Counting the Number of Axons from Mice Optical Bundles using Morphological Image Segmentation

Robert Misevski

EECS 490 Mid-Term Project
Case Western Reserve University, Cleveland, OH, Email: rxm183@cwru.edu

Abstract

This paper presents the design of an application with digital image processing algorithms in MATLAB. This algorithm allows for this application to count the number of axons of the optical bundle from mice, which were obtained from an image. The images were supplied by Prof. Howell at Case Western University who is researching retinal sensing. He is looking for a way that automates the location and counting of closed bundles of axons from the supplied images. This application attempts to do that with good precision. This GUI application allows a user to open and display the image that will be counted. The application uses default values to determine the best threshold for processing the image (threshold is also user selectable). A second image then displays the found axons, and a text field displays the numeric count and processing times.

KEYWORDS

Counting Axons, Morphological Image Segmentation, MATLAB, Auto Threshold

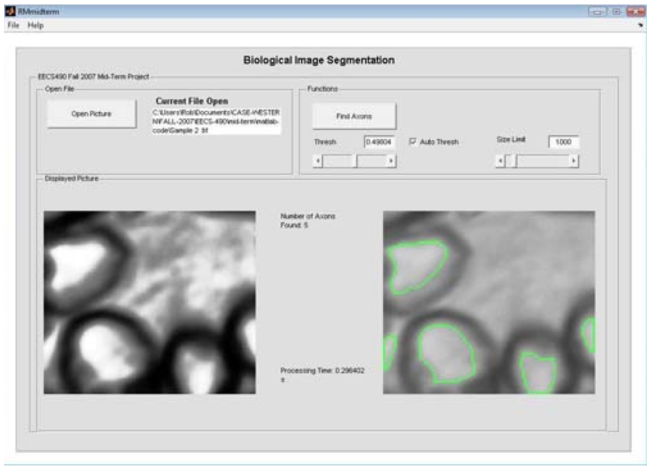
INTRODUCTION

Morphological Image Segmentation refers to using morphological processing steps in order to separate objects in a given image. Morphology in general is a segment of biology that deals with the appearance of a structure. This might include an organisms shape, structure, or pattern. Morphology in terms of mathematics of image processing, deals with extracting the general shape of an object that is of interest [1]. Segmentation is the process of dividing an image up into multiple regions in order to locate objects or determine boundaries [1]. This project deals with automating the process of counting axons. The role of the axon is to carry nerve impulses from the retinal sensors. The retinal sensors specifically the rods and cones could not be captured since the retina is curved. Instead the optical nerve bundle was sliced to capture the image of the axons. Though it is not fully understood why one would need to count or automate the counting of these axons. I can deduce that this is helping in some type of research, which might someday help humans fight disease. Since there are a high amount of similarities between mouse and human physiology, it has made working on mice the top choice in research. Hopefully this application can be useful in that research.

There is much difficulty in processing these biological images, since they are typically very blurry or noisy. Another problem is gray scale intensity and shades which very throughout the image. Lastly the objects of interest are not the same size and shape as each other making it very difficult to distinguish the background from the axon. With these problems and using some other techniques in digital image processing I have come up with an algorithm with very good accuracy that counts the axons. I was supplied with a variety of test images and the algorithm has done a good job on the ones that had a good contrast ratio (not to dark). Processing all the test images returned an average of 72% success rate of finding axons versus a trained human.

GUI AND ALGORITHM DEVELOPMENT

The graphical user interface I developed to showcase the algorithm using MATLAB is in Figure 1 it consists of 2 display images. Once a user selects the desired image using the open button, that image gets displayed in the lower left hand side of the window. The top right of this window consists of the controls to find the axons. The find axon button performs all the necessary steps in locating the axons. Then it displays the original image with the located objects outlined over the original in the lower right of the window. The GUI has several features for selecting a threshold when converting the image to black and white. One can enter a value between 0 and 1 or use the slide to see active feedback. If one so chooses the GUI has an auto threshold check boxes that calculates the best threshold for the given image, and the then displays that in the text and slider. The second slider located to the right of the first is used to limit the detection size of objects (connected pixels). The area between the two images is reserved for displaying the total number of axons the algorithm found and how long it took to find them in seconds. Since this project depended on a large number of steps to eventually find and count an image, the GUI design helped speed up corrections. With this GUI I was able to use a multitude of techniques to see which had the best results among all the images.



Current File Open
C:\Users\Rob\Documents\CASE-WESTERN\FALL-2007\EECS-490\mid-term\matlab-code\Sample 2.tif

Thresh 0.49804 ☑ Auto Thresh Size Limit 1000

Number of Axons Found: 5

Processing Time: 0.296402 s

Figure 1. The main graphical GUI

The image I chose to demonstrate this algorithm had %100 success rate at finding all 5 axons located in Figure 2, though the 5th one was not in the correct position as the test image. I can argue that these test images might not be the best gold standards to use since there are discrepancies in what to count within an image. Some images contain partial axons that are sometimes counted and sometime uncounted. When comparing this to a computerized algorithm, the computer will count all partial axons. An algorithm to subtract all connected pixels that touch the boarder could have been used but would result in very few counted axons.

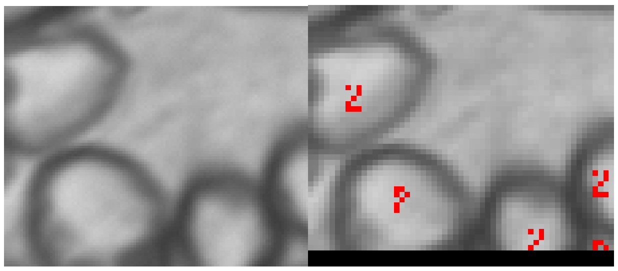


Figure 2. This is sample-2 test image that will be used to describe the algorithm

Figure 3 shows the image sample-2 after a wiener filter was applied, and then a histogram equalization. A wiener filter was selected in order to remove and resident noise in the test image. The function **wiener2** uses a low pass-filter on the image that has been degraded by constant power additive noise. **Wiener2** estimates the local mean and variance around each pixel [6]. MATLAB employees a nice function **histeq**, which enhances the contrast of image. It does this by a transform that generates an image where the intensity levels cover the entire range of the histogram [2]. The equation of this transform is as follows:

$$s = T(r) = \int_0^r \Pr(w)dw \quad (1)$$

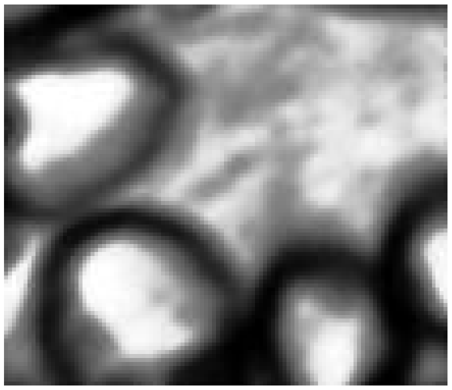


Figure 3. Sample-2 after a wiener filter to clean up any noise and then a histogram equalization to clean up contrast

The next step in this series is to transform the image to black and white (binary). The first step in doing this is selecting a threshold. The threshold decides what is an axon, and what is background information. This was done using the **graythresh** function, which employs the Otsu's method. [4] Otsu's method reduces the within-class variance defined by the following equation:

$$\sigma(t) = q_1(t)\sigma_1(t) + q_2(t)\sigma_2(t) \quad (2)$$

In general this algorithm does a good job in determining how to separate the foreground from the background using the histogram of the image.

The function **im2bw** converts the gray scale image to binary image, based on a threshold input. The output image replaces all pixels in the input image with either a one or a zero (black or white) depending on where the luminance falls on the scale between [0-1] [3]. The resulting black and white transform of the test image is shown in Figure 4 As one can see using the auto threshold produces a very good black and white image. Now that we have the image in black and white we need to do further processing in order to reach our goal. As seen in Figure 4. We have a small white spot in the top left of the image that is not an axon, neither is the large object that takes up most of the top right of the image. In the large object we have a black spot that needs to be taken care of. This image doesn't show the problems of having black spots with in our objects. Regardless we need to eliminate them.

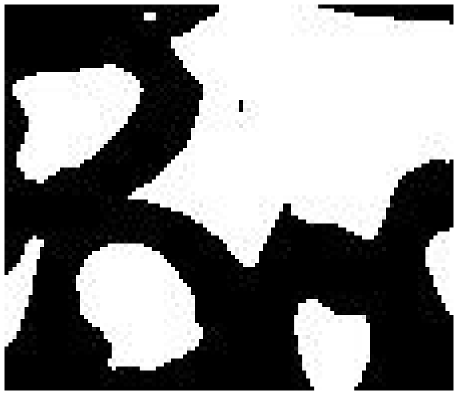


Figure 4. Sample-2 converted to black and white

In Figure 5 we use the MATLAB function **imfill** to fill image holes. A hole is a set of background pixels that cannot be reached by filling in the background from the edge of the image [7].

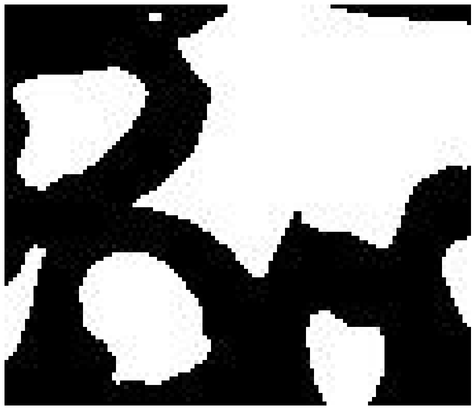


Figure 5. Sample-2 after all white objects have been filled

After we cleaned up the black spots we need to eliminate false axons produced from the black and white transformation. In Figure 6 we see using a morphological operation we can get rid of the small errors. The one I chose was the opening function **imopen** which did a good job in eliminating the small white spot. Opening is erosion followed by dilation which can be expressed in the following equation:

$$A \circ B = \bigcup \{(B)_z \mid (B)_z \subseteq A\} \quad (3)$$

This function needs some type of structuring element which is represented in equation (3) by B. The object is A. Using the MATLAB function **strel** I created a structuring element of a disk with size 3. This did a very good job in deleting the small spot and smoothing the rough edges.

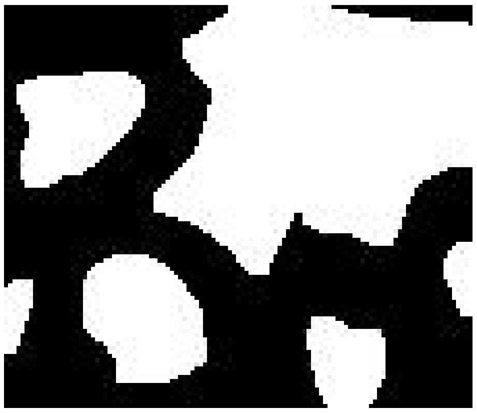


Figure 6. Sample-2 after an opening function which got rid of small artifacts

The next function I used **bwareaopen** is designed to remove small objects but in this case was used to remove large unwanted objects. This function works in finding the connected pairs and computing the area and discarding the unwanted objects. The resulting image in Figure 7 shows using **bwareaopen** with a input from the GUI on what type of large object to discarded. It then subtracts that from the result of doing a **bwareaopen** that passes all objects. The resulting operation produces a black and white image that only shows the axons.

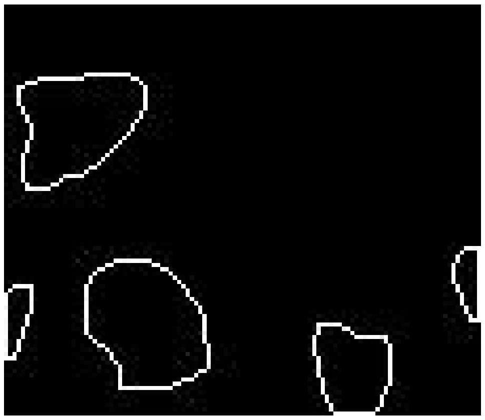


Figure 7. Sample-2 after a function that leaves perimeter pixels of objects while subtracting larger objects

Figure 8 shows the result of Figure 7 overlaid onto the original image using **imoverlay**. Now that we have found the objects in the image is a large step, but we still have to find a way to count those objects. The function **bwlabel** is the first step. This works by labeling connected components in binary image, which can be detailed as follows. [8] The function goes through the binary image and finds the connected 1's that comprise an object. Then it assigns is a value of 1 for the first, then 2's for the second and so on. If you take the max of the max of the matrix you will be left with the last found object which tells you how many total objects where found.

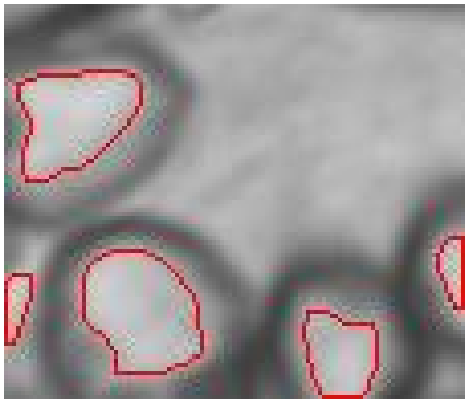


Figure 8. Sample-2 is then finally represented with an overlay of the outlined objects onto the original image

RESULTS AND DISCUSSION

Figures 9-14 show how my algorithm stacked up against the test images. In Figure 9 we see how the contrast of the image made it difficult to pick up individual axon in the middle of this image. The size limit removed the middle section because it showed up as one large axon. Table 1 shows that the application found 53% of the axons (21/40).

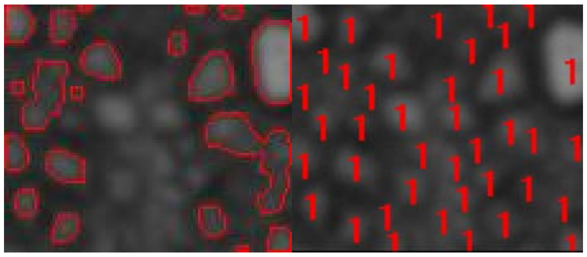


Figure 9. Sample-1, My algorithm vs. human

In Figure 10 and 11 the application had better luck with images Sample-3 and sample-4 finding (22/32) and (23/36) about 69% and 64% accuracy. Some error is evident in the fact some axons are grouped together being counted as one.

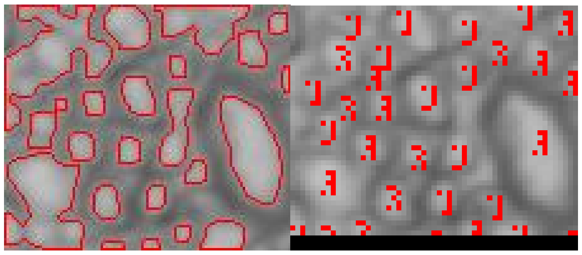


Figure 10. Sample-3, My algorithm vs. human

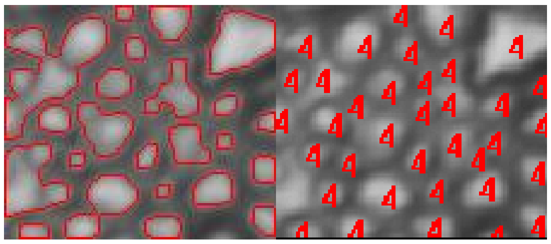


Figure 11. Sample-4, My algorithm vs. human

For the test image sample-5 in Figure 12 the algorithm had very good luck finding (12/13) axons with an accuracy of 92%. You can even argue that it could have been 100% if not for a human identifying an axon that is very small and vague.

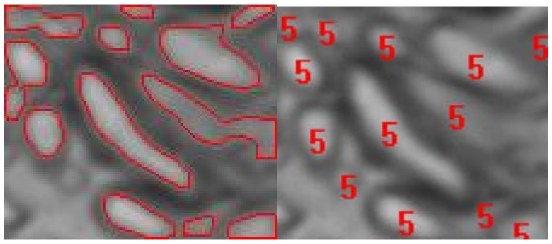


Figure 12. Sample-5, My algorithm vs. human

Test image sample-6 in Figure 13 had some trouble picking up some of the axons. I did very well in discarding the large portion from the left of the image. This produced 17 of 33 finds, which is 52% accuracy.

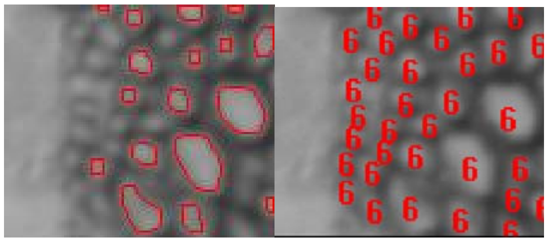


Figure 13. Sample-6, My algorithm vs. human

The original proposed image can be viewed in Figure 14 with an overlay of the axons that the algorithm found. I have no data to compare how well the algorithm worked when the same image was counted by a human. From a visual stance it did seem to pick up most of the obvious axons. Since I'm not a trained eye I done know what is a valid axon or not. According to Table 1 the algorithm determined that 255 axons were present in that image.

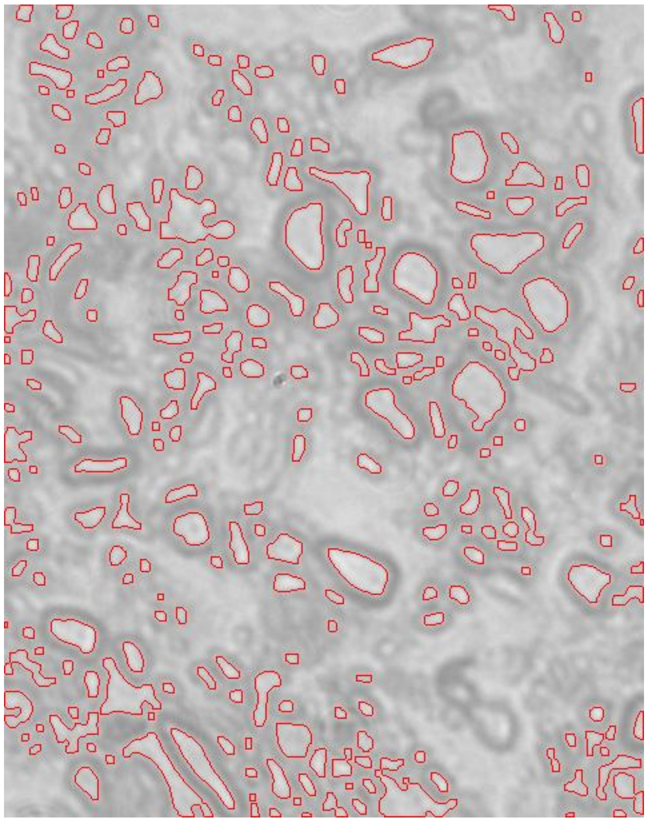


Figure 14. Original Axon Image, My algorithm on the original

Figure 15 shows the large cross section that was pieced together to form one image. MATLAB was unable to perform any operations on this since the image was too large, and caused memory problems in MATLAB.

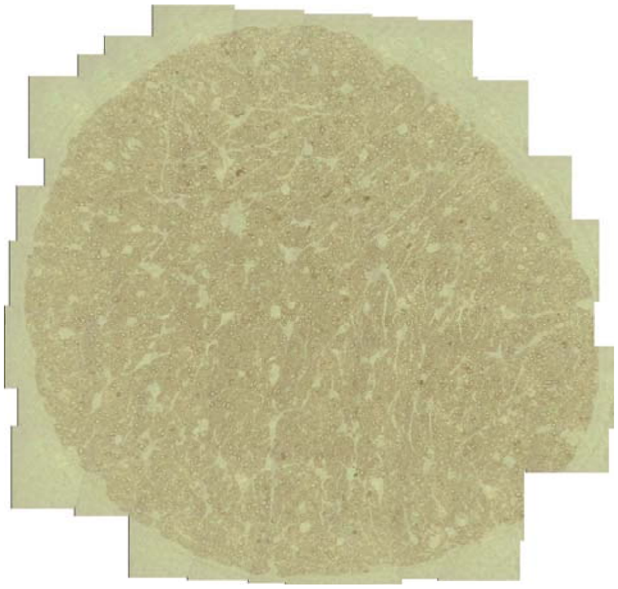


Figure 15. Large Axon Image

Table 1 lists how my algorithm stacked up against a trained human eye in identifying axons. Table 2 lists the percent error/success. Since every image was different in the fact they contained a wide variety of sized objects a different object pixel limit was required in order to eliminate some background information. Table 3 shows what limits were chosen for each test image. 2000 was an average number for 5 out of 7 images. The processing time was fairly quick with most of the test images as shown in Table 4. This was due to the images small size. The original was a much larger image and needed 1 second to complete on my PC.

Table 1. The results from 7 test images. This spreadsheet includes the number of axons a human identified vs. the algorithm.

Image	Test Image Count	Processed Image Count
Sample-1	40	21
Sample-2	5	5
Sample-3	32	22
Sample-4	36	23
Sample-5	13	12
Sample-6	33	17
Original	N/A	255

Table 2. The error and success rate of finding axons vs. a human

Image	Error	Success Rate
Sample-1	🟥 47.50%	🟥 52.50%
Sample-2	🟩 0.00%	🟩 100.00%
Sample-3	🟥 31.25%	🟥 68.75%
Sample-4	🟥 36.11%	🟥 63.89%
Sample-5	🟩 7.69%	🟩 92.31%
Sample-6	🟥 48.48%	🟥 51.52%
Original	N/A	N/A

Table 3. The pixel size limit used for each test image

Image	Pixel Size Limit
Sample-1	2000
Sample-2	2000
Sample-3	2000
Sample-4	2000
Sample-5	1000
Sample-6	2000
Original	1600

Table 4. The resulting processing time for each test image

Image	Processing Time (s)
Sample-1	0.312002
Sample-2	0.296402
Sample-3	0.296402
Sample-4	0.296402
Sample-5	0.312002
Sample-6	0.296402
Original	1.01401

CONCLUSION

The purpose of this project was to create an automated way in counting axons from the optical bundle from mice, which were obtained from an image. The algorithm I chose used black and white transformation with various morphological operators. One could have used a variety of other techniques such as edge operators to tackle this problem. I had good success with the method I chose. Since the axons varied so much in size and shape coupled with the poor quality of the sample images posed the most difficulty. The other problem was the inconsistencies in what was counted as an axon and what wasn't. If a standard were implemented to count the axons that consider a computerized algorithm, and with better quality images the percent rate of accuracy would have increased.

REFERENCES

[1] Rafael C. Gonzalez, Richard E. Woods, *Digital Image Processing, 3rd Edition*, Prentice-Hall, Inc., Upper Saddle River, New Jersey, USA, 2008, pp. 627-809

[2] Rafael C. Gonzalez, Richard E. Woods, Steven L. Eddins, *Digital Image Processing using MATLAB*, Prentice-Hall, Inc., Upper Saddle River, New Jersey, USA, 2004, pp. 170-172 and 334-48 3

[3] MATLAB HELP version 2007A The MathWorks, Inc. 3 Apple Hill Drive Natick, MA 01760-2098 UNITED STATES

[4] N. Otsu, "A threshold selection method from gray-level histograms," IEEE Trans. Sys., Man., Cyber., vol. 9, pp. 62–66, 1979.

[5] Steven L. Eddins, *Cell segmentation*, http://blogs.mathworks.com/steve/2006/06/02/cell-segmentation/, June 2nd, 2006

[6] Lim, Jae S., Two-Dimensional Signal and Image Processing, Englewood Cliffs, NJ, Prentice Hall, 1990, p. 548, equations 9.44 -- 9.46.

[7] Soille, P., Morphological Image Analysis: Principles and Applications, Springer-Verlag, 1999, pp. 173-174.

[8] Haralick, Robert M., and Linda G. Shapiro, Computer and Robot Vision, Volume I, Addison-Wesley, 1992, pp. 28-48.