Image Segmentation Algorithm Comparison

SOT, SSDLS, graph cut based, active contour based

Team 2

B. Part I: Image segmentation using MAT-ITK

B.1 MAT-ITK and image data

All the packages and data was downloaded. All packages were verified as functioning correctly.

Algorithm 1 overview: SOT (Otsu Threshold Segmentation / Otsu's method)

SOT: B.2.1 Segmentation of static images

Otsu's Method from MAT_ITK was run on the 2 sample images provided (shown below), with number of histograms equal to largest pixel intensity in image.
The only parameter accessible in Matlab is the number of histograms used. In Otsu's method this corresponds to the number of "groups" of pixels that the algorithm attempts to use to figure out what a good cutoff for what are background pixels.

When this parameter is set to 2, the lowest value possible, the resulting image segmentation is very poor. The very bright objects in the images are segmented out. These very bright objects could be part of actual objects to be segmented or they could be noise. When the number of histograms is 2, the image segmentation is very poor.

As the number of histograms is increased, the resulting image segmentation improves. However, this relationship only holds to a point. When the number of histograms is increased past a certain value, the image segmentation does not change, though by visual inspection, appears very good.

Thus to in general guarantee good performance, the number of histograms should be set to the highest pixel intensity in an image, as a rule of thumb, even if lower values could also give good segmentations. Number of histograms also does not appear to appreciably change the amount of time this algorithm is run.

This algorithm does however make errors, when visually inspected, when objects in an image have regions that are close in intensity to the background. However if signal to noise is high enough, this algorithm should give good results.

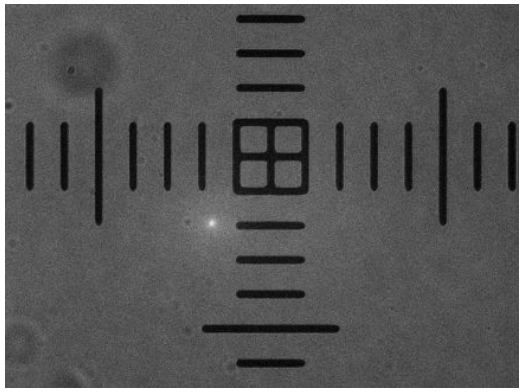


Figure 1a: Original 60X image

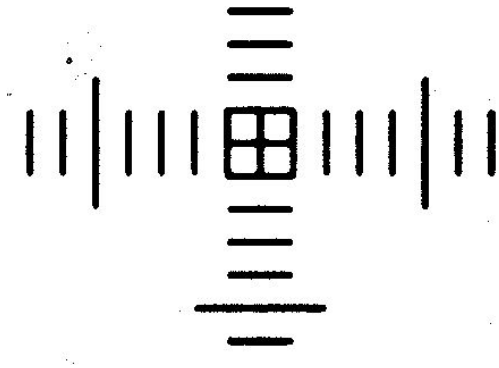


Figure 1b: Otsu's method run on 60X image with maximum intensity as number of histograms

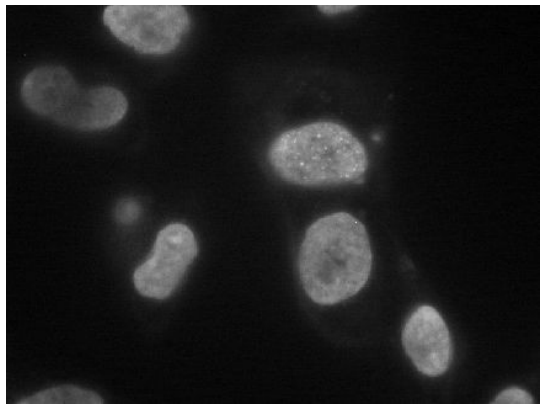


Figure 2a: Original "blue" image

Figure 1b: Otsu's method run on blue image with maximum intensity as number of histograms

SOT: B.2.2 Segmentation of image sequence

Otsu's Method from MAT_ITK was ran on the images sequence provided, with each image using its highest pixel intensity value as the number of histograms. Then the segmented images were combined into an AVI file in imageJ.

Below are the raw first image and the segmented first image for reference.

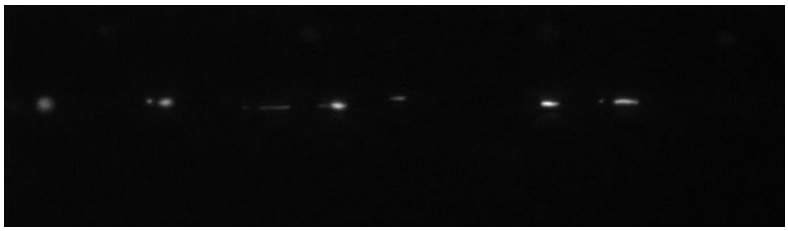


Figure 3a: First image in image sequence before segmentation

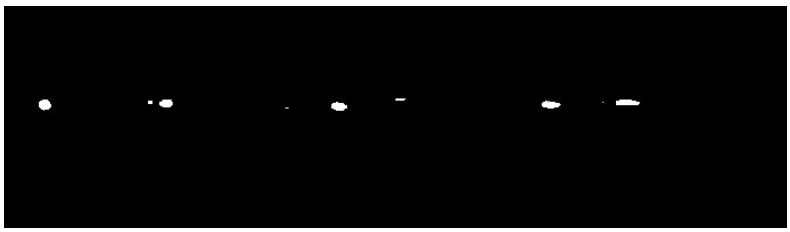


Figure 3b: First image in image sequence after segmentation

SOT: B.2.3 Segmentation of static images

The Otsu's method treats an image as a pixel matrix where each pixel belongs to one of L different intensity levels. Each intensity level, l, has a set of pixels in that image that are that intensity level, nl. Also all these pixels sets should add up to the total number of pixels in the image, $N = \sum_1^L n_l$. So the probability of a pixel belonging to a certain pixel set is $p_l = \frac{n_l}{N}$.

When the number of histograms is 2, Otsu's method works as follows. There are 2 classes C0 and C1, which have intensities below and above a threshold, k, respectively. Each class has its own ω and μ, as well as a total μ (μt, which is the mean intensity of the image), shown below. The parameters, ω and μ, correspond to the likelihood of a pixel belonging to that class and the average pixel intensity of that class.

$$\omega_0 = \sum_{l=1}^{k} p_l$$

$$\omega_1 = \sum_{l=k+1}^{L} p_l$$

$$\mu_0 = \sum_{l=1}^{k} l * p_l$$

$$\mu_1 = \sum_{l=k+1}^{L} l * p_l$$

$$\mu_T = \sum_{l=1}^{L} l * p_l$$

And each class has its own σ or class standard deviation of pixel intensities, as shown below:

$$\sigma_0^2 = \sum_{l=1}^{k} (l - \mu_0)^2 * p_l / \omega_0$$

$$\sigma_1^2 = \sum_{l=k+1}^{L} (l - \mu_1)^2 * p_l / \omega_1$$

Based off the above equations, the following variances, within-class variance, the between-class variance, and the total variance of levels, respectively, can be defined:

$$\sigma_W^2 = \omega_0 * \sigma_0^2 + \omega_1 * \sigma_1^2$$

$$\sigma_B^2 = \omega_0(\mu_0 - \mu_T)^2 + \omega_1(\mu_1 - \mu_T)^2 = \omega_0\omega_1(\mu_1 - \mu_2)^2$$

$$\sigma_T^2 = \sum_{l=1}^{L} (l - \mu_T)^2 * p_l$$

Otsu's method uses the metric, $\eta = \sigma_B^2/\sigma_T^2$, to determine the "goodness" of the segmentation threshold k, in other words Otsu's method seeks the "best segmentation threshold" k*, where k* maximizes η and is defined by:

$$\sigma_B^2(k^*) = \max_{1 \leq k < L} \sigma_B^2(k)$$

since σ_T^2 cannot be changed for a given image only σ_B^2 can be changed. The image is segmented by applying the following transformation for each pixel, x ,

$$classification(x) = \begin{cases} 1 \; if \; intensity \; of \; x \; is \; greater \; than \; \mathrm{k}* \\ 0 \; otherwise \end{cases}$$

η* is the optimal value for the metric for a given image or $\eta^* = n(k^*)$, it can be considered the separability of the classes. In the best case, it is 1, if there is not overlap between the 2 classes. In the worst case it is 0, if the 2 classes cannot be separated, or in fact there is only one class in the image.

However in practice, when Otsu's method uses only 2 classes, it gives poor results. However Otsu's method is easily extended for more classes. For 3 classes, the optimization necessary for the best segmentation becomes:

$$\sigma_B^2(k_1^*, k_2^*) = \max_{1 \leq k_1 < k_2 < L} \sigma_B^2(k_1, k_2)$$

For an arbitrary number of classes K, the equations introduced above become:

$$\sigma_B^2(k_i^* \; for \; 0 \leq i < K) = \max_{1 \leq k_i < k_{i+1} < L \; for \; each \; i \; in \; 0 \leq i < K} \sigma_B^2(k_i \; for \; 0 \leq i < K)$$

Thus the segmentation can be done by modifying the classification function above to:

$$classification(x) = \begin{cases} 1 \; if \; intensity \; of \; x \; is \; greater \; than \; k_i* \\ 0 \; otherwise \end{cases}$$

Where k_i , is the threshold that provides the most separability between background pixels and object pixels.
So Otsu's method with multi-thresholding is an algorithm with the following steps:

1. Compute the normalized histogram of the input image. Denote the components of the histogram by p_i, $i = 0, 1, 2, \ldots, L - 1$.
2. Compute the cumulative sums, $P_1(k)$, for $k = 0, 1, 2, \ldots, L - 1$, using Eq. (10.3-4).
3. Compute the cumulative means, $m(k)$, for $k = 0, 1, 2, \ldots, L - 1$, using Eq. (10.3-8).
4. Compute the global intensity mean, m_G, using (10.3-9).
5. Compute the between-class variance, $\sigma_B^2(k)$, for $k = 0, 1, 2, \ldots, L - 1$, using Eq. (10.3-17).
6. Obtain the Otsu threshold, k^*, as the value of k for which $\sigma_B^2(k)$ is maximum. If the maximum is not unique, obtain k^* by averaging the values of k corresponding to the various maxima detected.
7. Obtain the separability measure, η^*, by evaluating Eq. (10.3-16) at $k = k^*$.

Where m_G is identical to , $P_1(i)$ corresponds to ω_i ,m(i) corresponds to μ_i . The above algorithm was obtained from the textbook by Gonzalez.

Algorithm 2 overview: SSDLS (Shape Detection Level Set Filter)

SSDLS: B.2.1 Segmentation of static images

The function SSDLS has four parameters: propagation scaling, curvature scaling, maximum RMSE, number of iterations. After adjusting all four parameters iteratively, the resulting images shows little difference. As a result, I used the original settings as the parameters as seen under the "Settings" heading comment in both the b21_SSDLS.m and b22_SSLDS*.m codes.

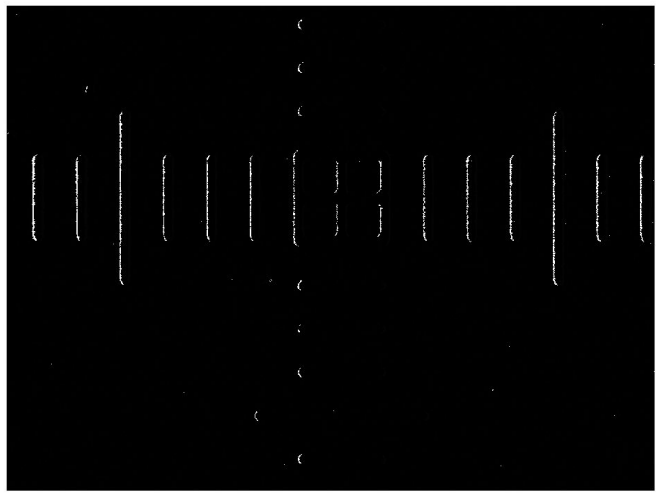


Figure 1c: SSDLS on 60X image with original parameters (1,1,1,1)

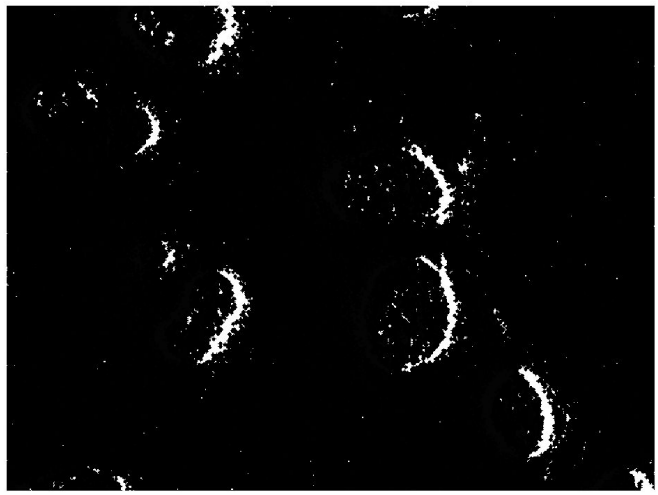


Figure 2c: SSDLS on blue0001 image with original parameters

SSDLS: B.2.2 Segmentation of image series
SSDLS from MAT_ITK was ran on the image sequence which each image using the original parameters because there were also minimal differences between the iterations of settings as show in the b21_SSDLS.m and b22_SSDLS*.m codes. After SSDLS segmentation of the image sequence, the results are combined into an AVI file using ImageJ.

Below is the segmentation of the first image of the image sequence for reference. The AVI can be found in .zip file as b22_SSDLS_video.avi.

Figure 3c: First image in the image sequence after SSDLS segmentation

SSDLS: B.2.3 Theoretical background

SSDLS from MAT_ITK is a segmentation algorithm that uses the level set method (LSM). LSM tracks feature shapes by passing a hyperplane at the image intensity axis (where the image has a x-axis and a y-axis and the image intensity is posed as z-axis). At each level on the image intensity axis, the hyperplane propagates down the axis. The function SSDLS initializes by propagating around the axis level until it registers to a shape boundary. It, then, propagates though the axis as specified by the propagation scale parameter.

There are two inputs that this SSDLS function takes. The first one is the original image and the second input is the feature image. The feature image in this case are edge features. Like the Canny edge detector and the anisotropic filter, the method used to get these line features is by calculating image gradient (or the change in intensity of the original image).

In this MAT_ITK function, the feature image is mathematically taken with the following speed term as it propagates along the axis:

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

where I is image intensity and $(\nabla * G)$ is the derivative of Gaussian operator.

To propagate through the intensity-axis levels, the level set function is:

$$\psi_{k+1} = \psi_k + S(\psi)(1 - |\nabla\psi|)$$

where ψ is the axis level and S is a snake function that wraps around the shape/line feature.

C. Part II: Image segmentation using graph-cut and active contour

C.1.1 Graph cut based image segmentation

C.1.1.1 Normalized Cut

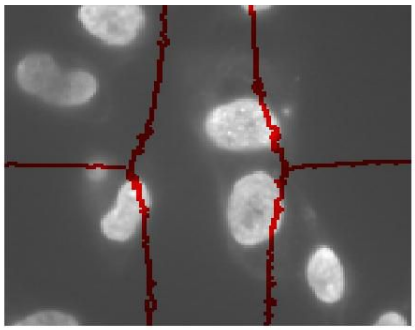
In the implementation of NCut algorithm, the construction of the affinity matrix W is one of the determinants for performance of image segmentation. Since the W is defined as

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

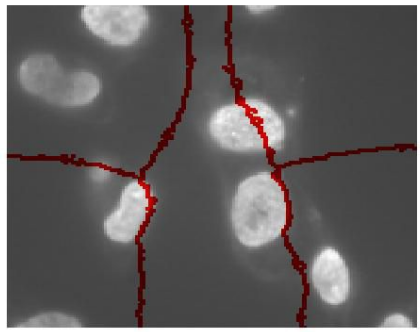
We could find out that σ_F and σ_X would act as tuning parameters that determines the degree of feature ||F(i) - F(j)|| and spatial information ||X(i) - X(j)|| that taken into account for computing the edge weights. With the decrease of σ_F value, the weight value w_{ij} would decease consequently, which contribute to a more local segmentation (more detailed), and it goes similar for σ_X.

The author set the default values for the parameters as $\sigma_F = 0.1$, $\sigma_X = 0.3$, and $R = 10$. Another parameter that would affect the segmentation performance is the number of segmentation groups k. The author set the default value for k as 10. According to different features of images, we have to tune these parameters to optimize the segmentation performance.

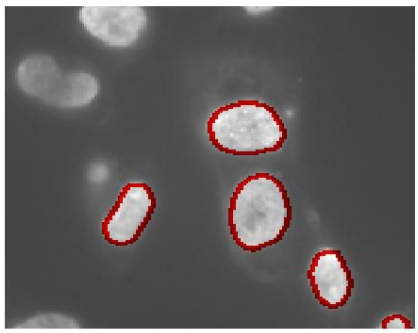
To illustrate the value tuned for parameter σ_F, we test it for image Blue0001.tif with k=5 and σ_F value range as 1.0, 0.5, 0.1, 0.05, 0.01 and 0.005. For other parameters, we use the default values. The figures are as follows.



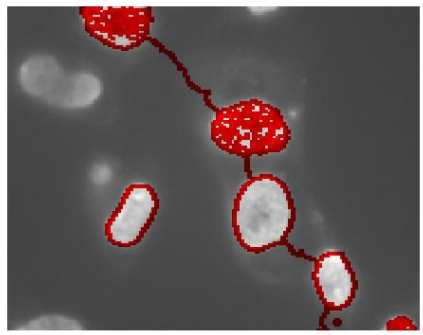
$\sigma_F = 1.0$



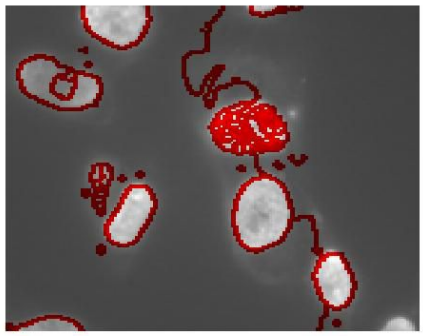
$\sigma_F = 0.5$



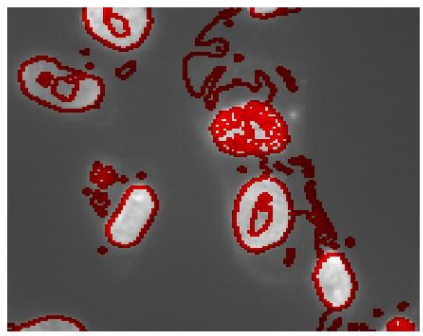
$\sigma_F = 0.1$



$\sigma_F = 0.05$



$\sigma_F = 0.01$



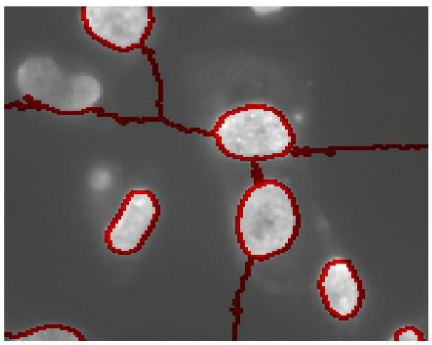
$\sigma_F = 0.005$

In the implementation of NCut algorithm, k is denoted as the field "nbSegments", σ_F is denoted as the field "dataW.edgeVariance", σ_X is denoted as the field "dataW.sample_rate", and R is denoted as the field "dataW.sampleRadius".

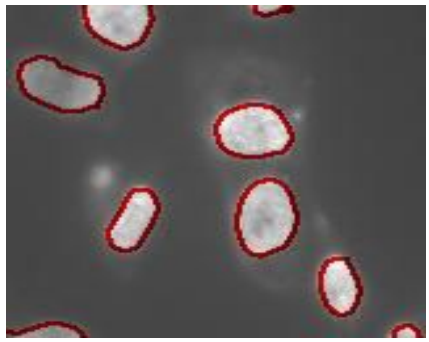
To obtain the fine-tuned parameters for particular images, we first set a value range parameter, then we implemented a for-loop program to go through all the values of

these four parameters and store the segmentation results in a folder. Finally we find out the one with best segmentation among these images and obtain the corresponding tuned parameters.

For image Blue0001.tif, we tune the parameters as k = 8, $\sigma_F = 0.045$ and $\sigma_X = 6$, and we use default value R = 10. The comparison of results before and after parameter-tuning is as follows. We could observe that after parameter-tuning, the performance of image segmentation is much better, since the segmentation result shows all the significant components.

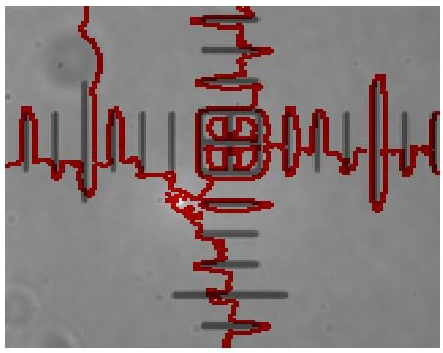


Before parameter-tuning



After parameter-tuning

For image 60x_02.tif, we tune the parameters as k = 24, $\sigma_F = 0.02$, $\sigma_X = 8$, and R = 15. The comparison of results before and after parameter-tuning is as follows. We could observe that after parameter-tuning, the image segmentation has a better performance. Since the Normalized Cut algorithm is more focused on global segmentation but not sensitive and less reasonable for local variation in the image, the segmentation result would ignore certain details.



Before parameter-tuning

After parameter-tuning

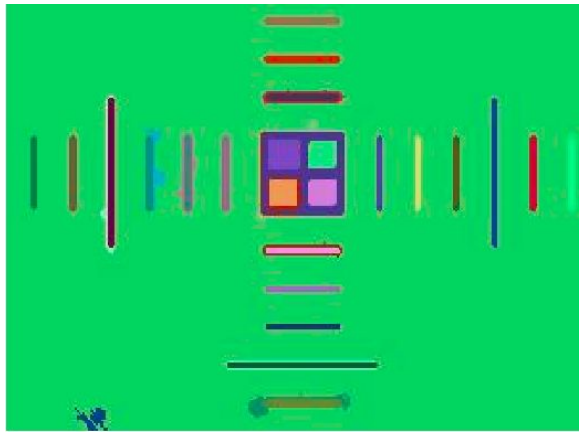
C.1.1.2 Efficient graph-based segmentation

We download the code for this algorithm, we found that we have to convert the size and format of our input images since the code can not deal with too large image input and can only process the ppm format images.

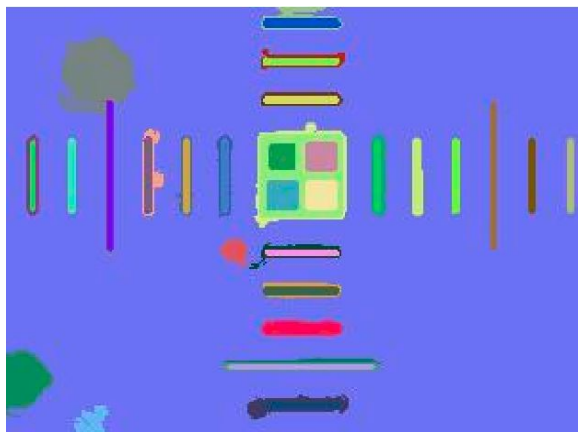
There are three main parameters for this program, which are:
sigma: Used to smooth the input image before segmenting it.
k: Value for the threshold function.
min: Minimum component size enforced by post-processing.

We tune the parameters by iterating through different combination of parameters, specifically [0,2] for sigma with step size of 0.5, [0,500] for k with step size of 100,and [0,200] for min with step size of 50. We test it over the two images and the image sequence. To see all of the test image sequence, it's in the zip file we hand in. Below we show some image results just to help analysis the parameters.

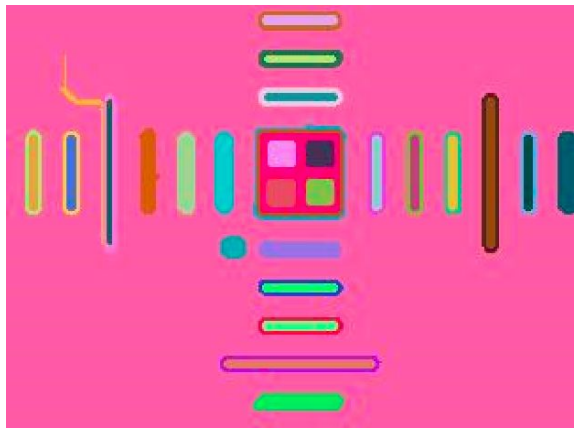
For 60x_02.ppm(some relatively good results as we tune the parameters):



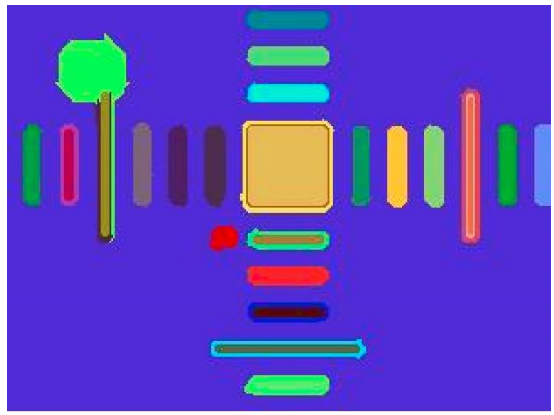
parameters:0.0_500_100



parameters:1.0_500_100

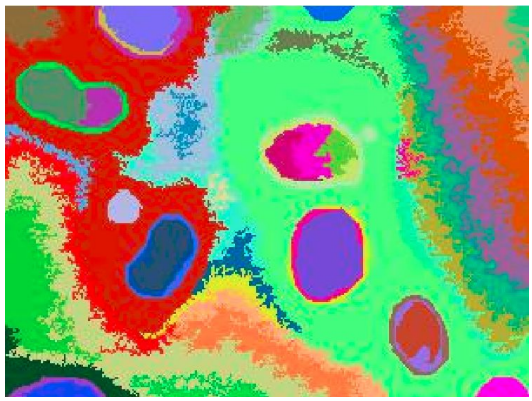


parameters:1.5_500_100



parameters:2_500_100

For Blue0001.ppm:



parameters:0_500_100

parameters:0.5_500_100

parameters:1_500_100

parameters:1.5_500_100

Some note when we were tuning the parameters:
1st parameter (sigma: used to smooth the input image before segmenting it)
- as param increase it picks up more noise (but not much difference);

- keep at 0.0 will remove noise on upper left for 60x_02;
- when >2, line becomes thicker and affect 3rd parameter (that the 3rd parameter can be larger while still getting good results)

2nd parameter (k: Value for the threshold function)
- can't be too small
- bigger is better

3rd parameter (min: Minimum component size enforced by post-processing)
- = 100 is good (=100 on most good images);
- if >200, line features begin to disappear

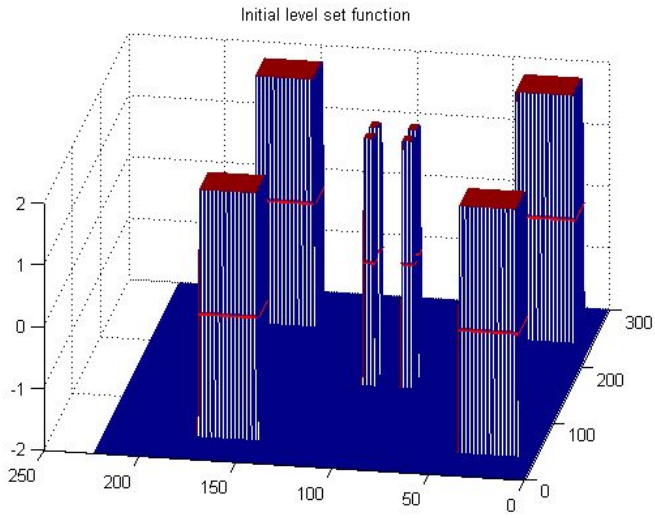
C.1.2 Active contour based image segmentation

C.1.2.1 Active Contour based on level set method-*DRLSE*

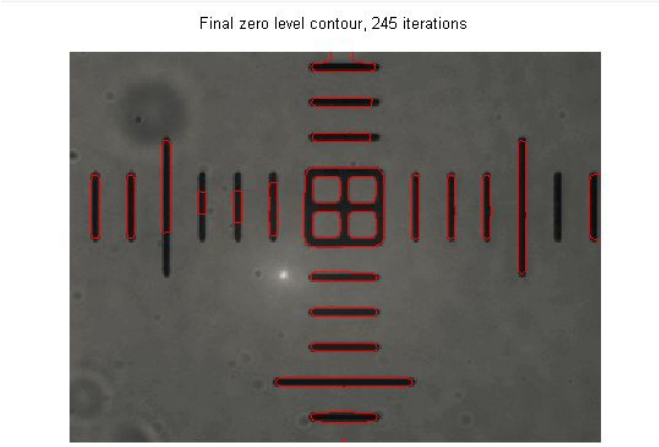
For optimizing the segmentation result, we picked some parameters for tuning, which includes the initial region/window for the algorithm to extend or shrink, *alfa* value which decide whether the algorithm performs extension or shrinking, also the number of iterations we choose for specific image inputs. We optimized the results mainly based on these three parameters.

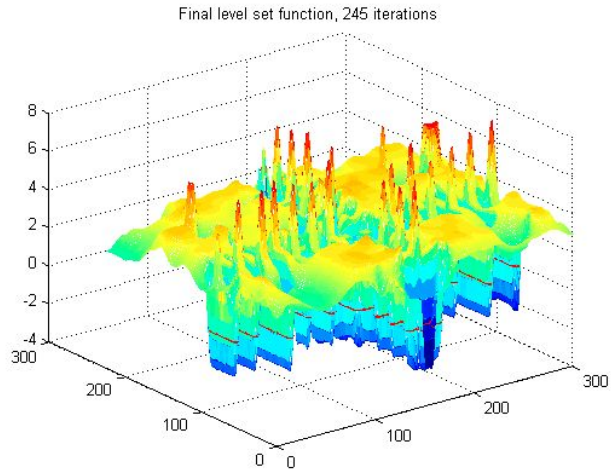
In the zip file we upload, we have the image for contour as it changing for each iteration which named as evolution for those contour images and the final segmentation results, for the image sequence, we just keep the final contour state for each frame to generate the avi video. Below we show some results for the two static image with different parameters.

For 60x_02 image, we picked eight region for initial level set function, or say windows. And we set a negative *alfa* value to make the regions extend.

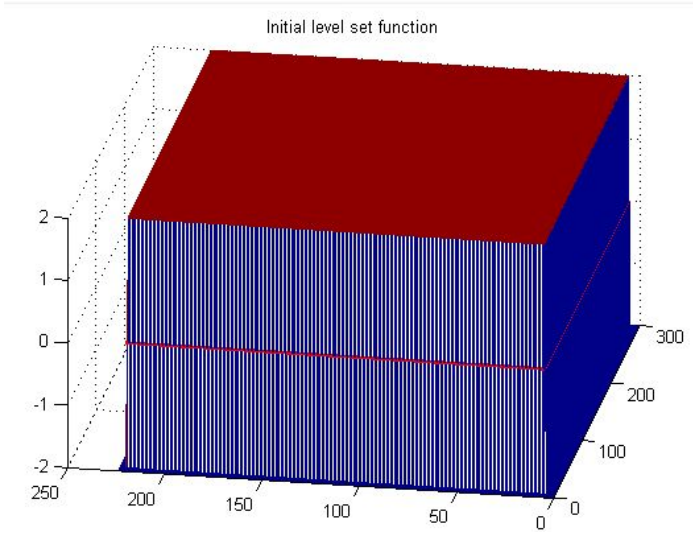


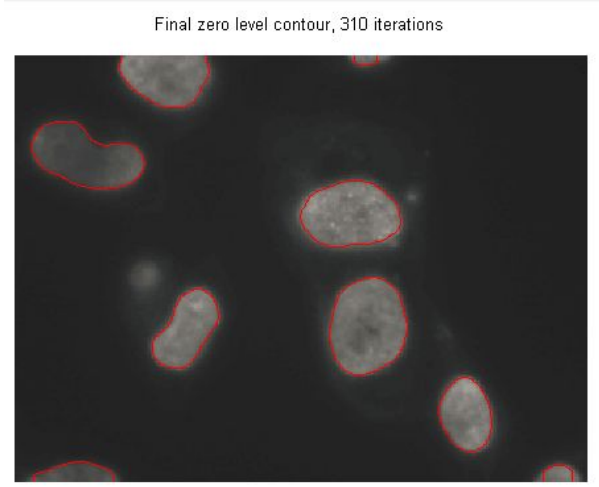
Here is the final result for segmentation as well as level set functions.



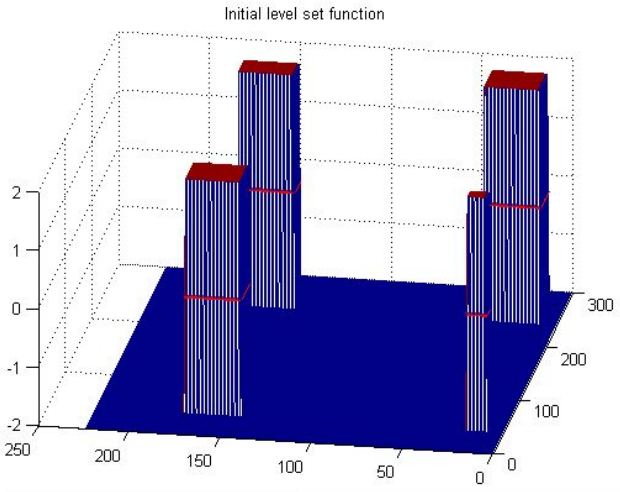


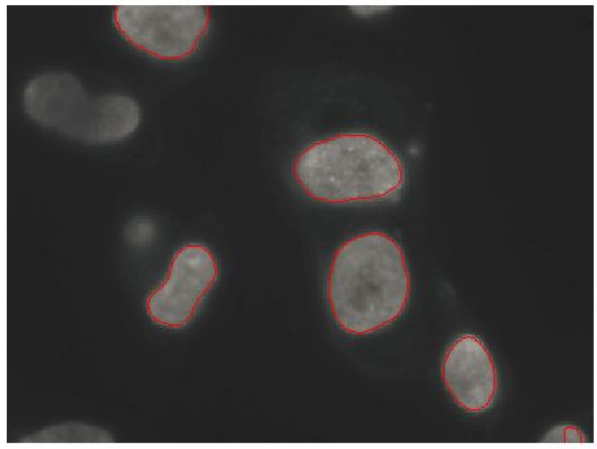
If we only pick one big region as initial function and set a positive *alfa* value, it shrinks and give us results as shown below. Since there is only one window for shrinking, so it take more iterations.





While for positive alfa value, it capture the up-left region but it got missed as the contour keep moving.





C.1.2.1 Active Contour based on Gradient Vector Flow

First we use the GVF snake program to test our input images. In this program, we have to convert the input image into pgm or raw first as required.

To obtain the optimal snake curve, we have to minimize the energy function known as

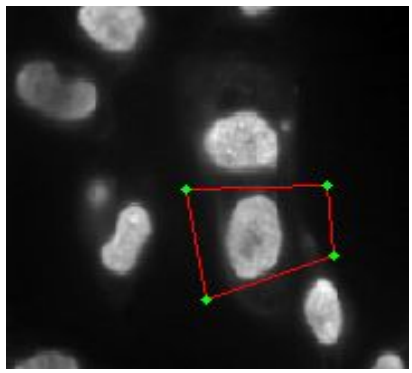
$$E = \tfrac{1}{2} \int [\alpha |x'(s)|^2 + \beta |x''(s)|^2] + \kappa E_{ext}(x(s))ds$$

where we could know that α (tension of the snake), β (rigidity of the snake) are the tuning parameters. Meanwhile, the GVF regularization parameter μ should also be the tuning parameter since it affects the calculation of GVF forces considerately. Besides, there are additional factors that would affect the contour performance, such as the number of iteration, the sigma σ (default as 0) for edge map, the initial contour of snake, the step size of one iteration γ (default as 1) and the external force weight parameter κ (default as 0.6).

The α and β parameters are associated with the internal force variables in the original snake model. Decreasing α or β will result in corners and self-intersections in the deforming contour, while increasing them too much will shrink the contour to a line or point. The κ parameter is a weighting variable applied to the external force and determines the strength of the effect of the image features that make up the external force. The parameter μ is a regularization parameter that should be set based on the amount of noise in the image.

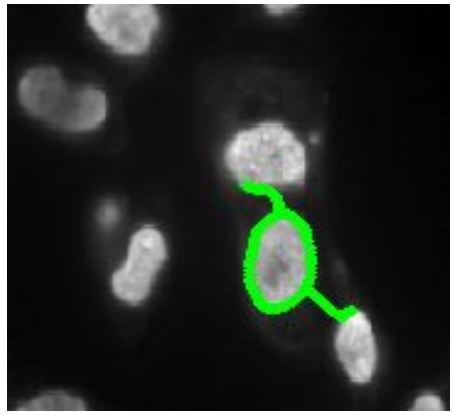
In our test, we select five distinct parameters, namely μ, α, β, γ and κ. First we used the GVF for Windows 95/NT version and implemented a for-loop program to go through all the values of these five parameters(each parameter has a value range) and store the results in a folder. However, we did not find significant result with good performance(the images would be attached and you could check that). Therefore, we use the snake demo instead, which is a GUI that the user could manually select the start contour. We tried many combinations of tuning parameters and finally we find out the one with best result among these images and obtain the corresponding tuned parameters. In the snake demo GUI, The author set the default values for these parameters as μ=0.1, α=0.05, β=0, γ=1 and κ=0.6.

For image Blue0001.tif, we tune the parameters as μ=0.2, α=0.6, β=0.01, γ=2 and κ=0.8. The comparison of results before and after parameter-tuning is as follows.



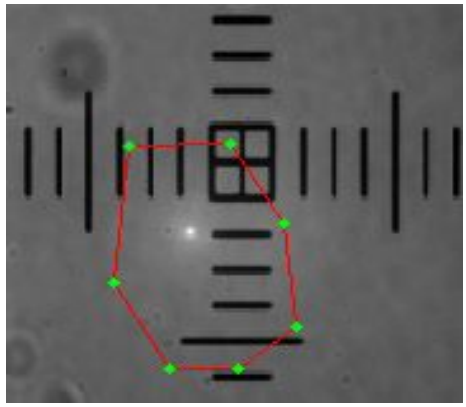
Manual Initial contour

Before parameter tuning

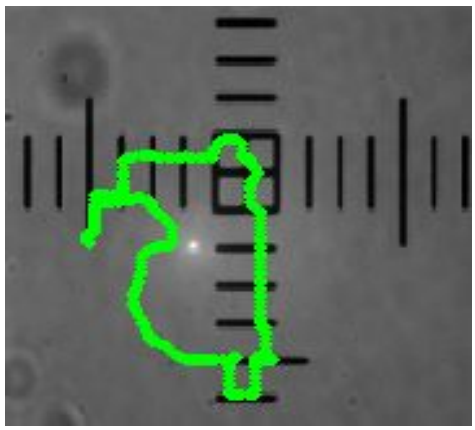


After parameter tuning

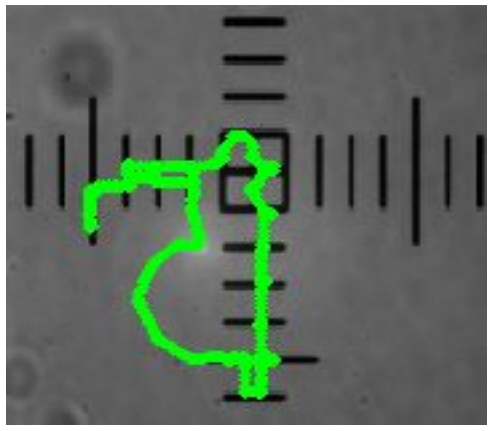
For image 60x_02.tif, we tune the parameters as μ=0.2, α=0.2, β=0, γ=3 and κ=1.5. The comparison of results before and after parameter-tuning is as follows.



Manual Initial contour



Before parameter tuning



After parameter tuning

C.1.3 Theoretical background

C.1.3.1 Normalized Cut

Normalized Cut Algorithm is a graph based segmentation algorithm, which extract the global features of an image, rather than focusing on the local features and their consistencies in the image data. The normalized cut criterion measures both the total dissimilarity between the different groups as well as the total similarity within the groups.

Taken the input image as an undirected weighted graph (denoted as G = (V, E)) and the pixels as the nodes in this graph (denoted as V, the set of nodes). Each edge between two nodes (i, j) has an associated weight w_{ij} , which represents the strength of the connection. High value of w_{ij} would indicate that these two nodes are very similar and it is very likely for them to be classified as one group. Using the notation of cut to measure a partition of graph nodes into two disjoint subsets:

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

where A and B are two disjoint subsets of the graph G. Since these two disjoint groups would not have strong connections with each other, we could determine the optimal segmentation of graph by finding the cut with smallest values, known as the minimum cut criteria.

However, the minimum cut criteria favors cutting small sets of isolated nodes in the graph. To avoid this bias of partitioning out small set of points, the notation of normalized cuts, known as NCut, should be introduced by taking both cut cost and total edge connections into consideration:

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

where *accoc(A, V)* represents the total connection from nodes in A to all nodes in the graph. With NCut criteria, the goal is to find a cut such that the connections between the partitioned groups are weak and that the nodes are evenly distributed so that the internal connections for the new groups are both evenly strong.

To minimize the value of Ncut, the following generalized eigenvalue system is introduced

$$(\mathbf{D} - \mathbf{W})y = \lambda \mathbf{D}\boldsymbol{y}$$

where D is N*N diagonal matrix with degree d on its diagonal and W is N*N symmetrical matrix with W(i, j) = wij which indicates the affinity of nodes. To solve this eigenvalue system, we find the eigenvector with the second smallest eigenvalue, which is used in NCut problem, and use it to segment the image.

The construction of the affinity matrix W is based on the magnitude value and the spatial locations of pixels, which indicates the likelihood that these two pixels are in the same group. We can define the edge weight for node i and node j as follows

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

where X(i) represents the spatial location of node i and F(i) represents the feature vector (intensity here) of node i.

Overall, the process for NCut algorithm would include the following steps. First, we take the input image I and construct the weighted graph G = (V, E) through constructing the affinity matrix W. Second, we would solve the eigenvalue system with NCut criteria to find the eigenvector with the second smallest eigenvalue. Finally, we check the stability of the cut and determine whether the current partition should be subdivided. Recursively partition the segmented parts if necessary and obtain the final results.

C.1.3.2 Efficient graph-based segment

For the efficient graph-based segment algorithm, the principle is based on graph theory. The theme underlying this Graph-based approach is the formation of a weighted graph, where each vertex corresponds to n image pixel or a region, and the weight of each edge connecting two pixels or two regions represents the likelihood that they belong to the same segment. The weights are usually related to the color and texture features, as well as the spatial characteristic of the corresponding pixels or regions. A graph is partitioned into multiple components that minimize some cost function of the vertices in the components and/or the boundaries between those components.

The input is a graph $G = (V, E)$, with n vertices and m edges. The output is a segmentation of V into components $S = (C_1, \ldots, C_r)$.

0. Sort E into $\pi = (o_1, \ldots, o_m)$, by non-decreasing edge weight.
1. Start with a segmentation S^0, where each vertex v_i is in its own component.
2. Repeat step 3 for $q = 1, \ldots, m$.
3. Construct S^q given S^{q-1} as follows. Let v_i and v_j denote the vertices connected by the q-th edge in the ordering, i.e., $o_q = (v_i, v_j)$. If v_i and v_j are in disjoint components of S^{q-1} and $w(o_q)$ is small compared to the internal difference of both those components, then merge the two components otherwise do nothing. More formally, let C_i^{q-1} be the component of S^{q-1} containing v_i and C_j^{q-1} the component containing v_j. 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 is obtained from S^{q-1} by merging C_i^{q-1} and C_j^{q-1}. Otherwise $S^q = S^{q-1}$.
4. Return $S = S^m$.

C.1.3.3 Active Contour based on level set method-*DRLSE*

The Active contour based image segmentation algorithm-*DRLSE*, is a level set method for capturing dynamic interfaces and shapes. The basic idea of the level set method is to

represent a contour as the zero level set of a higher dimensional function, called a level set function, and formulate the motion of the contour as the evolution of the level set function. Active contour model are formulated in terms of a dynamic parametric contour $C(s,t): [0,1]\times[0,\infty)\rightarrow R^2$, with a spatial parameter s in [0,1], which parameterizes the points in the contour, and a temporal variable t.

C.1.3.4 Active Contour based on Gradient Vector Flow

Snake model is widely applied in image processing and machine version, with computer-generated curves that move within images to find object boundaries. However, it has some limitations, such as the contour initialization issue and inability to converge to boundary concavities. To overcome these issues, a new type of external force field, known as gradient vector flow (GVF), is introduced, which is derived from the image by minimizing an energy function in a variational framework. The minimization is achieved by solving a pair of decoupled linear partial differential equations which diffuses the gradient vectors of a gray-level or binary edge map computed from the image.

The traditional snake curve, known as x(s) = [x(s), y(s)] where s $\in$[0,1], moves through the spatial domain of an image to minimize the energy function

$$E = \int_0^1 \frac{1}{2}(\alpha|x'(s)|^2 + \beta|x''(s)|^2) + E_{ext}(x(s))ds$$

where α and β represent the weighting parameters that control the snake's tension and rigidity respectively, and x'(s) and x''(s) are the first and second derivatives of x(s) with respect to s.

For the input image I(x,y), the external energy function E_{ext} could be computed as

$$E_{ext}(x,y) = -|\nabla(G_\sigma(x,y) * I(x,y))|$$

where $G_\sigma(x, y)$ is a two-dimensional Gaussian function with standard deviation σ. Though larger σ would lead to blur of boundaries, it would increase the capture range of the active contour. Thus, usually a larger σ is required. To obtain the snake curve that minimizes E, the following equation should be satisfied

$\alpha x''(s) - \beta x'''(s) - \nabla E_{ext} = 0$,which is equivalent to $F_{int} + F_{ext}^{(p)} = 0$

where $F_{ext} = v(x, y)$ is the external force field known as the gradient vector field. The GVF field is defined as the vector field v(x, y) = (u(x, y), v(x, y)) that minimizes the following energy function

$$\varepsilon = \int\int \mu(u_x^2 + u_y^2 + v_x^2 + v_y^2) + |\nabla f|^2|\mathbf{v} - \nabla f|^2 dxdy$$

The overall process for GVF snake would include following steps. First, we need to calculate the edge map of the given image with certain edge detection algorithm. Then we derive the gradient vector forces over the image domain using information of this

edge map. Finally, the GVF forces are used to force the snake towards the boundaries of the target object.

Compared to traditional snake models, GVF has many advantages. The most significant feature is that GVF is not sensitive to the initialization of contour, which means it could start far from the target object, and its ability to move into concave boundary regions. Besides, GVF does not require the prior information about the deforming direction, in other words, does not care whether dilate or shrink. Moreover, the capture range of GVF snake algorithm is comparatively large and its accuracy and robustness are relatively good.

C.1.3.5 Compare the results using the graph-cut and active contour algorithms with those of the ITK algorithms

Comparing the groups of MAT_ITK algorithms in part B with the group of graph cut based and active contour based algorithms in part C, the algorithms in part B runs much faster (around realtime to 2sec vs. 2 sec to **10sec**). For the visual performance in whether the algorithms segment the features that we want, part C's algorithms performed visually better for static images than part B's. For the image sequence, the performance is visually indifferent. However, because of there are much more parameters and combinations of parameters for the algorithms in part C, it is possible that there is some combinations that we might had missed due to the long computation time (many hours) that could result in an undeniably visually better image sequence.

We found that Otsu's Method was the fastest by far, each image took much less than a second to be segmented. All the segmentations that were examined by eye were very good, though when the intensity of part of an image is close to the background it will be missed segmented, as demonstrated in the blue0001.tif image. Otsu's method , as implemented in mat_itk, only takes one parameter for tuning, which is the number of histograms. this parameter however does not need to be tuned, since using the highest intensity in the image as the number of histograms is expected to always give a good segmentation. One possible reason for the speed of this algorithm is due to the fact that the algorithm is O(number of pixels) and no slower asymptotically.

We found that SSDLS was the next fastest algorithm, needing only 2 seconds per image processed. Its segmentation results were a little noisy though each major object was segmented out. There are 4 parameters to tune (propagation scaling, curvature scaling, maximum RMS error, and number of iterations). Even though each of the 4 parameters were iteratively tested over a wide number range, the resulting images had minimal difference. This is similar to the anisotropic filtering. One more thing to note is that while all the vertical line features in the 60x02.tif image was detected, many of the horizontal line features were not captured in the segmentation algorithm. This is also similar to the result of a directional Canny edge detector where features at parallel directions has difficulty in detecting, this is likely due to directional filtering.

We found that Efficient Graph-Cut was about as fast, needing 2 sec for each image. each image processed had to be relatively small(say 300 pixels wide) in order to get reasonably fast image segmentation, also it's because there are some places in the code that put a limitation over the input image size. But the parameter optimization is time-consuming as the parameters need to be tuned for a specific image to get a good segmentation.

The Normalized Graph Cut takes about 5 sec an image. Overall, the Normalized Cut algorithm is fast, since it takes a roughly linear runtime O(nlogn) with n nodes in the graph (say n pixels in the image). For its segmentation performance, the NCut algorithm is successful for giving a global segmentation, but not sensitive and less reasonable when picking up local variation in the image. We have to local tune the parameters for the construction of affinity matrix W to add flexibility and improve its performance.

For the Active contour based on Gradient Vector Flow, the execution speed is quite slow comparatively, due to its large range of capture and the time-consuming process for convergence to object contours. Besides, the parameters, such as the GVF parameter μ and the snake parameters α and β, have to manually tuned for particular input images.

The second Active contour based image segmentation algorithm-*DRLSE was* relatively slow, like for each frame of the image sequence which is 270kb, it may take up to 20 seconds as we tune the parameters. For example, the initial region R0(a rectangle) or more such regions is the key factor to get a good segmentation and they may also make the algorithm slow, also the number of iterations we choose for the level set evolution is another factor for the time performance of this algorithm.

Sources

[1] Otsu, Nobuyuki. "A threshold selection method from gray-level histograms." Automatica 11.285-296 (1975): 23-27.
[2] Malladi, Ravi, James A. Sethian, and Baba C. Vemuri. "Shape modeling with front propagation: A level set approach." Pattern Analysis and Machine Intelligence, IEEE Transactions on 17.2 (1995): 158-175.
[3] Shi, Jianbo, and Jitendra Malik. "Normalized cuts and image segmentation."Pattern Analysis and Machine Intelligence, IEEE Transactions on 22.8 (2000): 888-905.
[4] Felzenszwalb, Pedro F., and Daniel P. Huttenlocher. "Efficient graph-based image segmentation." International Journal of Computer Vision 59.2 (2004): 167-181.
[5] Xu, Chenyang, and Jerry L. Prince. "Snakes, shapes, and gradient vector flow." Image Processing, IEEE Transactions on 7.3 (1998): 359-369.
[6] Li, Chunming, et al. "Distance regularized level set evolution and its application to image segmentation." Image Processing, IEEE Transactions on 19.12 (2010): 3243-3254.