TRACING, EXTRACTING FEATURES, AND CLASSIFYING MICROGLIA FROM VOLUMETRIC IMAGES OF BRAIN TISSUE

By

Zachary Stephen Galbreath

A Thesis Submitted to the Graduate

Faculty of Rensselaer Polytechnic Institute

in Partial Fulfillment of the

Requirements for the Degree of

MASTER OF SCIENCE

Major Subject: COMPUTER AND SYSTEMS ENGINEERING

Approved:

_____

Kim Boyer, Thesis Adviser

_____

Charles V. Stewart, Thesis Adviser

_____

Chris Bjornsson, Thesis Adviser

Rensselaer Polytechnic Institute
Troy, New York

December 2011
(For Graduation December 2011)

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGMENT

I owe thanks to many people who have helped make this thesis possible.

First of all, I would like to thank Kim Boyer for guiding me through the process of obtaining my master's degree, and for patiently helping me improve the clarity and content of this document.

I acknowledge Badri Roysam as the main driving force behind my research. Without him, my journey into the world of brain imaging research probably would have never begun.

I would like to thank Chuck Stewart for teaching me how to effectively use a variety of powerful computer vision techniques.

I owe thanks to Chris Bjornsson and Karen Smith for providing me with high-quality images of brain tissue samples. Without these, my work would have been impossible.

I thank Amit Mukherjee for his neuron tracing algorithm. This provides a crucial component of the work that I present here.

I am grateful for all the help that Luis Ibáñez has provided me. He has consistently been eager to help me achieve my goals throughout the course of this project. The Insight Registration and Segmentation Toolkit that he has helped to create and maintain is an invaluable tool for biological imaging researchers.

Will Schroeder has offered me nothing but unwavering support and flexibility throughout my graduate studies. For this, I owe him my sincere gratitude.

I thank Bill Shain for sharing his knowledge of the brain with me and for taking an active interest in my research.

I would also like to thank the developers and maintainers of a number of software packages that have made my research possible: L-Measure, ITK, VTK, Farsight, ParaView, and ImageJ.

Words cannot properly the gratitude I feel towards my family for their unconditional love and support throughout the years. I also appreciate the fact that they have been genuinely interested and engaged in my research over the years.

Finally, I would like to thank my love Megan for all the support she has offered me throughout this busy time in my life.

ABSTRACT

Macrophages are an important component of our immune system. These are cells that hunt down and consume pathogens, foreign materials, and damaged cells in our bodies. By doing so, macrophages protect us from infection and help us recover from injury. Throughout most of the body, this role is played by a subset of the white blood cells.

White blood cells cannot be found within the central nervous system (CNS). This is due to the existence of the blood-brain barrier, which prevents many materials from entering the environment of the CNS. The blood-brain barrier protects us from substances found in the rest of the body that may be injurious to the brain. It also provides a consistent environment for the CNS.

Instead of white blood cells, the CNS has its own specialized type of macrophage known as the *microglia*. Microglia are intimately involved in a number of conditions affecting the CNS including Alzheimer's disease, multiple sclerosis, and spinal cord injury. In this thesis, we present an open-source system to automatically segment microglia from 3D images of brain tissue. This produces a model of each microglial cell in the original image.

In addition to segmentation, we also demonstrate how to use the L-Measure software package to generate an extensive set of features from these cellular models. These features allow us to perform Kernel Partial Least Squares regression. This technique produces a classifier that can predict whether or not a microglial cell was found in injured tissue with an accuracy of 86.4%.

This thesis is motivated by a desire to promote progress on open areas of biological research. It is our hope that the tools presented in this study will allow researchers to gain greater understanding from biological images. In turn, perhaps they will be able to develop more effective treatments for neuro-degenerative conditions.

1. INTRODUCTION

We begin this chapter by describing our motivation for pursuing microglia research. We then establish the work presented in this thesis within the context of prior research. The introduction concludes with a brief discussion of the techniques used to accomplish the goals of this thesis.

1.1 Motivation

Learning more about microglia may give us greater insight in how to maximize their protective capabilities. This may, in turn, lead to more effective treatments for the various disorders and diseases of the CNS. Here we examine a few selected research areas where microglia are of critical importance.

1.1.1 Neuro-Prosthetic Devices

Neuro-prosthetic devices have promise for treating neuro-degenerative disorders and helping patients regain lost CNS function. They also allow biological researchers to gain new insights into the inner workings of the brain. For example, neuro-prosthetic devices have led to an increased understanding of microglia structure and function [1]. Unfortunately, the effectiveness of these devices decreases over time. The brain reacts to the presence of a foreign body by encapsulating it within a protein and cellular sheath. This sheath is partially composed of microglia [2].

It is our hope that an increased understanding of microglia will allow us to develop more effective neuro-prosthetic devices. One way in which this could be accomplished is by suppressing the encapsulating behavior of microglia. It may also be possible to design a device in such a way that it will not induce this disabling response from the microglia.

1.1.2 Alzheimer's Disease

A distinctive symptom of Alzheimer's disease is the formation of plaques consisting of beta-amyloid protein. These plaques can be found surrounding damaged

neurons. Microglia are responsible for clearing away excess beta-amyloid in the brain [3]. Therefore, it is unsurprising that they can be found in abundance surrounding these plaques.

While microglia are involved in fighting this disease, they are also implicated in its progression because they release cytokines as they break down beta-amyloid. This causes an inflammatory reaction in the surrounding brain tissue. Microglia lose their ability to engulf beta-amyloid in chronically inflamed brain tissue. Making matters worse is the fact that microglia release neurotoxic substances when treated with cytokines [3]. This is one of the causes of neuron damage in Alzheimer's disease.

Microglia are intimately involved in our body's response to Alzheimer's disease. It is our hope that learning more about these cells may shed light on better methods of treatment and prevention for this disease.

1.1.3 Diagnostics

Microglia are the fundamental component of our brain and spinal cord's immune system. These cells undergo morphological changes when reacting to the presence of an injury or infection. We refer to microglia that have undergone such changes as *activated*. In this thesis, we demonstrate the capability to automatically distinguish between unactivated and activated microglia. As imaging techniques improve, this ability could become an important tool in diagnosing conditions affecting the CNS.

1.2 Prior Work

Rouchdy *et al.* demonstrated the ability to segment microglia from images obtained by confocal microscopes [4]. They used the Fast Marching Method to generate a tree structure from a single, user-specified point. This tree structure serves as a model of a microglial cell. In order to control the size of the tree, Rouchdy *et al.* used Harris Corner Points as the stopping criterion of Fast Marching propagation.

The segmentation system presented in this thesis has two capabilities that distinguishes it from the work presented by Rouchdy *et al.* First, our system allows for

3D segmentation of microglia. Second, our seed points are detected automatically, eliminating the need for the user to select such points manually.

Xiao *et al.* presented a method for tracking the movements of microglia in 3D [5]. While we are working towards the same goals, we are focusing on different methods of investigating microglia. *Tracking* attempts to gain information about a cell's position, and how this position changes over time. This thesis focuses (in part) on *tracing*. Tracing attempts to deduce the morphological structure of a cell.

1.3 Statement of Goals and Techniques Used

This thesis presents a tool that allows for more effective analysis of images of microglia. It allows biological researchers to automatically model individual microglial cells, reducing the need for painstaking manual tracing. We also demonstrate how to generate an extensive set of features for each cell. This enables quantitative analysis of various morphological aspects of these cells. Finally, we show how to use these features to distinguish among different types of microglia.

The goal of this thesis is to aid researchers in gaining greater insight into images of microglia. It is our hope that this will result in more effective treatments for neuro-degenerative conditions. This feeds into the common goal of all biomedical research: to improve overall health and well-being.

Before we describe the techniques used in pursuit of these goals, it is necessary to describe the morphology of a typical microglial cell. Figure 1.1 depicts a single microglia. The nucleus of a microglial cell is contained within the blob-shaped cell body. This is known as the *soma* (plural *somata*). Emerging from the soma are many long, narrow, twisting appendages. These are called the *processes* [6]. We exploit these morphological details when segmenting microglia from volumetric images of brain tissue.

We use a two step approach to segmenting microglia. First, we isolate the somata and calculate their centroids. These centroids are used as seed points for our process tracing procedure. Once the somata and processes have been traced, we recombine them to form tree-shaped models of microglia.

When isolating the somata, we begin by using *Robust Automatic Threshold*

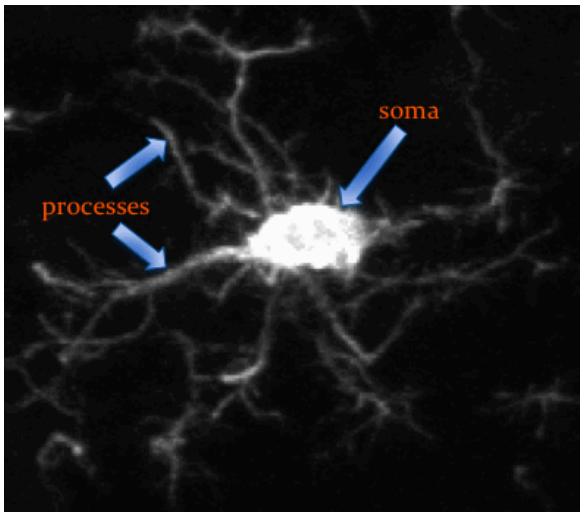


Figure 1.1: Maximum intensity projection (MIP) of a single microglia. Its soma and two of its processes are labeled.

Selection (RATS) to separate the image foreground from its background [7]. Afterward, we perform a *morphological opening* operation to erode away the narrow processes.

At this point we convert the remaining blobs of contiguous bright pixels into *labeled objects* [8]. Doing so allows us to calculate the volume and centroid of each blob. We measure the physical volume of each blob and remove those that are smaller than some experimentally-derived value. At this point we presume that the only blobs that remain are the somata. We record their centroids and move onto the process tracing step.

Process tracing uses a *Multi-scale Laplacian of Gaussian vessel detector* to identify points that are likely to lie along the centerlines of microglia processes [9]. We use these points to construct a Minimum Spanning Tree (MST) for each microglia. These trees are constructed using an adaptation of Prim's algorithm [10]. Tree building is done in such a way that we minimize the Euclidean distance between any two connected points.

Once segmentation is complete, we use the L-Measure software package to compute an extensive set of features for each cell [11]. These features allow us to perform Kernel Partial Least Squares (K-PLS) regression [12]. This gives us the ability to distinguish between two different types of microglia: those found in healthy

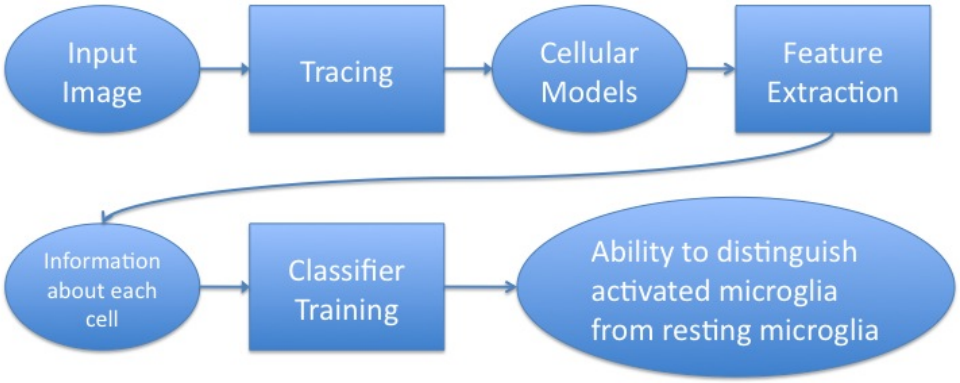


Figure 1.2: Workflow of the system presented in this thesis. Ovals are data or results. Rectangles are procedures or techniques used to generate data and achieve results. This figure highlights the primary accomplishments of this thesis: segmentation, feature extraction, and classification.

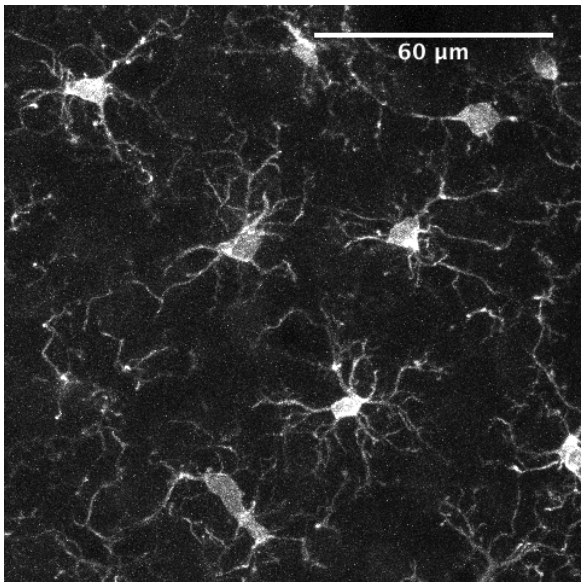
tissue, and those found near the site of an injury. Microglia found in damaged tissue are considered *activated*, while those found in healthy tissue are *inactive*. We can distinguish between active and inactive microglia because these cells undergo morphological changes when reacting to an invader. Figure 2.1 contrasts these two different states of microglia. While preparing this thesis we analyzed images of injured brain tissue taken at three distinct time periods: three hours, one week, and six weeks post injury.

In order to distinguish between active and inactive microglia we perform K-PLS regression using a uniform prior. This requires us to split our collection of microglia into two groups: a training set used to generate the classifier, and a testing set used to evaluate its performance. We randomly select approximately 20% of our microglia to serve as the training set. The training set is selectively sampled so that its ratio of activated and unactivated microglia matches that of the complete dataset. For example, if 40% of the cells in our dataset are activated, then 40% of the training set will also be activated cells. The remaining cells serve as the testing set. Our K-PLS classifier was found to be 86.4% accurate in distinguishing between activated and unactivated microglia. See Table 3.3 for more detailed classification results.

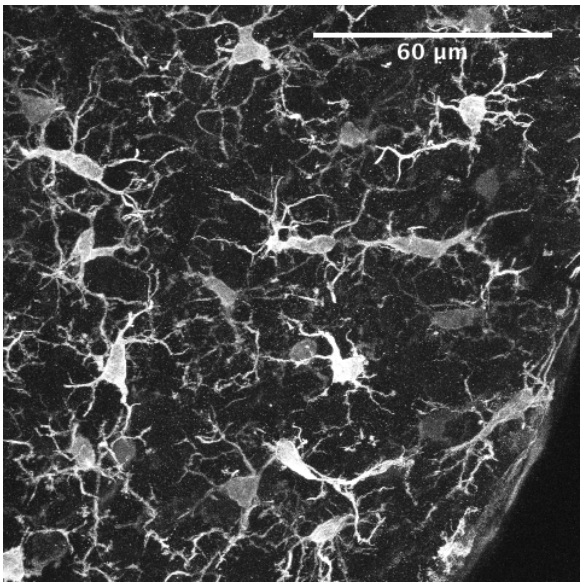
2. Method of Procedure

In this chapter we describe in detail the main achievements of this thesis: segmentation, feature extraction, and classification of microglia.

2.1 Tracing



(a) Inactive microglia



(b) Activated microglia

Figure 2.1: Maximum intensity projections of microglia. The black region in the lower-right corner of Subfigure b correponds to the location where a neuro-prosthetic device was implanted. Subfigure b was captured six weeks post injury.

We pursue a *divide and conquer* approach to tracing microglia. The first step is to isolate the somata and detect their centroids. These centroids are used as seed points to trace the long, narrow processes of the microglia. This combination of somata and processes allows us to model the morphology of microglial cells.

2.1.1 Somata

Before beginning segmentation it may be beneficial to apply some pre-processing techniques. Some of the images used in this thesis suffered from impulsive (salt &

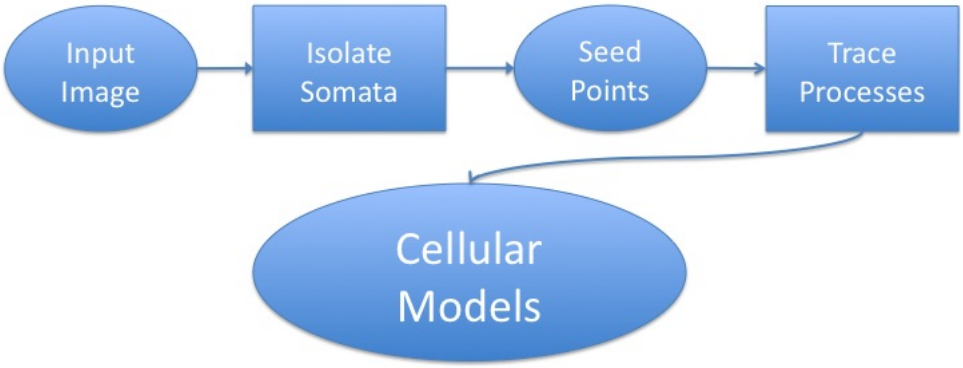


Figure 2.2: This figure presents a brief overview of the procedure that we use to generate cellular models from images of microglia.

pepper) noise. A median filter is useful in such situations. Histogram equalization can also improve results when working with images that exhibit poor contrast.

To aid in understanding our soma segmentation procedure, we will present the output generated by each step. Figure 2.1(a) will be the input image to our soma isolation procedure. We will continue to work with this image throughout the section.

Please note that all images used in this thesis are 3D, volumetric images. The images as presented in this thesis are *maximum intensity projections* (MIPs) of the actual images used in this project.

MIP is a method of visualizing a 3D image in two dimensions. A 3D image can be thought of a set of 2D images, where each Z value is considered a separate image, or *slice*. For each pixel index within the (X, Y) extent of the image, we analyze the intensity value at this index for each slice of the 3D image. The pixel intensity displayed in the 2D image is the maximum intensity found this way.

For example, suppose the pixel (10,10) had an intensity value of zero for each slice of a 3D image. In this case, the pixel (10,10) would also have an intensity value of zero in the 2D output image. If even a single slice of the 3D image had a value of 255 at (10,10), then the corresponding pixel in the 2D image would also be set to 255.

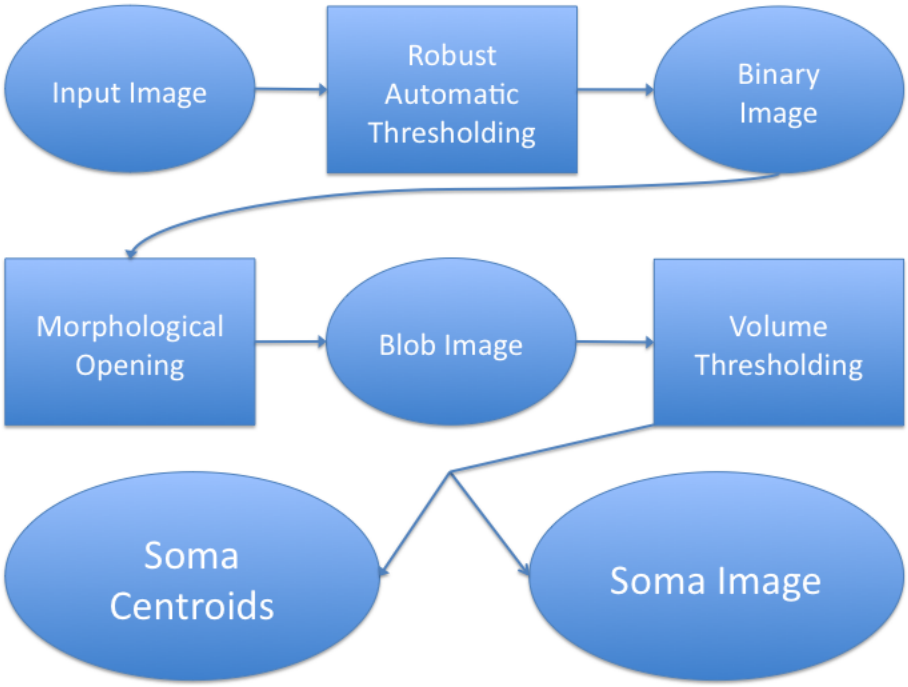


Figure 2.3: This flow chart shows each step of our soma segmentation procedure. Note that we acquire two outputs from this pipeline: a volumetric binary image depicting the somata, and a text file listing the locations of their centroids. These centroids will be used as the seed points for our process tracing step.

2.1.1.1 Robust Automatic Threshold Selection

The first step of isolating the somata is to separate the objects of interest from the image background. A common approach to this problem is to apply a threshold to the image. This yields a positive response for each pixel with an intensity value greater than the threshold. The output of this operation is a *binary image*. This means that each pixel has one of two possible logical values.

When using a threshold filter to isolate an image's foreground, the primary difficulty lies in selecting an appropriate value for the threshold. Set the threshold too high and important details will be lost. Set the threshold too low and the important details of the image will be obscured by noise from the image background.

This introduces a problem of *usability*. A tool becomes more difficult to use as the number of free parameters increases. For this reason, we use the Robust Automatic Threshold Selection (RATS) technique to calculate a good global threshold

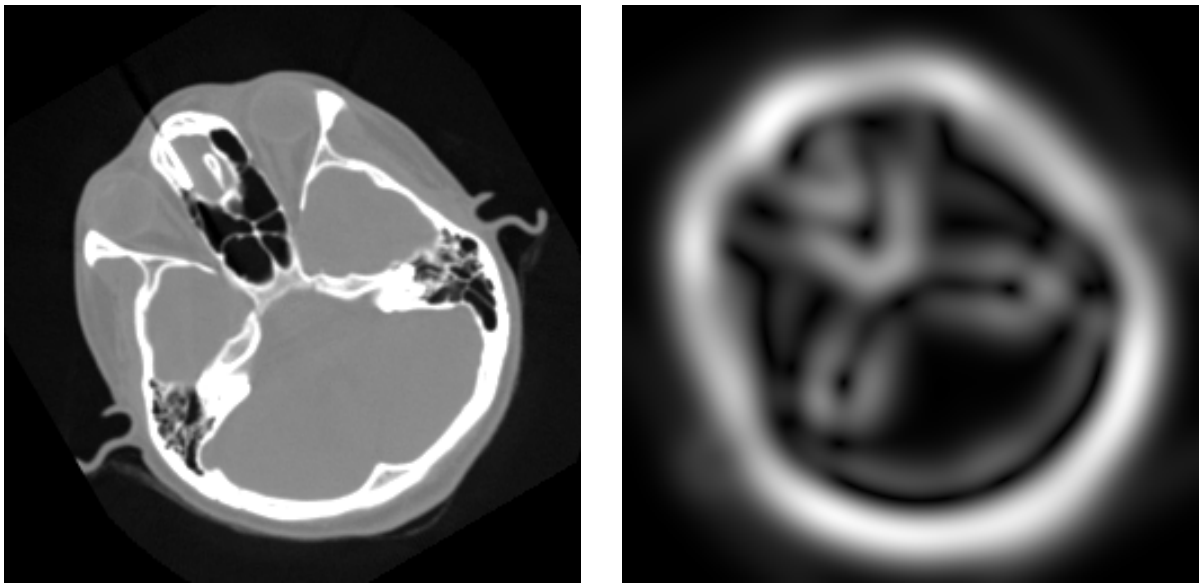


Figure 2.4: The left half of this figure shows a CT scan of a human head. The right image depicts the corresponding gradient magnitude image. Bright pixels in the gradient magnitude image indicate locations where the intensity of the original image is changing rapidly. The corresponding pixel intensity values in the original image are threshold candidates that are likely to be selected by RATS.

value for this image.

We can see the microglia in these images because they possess a protein known as ionized calcium binding adapter molecule 1 (Iba1). The antibody used to stain Iba1 has been found to demonstrate uniform labeling throughout a tissue slice [15]. If this were not the case, then an adaptive thresholding technique would be a more appropriate choice.

The RATS method uses the gradient of the image to calculate an appropriate value for a threshold whose purpose is to separate objects of interest from the image background. At each pixel location in an image, the gradient is a vector whose components are first partial derivatives of intensity with respect to position. The gradient of the 3D image I is defined in Equation 2.1.

$$\nabla I(x, y, z) = \frac{\partial}{\partial x} I\hat{x} + \frac{\partial}{\partial y} I\hat{y} + \frac{\partial}{\partial z} I\hat{z} \tag{2.1}$$

RATS uses the magnitude of the image gradient when computing an appro-

priate threshold value. The gradient magnitude is given in Equation 2.2.

$$\|\nabla I(x,y,z)\| = \sqrt{\left(\frac{\partial}{\partial x}I(x,y,z)\right)^2 + \left(\frac{\partial}{\partial y}I(x,y,z)\right)^2 + \left(\frac{\partial}{\partial z}I(x,y,z)\right)^2} \quad (2.2)$$

By calculating the image's gradient magnitude we obtain a numerical value for how rapidly the image intensity is changing with position at each pixel. As you can see in Equation 2.2, the gradient magnitude considers all directions when calculating intensity change for a particular pixel. In other words, the gradient magnitude considers only amplitude. It disregards the direction of fastest intensity change.

Image noise can be thought of as random variations of pixel intensity. Noisy pixels tend to have large gradients because they are unlikely to match the intensity of their neighbors. For this reason, it is customary to smooth the image before calculating its gradient.

A common smoothing technique is to convolve the image with a Gaussian kernel function. This smoothing process is described in Equation 2.3, and a 3D Gaussian kernel function is given by Equation 2.4.

$$I_{smooth}(x,y,z) = I(x,y,z) * G(x,y,z,\sigma) \quad (2.3)$$

$$G(x,y,z,\sigma) = \frac{1}{(\sqrt{2\pi}\sigma)^3} e^{-\frac{x^2+y^2+z^2}{2\sigma^2}} \quad (2.4)$$

As shown in Equation 2.5, we know that convolution is an associative operation [16].

$$f * (g * h) = (f * g) * h \quad (2.5)$$

Therefore, if we convolve our image with the first derivative of the Gaussian, this is equivalent to differentiating our image after smoothing it [17]. This is the approach that we use to compute the image gradient in Equation 2.6.

$$\|\nabla I(x,y,z)\| = I(x,y,z) * \left(\frac{\partial}{\partial x}G(x,\sigma)\right) * \left(\frac{\partial}{\partial y}G(y,\sigma)\right) * \left(\frac{\partial}{\partial z}G(z,\sigma)\right) \quad (2.6)$$

In Equation 2.6 we make use of the fact that the Gaussian kernel is *separable* [18]. A function $f(x,y)$ is considered separable if it can be expressed in the form $g(x)h(y)$. Equation 2.7 demonstrates the separability of our 3D Gaussian smoothing kernel.

$$G(x,y,z,\sigma) = \left(\frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{x^2}{2\sigma^2}}\right) \times \left(\frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{y^2}{2\sigma^2}}\right) \times \left(\frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{z^2}{2\sigma^2}}\right) \quad (2.7)$$

We use separability because it makes our Gaussian smoothing operation more efficient. For an isotropic 3D Gaussian kernel of width w, exploiting separability allows us to smooth an image pixel in $3w$ operations. By contrast, it would take w^3 operations to smooth this pixel if we directly used the 3D kernel to do so.

As noted in Figure 2.4, the fundamental idea underlying the RATS technique is that the threshold should be the pixel value where the image intensity changes most rapidly. These pixel locations correspond to brightness inflection points. This threshold is calculated by weighting the mean image intensity by the gradient image, as in Eq. 2.8.

$$T = \frac{\sum_{p\in D}\|\nabla I(p)\|^2 I(p)}{\sum_{p\in D}\|\nabla I(p)\|^2} \quad (2.8)$$

Here p indicates the index of a pixel in x,y,z space. D is the domain of the input image. Squaring the gradient magnitude places more emphasis on strong gradients. This yields better results on noisy images [7][19].

2.1.1.2 Morphological Opening

After isolating the objects of interest from the image background, our next step is to separate the blob-shaped somata from the long, narrow processes. In

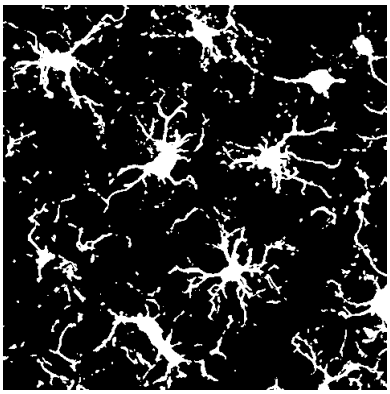


Figure 2.5: Example output from Robust Automatic Thresholding.

order to accomplish this task we use a morphological opening operation.

Opening is a technique from mathematical morphology that allows us to remove small objects from binary images. It makes use of a *structuring element*, which is simply a shape with a size.

In morphological terms, an opening consists of an erosion followed by a dilation. For a given image I and structuring element S, this is given by Eq. 2.9.

$$I \circ S = (I \ominus S) \oplus S \tag{2.9}$$

A more conceptual explanation follows. Imagine our structuring element moving around within the boundaries of a connected component of foreground pixels. Any pixel that the structuring element can reach remains within the foreground. A pixel is assigned to the background if no part of the structuring element can cover it without protruding out of the connected component. The only portions of a connected component that survive opening are those that can completely accommodate the structuring element.

For our purpose of process removal we use a simple spherical structuring element. This shape allows us to eliminate any portion of a connected component that is narrower than the diameter of the structuring element. The radius of this sphere is one of the important user-tunable parameters of this thesis. In order to make the tools presented here more widely useful, we have designed them to work in *physical units* (μm) instead of pixels whenever possible. Besides making it easier to select

appropriate values for parameters, this has the added benefit of allowing us to trace microglia from images of various magnifications.

For the images presented in this thesis, a structuring element with a radius of 1 μm was used. We testing this structuring element on approximately 2,000 microglia. We found it to eliminate all but the widest processes, while preserving even small or partial somata.

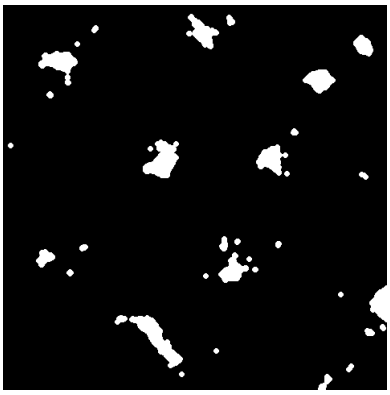


Figure 2.6: Example output from Morphological Opening.

2.1.1.3 Soma Pruning and Centroid Calculation

The next step in our soma isolation procedure is to examine the blobs that remain in our binary image after morphological opening has completed. Each contiguous set of pixels with a positive intensity value will be considered a separate *labeled object* [8]. Changing our focus from pixels to objects allows us to calculate useful information about each blob. The two attributes that we are interested in are volume and centroid. If a blob's volume is less than some threshold, we assume that it is not a soma, and we remove it from our final soma segmentation results. The centroids will be used as seed points for the tracing of the microglia processes.

These labeled objects are represented using *run length encoding* (RLE). This means that each separate labeled object consists of a set of lines. Each such line is fully described by a starting position and a length in the x dimension.

Given this representation, we can calculate the volume v of a labeled object by multiplying the number of voxels in the object by the size of a voxel in this image. This procedure is defined in Eqs. 2.10 - 2.12. Here L represents the set of lines that

compose a labeled object. $|l|$ denotes the length of a particular line l. $\vec{S}$ is a vector whose components represent the length of a voxel in each dimension. We use a vector to represent voxel size because the images captured by confocal microscopes are typically anisotropic. More specifically, the z step tends to cover more physical space than the x and y steps.

$$n = \sum_{l \in L} |l| \tag{2.10}$$

$$p = S_x \times S_y \times S_z \tag{2.11}$$

$$v = n \times p \tag{2.12}$$

Once we've determined the volume of a labeled object, we can use Eqs. 2.13-2.16 to calculate its centroid c. In the equations below, the start point of a line l is referred to as (l_x, l_y, l_z).

$$c'_x = \frac{\sum_{l \in L} |l| + (|l| \times (|l| - 1))/2}{n} \tag{2.13}$$

$$c'_y = \frac{\sum_{l \in L} |l| \times l_y}{n} \tag{2.14}$$

$$c'_z = \frac{\sum_{l \in L} |l| \times l_z}{n} \tag{2.15}$$

$$c = \frac{c'}{\vec{S}} \tag{2.16}$$

As before, the numerical parameter to this step is specified in physical units (μm) as opposed to pixels. From the images that we've examined, we've found the volume of an average soma to be around 300 - 400 μm^3. In order to capture somata that only partially appear in the image, we set this threshold a bit lower. For the results presented in this thesis, a volume threshold between 50 and 100 μm^3 did well at eliminating any spurious blobs while capturing all of the somata. It is important to note that this volume threshold parameter is partially dependent on the radius of the structuring element that we used during the morphological opening step. As the

size of the structuring element used for opening increases, the volume of the blobs that remain decreases. Once this thresholding is complete, each remaining object is assumed to represent a separate soma in the original image. At this point, we record the centroids of all the remaining blobs. These will be used as *seed points* during the tracing of the microglia processes.

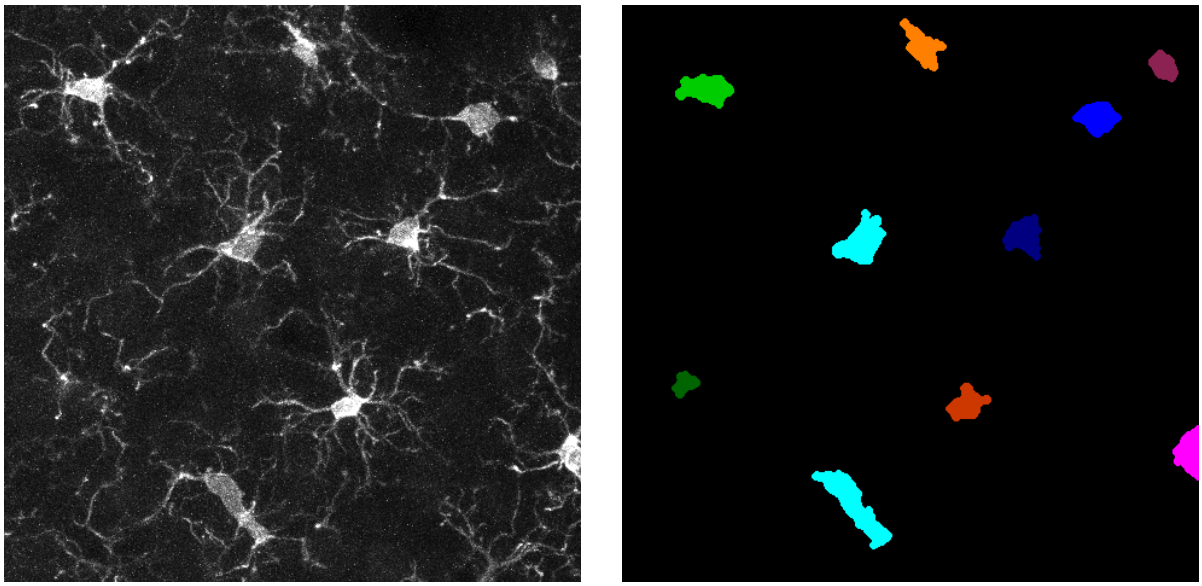


Figure 2.7: Soma segmentation results.

2.1.2 Processes

Now that we've successfully isolated the somata of the microglia, we turn our attention to the *processes*. These are the long, narrow, twisting appendages that emanate from the somata.

What follows is a brief overview of this portion of the tracing procedure.

1. Identify points of interest. Since we intend to isolate the centerlines of long, narrow objects, we will use a *ridge detection* algorithm for this purpose.

2. Generate tree structures from these points. Each microglial cell will be modeled by a separate tree.

3. Prune and interpolate these trees. This is done to smooth the traces, remove spurious branches, and reduce the system resources needed to model these

cells.

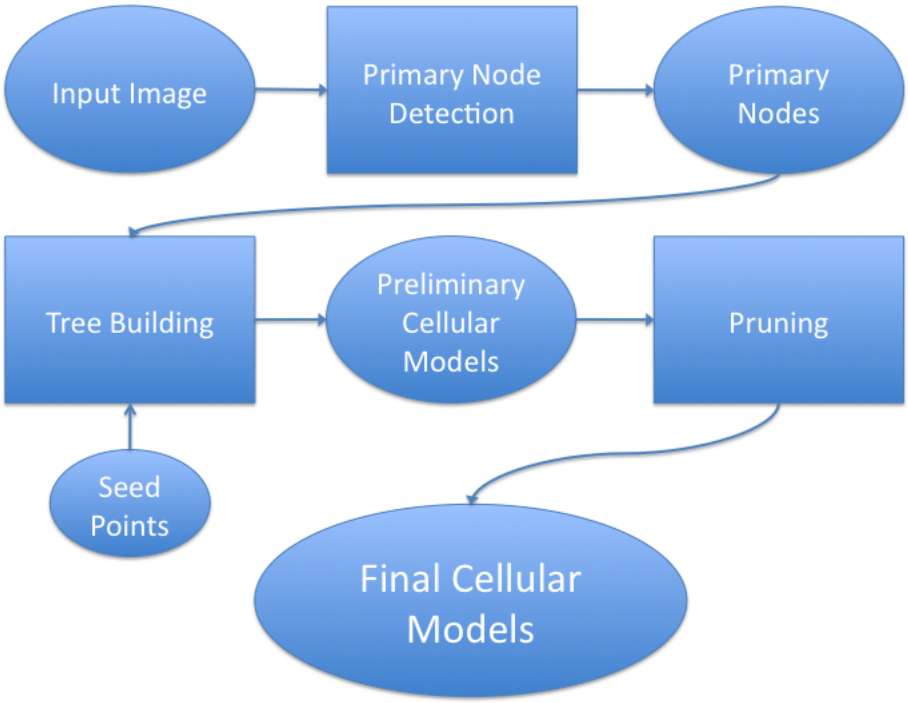


Figure 2.8: Here we depict the steps that make up our process tracing pipeline. At the conclusion of the procedure we obtain a morphological model for each microglia in the input image.

2.1.2.1 Primary Node Detection

Our goal for this step of the tracing process is to isolate pixels that lie on the centerline of the processes of the microglia in our input image. We refer to centerline pixels as *primary nodes*.

Once we have detected the primary nodes, we will connect them into tree-shaped structures. These trees will serve as our morphological models of the microglia cells.

In order to detect the primary nodes, we make use of an adaptation of a technique originally developed by Amit Mukherjee for segmenting neurons. Since this work is unpublished, we describe it in detail here.

The primary node detection procedure begins by computing the inverted Laplacian of Gaussian (LoG) of our input image. This is accomplished by convolving the

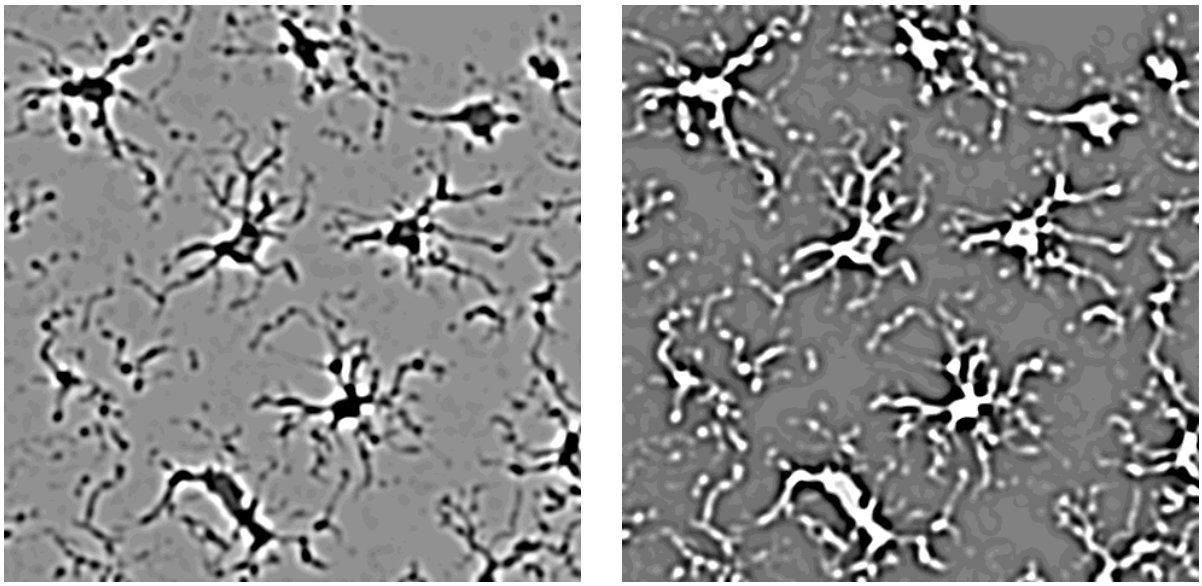


Figure 2.9: The Laplacian of Gaussian (LoG) of our input image when $\sigma = 1\ \mu m$. The two images depict the LoG response before and after inversion, respectively. We work with the inverted LoG image because we are interested in isolating points that lie along the bright processes of the microglia. The inverted LoG image exhibits a positive response along the processes.

image with the second derivative of a Gaussian (Eq. 2.17), and then inverting the results.

The Laplacian operator gives us information about the second derivative of image intensity with respect to space. It exhibits a strongly negative response for bright blobs of radius $= \sqrt{2}\sigma$. If we choose our values of σ carefully, this property will make it useful for detecting points near the centerlines of the microglia processes. We invert the LoG image so that it yields a strongly positive response at these points that we're interested in. This inversion step is also important later when we approximate the Hessian matrix for our input image.

$$LoG_\sigma = \Delta G_\sigma(x, y, z) = C\left(\frac{x^2 + y^2 + z^2}{\sigma^4} - 3\frac{1}{\sigma^2}\right)\exp\left(-\frac{x^2 + y^2 + z^2}{2\sigma^2}\right) \qquad (2.17)$$

C is a scale-invariant normalization coefficient.

As we've already hinted, the responses of the LoG operator are very scale

dependent [9]. For this reason, we compute the LoG response over a logarithmic range of scales.

$$\sigma = 2^p \text{ where } p = \{0, 0.25, 0.5, 0.75, ..., 1.25\} \tag{2.18}$$

As before, these numerical values are specified in terms of μm, not pixels. This range of scales was chosen to reflect the size of microglia processes. It would have to be changed to detect a different type of structure, but for a particular application it should be more or less static. After analyzing images of approximately 30 tissue samples captured at varying magnification levels, we have discovered that microglia processes can only be adequately imaged at 50x magnification or greater. Of the 50x images that we analyzed, the largest Z step of any of these images was approximately 0.5 μm per pixel. Hence our finest scale $\sigma = 1$ μm will correspond to a minimum of two pixels when it is discretized. For this reason, we are confident that that our LoG operator will provide meaningful responses, even at the smallest scale on the lowest magnification images.

Any pixel that survives the primary node detection procedure (outlined below) will be passed along to the tree building step, regardless of what scale it was detected at. For the remainder of this section $LoG(x, y, z, \sigma)$ will denote the intensity of the inverted Laplacian of Gaussian image at pixel location (x, y, z, σ). σ may be excluded from these coordinates for operations that are performed across scales.

Neighborhood Test

Consider a ridge of bright pixels passing through the center of a 3x3x3 neighborhood. In this case, the center pixel and two of its diametrically opposing neighbors will be brighter than the rest of the neighborhood. Intuitively, this is the motivation behind the neighborhood test.

We compute the maximum response from the inverted LoG filter for each pair of diametrically opposing pixels within the neighborhood. See Table 2.1 for an illustration of what pixels correspond to one another in a 3x3x3 neighborhood. We then compute a threshold by taking the average of these maximum responses (Eq. 2.19). If the center pixel's response is stronger than this threshold, it passes

A	B	C	J	K	L	I'	H'	G'
D	E	F	M	*	M'	F'	E'	D'
G	H	I	L'	K'	J'	C'	B'	A'
bottom			*middle*			*top*		

Table 2.1: Diametrically opposing pixels in a 3x3x3 neighborhood. Pixels A and A' are diametrically opposed to one another.

the neighborhood test. At this point, it is a candidate pixel.

$$\tau = \frac{\sum_{p \in pairs} max(LoG(p_1), LoG(p_2)}{card(pairs)} \tag{2.19}$$

Structure Test

The Hessian is similar to the Laplacian in that both give us second derivative information about our image. The difference is that the Laplacian gives us a scalar value at each pixel, while the Hessian returns a matrix of both mixed and unmixed second partial derivatives. This gives us more information about how image intensity is changing with respect to direction.

The Hessian matrix for a 3D image I is defined in as Eq. 2.20.

$$H = \begin{bmatrix} \frac{\partial^2 I}{\partial x^2} & \frac{\partial^2 I}{\partial x \partial y} & \frac{\partial^2 I}{\partial x \partial z} \\ \frac{\partial^2 I}{\partial y \partial x} & \frac{\partial^2 I}{\partial^2 y} & \frac{\partial^2 I}{\partial y \partial z} \\ \frac{\partial^2 I}{\partial z \partial x} & \frac{\partial^2 I}{\partial z \partial y} & \frac{\partial^2 I}{\partial z^2} \end{bmatrix} \tag{2.20}$$

We approximate these second order partial derivatives using the Laplacian that we calculated previously. This is demonstrated in Eqs. 2.21 - 2.26. Here the location of the candidate pixel is given as (x, y, z).

$$\frac{\partial^2 I}{\partial x^2} \approx LoG(x+2, y, z) + LoG(x-2, y, z) - 2 \times LoG(x, y, z) \tag{2.21}$$

$$\frac{\partial^2 I}{\partial y^2} \approx LoG(x, y+2, z) + LoG(x, y-2, z) - 2 \times LoG(x, y, z) \tag{2.22}$$

$$\frac{\partial^2 I}{\partial z^2} \approx LoG(x, y, z+2) + LoG(x, y, z-2) - 2 \times LoG(x, y, z) \tag{2.23}$$

$$\begin{aligned} \frac{\partial^2 I}{\partial x \partial y} = \frac{\partial^2 I}{\partial y \partial x} \approx\ & LoG(x+1, y+1, z) + LoG(x-1, y-1, z) \\ & -LoG(x-1, y+1, z) - LoG(x+1, y-1, z) \end{aligned} \tag{2.24}$$

$$\begin{aligned} \frac{\partial^2 I}{\partial x \partial z} = \frac{\partial^2 I}{\partial z \partial x} \approx\ & LoG(x+1, y, z+1) + LoG(x-1, y, z-1) \\ & -LoG(x-1, y, z+1) - LoG(x+1, y, z-1) \end{aligned} \tag{2.25}$$

$$\frac{\partial^2 I}{\partial y \partial z} = \frac{\partial^2 I}{\partial z \partial y} \approx LoG(x, y+1, z+1) + LoG(x, y-1, z-1) \\ -LoG(x, y-1, z+1) - LoG(x, y+1, z-1) \tag{2.26}$$

In Equations 2.21 - 2.23 we apply the simple convolution mask $\begin{bmatrix} 1 & -2 & 1 \end{bmatrix}$ to each dimension of our inverted LoG image. $\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$ is applied to each pair of dimensions to approximate the mixed partial 2nd derivatives of the inverted LoG image in Equations 2.24 - 2.26.

Equations 2.21 - 2.26 are actually computing 4th derivative information about our input image. Recall that our inverted LoG image exhibits strong positive values for bright blobs with a radius of $\sqrt{2}\sigma$. This implies that tubes with a radius of $\sqrt{2}\sigma$ are preserved in the inverted LoG image, while most other structures in the image are suppressed. For this reason, the inverted LoG image serves as a reasonable substitute for the smoothed input image in our regions of interest (bright vessels). You can verify this by comparing Figures 2.1a and 2.9b.

This property of our inverted LoG image allows the values calculated in Equations 2.21 - 2.26 to serve as a useful approximation for the components of the Hessian matrix. We have found this unorthodox method of approximating the Hessian to require roughly half as much computation time as a more traditional computation of the Hessian involving separated Gaussian kernels.

Frangi *et al.* demonstrated how to use the eigenvalues of the Hessian matrix to detect and enhance vessels (tubes) in an image [20]. We denote the three eigenvalues of the Hessian matrix as $\lambda_1, \lambda_2, \lambda_3$ such that $|\lambda_1| < |\lambda_2| < |\lambda_3|$.

Recall that the Laplacian of Gaussian provides a measure of the contrast between the regions inside and outside the radius of σ. The eigenvalues give us information about this contrast in each of three principal directions (the eigenvectors).

For tubes, λ_1 has a magnitude ideally ≈ 0. This corresponds to the direction of least intensity change (along the tube). λ_2 and λ_3 should have high magnitudes, because they represent the two dimensions of the cross section of the vessel. The sign of λ_2 and λ_3 depends on the whether the vessels are bright or dark.

We are looking for bright tubes. This means that we are interested in points

that have a low magnitude for λ_1 and large negatives values for λ_2 and λ_3. After computing the eigenvalues of the Hessian matrix, we use Eq. 2.27 to calculate a *vesselness score* for each candidate pixel.

$$score = -(\lambda_2 + \lambda_3) - |\lambda_1| \tag{2.27}$$

Non-Maximum Suppression

Now that we've assigned a score to this candidate pixel, we search its neighborhood for other candidates. This neighborhood is defined by the Euclidean distance threshold r_{avg}. This threshold represents the radius of an average microglia process.

We eliminate all candidates found within this neighborhood except for the one with the highest score. For the results presented in this paper, we used a value of 0.3 μm for r_{avg}. The purpose of this step is to ensure that our primary nodes lie along the centerlines of the processes.

Summary of Primary Node Detection

We perform the neighborhood test for each pixel in the LoG image. Those that pass are assigned a vesselness score from the structure test. We then perform non-maximum suppression on this pixel's neighborhood, so only the pixel with the highest vesselness score in the neighborhood remains.

This whole procedure is repeated across the range of scales given in Eq. 2.18. Once the entire range of scales has been analyzed, the pixels that remain are our set of *primary nodes*. At this point, we set out to determine how to connect these primary nodes into tree-shaped traces of microglia.

2.1.2.2 Graph Construction

Now that we've generated a set of nodes that lie along the processes of the microglia, our next task is to determine how they are connected. To address this problem, we use Prim's algorithm to construct a minimum spanning tree (MST) to model each microglia [10]. We use a tree because it can efficiently model the morphology of a microglial cell. This is true because a microglial cell consists of many branching processes, each of which emanates from a central soma. This is

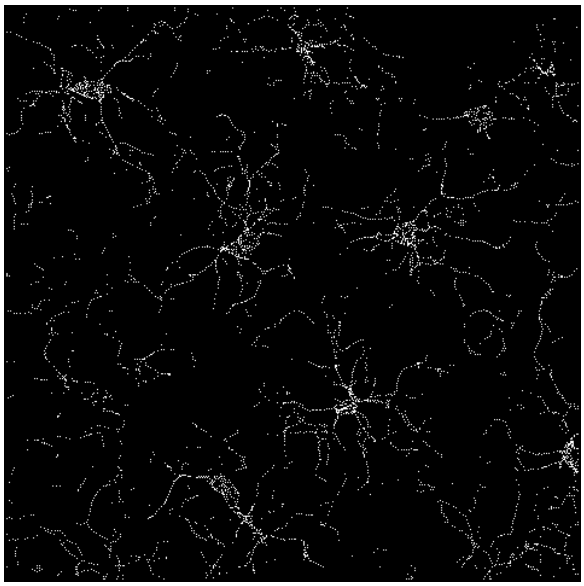


Figure 2.10: Example output from Primary Node Detection. This is a maximum intensity projection of a binary image. Each bright pixel is a Primary Node. We create morphological models of microglial cells by linking these bright nodes together.

identical to the structure of a tree: many branches emerging from a single root. Once tracing is complete we will be left with a forest, where each tree models the morphology of a single microglial cell.

Prim's algorithm takes as input a weighted, connected graph. All graphs consist of nodes and edges. In our case, each node is either a *primary node* (detected previously) or a *soma centroid*. These are pixels that are believed to represent some important morphological detail in our 3D input image of microglia. The edges of our input graph represent connectivity between nodes. Here we define the Euclidean distance threshold d_{max} as the maximum allowable distance between two connected nodes. We construct an edge between nodes p and q if the Euclidean distance between them is less than d_{max}. The weight of the edge (p, q) is equal to the Euclidean distance between these nodes.

Prim's algorithm constructs a MST by iteratively selecting the edge (p, q) with minimal weight such that p is a member of the MST and q is not. q is then added to the MST as a child of p. This procedure is completed when all of the nodes have been incorporating into the tree.

Our implementation of Prim's algorithm is laid out in Algorithm 1. It differs from the traditional algorithm in two ways. The first difference is that we are constructing multiple trees simultaneously from a common pool of nodes. The second difference is that we will not necessarily add all of the nodes to a tree before tracing is completed. This is a safety mechanism to prevent distant pixels from being erroneously incorporated into our models. We accomplish this by using d_{max}: a distance threshold measured in μm. Any node that is more distant to the nearest tree than this threshold will not be included in the tracing results. d_{max} is discussed in greater detail below.

In order to accomplish our goal of MST construction, we maintain two lists of nodes. The first is the *closed list*. This contains all the nodes that have already been added to a tree. Initially, it only contains the soma centroid nodes. The other list is the *open list*. This contains all of the nodes that are still awaiting assignment to a tree. This list is initialized to the list of primary nodes.

Algorithm 1 Build Trees

Require: $open \neq \emptyset$, $closed \neq \emptyset$, $adjMap \neq \emptyset$
  while $open.empty()$ =FALSE **do**
    $(parent, child) \leftarrow ClosestOpenNode()$ {Algorithm 3}
    if $parent = \varnothing$ or $child = \varnothing$ **then**
      break
    end if
    $child.parent = parent$
    $parent.children.push\_back(child)$
    $open.remove(child)$
    $closed.push\_back(child)$
  end while

In order to follow Prim's algorithm, we need to be able to find the node from the open list that has the shortest distance to a node in the closed list.

$$\arg\min_{closed,open} distance(closed, open) \tag{2.28}$$

The images presented in this thesis contain nodes numbering in the thousands to hundreds of thousands. If we're not careful, finding the closest open node at each iteration could be a very expensive operation. A naïve approach would have us

computing the distance between each open node and each closed node during each iteration.

Instead of doing this, we make use of an *adjacency map* so that these distances need only be calculated at the outset. This data structure allows us to look up a list of neighbors (sorted by distance) for a particular node. At the outset of tree construction, we compute the adjacencies for the soma centroids and the primary nodes.

Algorithm 2 Compute Adjacency Map For Nodes

Require: $srcNodes \neq \emptyset$
  $searchRegion.radius = maxDistance * 2$ {Double radius for scaled distances}
  for all $src \in srcNodes$ **do**
    $searchRegion.center = src.index$
    $neighbors = \emptyset$ {List of neighbors sorted by distance. Begins empty.}
    for all $dst \in searchRegion$ **do** {For each node in the search region}
      $d_{scaled} = scaledDistance(src, dst)$ {See Algorithm 4.}
      if $d_{scaled} > maxDistance$ **then**
        continue
      end if
      $d = distance(src, dst)$
      $neighbors.push\_front(d, dst)$
    end for
    $neighbors.sort()$
    $adjMap(src) = neighbors$
  end for

As an implementation detail, we generate a binary image where the only foreground pixels correspond to the primary nodes. This allows us to search for neighboring nodes by iterating over a region of the image. This saves us from having to compute the distance between two nodes just to discover that they are distant from one another.

As mentioned earlier, we use the distance threshold d_{max} to preclude distant nodes from being erroneously added to a model of a microglial cell. This threshold is used here to determine the size of the region in which we will search for nearby nodes. As you can imagine, the value that we choose for this parameter has a dramatic impact on the models that we generate. If the threshold is set too low then the models will be too small because of missed branches and connections. On

the other hand, if the threshold is set too high, then our models will erroneously connect to structures outside the cell.

Algorithm 3 Find Closest Open Node

Require: $open \neq \emptyset$, $closed \neq \emptyset$, $adjMap \neq \emptyset$
  $bestDistance \leftarrow \infty$
  $bestParent \leftarrow \varnothing$
  $bestChild \leftarrow \varnothing$
  for all $c \in closed$ **do**
    $neighbors = adjMap(c)$ {$neighbors$ is sorted by distance}
    $openNeighborFound \leftarrow$FALSE
    while $openNeighborFound$ =FALSE and $neighbors.empty()$ =FALSE **do**
      $(nbr, dist) = neighbors.front()$
      if $nbr \in closed$ **then**
        $neighbors.remove(nbr)$ {Some other node was closer to this neighbor and already added it to a tree.}
      else
        $openNeighborFound \leftarrow$TRUE
        if $dist < bestDistance$ **then**
          $bestParent \leftarrow c$
          $bestChild \leftarrow nbr$
        end if
      end if
    end while
  end for
  return $(bestParent, bestChild)$

In order to reduce the sensitivity of this parameter, we examine the pixels that lie between the two nodes that we're comparing. Our intuition here is that connecting across the background should be more expensive than connecting through foreground pixels. Thus we will calculate a *scaled distance* between nodes A and B. This distance will be less than the true Euclidean distance if the gap between A and B includes foreground pixels. We use the results of Robust Automatic Threshold Selection filtering (discussed previously) to determine whether or not a pixel belongs to the foreground.

Our scaled distance calculation begins by drawing a line between Node A and Node B. Each voxel on this line is subject to 50% reduction in size if it is one pixel away (or less) from a foreground point. This gives us the ability to connect nodes that are more distant from each other than our threshold would otherwise allow.

The motivation behind this technique is to widen gaps of background pixels, while shrinking distances that span regions of interest. This distance scaling procedure is further detailed in Algorithm 4.

Algorithm 4 Scaled distance from Node A to Node B

```
Draw a line from Node A to Node B (exclusive)
distance ← 0.0
for all pixel p on line do
  scale ← 1.0
  for all pixel p_0 in p's 3x3x3 neighborhood do
    if p_0 is foreground then
      scale ← 0.5
      break
    end if
  end for
  distance = distance + (pixelSize * scale) {pixelSize depends on offset from
  p − 1 to p because z steps are larger than x or y steps.}
end for
```

This scaled distance is only used to determine whether or not the connection between two nodes exceeds the distance threshold. Euclidean distance is still the metric used for the purpose of determining which node to add to a tree next. In most cases, using the scaled distance for tree construction would produce very similar results to those obtained by using the Euclidean distance. Occasionally the scaled distance tries too hard to avoid background pixels. This produces connections that do not closely match the underlying biological structure. This is particularly true when certain cellular regions are not captured well by our imaging techniques. In this case, regions that actually correspond to a portion of a microglia will be detected as background pixels by our segmentation algorithms. Under such circumstances, forming a connection across background pixels is the correct course of action. Generally speaking, the difference between using the scaled distance and the Euclidean distance for tree construction is very minor.

2.1.2.3 Radius Estimation

Next we calculate a radius estimate for each node in our MSTs. For a given node n, we initialize its radius estimate to 2 pixels. We then use an iterative pro-

cedure that allows our radius estimate to converge. The first step of the iterative procedure is to calculate the extent of our search space:

$$searchRadius = \max(2r, 5) \tag{2.29}$$

If we account for the center pixel in the neighborhood, Equation 2.29 implies that the minimum size of our search space is an 11 x 11 x 11 pixel neighborhood. This neighborhood is a subset of the input image centered on node n. For each pixel p in the neighborhood, we analyze its distance from n and its intensity value.

We separate the search neighborhood into two sets: *in* and *out*. A pixel is placed into the *in* set if it is closer to n than the current radius estimate. Otherwise it is placed in the *out* set. Once the search neighborhood has been partitioned into these two sets, we calculate two terms: I_{in} and I_{out}. These are the average intensity values for pixels that belong to these sets.

$$I_{in} = \frac{\sum I(p \in in)}{size(in)},\ I_{out} = \frac{\sum I(p \in out)}{size(out)} \tag{2.30}$$

While analyzing the search neighborhood, we also generate a set of points called B. This set represents the boundary points of our current radius estimate. It consists of any point that falls within 0.7 pixels of our current radius estimate.

$$|dist(n, p) - r| < 0.7 \Rightarrow B.push(p) \tag{2.31}$$

Using I_{in}, I_{out}, and B, we update our radius estimate using Equations 2.32 - 2.36.

$$\Delta r \leftarrow \frac{\sum_{b \in B} |I(b) - I_{in}| - |I(b) - I_{out}|}{size(B)} \tag{2.32}$$

$$\Delta r \leftarrow max(\Delta r, -1) \tag{2.33}$$

$$\Delta r \leftarrow min(\Delta r, 1) \tag{2.34}$$

$$r \leftarrow r - \Delta r \tag{2.35}$$

$$r \leftarrow max(r, 1) \tag{2.36}$$

2.1.2.4 Pruning near the Somata

As you can see in Fig. 2.10 we detect many pixels in and around the somata as primary nodes. If we don't account for these nodes they may cause inaccuracies in our cellular models.

A simple approach would be to mask away the somata from the image before we detect the primary nodes. Unfortunately, this technique does not work well for us because we tend to underestimate the volumes of the somata. This underestimation is a side-effect of our use of the morphological opening operation to remove processes in Section 2.1.1.2.

Our current goal is to clean up the traces around the somata. Doing so will allow us to compute meaningful features for the somata. Once we complete this step, we will have generated cellular models for the microglia in our image. Our somata pruning procedure consists of two steps:

1. Remove all nodes that fall within a soma except for those that are structurally significant (Algorithm 5).

2. Remove short, unforked branches emanating from a soma (Algorithm 6).

There are two types of nodes within a soma that we consider *structurally significant*. The centroid of the soma is one type. The nodes on the boundary of the soma are the other type. Boundary nodes are the parent of at least one node that falls outside the soma. Once we perform Algorithm 5 these two types will be the only nodes that fall within our estimated extent of the soma.

Because we underestimate the true volume of a soma, it is likely that we will be left with short branches that just barely protrude out of the border of a soma. Algorithm 6 was designed to solve this problem. It examines each branch emanating from a soma. Branches consisting of few nodes and no forks are removed.

Algorithm 5 Remove Intra-Somal Nodes

```
for all soma ∈ somata do
  searchRegion = soma.GetBoundingBox()
  for all n ∈ searchRegion do {For each primary node within the soma's bounds}
    keepThisNode ←FALSE
    for all child ∈ n.children do
      if child ∉ searchRegion then {We keep node n if it is the parent of a node outside the soma}
        keepThisNode ←TRUE
        break
      end if
    end for
    if keepThisNode =FALSE then
      delete n
    end if
  end for
end for
```

Algorithm 6 Prune Soma Branches

```
Require: branches ≠ ∅
for all branch ∈ branches do
  if branch.hasForks() = FALSE and branch.depth < 10 then
    delete branch
  end if
end for
```

2.2 Feature Extraction

At the end of the segmentation process, we are left with a separate tree model for each microglia in the input image. These models are stored in the .swc file format. This is a simple file format that allows us to describe the structure of a brain cell. Each line in the file describes a node, which represents a structurally significant point on a microglial cell.

Each node contains the following information:

- ID number. This is a unique identifier that allows us to unambiguously refer to this node.
- Type of node. In our case, this is either soma or process.

- Position: The (X,Y,Z) coordinates of the node's location in the image.
- Radius: The half-width of the underlying cellular structure at this point.
- Parent node. This provides information on how the nodes are connected together. For the nodes corresponding to soma centroids, the parent is denoted as -1.

One benefit from storing our models this way is that it allows us to use L-Measure (LM) [11]. This is a freely available software tool for quantitative characterization of neuronal morphology. Microglia and neurons have similar morphological structures. Both types of cells consist of branching appendages emerging from a bulbous cell body. This means that we can use LM to generate an extensive set of features for each microglial cell.

A full description of the features generated by LM is available in Appendix A.

2.3 Classification

LM has the benefit of providing us with a lot of information about the cells that we are studying. This additional information allows us to search for similarities and differences among the microglia. One such use of these features is to distinguish between different classes of microglia.

One class of cells that we are interested in is the *unactivated cells*. These microglia are found in healthy brain tissue. A second class is called the *activated cells*. These are microglia that are responding to a recent injury or infection. The activated microglia used in this project were found in samples of brain tissue where a neuro-prosthetic device was recently implanted.

To distinguish between these two types of cells, we use the *Kernel Partial Least Squares* (K-PLS) method [12][13][14].

2.3.1 Partial Least Squares

For the purpose of classification, we will refer to the myriad features provided by LM as *predictor variables*. Each cell will produce one vector of predictor variables,

denoted as x. X refers to the $n \times m$ matrix consisting of all the predictor variables for a set of n cells. m represents the number of features returned by LM for each cell.

The class that a cell belongs to will be referred to as the *target variable*. In the equations below, a single cell's class will be represented as y. Y is a column vector where each element indicates the class of the cell represented by the corresponding row of X.

Partial least squares (PLS) is a statistical method that allows us to compute a linear regression model of the relationship between the predictor variables and the target variables. PLS shares similarities with *Principal Component Regression* (PCR). Both techniques first reduce the dimensionality of the predictor data, and then compute the regression function by minimizing the sum of squared errors. PLS differs from PCR in that it incorporates both the predictor variables and the target variables when performing dimensionality reduction [12].

Before linear fitting is performed, PLS projects the predictor and target variables to a *latent variable* space. This allows us to find the direction in the original predictor variable space that accounts for the maximum variance in the target variable space. Linear regression assumes that there is a linear relationship between the predictor variables x and the corresponding target variable y. In matrix form, this relationship is given by the following equation:

$$Y = X\beta + \epsilon \tag{2.37}$$

Here β represents the regression coefficients. This controls the linear model that fits X to Y. ϵ is an error term that accounts for deviations from the linear model.

The use of a latent variable space reduces the dimensionality of our data. This makes it easier to understand, visualize, and partition. Latent variables are not directly observed or measured. Instead, they are inferred or calculated from observed variables.

PCR and PLS both generate latent variables by linearly combining the observed data. This is done in such a way that there is no correlation between the

latent variables that are used to generate the predictive linear model [21]. The relationship between the observed variables and the latent variables is given by the following equations:

$$X = TP + E \tag{2.38}$$

$$Y = TQ + F \tag{2.39}$$

Here T is a matrix of extracted latent variables. P and Q are *loading matrices* that indicate the relationship between the original data and the latent variables. E and F are error terms that account for data that is not explained by the latent variables.

This process projects the predictor variables of each sample cell x_i into a latent vector t_i. The factor score matrix T is extracted from X through the use of a weight matrix W:

$$T = XW \tag{2.40}$$

This is where PLS and PCR differ. PCR computes W so that T captures the maximum variance in X. PLS, on the other hand, computes W so that T maximizes the covariance between X and Y. In other words, PLS takes the response variables Y into account when performing dimensionality reduction, whereas PCR does not. W and P are computed using the Non-linear iterative partial least squares (NIPALS) algorithm. This is presented in Algorithm 7. In this algorithm the input variable H refers to the number of latent variables. We used a value of $H = 5$ to generate the classifier described in this thesis.

To compute the loading matrix Q from Eq. 2.39, we use ordinary least squares to compute the regression between T and Y. This results in the regression coefficients Q that satisfy $Y = TQ + F$. After Q has been calculated we have obtained our prediction model:

$$Y = XB + \epsilon \tag{2.41}$$

$$B = WQ \tag{2.42}$$

Algorithm 7 NIPALS

Require: X (predictor variables), Y (target variables), H (number of latent variables)
  $A \leftarrow X^T Y$
  $M \leftarrow X^T X$
  $C \leftarrow I$
  for $h = 1 \rightarrow H$ **do**
    $q \leftarrow$ the dominant eigenvector of $A^T A$
    $w_0 \leftarrow CAq$
    $w \leftarrow \frac{w_0}{\|w_o\|}$
    h^{th} column of W $\leftarrow w$
    $p_0 \leftarrow Mw$
    $c = w^T Mw$
    $p \leftarrow \frac{p_0}{c}$
    h^{th} column of P $\leftarrow p$
    $A \leftarrow A - cpq^T$ {Compute residuals}
    $M \leftarrow M - cpp^T$
    $C \leftarrow C - wp^T$
  end for
  return W,P

2.3.2 Kernel Partial Least Squares

Kernel Partial Least Squares (K-PLS) is an extension of the *Partial Least Squares* (PLS) method of linear regression. The idea behind K-PLS is as follows. If we map the original X space to a higher-dimensional feature space F before performing linear regression, this will allow us to obtain a nonlinear regression model in the original X space.

In practice, we can exploit PLS' use of inner products and employ the "kernel trick" [14]. This allows us to generate nonlinear regression models while avoiding explicit nonlinear mapping.

We use K-PLS instead of other classification techniques because it has been

shown to perform well in situations where the input variables are highly correlated [13]. As you can tell from examining the LM features in Appendix A, this is certainly the case for our data. K-PLS is also an appropriate choice for classification problems where the number of input variables exceeds the number of observations. We occasionally encounter this situation because LM provides multiple values for some features. If we take all of its information into account, LM can provide upwards of 100 separate numerical values for an individual cell. Hence, this property of K-PLS may also be useful for our experiments, depending on how many cells are being classified at once.

Parameter	Description & Purpose	Thesis Value	Suggested Range
σ	Standard Deviation of the Gaussian kernel used to smooth our image during Robust Automatic Threshold selection (RATS). This occurs during soma segmentation. W do not need to worry about obscuring fine details because the somata are relatively large when compared to the other structures in the image.	2 μm	1 - 4 μm
r_{max}	Maximum Process Radius. r_{max} is the radius of the structuring element used during the morphological opening step of soma segmentation. Our goal during this step is to eliminate the narrow processes, leaving behind only blob-shaped regions in the image (including the somata).	1 μm	$r_{max} < \sigma$
v_{min}	Minimum Soma Volume. This is used during soma pruning, the final step of the soma segmentation procedure. We only consider blobs larger than this value as somata; all others are discarded. The value of v_{min} should be inversely proportional to r_{max}.	75 μm	40 - 150μm, based on r_{max}
r_{avg}	Typical Process Radius. r_{avg} is used twice during the process tracing procedure. First, it determines the size of the neighborhood during non-maximum suppression. r_{avg} also serves as σ when we use RATS to generate a binary image. This binary image is used to compute scaled distances between nodes (Algorithm 4).	0.3μm	0.1 - 1μm
d_{max}	Maximum Connection Length. d_{max} comes into play during the tree building step of process segmentation. d_{max} represents the maximum distance a connection between a tree and a node can span. In other words, d_{max} controls the maximum allowable distance between connected nodes.	4.0μm	2 - 6μm

Table 2.2: Information about the user-tunable parameters of the tracing system presented in this thesis.

3. Results

3.1 Tracing

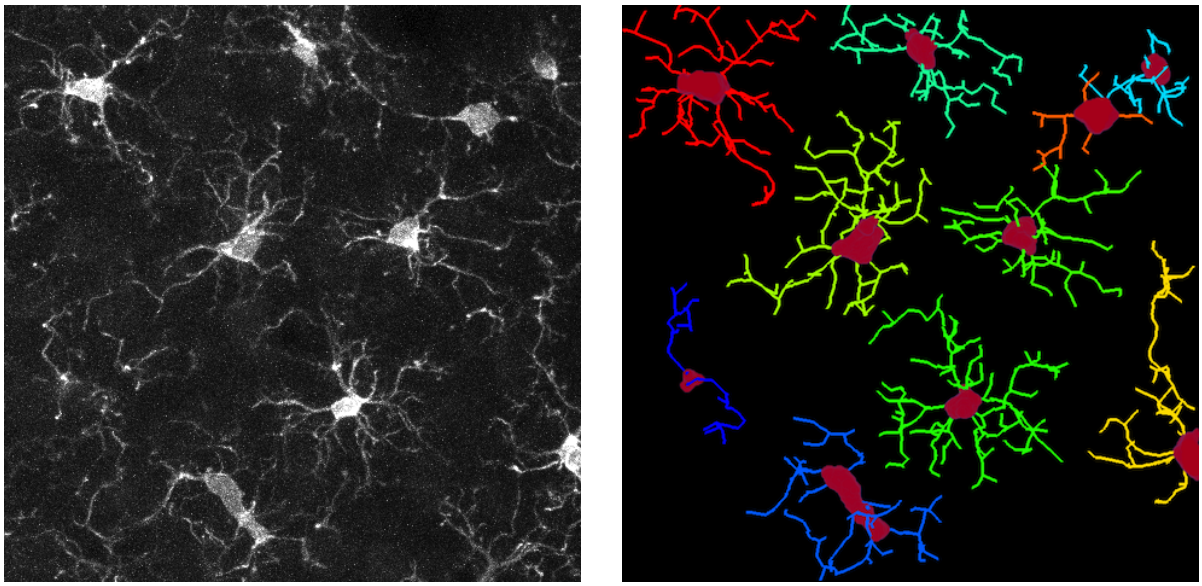


Figure 3.1: The results of tracing unactivated microglia. The image on the left is a maximum intensity projection of the original image, while the image on the right depicts the results of tracing. In the tracing results image, each color represents a distinct cell. The reddish-pink regions of each cell depict the volumes of the somata that were identified prior to the tracing of the microglia processes. Note how the processes in the bottom right of the original image were not added to any of the cellular models. This is because they are not attached to a soma that appears in the image.

Comparison to Manually Validated Images

In order to verify the accuracy of our automatic tracing system we compare its results to traces generated by humans. This is one of the few validation approaches available to us in the absence of any *ground truth* microglia data. A side-by-side comparison of such traces is presented as Fig 3.4.

One interesting discovery resulting from this experiment was just how much variation exists among traces generated by humans. Each manually generated trace is significantly different from the others, despite the fact that they were all generated

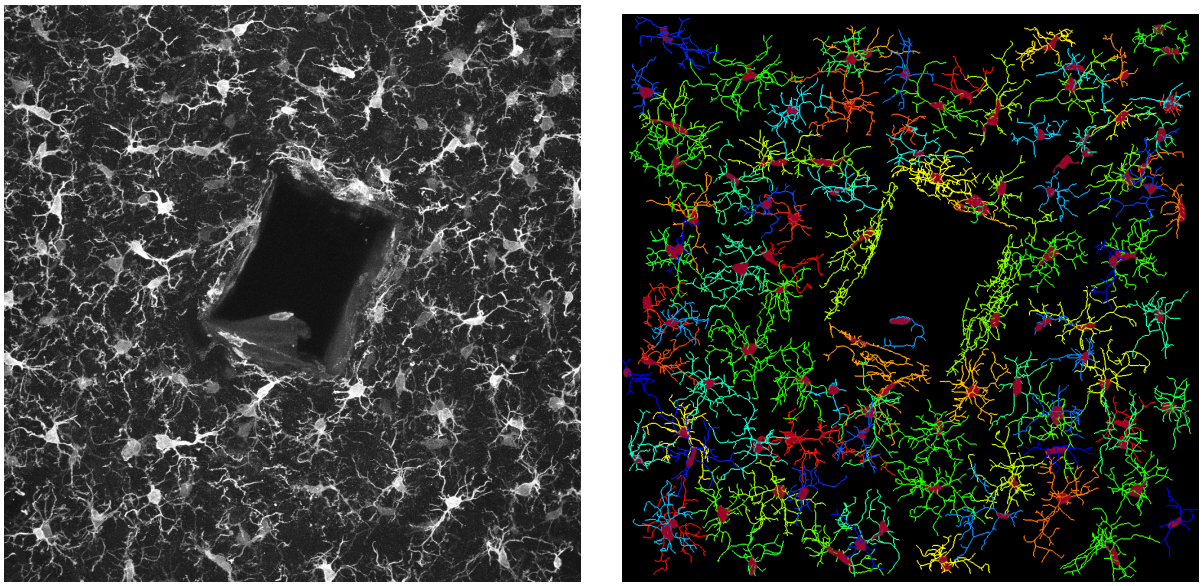


Figure 3.2: This figure depicts a sample of brain tissue tin which a neural prosthetic device was implanted. This is the cause of the rectangular "hole" in the center of the image. This image was taken six weeks post injury. Here we see some of the morphological differences between activated and unactivated microglia. Activated microglia tend to exhibit increased elongation. They also tend to have fewer processes than unactivated microglia. The processes of activated microglia are typically wider and contain more bifurcation points than those found in unactivated microglia. This figure demonstrates our system's ability to trace activated microglia.

by people using the same software to analyze the same cell. This raises worrying questions. If humans cannot agree on the correct shape and structure of a microglial cell from a 3D image, then what hope do computers have? Even if computers can generate accurate traces, how would we be able to tell?

Fortunately, our feature extraction techniques inject some objectiveness back into what might now seem like a subjective procedure. Comparing important feature points (such as branches or terminations) gives us another way to analytically compare traces. Using techniques such as these allows us to seek consensus between traces. If multiple traces all agree with one another it increases our confidence in all of them.

Another surprising result was realized while visually inspecting the differences between the human and machine generated traces. In one such case, there was a

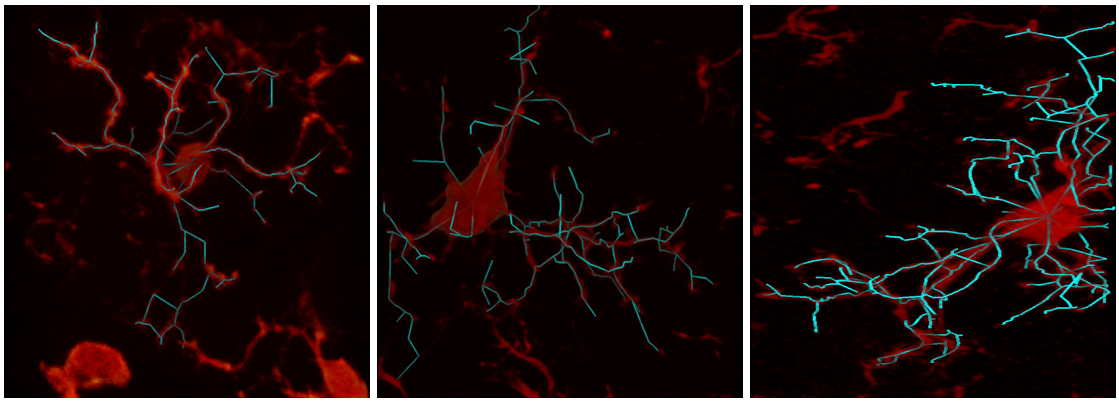


Figure 3.3: Tracing results from images of three different magnifications. These images were captured at 50x, 63x, and 94.5x magnification. Because our tracing system is grounded in physical units it can trace microglia from images of different scales.

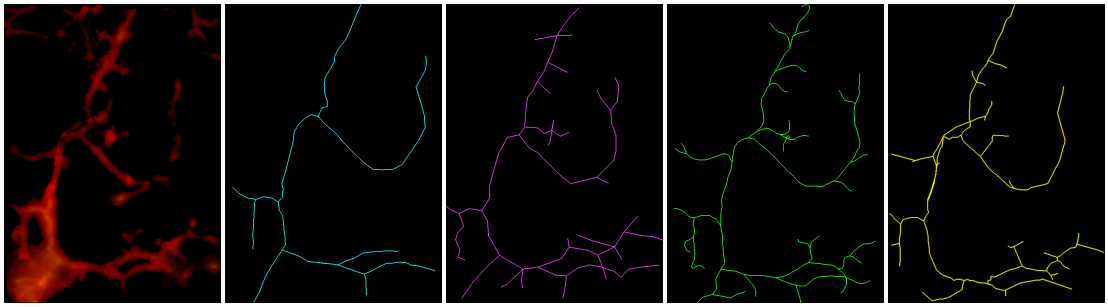


Figure 3.4: Comparison of an manually traced cell and automatic results. The first three traces, displayed in cyan, purple, and green, were manually generated by undergraduate students studying biology and biomedical engineering. The final trace, displayed in yellow, was generated by the software presented in this thesis.

large branch in the machine-generated results that did not appear in any of the others. When comparing this to the original image, we were surprised to find that this branch actually corresponded to a faint microglia process. Thus, in certain cases, our automatic tracing procedure can identify physical structures in images that humans may fail to detect.

3.2 Classification

We used K-PLS to classify a population of 793 microglia based on the features provided to us by L-Measure. We selected a Gaussian kernel function with $\sigma = 20$

	Human # 1	Human #2	Human #3	Machine
Human #1	100%	100%	100%	85.7%
Human #2	44%	100%	94.4%	72.2%
Human #3	32%	80%	100%	62.5%
Machine	25%	62.5%	60%	100%

Table 3.1: Percentage of branch points that were accounted for by other traces. Each row shows how many of this trace's branches were accounted for by its neighbors. For instance, the final field of the first row shows that our automatic tracing results contained 85.7% of the branch points that the first human examiner found. A branch point in Trace *A* is considered "accounted for" by Trace *B* if *B* contains a branch point within 5.0 μm of the original. Branch points were selected as a basis of comparison because they are among the most defining structural features of a cellular trace.

Trace	# Bifurcations	# Branches	# Tips	Width	Height	Depth
Cyan	6	16	8	156.65	231.94	21
Purple	17	44	22	450.75	256.25	24
Green	23	52	26	174.95	276.03	22
Yellow	24	52	26	448	235	24

Table 3.2: Features extracted from five traces of a microglial cell

for use with K-PLS. The cells were classified into one of two groups: activated or inactive.

From this total population, we randomly selected 158 cells to compose our training set. This represents 20% of the total population. The remaining 635 cells were used as our test set. The training set was selected such that the representation of each class is the same as it is in the test set.

Using the training data, K-PLS generated a classifier. This classifier identified the correct class for each cell in the test set with an accuracy of 86.4%.

In order to evaluate the performance of K-PLS, we also classified this data using Support Vector Machines. Using this approach, we achieved a classification accuracy of 83.5%. The fact that K-PLS and SVM resulted in similar accuracy shows that the the results we have achieved with K-PLS are not a fluke. This comparison experiment also helps to justify our choice of classifier, as K-PLS provided us with a more accurate discriminant.

As you can see in Table 3.3, our classification error consists mostly of false

		Predicted class: Active	Predicted class: Inactive
True class	Active	195 (30.7%)	51 (8.0%)
	Inactive	35 (5.5%)	354 (55.7%)

Table 3.3: Confusion matrix resulting from our cellular classification experiment.

negatives. These are activated microglia that were classified as unactivated. In order to improve the classification accuracy beyond 86% we will need to discover why so many activated microglia are considered unactivated by our classifier.

4. Discussion and Conclusions

4.1 Statement of Principal Findings

In this thesis, we demonstrated a novel method for automatic 3D tracing of microglia using state of the art image processing techniques. Using the traces generated by this procedure, we assembled an extensive set of features for each cell. With these features, we were able to classify microglia as activated or inactive within 86.4% accuracy.

4.2 Strengths of the Study

This thesis advances the state of the art of a specific domain topic within brain tissue analysis. We know of no other published work that demonstrates automatic 3D microglia tracing. This was not a trivial task. Fig. 2.10 gives a sense of the difficulty in deciding what cell a given node should be assigned to. Images of microglia often contains processes belonging to a cell whose soma is not depicted. If we do not carefully handle these edge cases our tracing results will be invalid. This is another motivation behind our use of a maximum distance threshold during the creation of cellular models.

The 86.4% classification accuracy that we achieved demonstrates the feasibility of using machine learning techniques to distinguish between different states of microglia.

4.3 Weaknesses of the Study

Despite our best efforts, the system still has a number of parameters that may need tuning. An improper value for any of these parameters could render the results of tracing as inaccurate. In order to make these parameters as easy to tune as possible, we have based them in physical units (μm) wherever possible. We also save an intermediate image after each step of the tracing procedure. This allows the user to determine which step (if any) failed. This information usually identifies the

parameter that needs further tuning.

The other weakness of the approach presented here is that parts of it are computationally intensive, particularly the tree building step. There may be other parts of our implementation that could benefit from a more efficient solution, as well. So far our focus has been on achieving results of the highest quality possible. Thus we have not shied away from using methods that may be slow or demanding of system resources.

Approaches to lessen or eliminate these inconveniences will be explored in Section 4.4.

4.4 Future Research

4.4.1 Improved Memory Usage

The work presented in this thesis could benefit from improved memory usage, particularly for larger images. Currently, our tracing system necessitates that the entire input image be loaded into system memory.

As imaging technology continues to advance, so does the memory footprint of these images. Therefore, we intend to re-factor our tracing system to use a streaming pipeline. This technique divides the image into several pieces, processes those pieces, and then tiles them back together into a single large output image. If we adopted such an approach we would no longer need to keep the whole image in memory at once. It is our hope that this will improve runtime performance for typical workstations and laptops.

4.4.2 More Intuitive Parameter Selection

If tunable parameters cannot be completely eliminated, we should develop more intuitive interfaces to make it easier for the user to select appropriate parameter values. Here we describe one such system that could accomplish this goal.

We begin by asking the user to select a subset of the image to represent a region of interest (ROI). We then perform one step of the tracing process on this ROI using three different parameter values: the default value, a positively perturbed value, and a negatively perturbed value. Once this tracing step is complete, we display these

three different results to the user. At this point, the user indicates whether or not they are satisfied with any of the results. If not, we ask that they indicate which of the results was best. This lets the system know which direction to further perturb the parameters.

4.4.3 More Accurate Classification

As mentioned previously, our classification error consists mostly of activated microglia that were considered unactivated. In order to improve our classification results above 86% we will need to analyze these particular cells more closely to discover why they were misclassified.

4.4.4 Astrocyte Analysis

We are excited about the possibility that our tracing system is general enough to be useful for investigating astrocytes. These are another type of support cell in the CNS. If our system can also accurately trace astrocytes, we should be able to gain a wealth of knowledge by examining the associations between these two different types of brain cells. One important scenario that involves both microglia and astrocytes is the body's response to brain and spinal cord injury.

LITERATURE CITED

[1] L. Spataro *et al.*, "Dexamethasone treatment reduces astroglia responses to inserted neuroprosthetic devices in rat neocortex," in *Experimental Neurology*, vol. 194, no. 2, pp. 289-300, Apr. 2005.

[2] D. H. Szarowski *et al.*, "Brain responses to micro-machined silicon devices," in *Brain Research*, vol 983, no. 1-2, pp. 23-25, Sep. 2003.

[3] R. Fields, "Aging: Glia Rage against the Dying Light," in *The Other Brain*, 1st ed. New York: Simon & Schuster, 2009, pp. 241-246.

[4] Y. Rouchdy *et al.*, "Segmentation of microglia from confocal microscope images combining the Fast Marching Method with Harris Points," in *Proc. 3rd Int. Workshop on Microscopic Image Analysis with Applications in Biology*, New York, NY, 2008.

[5] H. Xiao *et al.*, "Ct3d: tracking microglia motility in 3D using a novel cosegmentation approach," in *Bioinformatics*, vol. 27, no. 4, pp. 564-571, Dec. 2010.

[6] J. Glenn *et al.*, "Characterisation of ramified microglial cells: detailed morphology, morphological plasticity and proliferative capability," in *J. Anatomy*, vol. 180, no. 1, pp. 109-118, Feb. 1992.

[7] G. Lehmann, "Robust Automatic Threshold Selection," in *Insight J.*, Nov. 2006. Available: http://hdl.handle.net/1926/370 (Retrieved on Nov. 17th, 2011).

[8] G. Lehmann, "Label object representation and manipulation with ITK," in *Insight J.*, Aug. 2008. Available: http://hdl.handle.net/1926/584 (Retrieved on Nov. 17th, 2011).

[9] T. Lindeberg, "Feature detection with automatic scale selection," in *Int. J. of Computer Vision*, vol 30, no. 2, pp. 79-116, Aug. 1998.

[10] R. C. Prim, "Shortest connection networks and some generalizations," in *Bell System Tech. J.*, vol. 36, pp. 1389-1401, 1957.

[11] R. Scorcioni *et al.*, "L-Measure: a web-accessible tool for the analysis, comparison and search of digital reconstructions of neuronal morphologies," in *Nature Protocols*, vol. 3, pp. 866-876, Apr. 2008. doi: 10.1038/nprot.2008.51

[12] R. Rosipal and L. Trejo, "Kernel Partial Least Squares Regression in Reproducing Kernel Hilbert Space," in *J. Machine Learning Research*, vol. 2, pp. 97-123, Dec. 2001.

[13] K. Bennett and M. Embrechts, "An Optimization Perspective on Kernel Partial Least Squares Regression," In J. A. K. Suykens *et al.*, editors, *Advances in Learning Theory: Methods, Models and Applications*, vol. 190 of NATO Science Series, Series III: Computer and System Sciences, pp. 227-250. IOS Press, Amsterdam, The Netherlands, 2003.

[14] R. Rosipal, N. Kramer, "Overview and Recent Advances in Partial Least Squares," *Lecture Notes in Comput. Sci.*, vol. 3940, pp. 34-51, 2006.

[15] C. Bjornsson *et al.*, "Associative image analysis: A method for automated quantification of 3D multi-parameter images of brain issues," *J. Neurosci Methods*, vol. 170, no. 1, pp. 165-178, May 2008. doi:10.1016/j.jneumeth.2007.12.024.

[16] R. Bracewell, "Convolution," in *The Fourier Transform and Its Applications*, 1st ed. New York: McGraw-Hill, 1965, pp. 27.

[17] D. Hale, "Recursive Gaussian filters," Center for Wave Phenomena, Colorado School of Mines, Golden, CO, CWP Rep. 546, 2006.

[18] D. Forsyth and J. Ponce, "Edge Detection," in *Computer Vision A Modern Approach*, 1st ed. Upper Saddle River, NJ: Prentice Hall, 2003, pp. 173-174.

[19] M. Wilkinson, "Optimizing edge detectors for robust automatic threshold selection: coping with edge curvature and noise," in *Graph. Models Image Process.*, vol 60, pp. 385-401, Jun. 1998.

[20] A. Frangi *et al.*, "Multiscale vessel enhancement filtering," in *Lecture Notes in Comput. Sci.*, vol 1946, pp. 130-137, 1998.

[21] StatSoft, Inc. "Partial Least Squares," in *Electronic Statistics Textbook*. Tulsa, OK: StatSoft, 2011. [Online]. Available: http://www.statsoft.com/textbook/partial-least-squares/ (Retrieved on Nov. 17th, 2011).

[22] S. Polavaram. (2011, Jan. 31). *L-Measure functions* [Online]. Available: http://cng.gmu.edu:8080/Lm/help/index.htm (Retrieved on Nov. 17th, 2011).

APPENDIX A
Features calculated by L-Measure

Recall that the input to L-measure is a .swc file. This is a simple file format originally designed to describe the morphology of neurons. Each line of a .swc file describes a point, or *node*, on a cell.

Many of the feature definitions in this Appendix use the term *compartment*. This is defined as a line segment whose endpoints are a node and its parent. All compartment-level measurements return zero for somata nodes, as they have no parent.

The following information was retrieved from *L-Measure Online Help* [22].

A.1 Topological measurements

These measurements typically return a single value for a given cell.

A.1.1 Soma Surface

This function computes the surface of the soma (Type=1). If the soma is composed of just one compartment, then it uses the sphere assumption, otherwise it returns the sum of the external cylindrical surfaces of compartments forming the soma. There can be multiple somata, one soma or no soma at all.

$$4\pi r^2 \tag{A.1}$$

A.1.2 Number of Stems

This function returns the number of stems attached to the soma. When the type of the Compartment changes from type=1 to others it is labeled a stem. These stems can also be considered as independent subtrees for subtree level analysis.

A.1.3 Number of Bifurcations

This function returns the number of bifurcations for the given input cell. A bifurcation point has two daughters.

A.1.4 Number of Branches

This function returns the number of branches in the given input cell. A branch is one or more compartments that lie between two branching points or between one branching point and a termination point.

A.1.5 Number of Tips

This function returns the number of terminal tips for the given input neuron. This function counts the number of compartments that terminate as terminal endpoints.

A.1.6 Width

Width is computed on the x-coordinates and it is the difference of minimum and maximum x-values after eliminating the outer points on the either ends by using the 95% approximation of the x-values of the given input neuron.

If the user believes that the input neuron is not oriented properly, one can do principal component analysis (PCA) first. In this case, width is computed using the same 95% approximation on the new x-axis after shifting and rotating the cell to a different axis based on component analysis.

A.1.7 Height

Height is computed on the y-coordinates and it is the difference of minimum and maximum y-values after eliminating the outer points on the either ends by using the 95% approximation of the y-values of the given input neuron.

See the width metric for a notice about orientation.

A.1.8 Depth

Height is computed on the z-coordinates and it is the difference of minimum and maximum z-values after eliminating the outer points on the either ends by using the 95% approximation of the z-values of the given input neuron.

See the width metric for a notice about orientation.

A.1.9 Terminal Segments

A terminal segment is a branch that contains a terminal tip. The standard way to use this function gives the total number of compartments that are a part of a terminal branch. This function can also be used to differentiate terminal branches from internal branches.

A.2 Branch level measurements

The functions in this section measure features for a given branch or bifurcation point. When calculated for an entire input tree, these measurements return the following values:

minimum, average, maximum, total sum across all branches, standard deviation.

A.2.1 Burke Taper

This function measures the actual diameter of the bifurcation compartment D_a minus previous bifurcation compartment diameter D_c, divided by the total length L of the branch

$$\frac{D_a - D_c}{L} \tag{A.2}$$

A.2.2 Hillman Taper

Given a branch, this function computes the ratio between the final diameter D_a and the initial diameter D_c.

$$\frac{D_a - D_c}{D_a} \tag{A.3}$$

A.2.3 Branch Path Length

This function returns the sum of the length of all compartments forming the given branch. Given a branch consisting of nodes A, B, C, D, and E, this value would be given by the following equation:

$$\|\overrightarrow{AB}\| + \|\overrightarrow{BC}\| + \|\overrightarrow{CD}\| + \|\overrightarrow{DE}\| \tag{A.4}$$

A.2.4 Contraction

For a given branch, contraction is calculated as the ratio of Euclidean Distance and Path Length.

$$\frac{\|\overrightarrow{AE}\|}{\|\overrightarrow{AB}\| + \|\overrightarrow{BC}\| + \|\overrightarrow{CD}\| + \|\overrightarrow{DE}\|} \tag{A.5}$$

A.2.5 Fragmentation

This function simply measures the number of compartments that constitute a given branch.

A.2.6 Daughter Ratio

This function returns the ratio between the radii of the two daughters at a branch point. The ratio is always calculated such that the wider daughter D_a is taken as the numerator.

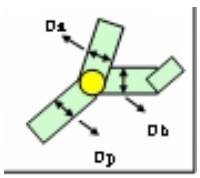
$$\frac{D_a}{D_b} \tag{A.6}$$

A.2.7 Partition Asymmetry

Let us define n_1 as the number of terminal tips that have daughter D_a as an ancestor. Similarly, n_2 is the number of terminal tips that can be reached from daughter D_b. The Partition Asymmetry is calculated using the following formula:

$$\frac{|n_1 - n_2|}{n_1 + n_2 - 2} \tag{A.7}$$

A.2.8 Rall's Power



The Rall value n is computed as the best value that fits the following equation:

$$D_p^n = D_a^n + D_b^n \tag{A.8}$$

Here D_p, D_a, and D_b represent the diameters of the parent, first daughter, and second daughter, respectively.

According to Rall's rule, we compute Rall's power by linking the diameter of two daughter branches to the diameter of the bifurcating parent. We compute the best fit for Rall's power within the boundary values of [0, 5] at incremental steps of 1,000 compartments. The final Rall value is the idealistic n value that can propagate the signal transmission without loss from the starting point to the terminal point in a cable model assumption.

A.2.9 Rall's Ratio

After computing the value for Rall's Power, this function returns the following ratio:

$$\frac{D_a^n + D_b^n}{D_p^n} \tag{A.9}$$

A.2.10 Rall's Ratio (classic)

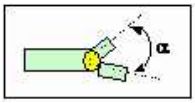
The only difference between this metric and Rall's Ratio is that n is set to 1.5

A.2.11 Rall's Ratio (2)

The only difference between this metric and Rall's Ratio is that n is set to 2

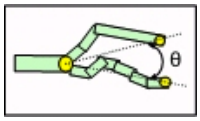
A.2.12 Local Bifurcation Angle

This function returns the angle between the first two nodes following a bifurcation point.



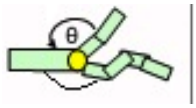
A.2.13 Remote Bifurcation Angle

Given a bifurcation, this function returns the angle between that bifurcation and the end of the two growing segments. In other words this function returns the angle between two branches.



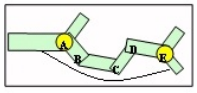
A.2.14 Local Bifurcation Tilt

This function returns the angle between the previous compartment of a bifurcating father and the two daughter branches of the same bifurcation. The smaller of the two angles is returned as the result.



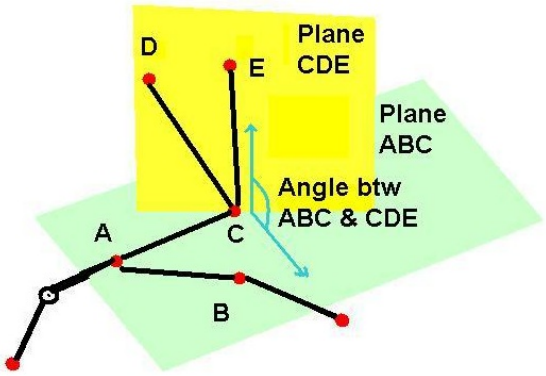
A.2.15 Remote Bifurcation Tilt

This function returns the angle between the previous father compartment of the current bifurcating father and its two daughter compartments. Since there are two compartments exiting the bifurcation, the angle returned is the smaller of the two possible.



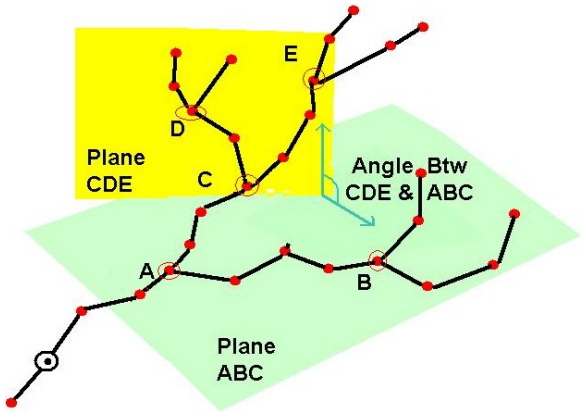
A.2.16 Local Bifurcation Torque

This function returns the angle between the plane of previous bifurcation and the current bifurcation. The bifurcation plane is identified by the two daughter compartments leaving the bifurcation.



A.2.17 Remote Bifurcation Torque

This function returns the angle between, current plane of bifurcation and previous plane of bifurcation. See the figure below. The current plane of bifurcation is formed between C D and E where all three are bifurcation points and the previous plane is formed between the bifurcation points A B and C.



A.2.18 Last Parent Diameter

This function returns the diameter of last bifurcation point before a terminal tip.

A.2.19 Diameter Threshold

This function returns the diameter of the first compartment after the last bifurcation leading to a terminal tip.

A.2.20 Hillman Threshold

This function returns the weighted average between 50% of father and 25% of daughter diameters of the terminal bifurcation.

$$\frac{1}{2}D_p + \frac{1}{4}D_a + \frac{1}{4}D_b \tag{A.10}$$

A.3 Node level geometrical measurements

These measurements are performed on each node in a cell.

For a given tree, these measurements return the following values:

minimum, average, maximum, total sum across all nodes, standard deviation.

A.3.1 Diameter

This function returns diameter of each compartment in the neuron.

$$D = 2r \tag{A.11}$$

A.3.2 Length

This function returns the length of compartments by computing the distance between the two end points of a compartment.

A.3.3 Surface

This function returns the size of the surface of the compartment. This is calculated using the formula for the surface area of the side of a cylinder.

$$2\pi rh \tag{A.12}$$

A.3.4 Section Area

This function returns the size of the cross section of the compartment. This is calculated using the formula for the area of a circle.

$$\pi r^2 \tag{A.13}$$

A.3.5 Volume

This function returns the volume of the compartment. This is calculated using the formula for the volume of a cylinder.

$$\pi r^2 h \tag{A.14}$$

A.3.6 Euclidean Distance

This function returns the straight line distance from the soma to a given point.

$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2} \tag{A.15}$$

A.3.7 Path Distance

This function returns the sum of lengths of each compartment from the soma to a given point.

A.3.8 Branch Order

This function returns the order of the branch with respect to soma where soma has order = 0. The first bifurcation has Branch Order of 1, second bifurcation has Branch Order of 2 and so on.

A.3.9 Terminal Degree

This function gives the total number of tips that each compartment will terminate into. If you back track the tree starting from terminal tips, the farthest compartment from a terminal tip will have the highest degree.

A.3.10 Helix

The function computes a helix by choosing four points at a time. It computes the normal form of the three vectors to find the 4th vector. We denote the following vectors:

$\vec{A}$: previous vector, $\vec{B}$: 2nd previous vector, $\vec{C}$: current vector

The helix is given by the following formula:

$$\frac{(\vec{A} \times \vec{B}) \cdot \vec{C}}{3(\hat{A} \cdot \hat{B} \cdot \hat{C})} \tag{A.16}$$