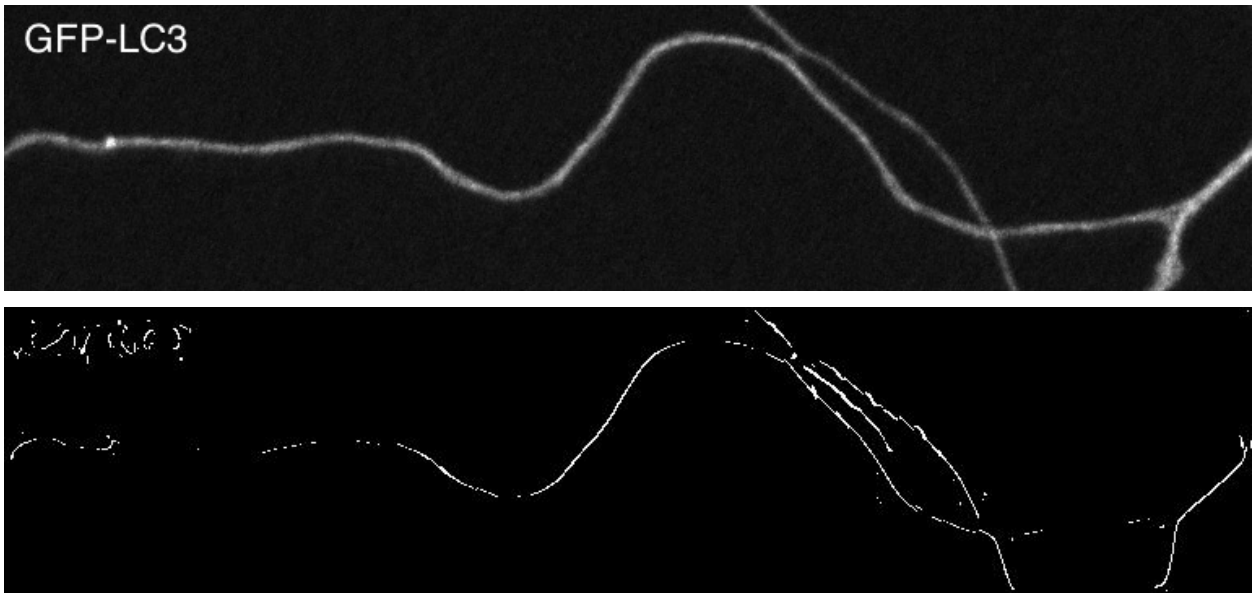
Bioimage Informatics Project Assignment 04

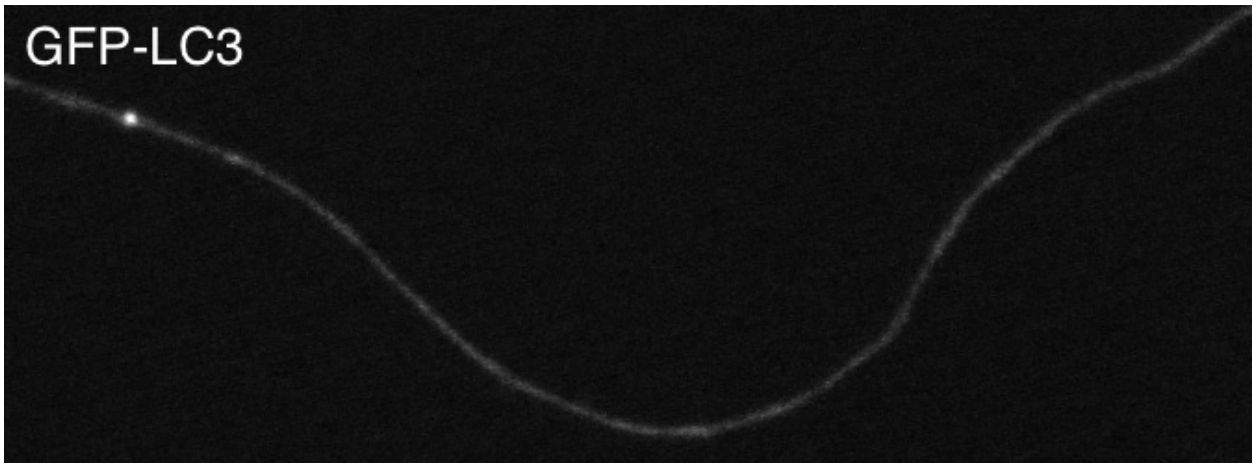
Aniket Rao (aniketr), Easwaran Ramamurthy (eramamur), Patrick Ropp (propp), Siddharth Gurdasani (sgurdasa)

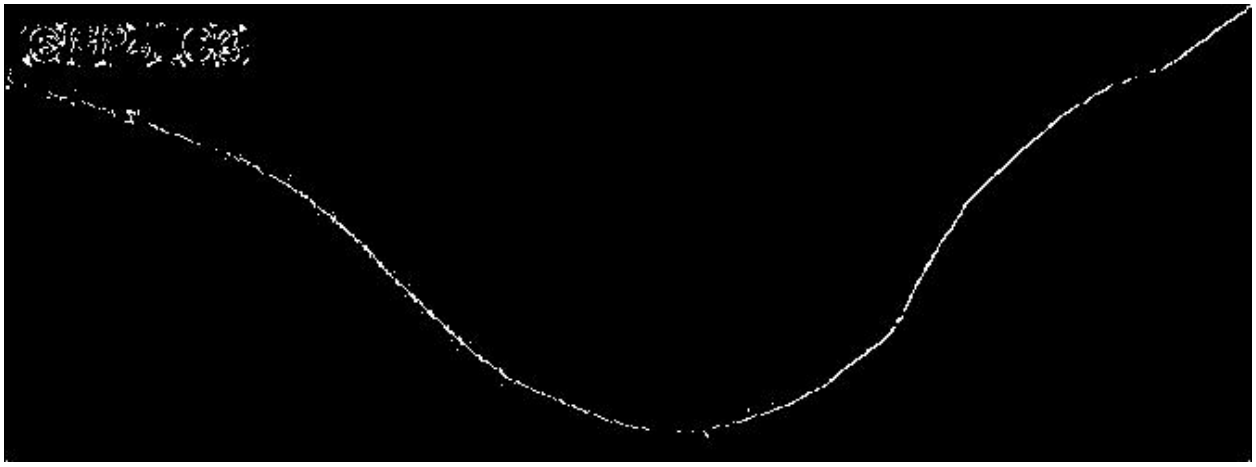
B. Curvilinear feature detection

We implemented the Steger's Curve Detection Algorithm as described in the paper, An Unbiased Detector of Curvilinear Structures, Steger et al., 1998. We analyzed the directional gradients to find maximal ridges and then applied a mask to remove noise.

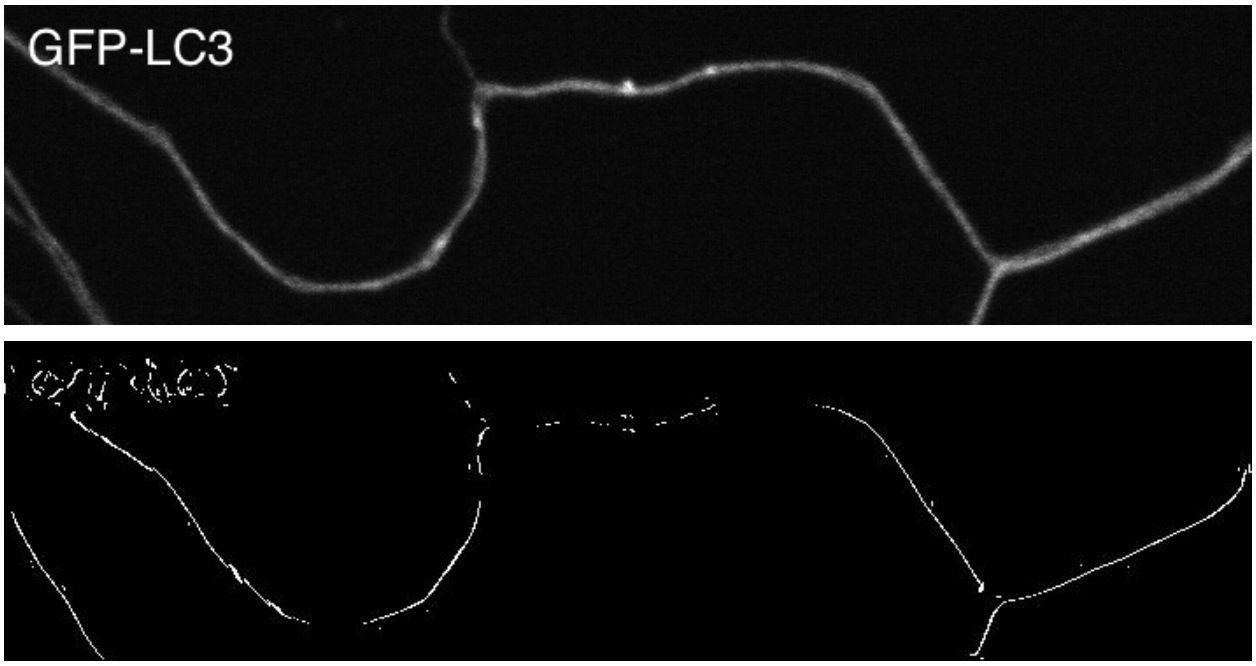


'Curv_det_01.tif' and 'Curv_det_out01.tif', set with a kernel width of 6 pixels and a threshold of 0.2.





‘Curv_det_02.tif’ and ‘Curv_det_out02.tif’, set with a kernel width of 2 pixels and a threshold of 1.0.



‘Curv_det_03.tif’ and ‘Curv_det_out03.tif’, set with a kernel width of 6 pixels and a threshold of 0.2.

We found that using a lower kernel width with a higher threshold retained more of the line, but with a higher rate of false positive noise just off of the curve. A higher kernel width with a lower threshold worked better at suppressing noise, but at the cost of weaker detection of faint regions of the curves.

Code is called ‘StegerDetection.m’ in folder B and takes an image name, kernel width, and threshold as parameters. It returns a bitvector of the detected curves. The supporting code for ‘StegerDetection.m’ is also found in folder B.
Ouputted images (*Curv_det_out01.tif, Curv_det_out02.tif,* and *Curv_det_out01.tif*) can be found in the selected images folder.

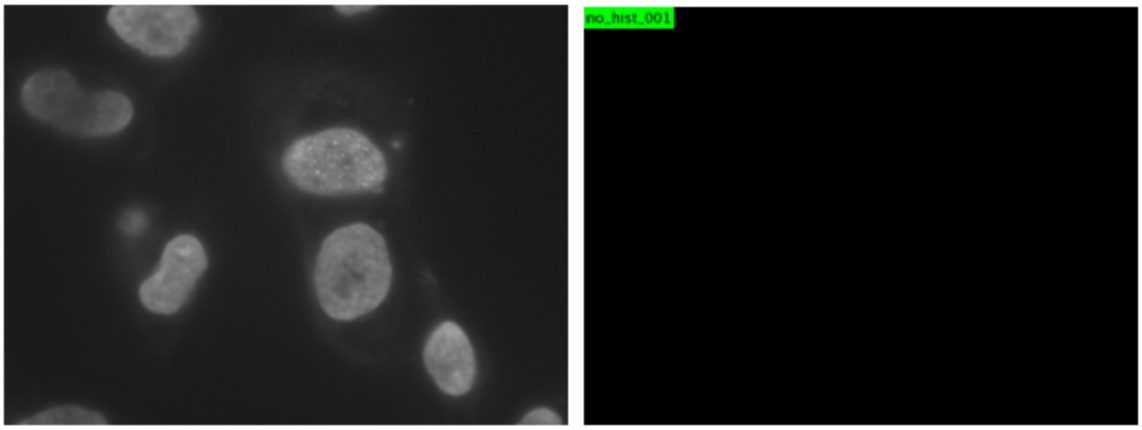
C. Image Segmentation

C.1 MATITK was provided in the homework handout and we downloaded it, extracted the windows executables and added the path to MATLAB. Dataset was also download from the homework handout.

C.2.1 We applied both Otsu Threshold Segmentation (SOT) and Shape Detection Level Set Filter (SSDLS) to the two static images 60x_02.tif and Blue0001.tif. The results of segmentation of the static images using both techniques are summarized below:

OTSU THRESHOLD SEGMENTATION (SOT)

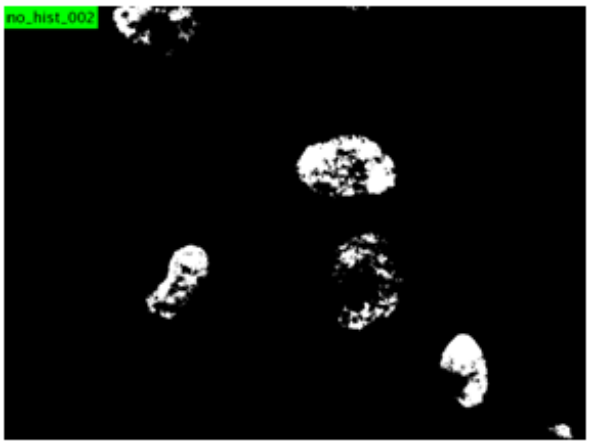
SOT takes in one argument which is the number of histogram bins (n) that the algorithm splits the image pixels into for thresholding. The recommended value for this parameter is the maximum intensity of a pixel in the image being segmented. However, we expect better results at even a small number of histograms and so we decided to tune this parameter using a simple for loop that explores the parameter range from 0 to $max(I(:))$ histograms for the Blue0001.tif image. For the 60x_02.tif image, since the image was a 16 bit image, the maximum intensity was very high (and difficult to tune) and so we decided to use 255 as the number of histograms to make it easier to find the right parameter value in a smaller range. The results were then collected and converted into a video for easier visualization. The resulting video was then visually inspected to figure out which parameter setting worked best for the segmentation. The videos are submitted in the files Mov_60x_02_SOT and Mov_Blue0001_SSDLS. For the Blue0001.tif image, the results are shown below:



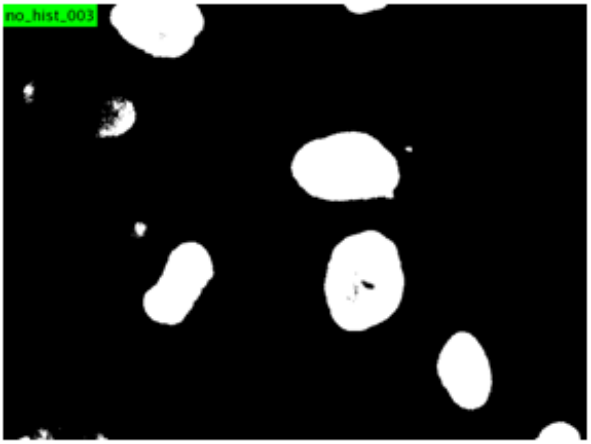
Original Image n = 1

As expected, with 1 histogram, all image pixels went into the same histogram bin and could not be separated into objects.

As we started increasing the number of histograms starting from n=2, we see objects in the image start to get segmented from the background.

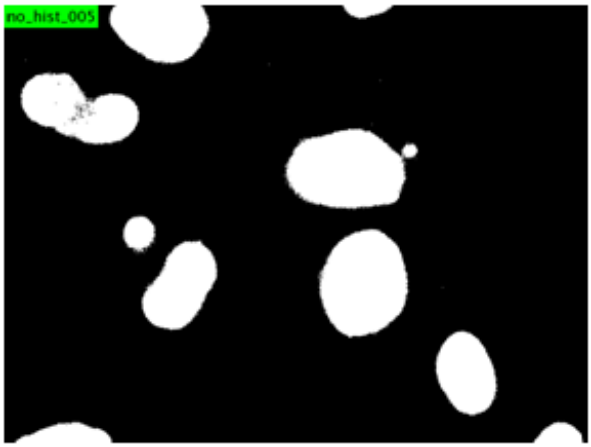


n = 2

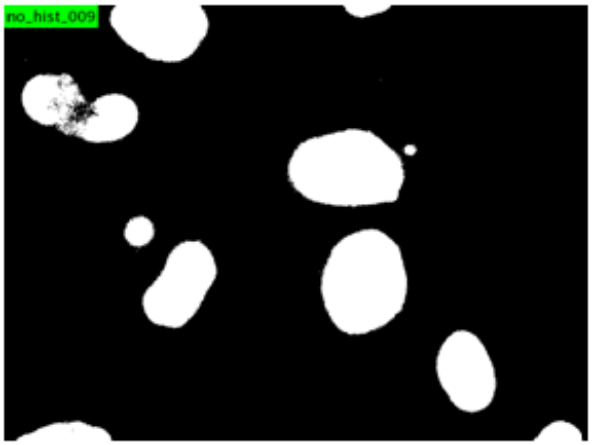


n = 3

We found that n = 5 gave the best segmentation visually although a good segmentation could also be obtained at n = 9 since these two settings weren't breaking up the object at the top left in the image.

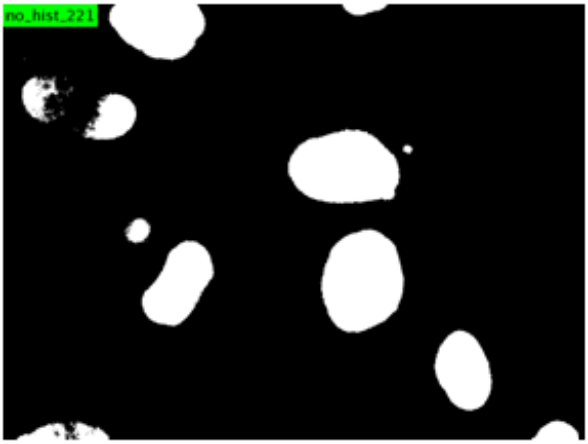


n = 5



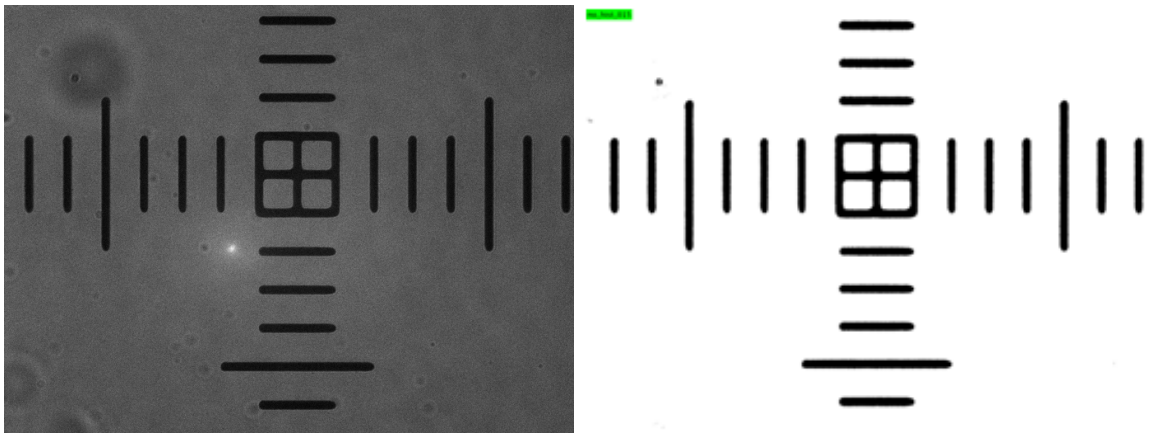
n = 9

Further, we observed that the image segmentations altered periodically in steps of 4 i.e. the image segmentation with n = 8 was similar to the image segmentation with n = 12. Furthermore, the segmentation quality visibly decreased as we increased the number of histograms and therefore, we report n = 5 as the best parameter setting for SOT. At n = 221 (shown below) which was the maximum intensity of a pixel in the image, we found that it was not quite as good a segmentation as when n = 5.



n = 221

For the 60x_02.tif image, we observed again that there was periodicity in the results although almost all the parameter settings gave similar results as shown below. However, some segmentations in the period were visibly better than others as a few of the settings failed to remove out the noise from the images. A good value of n for this image was n = 15 although several other values gave similar segmentation results. The segmentation for n = 15 along with the original image are shown below:



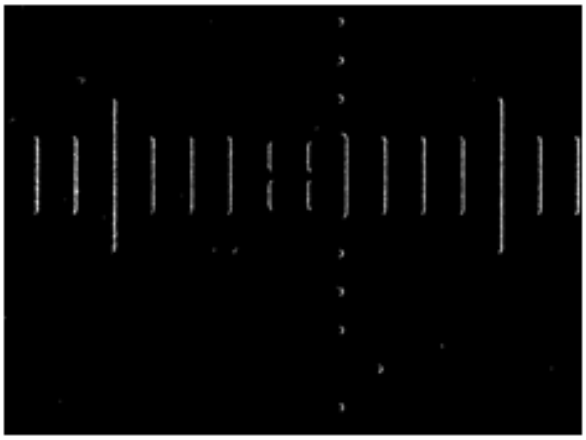
Original Image Segmentation with n = 15

The code for OTSU segmentation and for tuning parameters is included in the run_tuneSOT.m, tuneSOTParam.m and performSOT.m. The results for SOT parameter tuning on the two static images along with the corresponding image sequences and movies are submitted in the folders C2_1 results_60x_02_SOT and C2_1 results_Blue0001_SOT.

SHAPE DETECTION LEVEL SET FILTER (SSDLS)

There are 2 major parameters that can be tuned for the SSDLS algorithm namely, the propagation scaling parameter, the curvature scaling parameter. The other two parameters: maximumRMSerror and the number of iterations are usually used with the default values which are 0.02 and 100 respectively.

Having experimented with several different values of the first two parameters and not observing any changes, we could not observe any visual improvement in the segmentation. Therefore, we went with the default parameters which was scaling = 1, curvature = 1 and the result is shown below for both the static images:

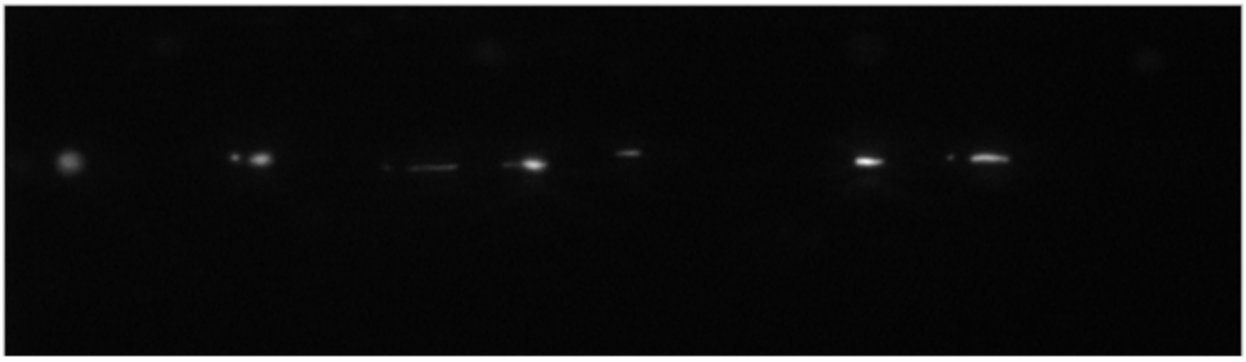


60x SSDLS result

Blue SSDLS result

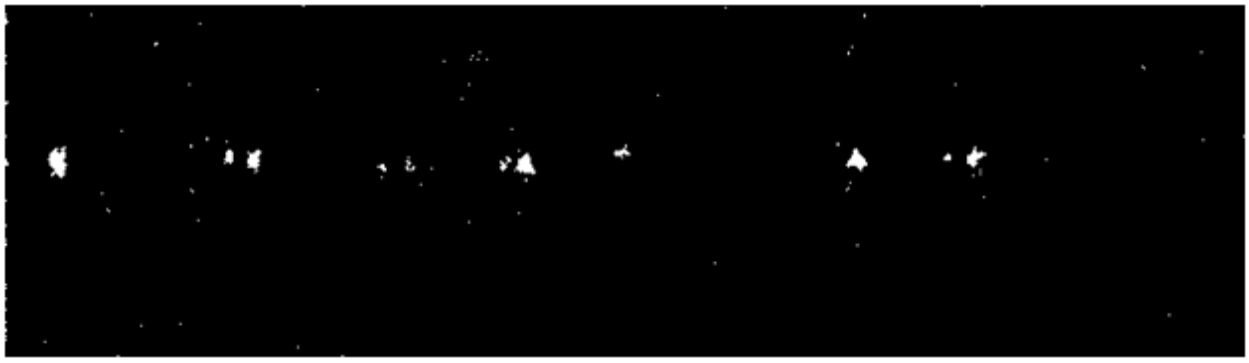
SSDLS in general performs worse than SOT on both the images as can be seen from above. The code for SSDLS is submitted in the file performSSDLS.m. The results for the blue and 60x image with the above given parameters are also submitted in the folder C2_1 results_SSDLS.

C.2.2 The two segmentation algorithms were applied to the entire image sequence and both the movies are submitted in the avi files named C2_2_SOT_results and C2_2_SSDLS_results. For SOT, the maximum image intensity was used as the number of histograms and the default parameters were used for SSDLS. We can see from the movies that again SOT performs much better and SSDLS fails to recognize simple shapes that are present in the images. As an example, the segmented first frame is shown below along with the original image below:



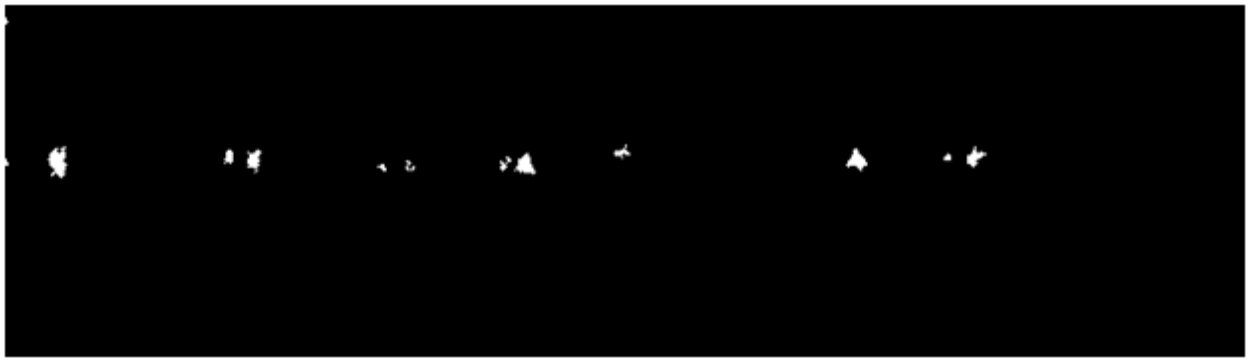
Original Image (Frame 1)

SOT Segmentation



SSDLS Segmentation (without using morphology operations to remove noise)

As can be seen the SSDLS results give a lot of background pixels which were removed by we removed by applying a morphology operation using the matlab function bwareaopen(I,4) to remove all hits that were surrounded by less than 4 pixels.



SSDLS Segmentation (using morphology operations to remove noise)

The code for performing image segmentation on the MitoGFP image sequence is submitted in performSOTOnImageSequence.m and performSSDLSOnImageSequence.m. Further, the results for the segmentation are submitted in folders named C2_2_SOT_results, C2_2_SSDLS_results and C2_2_SSDLS_results_with_filtering. These folders contain the segmented image sequences along with an accompanying movie in avi format.

Manual segmentations for the first three images in the image sequence were constructed by free hand drawing in ImageJ followed by removal of background pixels. The manually segmented images are submitted in files frame1, frame2 and frame3 in selected_images folder. The volumetric overlap error, false positive rate and false negative rate were calculated for the first three images in the sequence using both algorithms (code submitted in file evaluateSOT.m, evaluateSSDLS.m and calcErrors.m) and the results are tabulated below:

SOT:

Performance Measure	**Frame1**	**Frame2**	**Frame3**	**Average**
Volumetric Overlap Error	0.0053	0.0055	0.0051	0.0053
False Positive Rate	0	0	0	0
False Negative Rate	0.4170	0.4245	0.3826	0.4081

SSDLS:

Performance Measure	**Frame1**	**Frame2**	**Frame3**	**Average**
Volumetric Overlap Error	0.0066	0.0059	0.0058	0.0061
False Positive Rate	0.5345	0.4189	0.4899	0.4811
False Negative Rate	0.7900	0.7306	0.7527	0.7578

As can be seen, SSDLS performs worse than SOT on all different measures. SOT does not give any false positives at all. The high false positive rate in SSDLS occurs because it predicts several objects in background regions that occur due to noise. Even after careful filtering of the output using the MATLAB function bwareaopen( ) we weren't able to remove all of these. The high false negative ratee issue in SSDLS might also be due to the fact that SSDLS results in a grayscale output and we have to threshold it to convert it to a binary image which might filter out some true positives. The volumetric overlap error in both SSDLS and SOT is comparable however, which shows that SSDLS is still a viable method. Overall, however, it can be seen that SOT is better than SSDLS for all considered images. However, SSDLS could perform much better if a different initiating contour was used and this is something to explore in the future.

C.2.3. Theoretical Background

OTSU THRESHOLD SEGMENTATION

Otsu's method is an image processing method that is used for image thresholding and does so by finding a threshold that minimizes the intra class variance σ_w^2 (i.e. the intensity variances within pixels belonging to the same binary class) which is derived as follows:

$$\sigma_w^2(t) = w_1(t)\sigma_1^2(t) + w_2(t)\sigma_2^2(t)$$

where t is a given threshold, $w_i(t)$ is the probability of each class i at threshold t and $\sigma_i^2(t)$ is the variance of class i. Otsu also showed that minimizing the intra-class variance is equivalent to maximizing the inter-class variance given by:

$$\sigma_b^2(t) = \sigma^2 - \sigma_w^2(t) = w_1(t)w_2(t)[\mu_1(t) - \mu_2(t)]^2$$

The class probability $w_1(t)$ is calculated as follows:

$$w_1(t) = \sum_{i=0}^{t} p(i)$$

Here $p(i)$ is the probability of histogram bin i. $w_2(t)$ is calculated as:

$$w_2(t) = 1 - w_1(t)$$

while the class mean $\mu_1(t)$ is calculated as:

$$\mu_1(t) = [\sum_{i=0}^{t} p(i)x(i)]/w_1(t)$$

where $x(i)$ is the center of histogram bin i. These values can be computed similarly for the other class.

The algorithm proceeds by first assigning pixels to histogram bins according to intensity and then stepping through each possible threshold intensity to find the intensity at which σ_b^2 is maximum. At any given step in the algorithm all pixels in histogram bins above the threshold are assigned a value of 1 and all pixels below are assigned a value 0. The variances are then calculated and the maximum inter-class variance is found at a particular threshold t_*, which is used to produce the final binary segmentation. This leads to an accurate and powerful segmentation that incorporates global information

from the image. Otsu's method can also be extended to incorporate multiple classes in a technique called multi-thresholding and this is another viable alternative when the binary thresholding fails.

SHAPE DETECTION LEVEL SET FILTER (SSDLS)

The SSDLS method is a level set based segmentation method by propagating an initial contour outwards until it slows down near object edges. The initial set is the original image which is considered as the zero contour and it is propagated outwards based on a speed function which is an edge potential map that depends on the image gradient. The edge potential feature image is defined as follows:

$$g(I) = 1/(1 + |(\nabla * G)(I)|)$$

where I is the image intensity and $(\nabla * G)(I)$ is the gradient of gaussian convoluted image. However this can also be read as the gaussian of the image derivative. The value of $g(I)$ is low at shape boundaries and hence the propagation step proceeds slowly at these points. The SSDLS algorithms takes in two parameters namely the propagation scaling parameter which specifies whether to propagate the contour outwards or to propagate it inwards. Usually it is propagated outwards and so the propagation scaling parameter is set to 1. The curvature scaling parameter defines how smooth the contour is and a higher value of this parameter results in a much smoother contour and a much smoother final segmented object.

C.4

Algorithm 1: Normalized Cuts and Image Segmentation

Normalized cut algorithm is a graph partition algorithm which forms a weighted undirected graph G = (V, E) from the image, where V is the set of nodes or the points in the feature space and E is the set of edges between every pair of nodes. Weight of the edge is represented by w(i,j) and is the function of similarity between the nodes i and j.

The partitioning of the graph G into two disjoint sets is called a cut and is defined as the total weight of the edges that have been removed to form the partition.

$$cut(A, B) = \sum_{u \varepsilon A, v \varepsilon B} w(u, v)$$

Further to remove the bias of partitioning small components, the above equation of cut is normalized by the total edge connections of a node with all other nodes in the graph i.e.

$$assoc(A, V) = \sum_{u \varepsilon A, t \varepsilon V} w(u, t)$$

Following is the equation for normalized cut:

$$Ncut(A,B) = \frac{cut(A,B)}{assoc(A,V)} + \frac{cut(A,B)}{assoc(B,V)},$$

In the similar fashion, a measure for total normalized association within a group or component is as follows:

$$Nassoc(A,B) = \frac{assoc(A,A)}{assoc(A,V)} + \frac{assoc(B,B)}{assoc(A,V)}$$

where assoc(A, A) and assoc(B,B) are the total weights of the edges connecting nodes in the component or group A and B respectively.

Thus, normalized cut algorithm minimizes the similarity or association between the groups and maximizes the association within the groups.

Normalized cuts algorithm works as follows:

1. Construction of a weighted graph G = (V, E) is done by considering each pixel as a node and connecting those nodes by an edge. The weight of the edge is the likelihood of the two pixels belonging to a single component or cluster. Following is the equation to determine edge weight:

$$w_{ij} = e^{\frac{-\|F(i)-F(j)\|_2^2}{\sigma_I^2}} * \begin{cases} e^{\frac{-\|X(i)-X(j)\|_2^2}{\sigma_X^2}} & \text{if } \|X(i)-X(j)\|_2 < r \\ 0 & \text{otherwise.} \end{cases}$$

where X(i)-X(j) is the spatial location difference between the two nodes/pixels i and j. Similarly F(i) - F(j) is the feature difference for example brightness or intensity. σ_I^2 and σ_X^2 are the two parameters in the algorithm that can be tuned for getting segmentations. σ_I^2 is the standard deviation of the feature difference and σ_X^2 is the standard deviation of the spatial difference in the location of the two nodes i and j. If we decrease any of these standard deviations, the weight of the edge decreases resulting in more segmentations i.e. likelihood that the two objects belong to one component decreases.

2. Solve the following equation :

$$(D - W)x = \lambda Dx$$

to get the eigenvectors with the smallest eigenvalues, where D is a NxN diagonal matrix with d on the diagonal,

$$d(i) = \sum_j w(i,j)$$

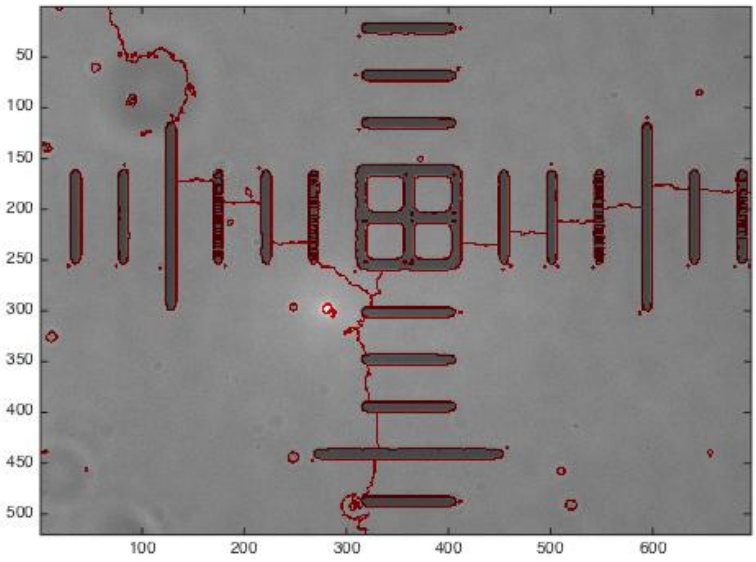
So d is the summation of the edge weights from node i to all other nodes in graph G.

W is a NxN symmetric matrix of weights of all the edges i.e. $W(i,j) = w_{ij}$

3. The graph is then bi-partitioned using the second smallest eigenvector and as eigenvectors can have continuous values, a splitting point has to be chosen. Splitting point can be median value or 0 or any point which minimizes the Ncut value.

4. Based on a stability criterion which measure the degree of smoothness in the eigenvector values, check the stability of the cut and check for the Ncut value to be below a specified threshold. These checks are done to make a decision whether the current partition should be further divided or not.

5. Recursively run the algorithm and repartition the segmented regions of the image if needed.



Segmentation of 60x_02.tif image

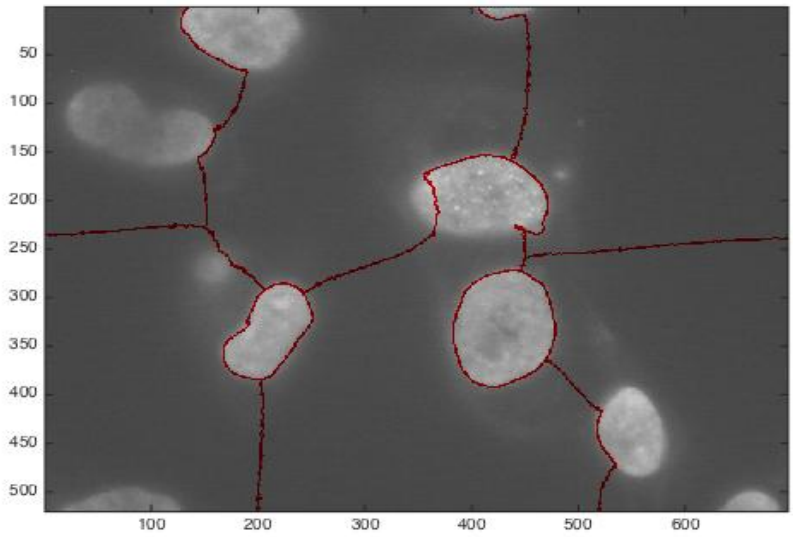


Image: Blue0001.tiff edge variance = 0.1 sample_rate = 0.3 radius=10 number of segments=8

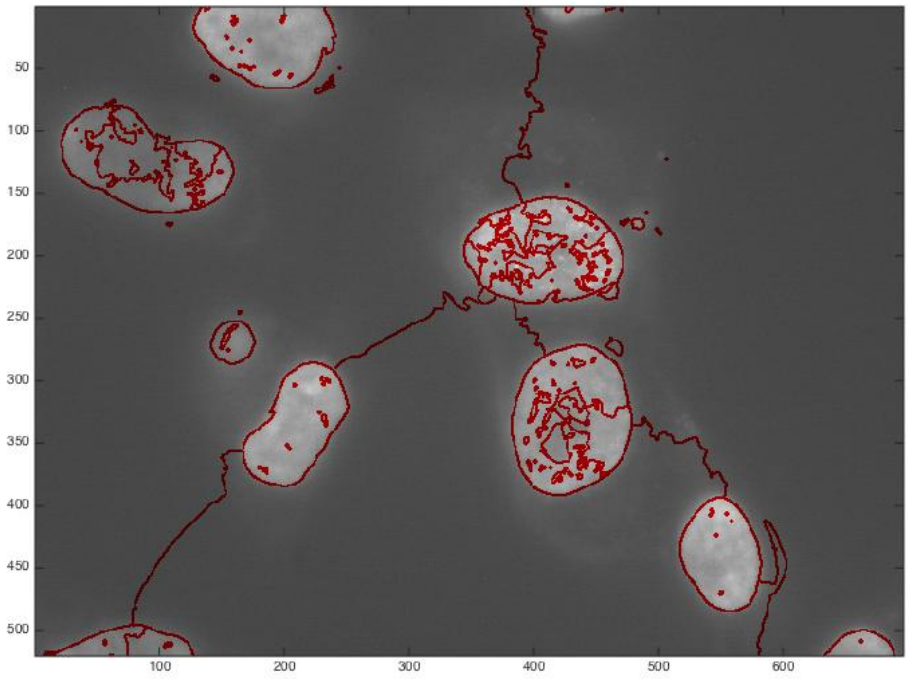


Image: Blue0001.tiff edge variance=0.015 sample_rate=0.3 radius=10 number of segments=8

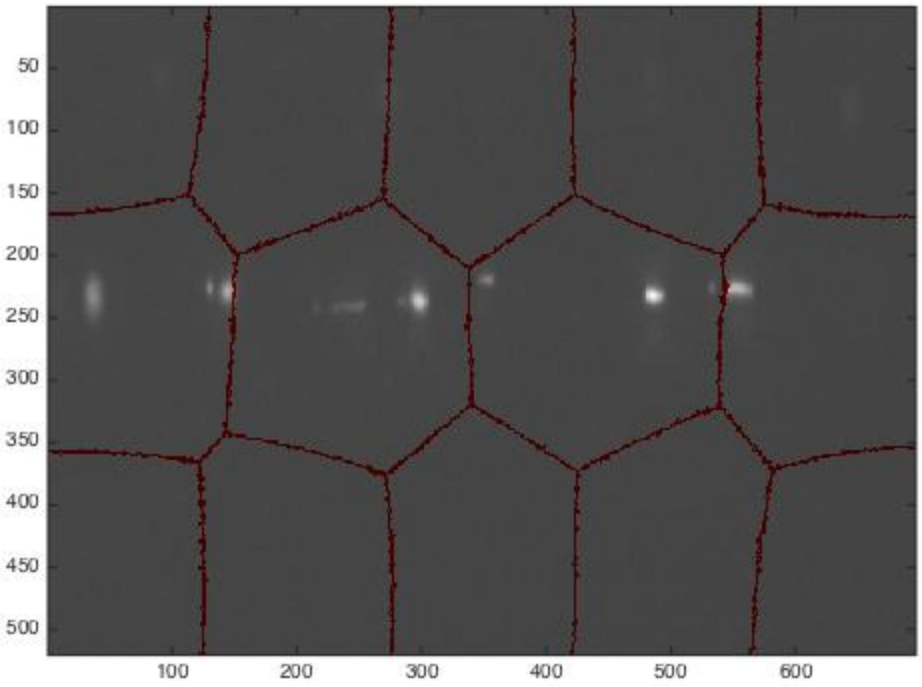


Image:MitoGFP_LgtGal4_a01r01s02001.tif, number of segments=14 with the default parameters i.e. edge variance or sigma f 0.1 sigma x 0.3 and R=10

Code submitted online for parameter tuning: AlgorithmjshiTest.m. This function is just a wrapper function, basically to tweak parameters and is taken from demoNcutImage.m (provided by the authors of the algorithm). This function takes an input as the name of the image file and also the number of segments to form within the image as a parameter.

Algorithm 2: Efficient Graph based image segmentation

The graph based segmentation approach by Pedro et al. assumes an undirected graph made of pixels as nodes and edges connecting neighboring vertices. The weight of an edge is the non-negative measure of dissimilarity between the vertices in the form of pixel intensities, color motion etc. According to this method, a segment S in the image is a partition of set of vertices V such that a component C ε S is connected component representing the graph $G' = (V, E')$ where $E' \subseteq E$, here E is set of edges in the graph G. Thus this metric favors similarity (low weighted edges) within the components and dissimilarity (high weighted edges) between components.
The internal difference of a component $C \subseteq V$ is defined as follows:

$$Int(C) = max_{e \varepsilon MST(C,E)} w(e)$$

That is maximum weighted edge (high dissimilarity or low similarity) in the MST. And the difference between two components is $C_1, C_2 \subseteq V$ is defined as the minimum weighted edge (lowest dissimilarity i.e. high similarity) connecting the C_1 and C_2:

$$Dif(C_1, C_2) = min_{v_i \varepsilon C_1, v_j \varepsilon C_2, (v_i, v_j) \varepsilon E} w((v_i, v_j))$$

In case of no edge connecting the two components, the difference is $Dif(C_1, C_2)$. If the difference between the two components C_1 and C_2 satisfies the below given equation i.e. the difference between two components is larger than the internal difference within at least one of the components Int(C_1) or Int(C_2), then there is an evidence of boundary and the two components can be segmented or else they will have to be merged. Following is the equation for pairwise comparison of two components:

$$D(C_1, C_2) = \begin{cases} true\ if\ Dif(C_1, C_2) > M\ Int(C_1, C_2) \\ false\ otherwise \end{cases}$$

where M Int(C_1,C_2) is

$$M\ Int(C_1, C_2) = \min\left(Int(C_1) + \tau(C_1), Int(C_2) + \tau(C_2)\right)$$

Furthermore, there is a threshold function τ that controls the extent to which the difference between the two components should be greater than the internal difference within any of the two components. It is defined as follows:

$$\tau(C) = k/|C|$$

τ depends on the size of the component |C| as in the worst case when |C| is 1, then Int(C) would be 0 and so the method will be biased towards small components. Here k is a constant parameter, which can be changed to prefer larger components. Also different shapes of the components can be preferred, by changing the τ threshold function, to larger values for undesired shape and smaller values for desired shapes. However, in accordance with the equation for M Int(C_1,C_2), only one of the two neighboring components of the desired shape will be selected.

Following is the algorithm by Pedro et al:

Input to the algorithm is a graph G =(V, E) with n vertices and m edges and the output is a segmentation of V into components $C_1, C_2, \ldots C_n$

1. First sort the edges by non decreasing edge weight, so that low weighted edges (less dissimilar or more similar) come before the high weighted edges (more dissimilar).
2. Run a loop for q = 1, …. , m and inside the loop update S^q from S^{q-1} as follows:

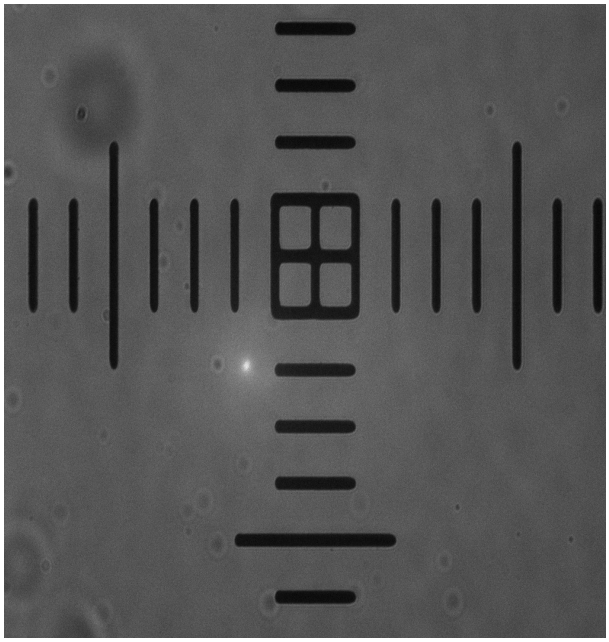
   a. If C_i^{q-1} is a component of S^{q-1} and $v_i \varepsilon\ C_i^{q-1}$ and $v_j \varepsilon\ C_j^{q-1}$, then if $C_i^{q-1} \neq C_j^{q-1}$ and

   $$w(o_q) \leq MInt(C_i^{q-1}, C_j^{q-1})$$

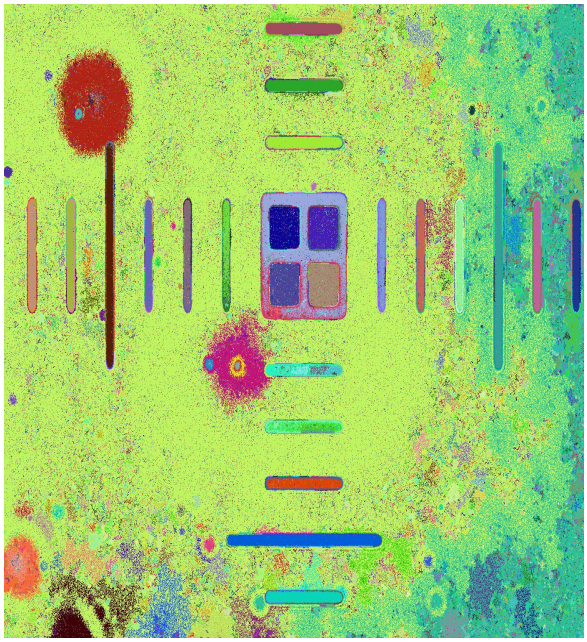
   then S^q would be formed from S^{q-1} by merging C_i^{q-1} and C_j^{q-1}, and otherwise S^q will be same as S^{q-1}.

3. Return $S = S^m$ at the end of m iterations.

Following are the results of parameter tuning with the above mentioned algorithm:

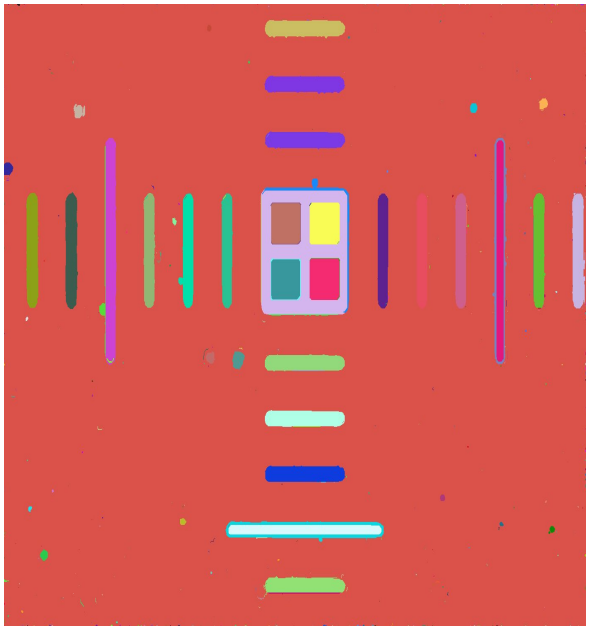


original image (60x_02.tif)

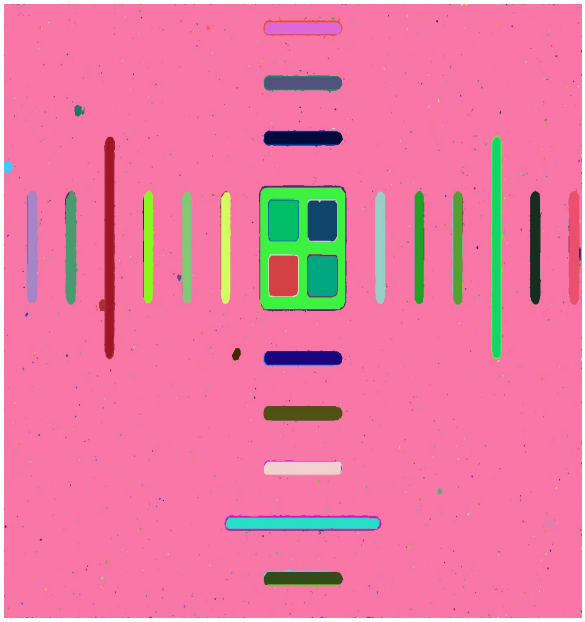


sigma 0 k 0 min 0

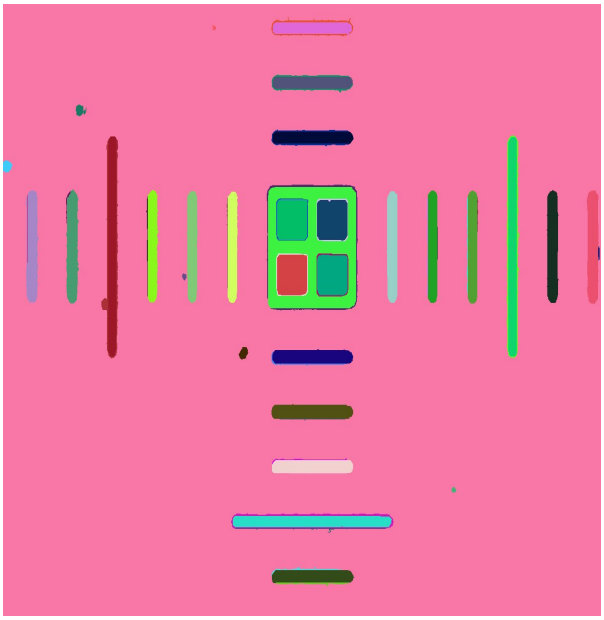
sigma 0.5 k 500 min 50



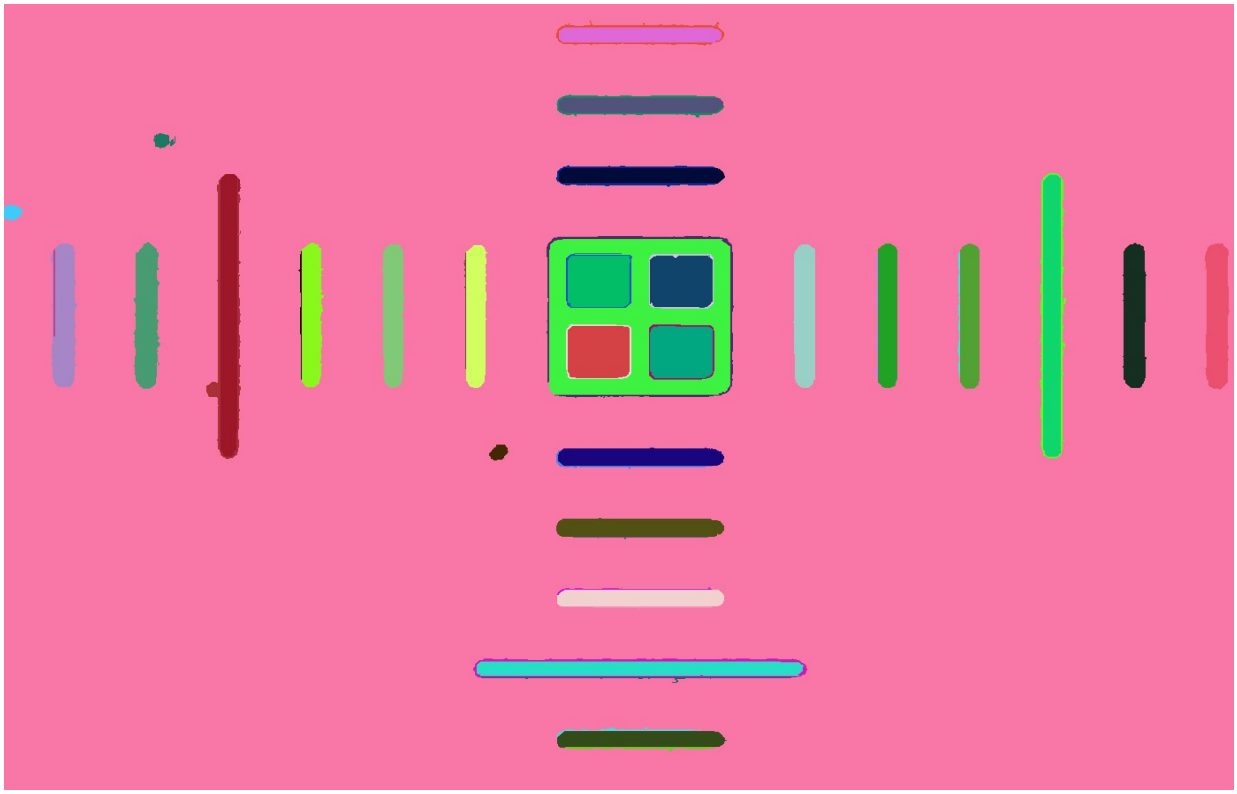
sigma 1.5 k 100 min 0



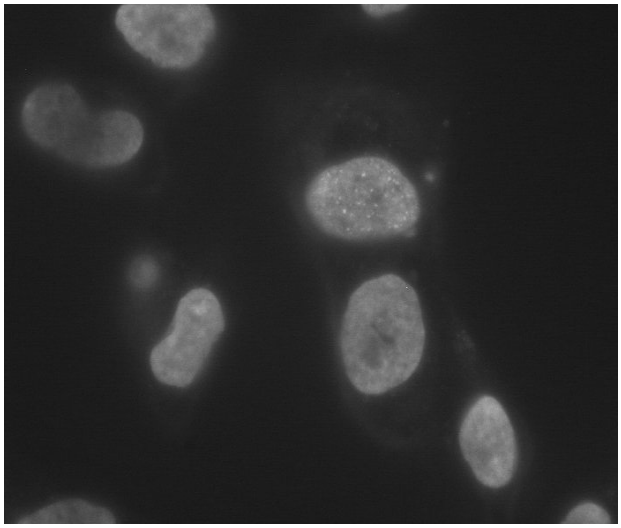
sigma 1 k 100 min 0



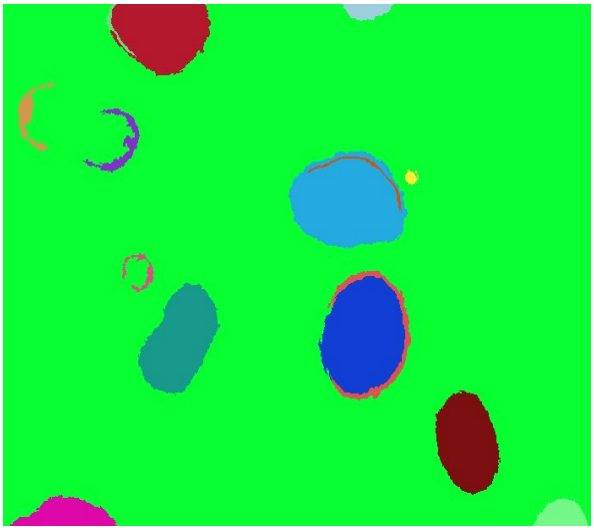
sigma 1 k 100 min 50



sigma 1 k 100 min 100



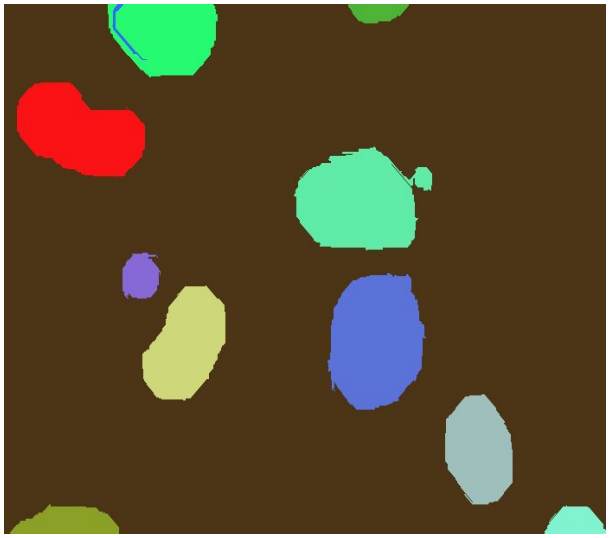
original image (Blue_0001.tif)



sigma 0.5 k 700 min 100

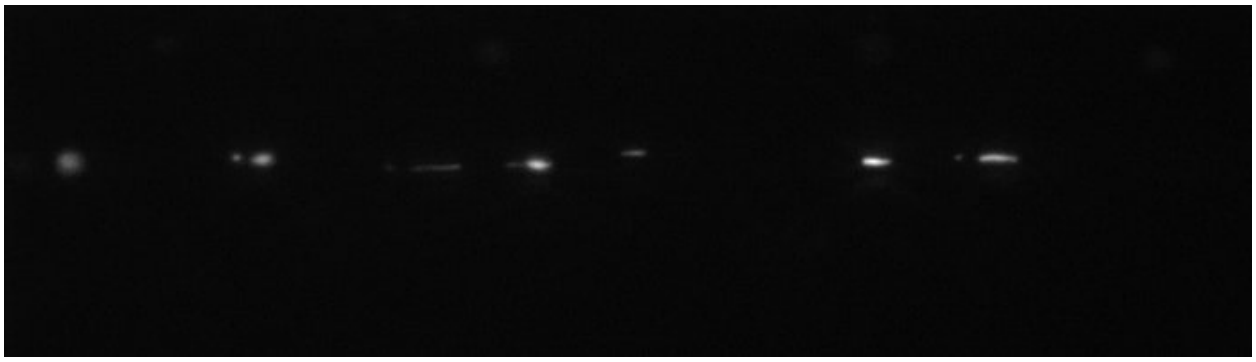
sigma 1 k 500 min 150

sigma 1 k 900 min 150



sigma 1.5 k 600 min 200

Selected Image 1 from the Mito_GFP_a01



original image

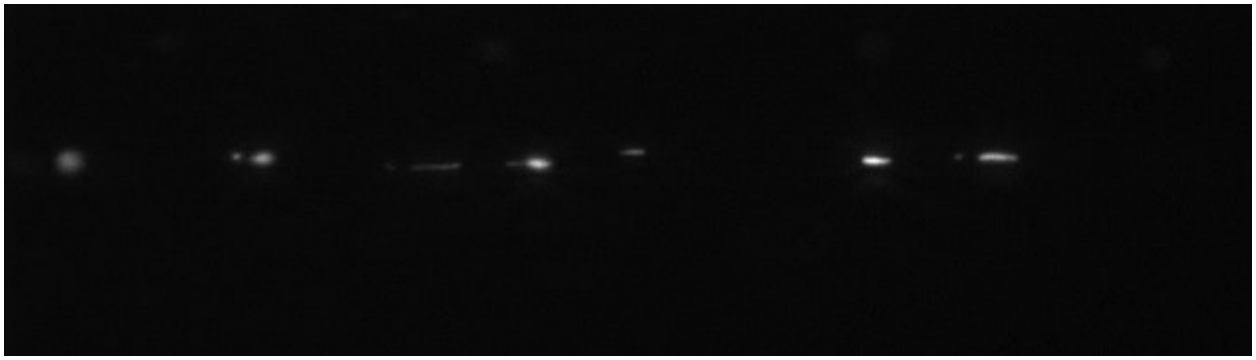
sigma 0 k 0 min 0

sigma 0 k 0 min 100

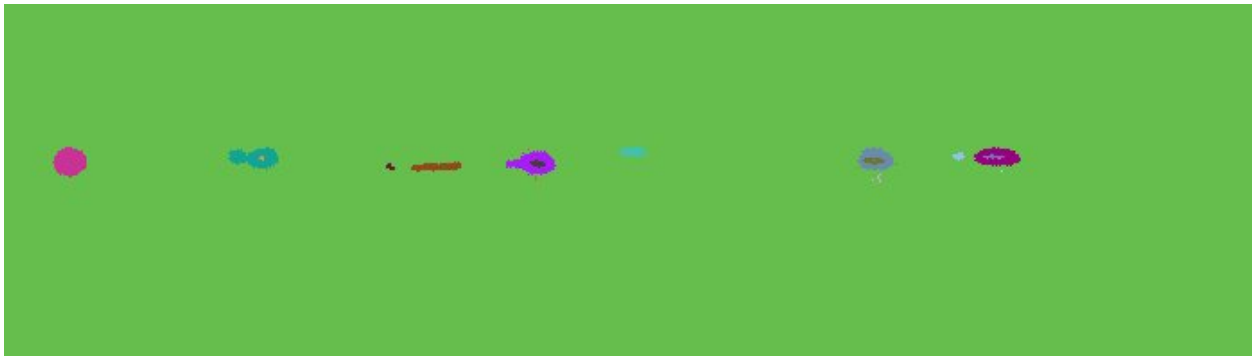
sigma 0 k 0 min 150

sigma 0 k 0 min 200

Image 2 from the Mito_GFP_a01

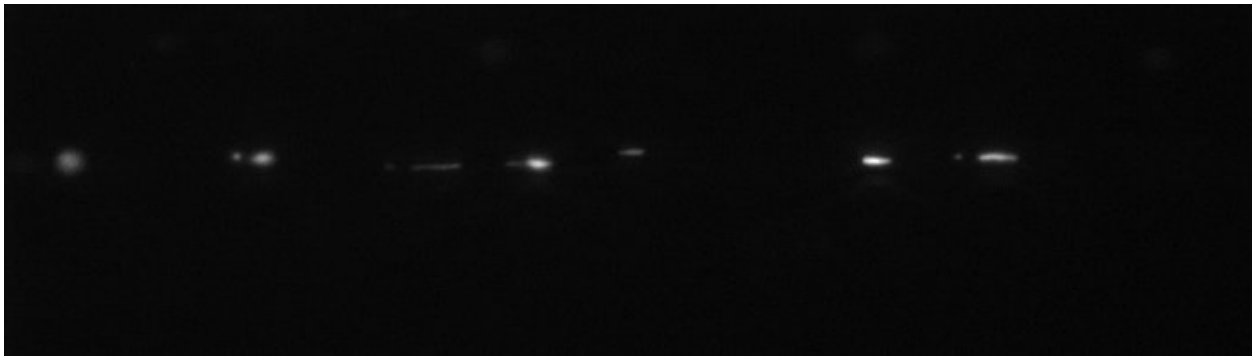


original image



sigma 0 k 0 min 0

Image 3 from the Mito_GFP_a01



original image

sigma 0 k 0 min 0

Code submitted for parameter tuning in python:

python parameter_tunesegment.py <input_file>

input_file should be in the ppm format strictly according to the specification of the algorithm. Output files will be generated in the following format:

output_sigma_<floatnumber>_K<integernumber>_min<integernumber>.tif

References:

1. P. F. Felzenszwalb & D. P. Huttenlocher, Efficient graph-based image segmentation, International Journal of Computer Vision, vol. 59, pp. 167-181, 2004.
2. Normalized Cuts and Image Segmentation, Jianbo Shi and Jitendra Malik, IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI) 2000
3. Normalized Cut Segmentation Code, Timothee Cour, Stella Yu, Jianbo Shi. Copyright 2004 University of Pennsylvania, Computer and Information Science Department.
4. Nobuyuki Otsu (1979). "A threshold selection method from gray-level histograms". IEEE Trans. Sys., Man., Cyber.
5. C. Steger, An unbiased detector of curvilinear structures, IEEE Trans. Pattern Analysis and Machine Intelligence, vol. 20, pp. 113-125, 1998.
6. R. Malladi, J. A. Sethian and B. C. Vermuri "Shape Modeling with Front Propagation: A Level Set Approach". IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol 17, No. 2, pp 158-174, February 1995