Location of Closed Bundles in an Optical Nerve Using Boundary Tracking of a Digital Image

David Young
Department of Electrical Engineering and Computer Science,
Case Western Reserve University, Cleveland, OH, Email: dly@cwru.edu

Abstract

This paper presents an application of image segmentation to locate and count individual nerve bundles from the optical nerve of mice. Automated image processing techniques such as color-space manipulation, histogram analysis, and boundary definition are used to reach the solution. To further increase the accuracy of the algorithm, minimal input is required from the user to allow the program to automatically set parameters.

KEYWORDS

Image processing, image segmentation, histogram, binary image, nerve bundles

INTRODUCTION

The task of counting bundles in an optical nerve is not only a tedious task, but one that must be executed by a person who is knowledgeable in the field to recognize a legitimate bundle as opposed to noise or background. It makes sense that a program is therefore able to help alleviate the lab staff's workload and expedite the time it takes to achieve an accurate bundle count. However, an automated system that efficiently recognizes bundles is difficult to realize for the same reasons a trained technician is required for manual counting. While the images are of reasonable resolution, they have very low contrast and bundles vary individually in both shape and size. Additionally, the area inside a bundle can appear homogeneous with the areas between the bundles. Bundle counts completed by two different technicians can even produce conflicting data. An example of a full optical nerve is shown below in Figure 1.

Figure 1 – Image of a full optical nerve.

Due to the ambiguities related to the problem, an acceptable accuracy from the program will be defined as +/-15% from an expert's count.

Boundary following is a method widely used for image representation applications. The Moore Boundary Tracking Algorithm [1968] outlines a reliable method for defining boundaries in an image. To ensure simplicity of the process, the algorithm can only be run on binary images. The five basic steps below complete the algorithm, and are illustrated by Figure 2.

1. Let the starting point, b_0, be located in the upper-left pixel of the binary image of value 1. C_0 is defined by the neighbor directly to the left of b_0, and must be of value 0 (Figure 2b). Moving from c_0 in a clockwise direction find the next pixel of value 1 and define it as b_1 and the pixel preceding it as c_1. [1]
2. Let $b=b_1$ and $c=c_1$ (Figure 2c).[1]
3. Perform the same operation as in step 1, looking at adjacent pixels beginning with c in a clockwise fashion for the next location of value 1.[1]
4. Name the next pixel b, and the pixel preceding it c. [1]
5. Repeat steps 3 and 4 until two conditions are met:
    a. $B=b_0$
    b. After completing step 3 once more, the next pixel is found to be b_1.[1]

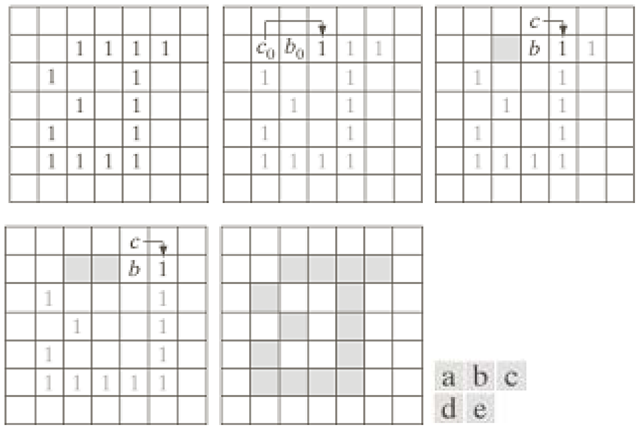

Figure 2 [1] - Illustration of the Moore Boundary Tracking Algorithm. [book]

It is important to note the dependence on step 5 for universal operation. This last step prevents "spurs" from being recorded as complete boundaries [2]. Spurs are small tangents of a larger boundary. An example of the significance of step 5 is illustrated in Figure 3[1]. Without the qualifier for verifying the location of b_1, the image on the left of Figure 3 would be improperly characterized. While it would find the first two points easily, as shown in the middle image, the image on the right would consider the boundary complete without checking against b1. When step 5 is incorporated, the algorithm would understand the right side of Figure 3 as a "spur" and decide to traverse the lower part of the object [2].

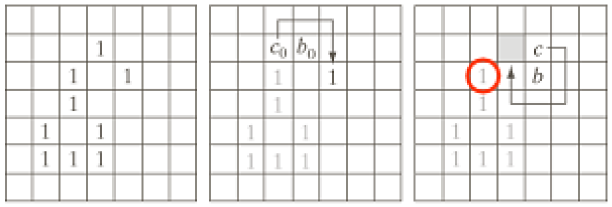

Figure 3[1] – Note the smaller boundary that would be defined if step 5 was not included in the algorithm.

A notable benefit to using boundary following over other image segmentation operations (i.e. the watershed algorithm) is the allowance for recognition of boundaries within boundaries. This is especially important for the images related to this paper because distinguishing the inside and outside of boundaries is often difficult. This characteristic makes it possible for algorithms to recognize an empty area surrounded by bundles as a false-bundle. This error is rarely seen and thus acceptable. However, it is possible for a significant number of bundles to be present as an "island" inside of the "false-boundary." Because boundary following searches for boundaries within boundaries, these extra bundles would still be counted. However, the watershed operation would fail to identify the bundles within false-bundles, dramatically increasing inaccuracies. This point is illustrated below in Figure 4.

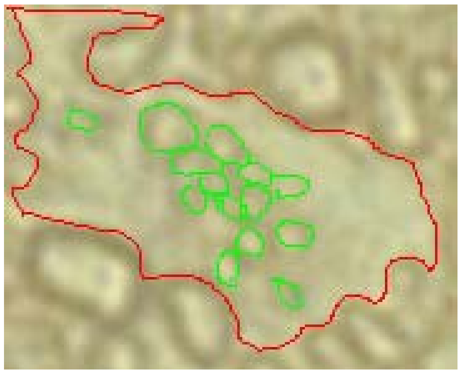

Figure 4 – The false boundary is highlighted in red; the true boundaries inside of the false boundaries are highlighted in green.

This additional benefit of increased accuracy does not come without a cost. The boundary following operation is much more computationally intensive than the watershed method. However, in this biological application the optimization of accuracy is preferred over time-course of analyses; thus, the boundary following method is implemented.

DEVELOPMENT OF THE PROGRAM

Figure 5 shows an excellent example of the type of image that must be processed to determine a final bundle count.

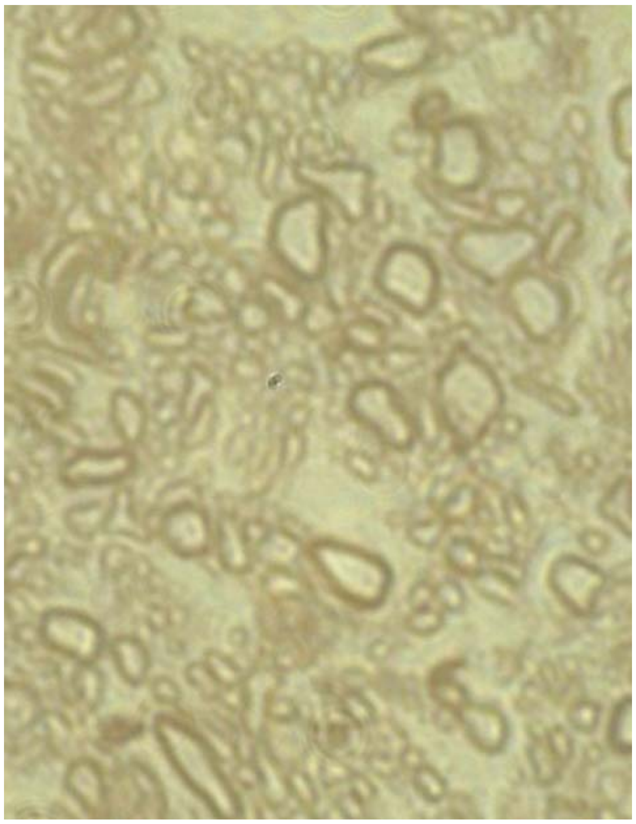

Figure5– Note the low contrast, random shapes, and color similarities between the inside and outside of each bundle.

The first challenge of the program was to implement a meaningful application of the boundary following algorithm defined in the previous section. Fortunately, MATLAB has a built-in function in the Image Processing Toolbox that implements Moore's Algorithm called '*bwboundaries*'.[3]

However straightforward the implementation of the boundary following application may be, getting the image to a point that will allow accurate execution of the *bwboundaries* function creates a challenge. The image shown in Figure 5 must be converted to a binary image that accurately defines the boundaries. Additionally, the program must be able to accommodate images that may have different contrast, brightness, or resolution than the one shown in Figure 5.

A complicated and computationally intensive method such as edge detection could be selected to convert the color image to binary. However, the information in the image is simple and somewhat underdefined by the lack of contrast and similar boundary intensity. Therefore a thresholding operation would prove to be more efficient without sacrificing conversion quality or edge definition.

Any function that requires parameterization to operate correctly carries with it the difficulty of setting the correct parameters. Image variety introduces the highest level of error when attempting to create a binary image with an intensity threshold.

The preferred method to determine the optimum threshold intensity is by creating a histogram of the image. The threshold is then set at the peak intensity level of the histogram for images such as the optical nerve in Figure 5. The histogram that relates to a grayscale version of Figure 5 is shown below as Figure 6.

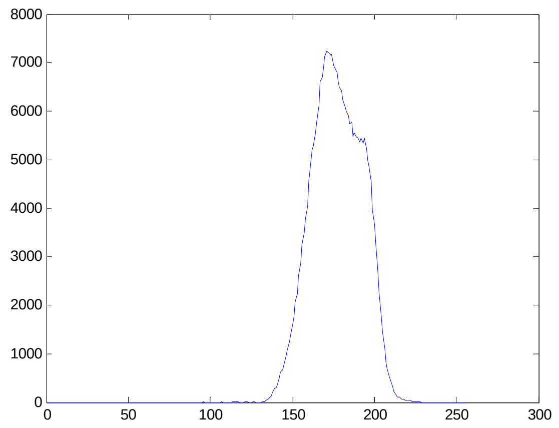

Figure 6 – Histogram of the image shown in Figure 5.

The ultimate goal stated in the introduction is to optimize for count accuracy. Two situations can arise to skew results based on a binary image created from a single threshold parameter for all images. Firstly, there may be sections of the image that are not representative of the optical nerve, such as a border on the image, creating errors in the histogram. Secondly, the threshold set at the peak of the image may be good, but may not be the optimal intensity level at which to create a binary image.

To overcome these two hurdles and get the highest accuracy possible, the user is requested to make two selections interactively. First, the user is directed to select a small area of bundles representative of the image. A histogram is performed on the small image to find its peak. The binary image is then created using twelve threshold levels close to the peak value on the histogram. A brief analysis of the '*bwboundaries*' function is completed on the small selected area utilizing twelve different binary images. The boundaries are displayed over the original small image section to allow the user to easily define the best outcome. The user is then asked to select the image which represents the optimal threshold level for that particular image. A sample of the twelve different options is shown below in Figure 7.

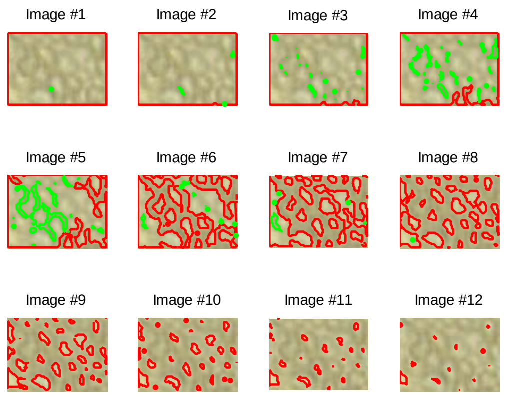

Figure 7 – Note the precision gained by having the user help the program select the optimal threshold level.

The two user input steps above allow for the histogram-based threshold to be set at the optimal level for each image. Although this increases the algorithm's ability to accurately count the bundles in a variety of images, it does move away from the goal of minimizing user input. However, the program is written to have all user input completed at the beginning of the program's operations. All calculations needed for the user input are based on small versions of the image, ensuring that the user does not have to wait for the computer to process what could be a very large image. Regardless of the image size, the user can select the file to be analyzed, define the representative area and the optimal threshold in less than one minute. The additional user input was deemed insignificant by the author compared to the increase in accuracy.

With the optimal threshold set by the computer's histogram and the image-based selection made by the user, the '*beboundaries*' function can be invoked to apply boundaries to the entire image. The '*bwboundaries*' function only requires an input of the binary image, but has 4 outputs: B, L, N, and A.

B is a P-by-1 cell array, where P is the number of objects and holes. Each cell contains a Q-by-2 matrix, where Q is the number of boundary pixels for the corresponding region. Each row of these Q-by-2 matrices contains the row and column coordinates of a boundary pixel. The coordinates are ordered in a clockwise direction [3]. L is a two-dimensional array of nonnegative integers that represent contiguous regions [3]. N is defined by the number of objects found and A is an adjacency matrix. A represents the parent-child-hole dependencies [3].

To find, record, and plot all of the boundaries, all 4 outputs of the function are required. The program is able to locate all boundaries as well as denote the parent/child classification in variables.

The final step is to plot the boundaries directly over the original image is used to give the best possible graphical output to accompany the bundle count. This allows the technician to review the results for potential errors and inaccuracies.

RESULTS AND DISCUSSION

To continue with the processing of the image used in figures 5, 6, and 7, the program was allowed to complete the processing. Image number 7 was selected to determine the thresholding parameters. The total bundle count came to 476, and the output image is shown below in Figure 8.

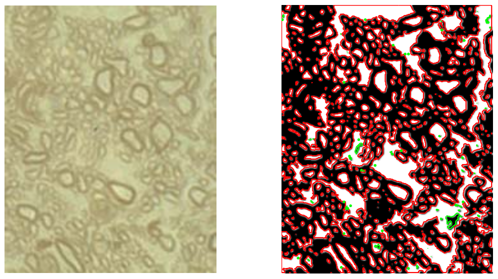

Figure 8 – Output image shown side-by-side with original image to verify the executed operations. Note the child boundaries that are highlighted in green. Final bundle count was 476.

The accuracy of the results of various images are dependant on the image as well as the input provided by the user. To evaluate the effectiveness of the program, "golden samples" were created by asking a doctor to identify bundles. There were 6 samples examined by both the doctor and the program.

Figure 9 below represents a processed image with very accurate results. For repeatability, image 11 was selected by the user causing the program to determine the threshold value of *thresh=0.5184*. The results were 97% accurate when compared to the golden sample, and Figure 9 shows the correlation between counted boundaries and bundle edges. The only visible error is seen in the lower-left section on the border of the image.

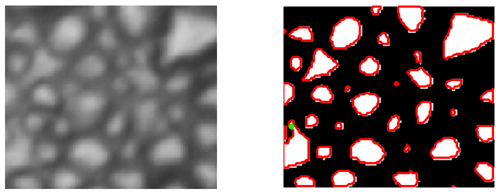

Figure 9 – Program output from analysis of sample4.tif with 97% accuracy.

Allowing the user to select the histogram characterization area proved to be valuable when examining the sample shown in Figure 10. The left side of the image was devoid of any bundles, and would skew the histogram analysis because it was of a single intensity. The user was able to select the area with only bundles to create a threshold level that provided more accurate results. The final count located 85% of the bundles identified by the doctor.

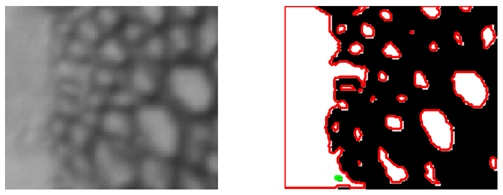

Figure 10 – Result from bundle count of sample6.tif analysis. Accurate count (85%) despite a large section of the image that has no bundles.

One instance where the algorithm did not yield good results is shown in Figure 11. This shows the specific weakness of processing images that do not have many bundles. Furthermore, the fact that the image had very few full bundles viewed made this image less representative of what would be expected to be found in a full image. The final count from processing this image was 240% of the doctor's count.

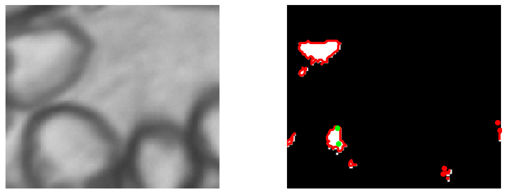

Figure 11 – Result from Sample2.tif analysis. An example of poor accuracy (240%) due to the small number of bundles and the close-zoom of the image area.

An example of an image that acheived reasonable results is shown in Figure 12. In this case, there were a few instances of visible errors. The areas highlighted in green appear to be false-bundles that had been recognized and counted. While unfortunate, results within 15% of a doctor's estimate were still acceptable.

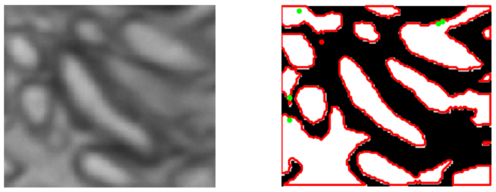

Figure 12 – Analysis of Sample5.tif with a total count 115% of the doctor's count. Note the false bundles in green.

An example of the program being able to handle an image of different contrast is shown below in Figure 13. Sample1.tif resulted in finding 85% of the bundles found by the doctor. Clearly this image is much darker than the others, which showed the effectiveness of the histogram analysis as a means of setting thresholds.

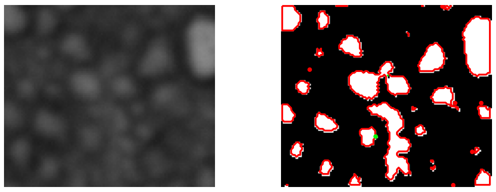

Figure 13 – Sample1.tif was analyzed with 85% accuracy.

The final golden sample named sample3.tif counted 115% of what the doctor counted and is shown in Figure 14. Many of these erroneous tallies were a result of partial bundles and partial empty spaces on the border of the image. While this error would not be accounted for in the program, an image with a higher bundle-to-border perimeter ratio, such as the one shown in Figure 5, would minimize these errors.

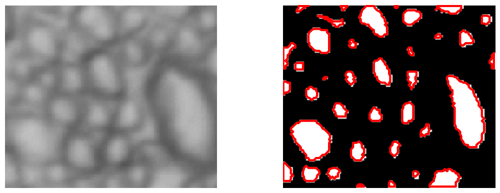

Figure 14 – 15% more bundles were counted compared to the doctor's count. Most errors are found to be due to lines at the border.

Table 1 summarizes the results and the related parameters. The Sample # column refers to the image sample number. The Image # column refers to the user-selected image. Threshold is the variable 'thresh' which was set by the histogram peak and the user selection. Doc count gives the total count as tallied by the doctor. Prog Count gives the total count found by the program.

Sample #	Image #	Threshold	Doc Count	Prog Count	% of Doc
1	8	0.2475	40	34	85%
2	9	0.7753	5	12	240%
3	9	0.6152	32	37	116%
4	11	0.5184	36	35	97%
5	7	0.4267	13	15	115%

Table 1 – A compilation of a doctor's count totals compared to the program's count totals of the golden samples.

SUMMARY

While the program is not as accurate as human analysis, it is clear that there are circumstances in which it can be used with acceptable accuracy (i.e. 85%) to execute a count in images that have a large number of bundles.

ACKNOWLEDGMENTS

This work was done in partial fulfillment of EECS 490, Digital Image Processing at Case Western Reserve University, Cleveland, Ohio. Course funding was provided by Keithley Instruments, Cleveland, Ohio.

REFERENCES

[1] R. Gonzolez and R. Woods, "Digital Image Processing", 3rd ed. vol. 3, M. McDonald, Ed. New Jersey: Pearson Prentice Hall, 2008, pp. 795-798.

[2] Francis Merat, "Lecture 23," *Proceedings from lecture 23 from the Fall 2007 EECS490 lecture series on Digital Image Processing*, pp. 5-6, 2007.

[3] The Math Works. (2007). Image Processing Toolbox – bwboundaries. The Math Works, Natick, MA. [Online]. Available: http://www.mathworks.com/access/helpdesk/help/toolbox/images/index.html?/access/helpdesk/help/toolbox/images/bwboundaries.html