Homework 3

Problem 1:

Algorithm

Given 2 lists of points, fixed_point set and corresponding moving_point set. Apply the 2 lists to the Least-Squares Estimation model to estimate the $\boldsymbol{\theta}$, the parameters of similarity transformation. Using the result $\boldsymbol{\theta}$ to transform the moving image and mask back to fix space.

```
function transformed_image= transform(fixed_pts, moving_pts, moving_image)
```

Process Step:

1. Using fixed_pts and moving_pts to estimate $\boldsymbol{\theta}$ by Least Squares estimate model. (Solve for the parameters through matrix inversion)
2. Inverse moving_image point
3. Transform all points original in moving_image to fixed image space only. Uncover out boundary space set to gray.

Euclidean error of a similarity transformation: **e=X$\boldsymbol{\theta}$-f**

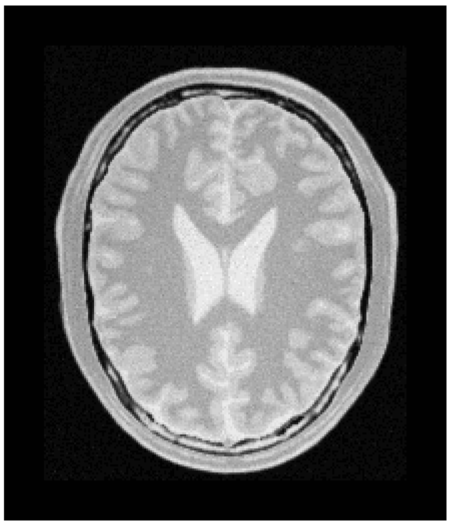
- f: Vector of fixed points, which are the coordinates of points in the fixed image.
- $\boldsymbol{\theta}$: Vector contains the parameters of similarity transformation. Degree of freedom=4. It contains the scaling scalar, rotation angle, x and y translation parameters.
- X =

$$\begin{bmatrix} sx\cos(\boldsymbol{\theta}) & -sy\sin(\boldsymbol{\theta}) & 1 & 0 \\ sx\sin(\boldsymbol{\theta}) & sy\cos(\boldsymbol{\theta}) & 0 & 1 \end{bmatrix}$$

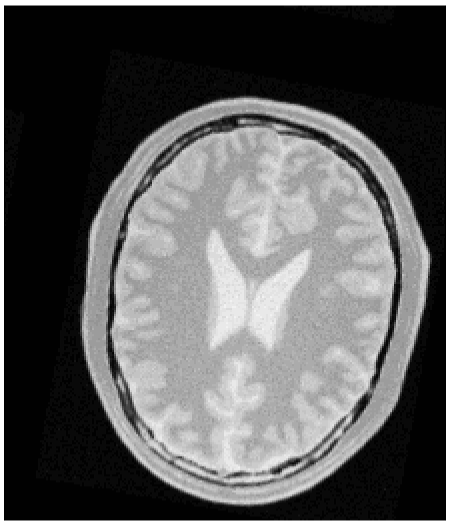
X$\boldsymbol{\theta}$ is the transformed point

Image

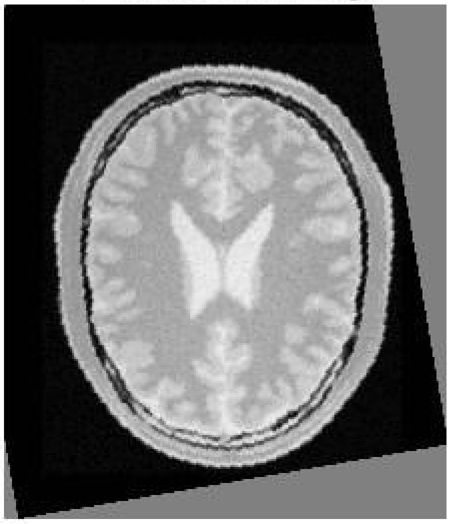
The fixed image



The moving image

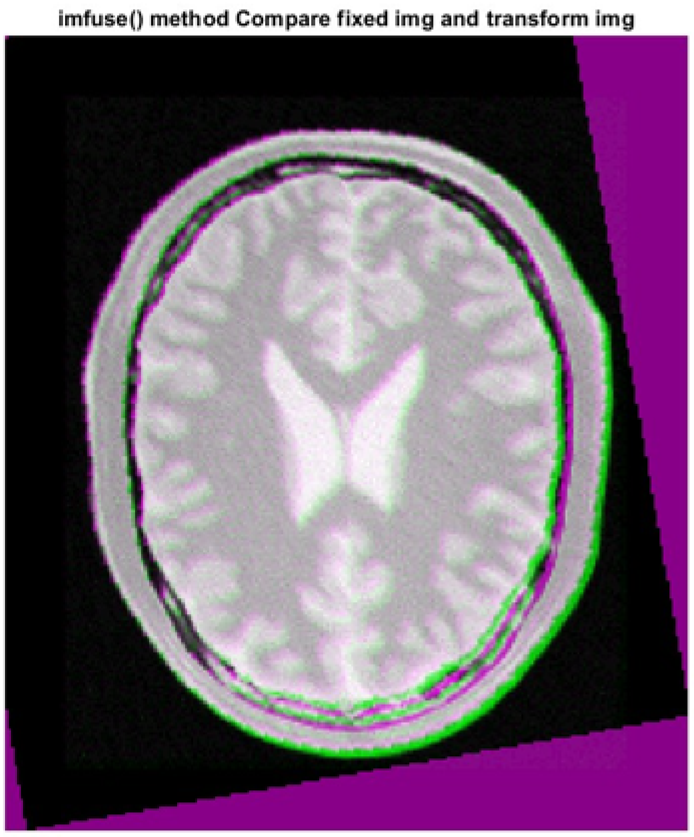


The transform image



Result

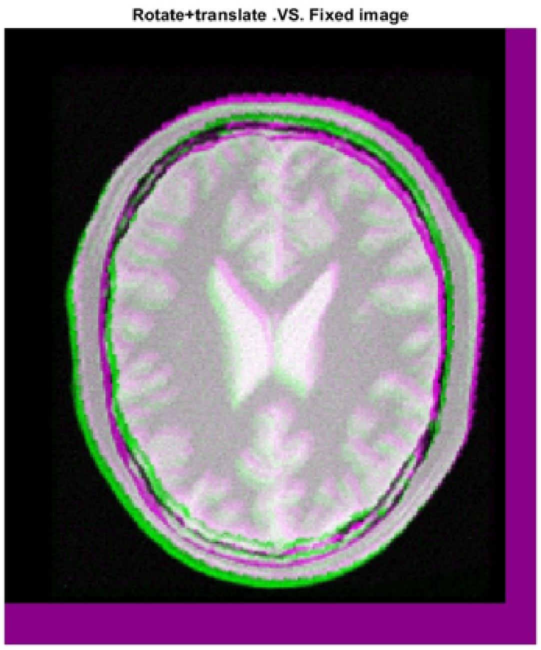
The result of the transformation is looking perfect on satisfying our experiment goal, that "The parts of the transformed image not originally in the moving image should be in gray, so we can tell apart the fixed image and the moving image that is transformed into the space of the fixed image." To further check how well is the transformation back, the image below is using the imfuse() method to overlay the fixed image and the transform image.



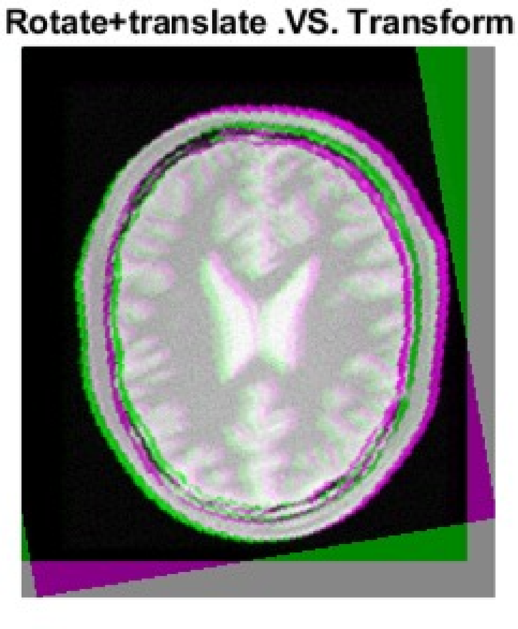
The green regions show where the intensities of two images are different. In this image, very little green regions are found, so we can tell that the transformation is nearly perfect.

However, the transformation parameter's value acquired from my program is significantly different to the actual. The given moving image, "BrainProtonDensitySliceR10X13Y17.png", whose name tells the rotation angle is 10, moves X 13 and Y 17. I believe the scalar should be 1 just based on the observation from the fixed and moving image. The $\boldsymbol{\theta}$ i had from my program is = [0.9912; -0.1718; -36.8440; 6.6674] respect to [Scalar, Radius, X, Y]. Except the scalar and radius(-0,1718 radius = -9.8434149 degree) are close to the actual value, the X and Y are very different.

Below is the comparison result of the fixed image and the moving image after performing imrotate() to rotate and imtranslate() to translate back by using the actual parameter value.

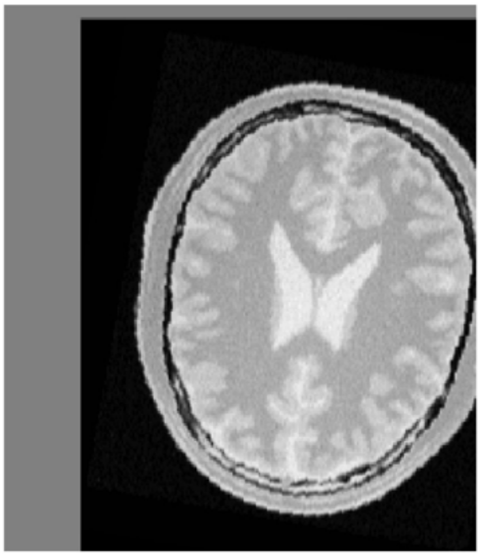


And below is the result that compares with the transform image.(Purple region is the transform image)

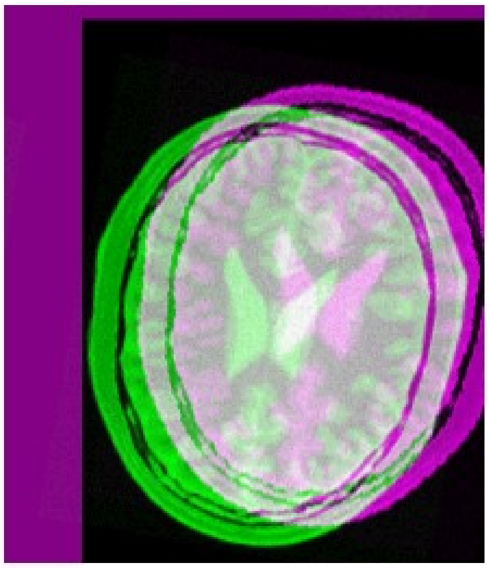


Based on these two comparisons and all the data have, the scalar and angle parameter can be concluded correct. For the X and Y, why they don't seem to be having any effect on any image transformation. I guess it's because my program accidentally covered the mistake during the mask process. I assumed a few reasons for that, such as I had limited the space area from 1 to maxRow and maxCol, or a wrong starting coordinate to mask transform image to fixed space. But unfortunately, none of the reasons seem to be persuasive and I can't even tell.

Now that I can't figure out the cause of the error, let's prove the X and Y from my program is 100% incorrect. Using the parameter from my program output to do a similarity transform to the fixed image, and below is the result.



Obviously not the same as the moving image, and let's compare it with the moving image by imfuse() method. Below is the comparison.(Green region represent moving image)



My computed X and Y are 36.844 and -6.6674. The actual X and Y should be 13 and -17. The comparison result tells the same information as the data. The purple region staying over right corresponds to 36.844 > 13 and staying a little higher corresponds to -6.6674 > -17. Clearly that my computed X and Y are incorrect.

Resources that Helped Me

- https://zhuanlan.zhihu.com/p/87582571
- https://nghiaho.com/?page_id=671
- https://www.youtube.com/watch?v=smwhve0hbtU
- https://www.youtube.com/watch?v=cYCflOM86T0
- https://www.aplitop.com/subidas/ayuda/en/MDT-Topografia/index.html#!helmertD
- https://proj.org/en/9.4/operations/transformations/helmert.html#:~:text=vector%20belong%20to.-,2D%20Helmert,-%C2%B6
- https://byjus.com/maths/least-square-method/#:~:text=Least%20Square%20Method%20Formula
- https://global.oup.com/booksites/content/0199268010/samplesec3

Problem 2:

Algorithm

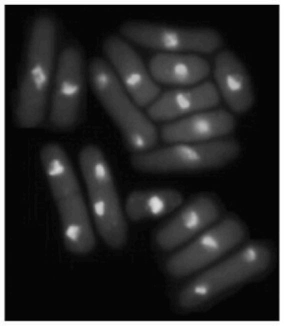
1. Read the given image.
2. Perform preprocessing operations to enhance the contrast or reduce noise, such as histogram equalization, filtering or smoothing.
3. Determine threshold value. (Computed or manually input)
4. Apply the threshold value to turn the image to binary.
5. Ensure all objects are segmented, apply further processing operation if not.
5. Count total cells and calculate total segment area to compute the average cell area.

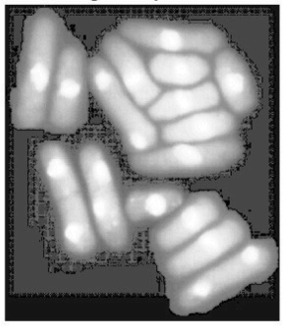
(Some step might need to repeat for better segment result)

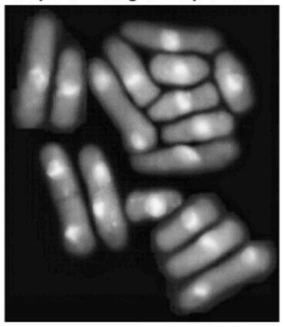
Output:

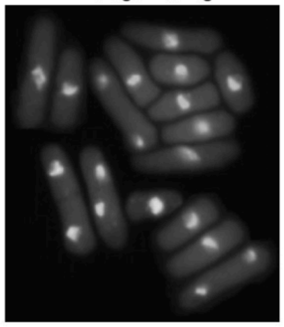
1. Segmentation image
2. Average cell area
3. List of threshold value might provide better result

Output Image


Gaussian filter

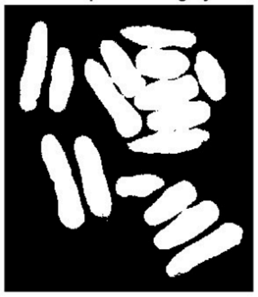

Histogram equalization


Adaptive histogram equalization

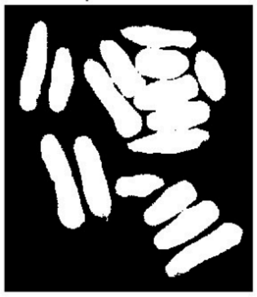

Average filtering

Gaussian filter image after binary threshold

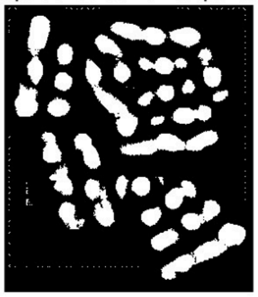
Global threshold computed from graythresh() method



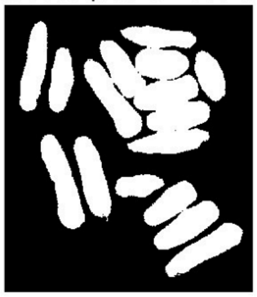
Global threhold computed from otsuthresh() method



Locally adaptive threshold from adaptthresh method

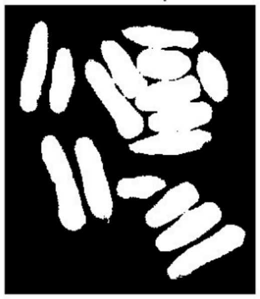


Single threshold computed from multithresh method

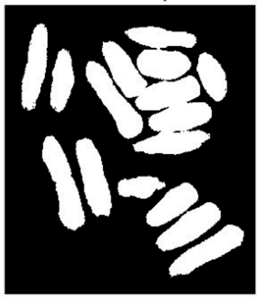


Adjust Threshold value

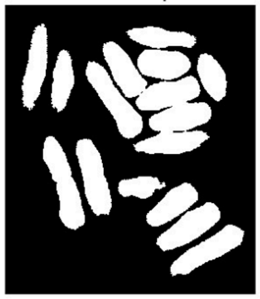
Threshold:42 Component#:11



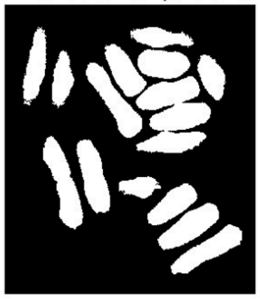
Threshold:45 Component#:12



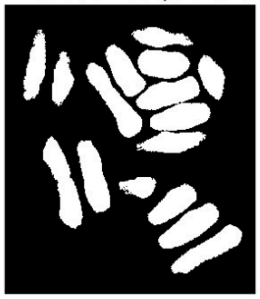
Threshold:46 Component#:13



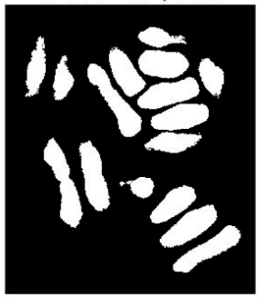
Threshold:50 Component#:15



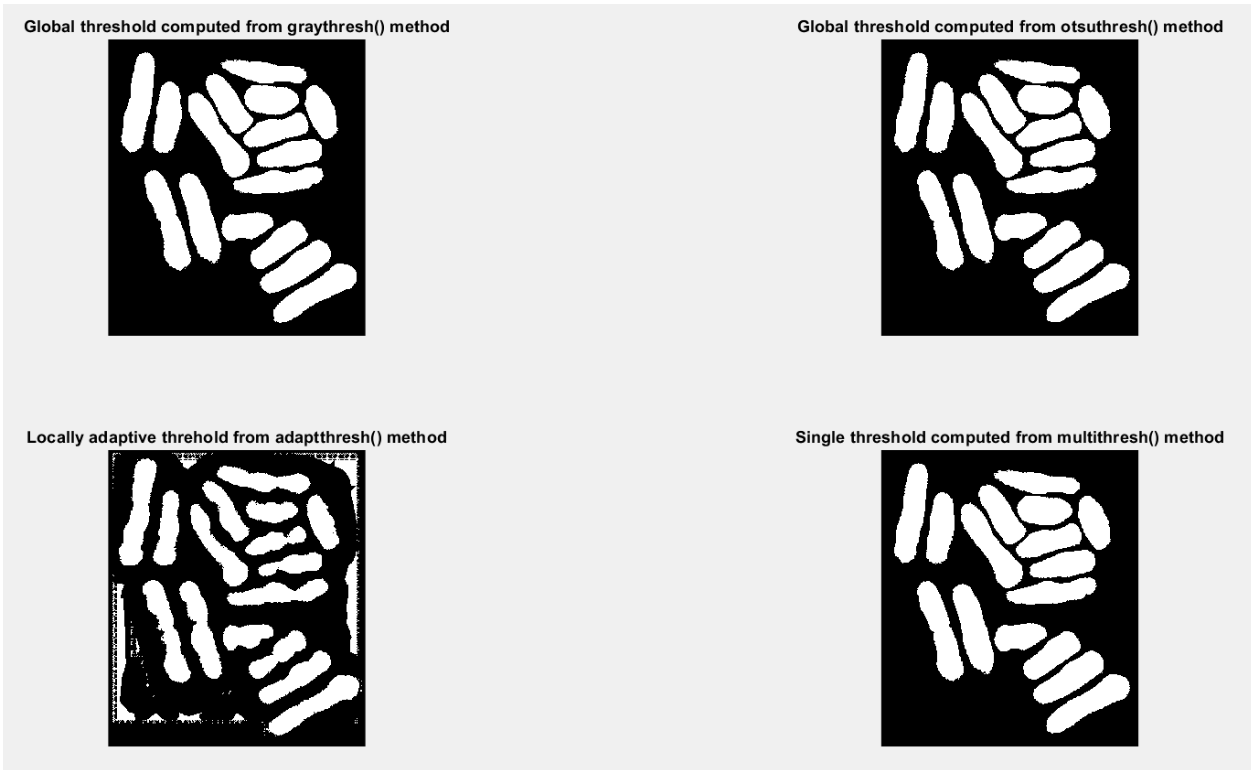
Threshold:51 Component#:16



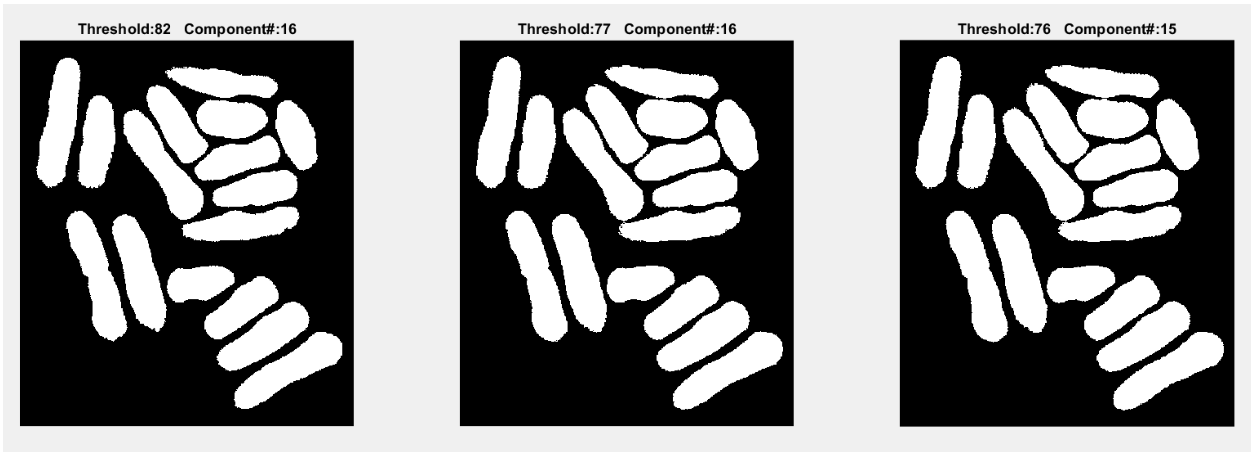
Threshold:54 Component#:17



Adaptive histogram equalization image after binary threshold

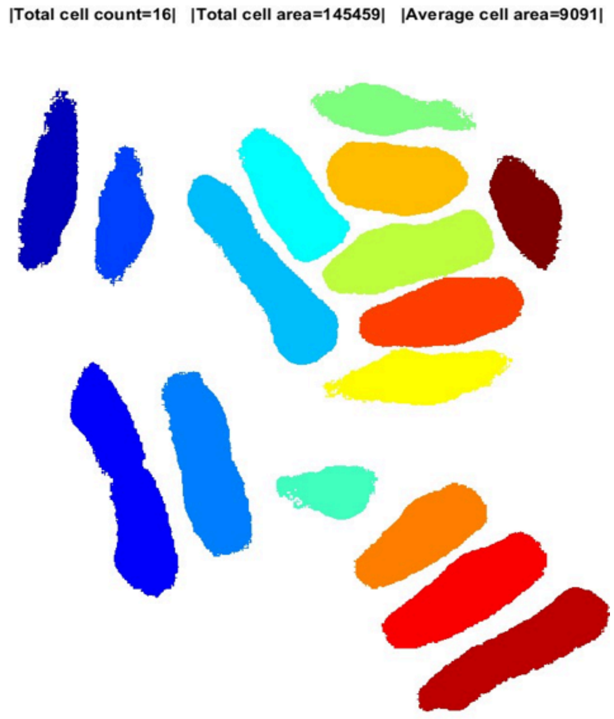

Global threshold computed from graythresh() method
Global threshold computed from otsuthresh() method
Locally adaptive threhold from adaptthresh() method
Single threshold computed from multithresh() method

Adjust Threshold value

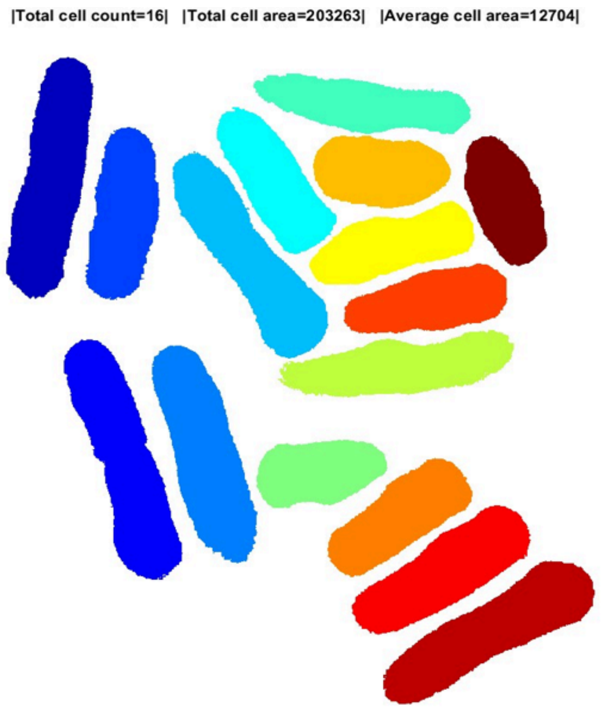

Threshold:82 Component#:16
Threshold:77 Component#:16
Threshold:76 Component#:15

Two Result Comparisons

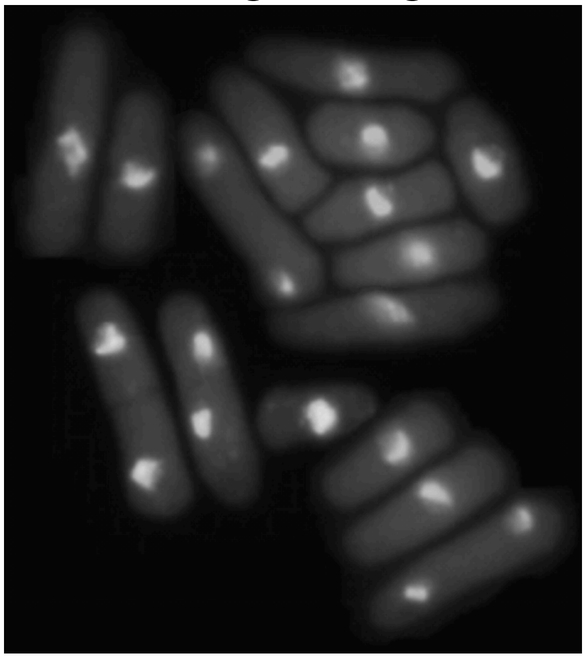
Gaussian



Adaptive Histogram Equalization



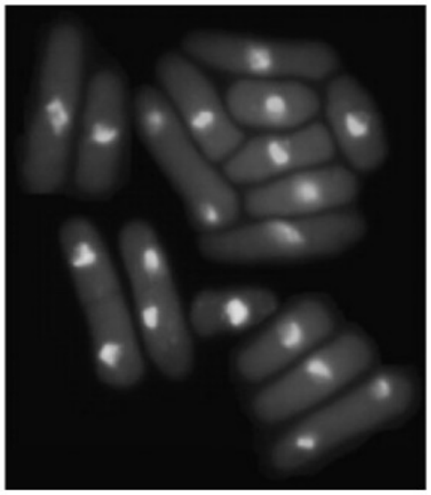
Original image



Result

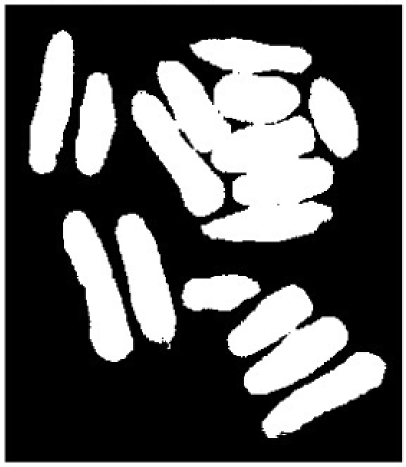
Two of the best results are selected to show in this report. One approach is starting by smoothing the image with Gaussian filtering. Below is the result after smoothing.

Gaussian filter



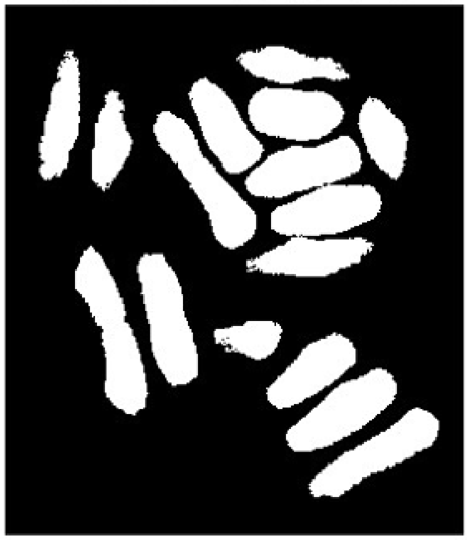
The smooth image doesn't seem to be any different to the original image. Anyway, four different approaches(matlab built-in function) are applied to get a threshold value for the smooth image. They are, respectively, graythresh(), otsuthresh(), adapthresh() and multithresh(). The result of using these four approaches to compute the threshold to binarize the smooth image is shown in the above **output image** section(**Gaussian filter image after binary threshold**). All approaches except the adapthresh() one seem to be close to our segmentation goal, although it's not completely separate from all objects yet. So, a further process is needed to refine the result. Since the 3 better approaches are giving the same result(by observation), the multithresh() approach is selected to be refined for the convenience of threshold value control. The multithresh() function always outputs a positive integer, which is more intuitive and easy to increase and decrease the threshold value for us to compare. (below is the unrefined image)

Single threshold computed from multithresh method



The threshold value 42 produced the unrefined image. As we can see, the top right cells are having many connections. To eliminate those connections, I tried to increase the threshold value. But first we need to decide how much should increase since the higher threshold value will shrink our cell size at the same time. The goal is to find the threshold value that will separate all cells(produce correct number of cells in image) and preserve the perfect shape or area of each cell. So, I put a cap 55 for the threshold value, which is equal 1.3*(threshold value). Then, apply all threshold values from the range of 42-55 to the smooth image and filter out only the result that has a different number of cells separate. The result is shown in the output **image** section(**Adjust Threshold value**). By observation, the result with threshold value 51 is the best fit to our goal, in which all cells are separated. (Below is the image)

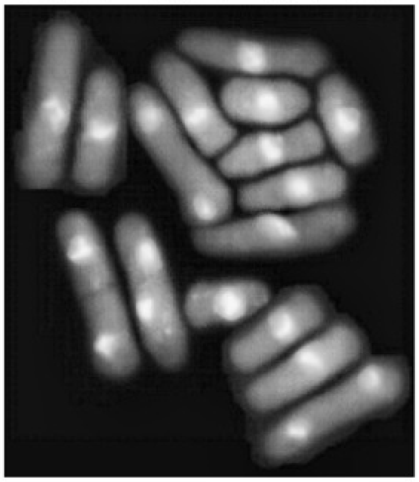
Threshold:51 Component#:16



However, this result doesn't preserve all cells' original shape well enough. Especially the middle small cell, it's significantly smaller compared to its original shape. Although we can refine the shape by doing a dilation morphology, it also recovers some connection between cells. Anyhow, this approach doesn't work perfectly on preserving the cell area but it actually gets the correct number of cells. The average cell area for this result is 9091 pixels. (Result image on **output image** section **Two Result Comparisons**)

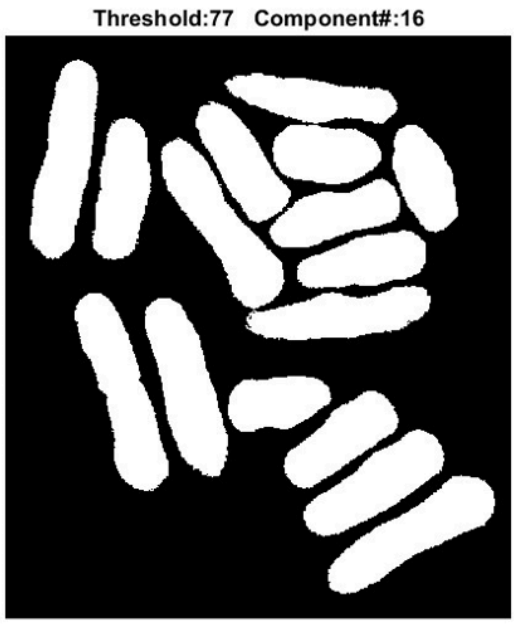
The other approach is smoothing the image by the adaptive histogram equalization technique. Below the result of this smoothing, which significantly enhances the contrast.

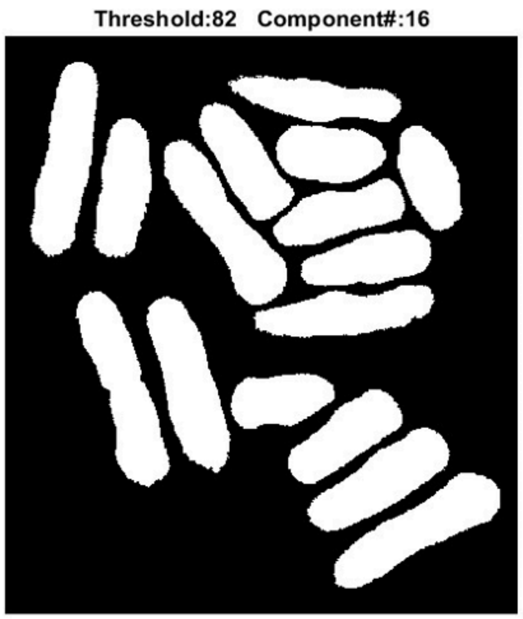
Adaptive histogram equalization



The following process is the same as the approaches mentioned previously. Applying four different threshold selections and binarizing the smooth image. The result is quite surprising because it looks perfectly like our goal. However, there might be some improvement for the cell area by decreasing the threshold value. In this case, the multithresh() approach is continued to be used for testing. The threshold value computed from multithresh() is 82. The goal of this testing is finding the minimum threshold value that can maximax the cells area and preserve the number of separate cells. The process is decreasing the threshold value by one and applying it to binarize the smooth image every time. If a threshold value is applied and the number of separate cells in that result image is decreased, then the minimum threshold value will be this threshold value + 1. (This process result is shown on **output image** section **Adjust Threshold value**)

The minimum threshold value I found from this testing is 77. Below is the result of it and the result of original threshold value 82.





It's really hard to see the difference between the two results. But in fact, the total area cell of the 77 threshold results 203263 pixels which is greater than the 82 threshold result 195264 pixels. Thus, the best average cell area is 203263\16 = 12704. (Result image on **output image** section **Two Result Comparisons**)

Comparing the two results of Gaussian and Adaptive histogram equalization, the second approach is producing a much better result than the first approach. (12704 > 9091)

Resources that Helped Me

- https://www.ibm.com/topics/image-segmentation#:~:text=Image%20segmentation%20is%20a%20computer,faster%2C%20more%20advanced%20image%20processing.
- https://encord.com/blog/image-segmentation-for-computer-vision-best-practice-guide/
- https://www.mathworks.com/help/images/