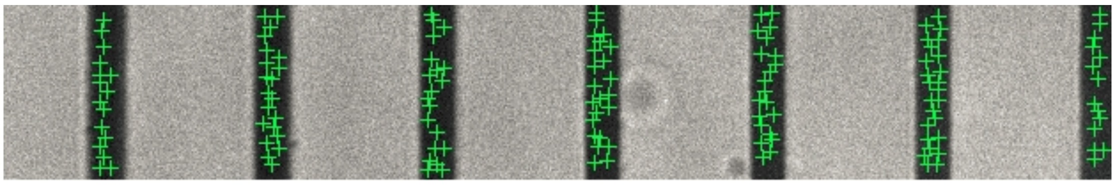
B.4.2 semi-automated calibration

Since the original image contains the number mark like 10, 20, etc, we decide to crop a region for analysis, which makes it a semi-automated calibration. First, we crop a long region with short height so that it does not contain black line in horizontal direction(like '-'). Then we find the significant local minimums using a way like the what we do in the project 2. After that we calculate the average X-axis position for the local minimums within a black line, then we get the average distance between every two next to each other, which tells us an approximate number of pixels within a division(0.01mm). We note that the black line width defer among the images we use(10x,20x,60x,100x), so we have to adjust the ratio and distance within our code when doing calibration for different images. Here, we show some result pixel sizes we got with the local minimum detection images.

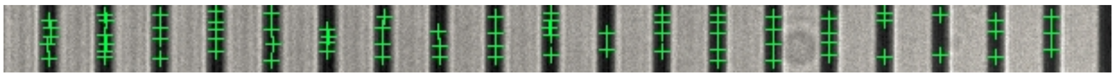
100x_calibration: 6.3807e-07



60x_calibration: 6.4071e-07



20x_calibration:6.4418e-07



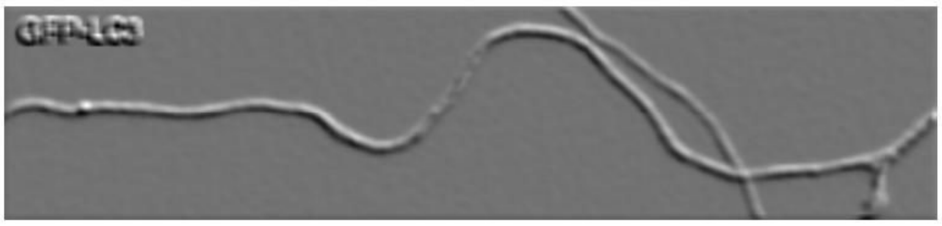
10x_calibration: 6.4302e-07

B.5 Implementation of a directional anisotropic filter

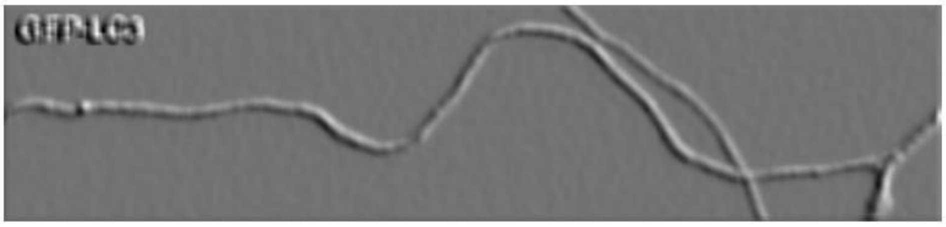
The goal of this section is to visualize the difference between anisotropic filtered image sheared at different angles. We were unable to get the fast anisotropic gauss filtering method listed in the Geusebroek and Smeulder's paper to work correctly so our approach for this extra-credit task was to modify the directional anisotropic gauss filter found in lecture 14's class demo code on Canny edge detection.

For the following images, please run test_b5.m. Shearing angle and sigmas settings are inside that file.

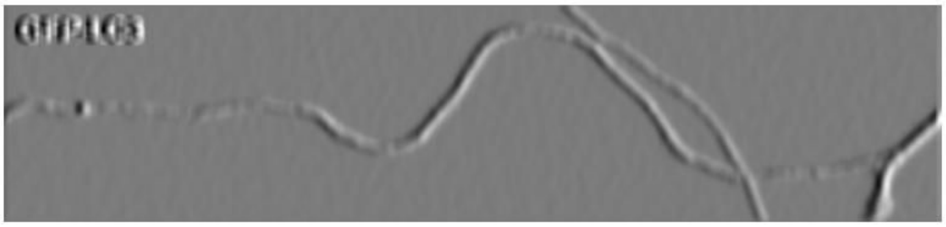
Visualizing t-direction sheared at 30 degrees with longitudinal sigma = 10 and lateral sigma = 5



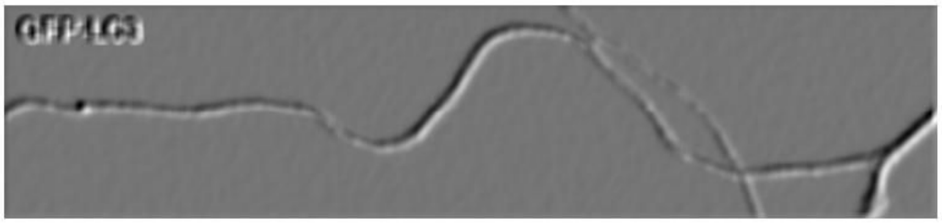
Visualizing t-direction sheared at 60 degrees with longitudinal sigma = 10 and lateral sigma = 5



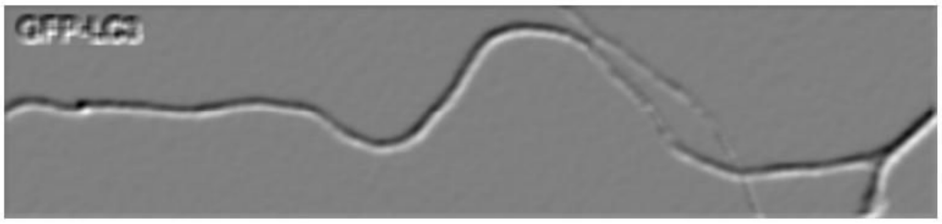
Visualizing t-direction sheared at 90 degrees with longitudinal sigma = 10 and lateral sigma = 5



Visualizing t-direction sheared at 120 degrees with longitudinal sigma = 10 and lateral sigma = 5



Visualizing t-direction sheared at 150 degrees with longitudinal sigma = 10 and lateral sigma = 5



C. Curvilinear feature detection

C.1 Implementation of the Steger's algorithm

We implement Steger's algorithm in several steps as follows.

Step 0: Identify maximal line width

We crop an area that exactly covers the line for 3 times and get the average as the output line width. Then we calculate the sigma for gaussian kernel which satisfies σ = width/sqrt(3). Then line width we get is around 7 pixels and the sigma is around 4.6 pixels.

Step 1: Hessian matrix calculation

First, we implement second derivatives for 2D Gaussian function and denote these partial derivatives as G_{xx}, G_{yy}, G_{xy} respectively. Then we convolve the original image with different second derivatives of Gaussian and obtain corresponding I_{xx}, I_{yy}, I_{xy} ($I_{xy} = I_{yx}$).

Thus, we could get the Hessian matrix

$$H = \begin{bmatrix} \mathrm{Ixx}, \mathrm{Ixy} \\ \mathrm{Iyx}, \mathrm{Iyy} \end{bmatrix}$$

For each pixel's Hessian matrix, we search for the eigenvector corresponding to the maximum eigenvalue, and we get (n_x, n_y) for this eigenvector.

Step 2: Calculate the first and second directional derivative

According to the equation, we get r' and r'' for each pixel.

$$r' = [n_x \quad n_y] \begin{bmatrix} \mathrm{fx} \\ \mathrm{fy} \end{bmatrix} \qquad r'' = [n_x \quad n_y]\, H \begin{bmatrix} nx \\ ny \end{bmatrix}$$

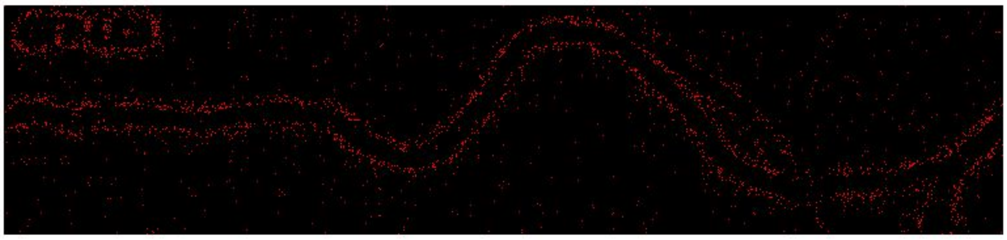
Step 3: Determine location of the local intensity maximum

We implement and get x^* as $x^* = r'/r''$.

Step 4: Test whether it is a center line point

For each x^* we check whether it is within the interval [-0.5, 0.5] and mark such points which match the condition as red.

To get the result better, we set up threshold to filter those noise points. For example, we filter those points whose maximum eigenvalue is negative. We also find that the smaller the sigma is, the more closely the marked points would be to the line of image.



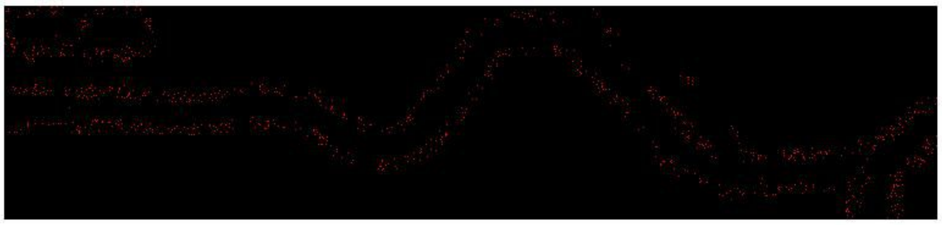
C.2 Implementation of the pixel linking operation

The goal of this section is to join the detected edge pixels in order fully show the edges of the original image. We tried a couple of methods: seeded growth method, and plotting method.

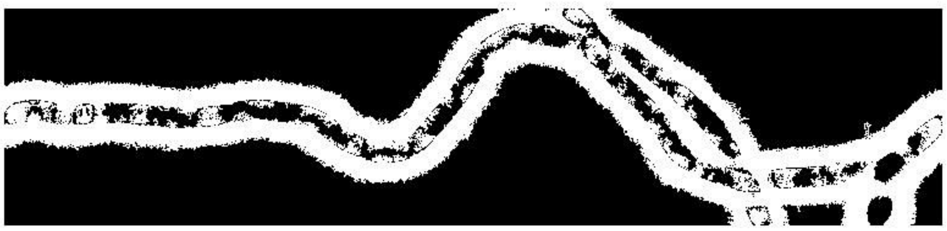
First, we convert the x* points to binary (0 or 1) on a coordinate system (two column matrix corresponding to x and y) and a corresponding column vector with the point's x* magnitude. Then, we applied the pixel linking methods.

In the pixel linking method, we first find a point with signal as a seed. Then, from the seed, we search its surrounding for the nearest pixel with a x* signal and interpolate until the pixels are joined.
In the following image, we seeded with pixel at coordinate (131, 760).

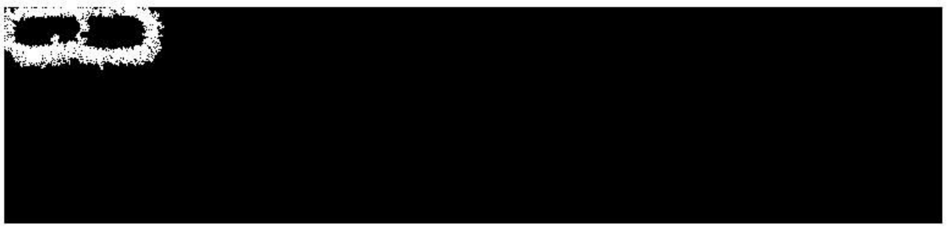
Coordinates of edge features from C.1



Joined with seed at (131, 760); please run test_c2.m (requires result from test_c1.m)



With region growth method, pixel linking operation has to be guided by a human. This is because the initial seed is important for region growth. For example, a bad seed at (6, 95) will result in pixel linking at an area that is not of our interest. To get this result, please uncomment the line with the comment "Bad seed example" in test_c2.m.



References

[1] C. Steger, An unbiased detector of curvilinear structures, IEEE Trans. Pattern Analysis and Machine Intelligence, vol. 20, pp. 113-125, 1998.
[2] J.-M. Geusebroek, A. W. M. Smeulders, and J. van de Weijer, Fast anisotropic Gaussian filtering, IEEE Trans. Image Processing, vol. 12, no. 8, pp. 938-943, 2003.