5-2020

Evaluation of tracing techniques in the rat spinal cord using a custom MATLAB application.

Rachel M. Zalla
University of Louisville

EVALUATION OF TRACING TECHNIQUES IN THE RAT SPINAL CORD USING A CUSTOM MATLAB APPLICATION

By

Rachel M. Zalla
B.S., University of Louisville, 2019

A Thesis
Submitted to the Faculty of the
University of Louisville
J.B. Speed School of Engineering
as Partial Fulfillment of the Requirements
for the Professional Degree

MASTER OF ENGINEERING

Department of Bioengineering

May 2020

EVALUATION OF TRACING TECHNIQUES IN THE RAT SPINAL CORD USING A CUSTOM MATLAB APPLICATION

Submitted by: Rachel Zalla
Rachel M. Zalla

A Thesis Approved On

5-19-2020
(Date)

by the Following Reading and Examination Committee

David S.K. Magnuson, PhD, Thesis Director

Guruprasad Giridharan Digitally signed by Guruprasad Giridharan Date: 2020.05.19 12:11:53 -04'00'
Guruprasad A. Giridharan, PhD, Thesis Co-Director

Jonathan Kopechek
Jonathan A. Kopechek, PhD

ACKNOWLEDGEMENTS

The author is grateful to the committee of David S.K. Magnuson, PhD, Guruprasad Giridharan, PhD, Jonathan Kopechek, PhD, and Scott R. Whittemore, PhD, for their support throughout the research process. The author is also grateful to the members of the labs of Dr. Magnuson and Dr. Whittemore, especially Brandon Brown, for their guidance in completing this research project.

ABSTRACT

The objective of this research project is to evaluate the efficiency of traditional chemical tracing compared to dual-viral tracing for labeling long ascending propriospinal neurons (LAPNs) in the uninjured rat spinal cord, and to develop a MATLAB program which will quantify this labeling efficiently. Chemical tracers such as fluorescent dextrans have traditionally been used to retrogradely label projection neurons in the nervous system, however, these tracers lack specificity and can label fibers of passage. To label a specific population of long-range projection neurons, such as LAPNs, we show here that dual-viral systems are necessary.

An animal experiment was performed to directly compare the efficiency of chemical tracers versus dual-viral systems to label LAPNs. To evaluate chemical tracing, Fluoro-Ruby (FR) was injected at the level of the axons terminals, cervical level 5/6 (C5/6), and the number of ipsilateral labeled cell bodies at lumbar level 2/3 (L2/3) was quantified. Similarly, two dual-viral systems were evaluated, by either injecting retro-AAV-Cre or HiRet-Lenti-Cre unilaterally at C5/6 in combination with a Cre-dependent adeno-associated virus (AAV2-FLEx-EGFP) injected unilaterally at L2/3, the level of the LAPN cell bodies. We hypothesized that 1) the HiRet-Lenti group would label and identify greater numbers of LAPNs than retro-AAV group, 2) the HiRet-Lenti group would provide greater specificity than FR, and 3) FR would label more neurons than either dual-viral labeling group.

The HiRet-Lenti and FluoroRuby groups labeled significantly greater numbers of LAPNs than the retro-AAV group. These results show that despite the retro-AAV being a robust tool for tracing corticopontine neurons, retro-AAV is inefficient for labeling long propriospinal neurons such as LAPNs. The similar numbers of LAPNs labeled by HiRet-Lenti and FR is likely due to similar rostral-caudal spread of FR and viruses at the injection site(s). However, the HiRet-Lenti

dual-viral system provides a greater specificity of labeling, as the expression of EGFP is dependent on the presence of both injected viruses, while FR can be taken up by fibers of passage, and labeling neurons that were not directly targeted for tracing. Despite the number of LAPNs labeled being similar, the dual-viral labeling utilizing HiRet-lentiviruses is preferred due to greater specificity and more prominent labeling.

Methods to quantify labeled spinal cord neurons include either manual counting or the utilization of available image processing software. Current imaging processing software are difficult for users to navigate and are not optimized for counting cells in spinal cord tissue sections. Manual counting is highly accurate, but it is inefficient and biased. To automate the cell counting process, a MATLAB program was developed to accurately determine the number of ipsilateral cell bodies labeled by each of the tracing techniques analyzed in this experiment. To validate the accuracy of the MATLAB program, the number of labeled cells counted for each tracing technique by manual counting was compared to the number generated by the MATLAB program.

The number of LAPNs counted manually did not significantly differ from the number of LAPNs counted by the MATLAB application for any of the labeling groups, and there was a highly significant correlation between the two methods. Based on these results, the custom MATLAB application accurately determines the number of ipsilateral cell bodies labeled by each of the tracing techniques analyzed in this experiment. Overall, the interactive application provides an automated, efficient, and unbiased method of counting cells in spinal cord tissue sections.

TABLE OF CONTENTS

NOMENCLATURE

SCI	=	spinal cord injury
CPG	=	central pattern generator
LAPN	=	long ascending propriospinal neuron
LDPN	=	long descending propriospinal neuron
KSCIRC	=	Kentucky Spinal Cord Injury Research Center
FLEx	=	flip excision switch
Cre	=	Cre-recombinase
EGFP	=	enhanced green fluorescent protein
retro-AAV	=	retrograde adeno-associated virus
HiRet-Lenti	=	highly efficient retrograde lentivirus
ROI	=	region of interest
FR	=	FluoroRuby
PBS	=	phosphate-buffered saline
PFA	=	paraformaldehyde
ANOVA	=	analysis of variance
CTB	=	cholera toxin b

LIST OF TABLES AND FIGURES

TABLES Page

FIGURES

I. INTRODUCTION

There are approximately 18,000 new cases of spinal cord injury (SCI) per year in the United States and < 1% of SCI patients achieve complete neurological recovery.[1] The majority of patients experience incomplete tetraplegia, incomplete or complete paraplegia, or complete tetraplegia. To develop novel treatments for SCI, the structure and function of the spinal cord must be further understood at a systems, network, and cellular level. Animal models of SCI are being used to address this lack of knowledge, with the ultimate goal of translating effective treatment strategies from animal models to human patients in the clinic.

The rat model of SCI has been used to better understand the pathology of SCI as well as evaluate treatment strategies.[2] For example, rat models have led to the development of tests to assess the locomotor and sensory functional recovery. Rat models have also resulted in a better understanding of the changes in the neuronal circuity following SCI, and how the enhancement of spontaneous regenerative mechanisms can promote recovery. The therapeutic impact of manipulating myelination, glial scarring, and/or inflammation have also stemmed from studies utilizing rat models.

A. Rat Locomotion and Propriospinal Neurons

Rat locomotion is characterized by the precise coordination of muscle activity to produce regular patterned stepping based on rhythm and pattern of locomotion. At the heart of this control are central pattern generators (CPGs), which are neuronal circuits that produce rhythmic

neural outputs to control rhythmic behaviors, such as walking or breathing. The four spinal CPGs (one for each limb) for locomotion are housed within the cervical and lumbar spinal enlargements, and produce and stepping behaviors of the forelimbs and hindlimbs, respectively.[3] Communication between the locomotor CPGs is carried out by propriospinal neurons, which are neurons housed completely within the spinal cord.

Propriospinal axons make-up approximately one-third of axons/fibers in the rat lateral and ventral spinal white matter. Most propriospinal neurons are short in length, projecting only four spinal segments or less. However, there are also long propriospinal neurons that project more than four spinal segments. These long propriospinal neurons, can either ascend or descend within the spinal cord, and if they project to and from the spinal enlargements they are anatomically suited to mediate coordination between the forelimb and hindlimb CPGs.[4,5] The pathway(s) most anatomically suited for this coupling of the lumbar and cervical CPGs, and in turn the mediation of interlimb coordination of the forelimbs and hindlimbs are long ascending (LAPNs) and the reciprocal pathway, long descending (LDPNs) propriospinal neurons.[5,6] LAPNs, which are the focus of this thesis, are defined herein as having cell bodies in the rostral lumbar spinal cord (L1-3) and having at least one projection to cervical level 5-6 (C5/6).[7] Both anatomical and electrophysiological studies support the concept of reciprocal long propriospinal neurons communicating between the cervical and lumber CPGs to mediate the rhythm and pattern of locomotion. Additionally, these long propriospinal neurons have become increasingly important following SCI as reorganization of these propriospinal connections has been suggested to contribute to functional recovery after SCI.[8]

To determine the specific role of LAPNs in functional recovery, the laboratories of Dr. David S.K. Magnuson and Dr. Scott R. Whittemore at the Kentucky Spinal Cord Injury Research

Center (KSCIRC) have performed studies to determine the behavioral role of LAPNs in rats before and after contusive SCI. Using a dual-viral system described in more detail below, they were able to conditionally and reversibly silence LAPNs and analyze the resultant locomotor behavior.[3,9] These studies showed that when LAPNs were silenced in uninjured rats, left-right hindlimb alternation during stepping is disrupted, resulting in a "bounding" gait that is not normally seen at the lower speeds at which the rats were locomoting. Surprisingly, when LAPNs are silenced after contusive SCI, locomotor outcomes were improved. These results have led to specific questions about the role of LAPNs in functional recovery and their inherent anatomical plasticity after SCI. Methods developed during this thesis work are essential to answering that latter question.

B. Tracing Techniques

Work is currently being done to specifically label LAPNs for somatic and dendritic characterization, as well as determine the monosynaptic inputs of LAPNs. The tracing techniques available to label neurons include transgenics, traditional chemical tracers, single virus tracers, and multi-viral systems.[10] Transgenic tracing utilizes genetically modified mice to express genes targeted to specific organelles, cells, or tissues.[11] Through specific genetic modification to the mouse germ line, robust fluorescent labeling is available which enables the labeling of a specific subset of cells such as neurons. However, current transgenic rat models are not yet capable of such tracing.

Traditional chemical tracers, such as Fluorogold™ and similar fluorescent inorganic compounds and single virus tracers can act as either anterograde or retrograde tracers, but lack specificity of labeling. Anterograde tracers are taken up by neuronal cell bodies at the injection site, while retrograde tracers are taken up any/all axon terminals at the injection site and traverse back to the cell body. Chemical tracers such as fluorescent dextrans can be taken up by axon terminals and retrogradely transported. Chemical tracers are also known to inadvertently label fibers of passage.[12,13] Similarly, single virus tracers involve injecting a single virus, that may be taken up by cell bodies and/or axon terminals at the injection site, and ultimately does not offer more specificity than chemical tracers.

To overcome the limitations of chemical and single virus tracing, and specifically label an anatomically defined population of long-range projection neurons such as LAPNs, dual-viral systems were developed and further modified.[14,15,16]. For specific LAPN labeling, the dual-viral systems used in this project involve two viruses and two sets of injections: one at the level of the cell bodies for the pathway of interest (L2/3), and one virus at the level of the axon terminals (C5/6).

The virus injected at the level of the cell bodies (L2/3) contains a Flip-Excision Switch (FLEx), which allows for Cre-recombinase (Cre)-dependent expression of enhanced green fluorescent protein (EGFP) to label and visualize neurons. The FLEx switch utilizes site-specific recombination to conditionally manipulate gene expression, allowing the expression of a gene of interest in the presence of Cre.[14] The FLEx switch used in this experiment relies on the orientation specificity of Cre-recombinase, which binds lox P sites to induce recombination. More specifically, the DNA coding sequence for EGFP is flanked by target sites in opposing orientations, so the DNA sequence is first inverted by Cre, allowing for expression of the gene of

interest, EGFP here. Cre then excises the heterotypic lox P sites in the same orientation, resulting in stable EGFP expression, and cell specific labeling. The virus injected at the level of the axon terminals (C5/6) must code for Cre, infect axon terminals, and be retrogradely transported to the cell bodies so that EGFP is expressed. The FLEx switch for Cre-dependent expression of EGFP is detailed in **Figure 1** below.

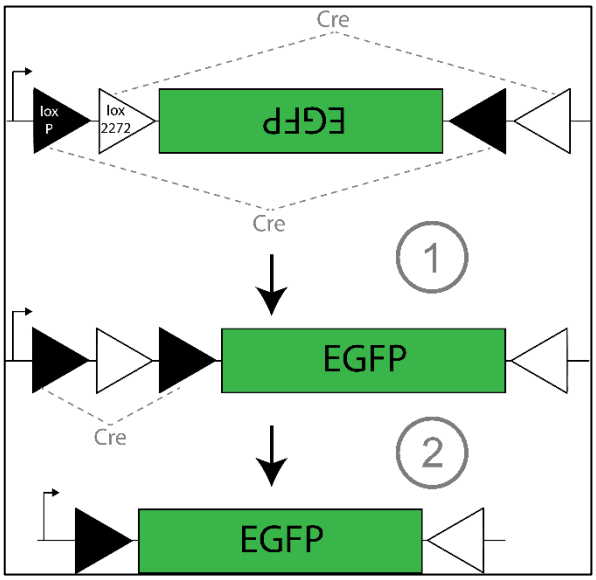

FIGURE 1 – Visualization of FLEx switch: 1) Cre-mediated inversion of coding sequence via either of the heterotypic antiparallel lox sites 2) Excision of lox P sites results in orthogonal and antiparallel lox sites that are incapable of further recombination and allows for stable EGFP expression.

While this dual-viral system provides high specificity, robust labeling of the pathway of interest is also paramount. For robust labeling, Cre delivery to the cell body must be high. Numerous virus that may confer robust Cre delivery have been well characterized in the brain, but have yet to be used and characterized in the rat spinal cord. One candidate for robust Cre delivery is a retrograde adeno-associated virus (retro-AAV) which provides efficient labeling

when targeting cortical neurons.[16] A second candidate for robust Cre delivery is a highly efficient retrograde lentivirus (HiRet-Lenti), which our laboratories have used previously.[3] A major goal of this thesis was to compare these chemical tracer labeling with both retro-AAV-cre- and HiRet-Lenti-cre-mediated labeling.

C. Quantifying the Number of Labeled Neurons

Current methods to quantify labeled spinal cord neurons include either manual counting or the utilization of available image processing software. Open source image processing software such as ImageJ (National Institutes of Health / https://imagej.nih.gov) or CellProfiler (Broad Institute / https://cellprofiler.org) have a wide variety of functions. These open source software programs are often difficult for users to navigate and are not optimized for counting cells in spinal cord tissue sections. Other image analysis software packages, such as MetaMorph (Biovision Technologies Inc. / https://www.biovis.com/metamorph), are expensive and not often utilized. Due to the non-specificity of current image processing software, manual counting is often used when analyzing spinal cord section images. While the accuracy of manual cell counting is high, it also inefficient and biased, as it relies on the counter's perception of what defines a cell.

To automate the cell counting process, a MATLAB program was developed to accurately determine the number of ipsilateral cell bodies labeled by each of the tracing techniques analyzed in this experiment. The program utilizes MATLAB image processing techniques, including color thresholding and boundary determination, to automatically determine the number of labeled cells within a user-specified region of interest (ROI). The program is integrated within an interactive

application, which enables the user to load an image of a spinal cord section, select a ROI, count only the labeled cells within that region, and overlay lamina to determine where the cells are located in the spinal cord. The interactive application enables users to seamlessly navigate through a large number of images, while the automated cell counting function both eliminates variability between users and significantly reduces counting time. To validate the accuracy of the MATLAB program, the number of labeled cells counted for each tracing technique by manual counting was compared to the number generated by the MATLAB program.

II. METHODS

A. Evaluating Efficiency of Chemical and Viral Tracing

An animal experiment was performed to directly compare the efficiency of chemical tracers - which have been traditionally used for labeling neurons and their projections in the nervous system – with dual-viral systems to label LAPNs, in uninjured (non-SCI) rats. To evaluate the efficiency of a commonly used retrograde chemical tracer, FluoroRuby (FR) was injected at cervical level 5/6 (C5/6) on the animals' left side, and the number of ipsilateral cell bodies at lumbar level 2/3 (L2/3) was quantified. To evaluate the dual-viral systems, either two boluses of retro-AAV-Cre or two boluses of HiRet-Lenti-Cre were injected unilaterally at C5/6. In the same surgery, two boluses of AAV2-FLEx-EGFP were injected unilaterally at L2/3. Based on preliminary work, we hypothesized that: 1) HiRet-Lenti would label and identify greater numbers of LAPNs than retro-AAV, 2) HiRet-Lenti would have greater specificity than FR, and 3) FR would label more neurons than either dual-viral labeling group. The number of ipsilateral cell bodies at L2/3 was quantified for each set of injections by both manual counting and automatic counting utilizing a custom MATLAB program.

B. Chemical and Viral Tracer Injections

A total of $N = 12$ adult female Sprague Dawley rates (220-250 g; Envigo, City, IN) were used in this experiment. Animals were housed two per cage with *ad libitum* food and water under 12 h light/dark cycle. Procedures were performed in accordance with the University of Louisville

Institutional Animal Care and Use and Institutional Biosafety Committees, as well as the Public Health Service Policy on Humane Care and Use of Laboratory Animals.

For the chemical tracer injections, rats (n = 4) were anesthetized (ketamine/xylazine/acepromazine, 0.5 ml/250 g i.p.), placed into a spinal stabilization unit, and received a C5/6 laminectomy to expose C6. FluoroRuby was ipsilaterally injected (0.25 µl, 1.3 mm rostrocaudal) into the intermediate gray matter (0.55 mm mediolateral, 1.2 mm dorsoventral) using a stereotaxic device. Injections were given in one 0.25 µl bolus with the needle left in place for another 2 minutes to allow for tracer uptake and to prevent leakage out the needle track. The 0.25 µl volume was used for FR injections, as this volume provided the same rostral-caudal spread within the spinal gray matter as the volume of virus that was injected (outlined below).

For the viral injections, rats (n = 8) were anesthetized (ketamine/xylazine/acepromazine, 0.5 ml/250 g i.p.), placed into a spinal stabilization unit, and received a C5/6 laminectomy and T12 laminectomy to expose C6 and L1/2, respectively. For retro-AAV2-Cre, rats (n = 4) were ipsilaterally injected (0.5 µl/site, 1.3 mm rostrocaudal) into the intermediate gray matter (0.55 mm mediolateral, 1.2 mm dorsoventral) of C5/6 at two sites. For HiRet-Lenti-Cre, rats (n = 4) were ipsilaterally injected (0.5 µl/site, 1.3 mm rostrocaudal) into the intermediate gray matter (0.55 mm mediolateral, 1.2 mm dorsoventral) of C5/6 at two sites. All rats receiving viral injections at C5/6 (n = 8) were then given ipsilateral injections of AAV2-FLEx-EGFP at L2/3 (0.5 µl/site, 1.3 mm rostrocaudal) into the intermediate gray matter (0.5 mm mediolateral, 1.35 mm dorsoventral) of L2/3 at two sites. All injections were given in 0.25 µl boluses with the needle left in place for another 2 minutes to allow for viral uptake. Injection sites are schematized in **Figure 2**.

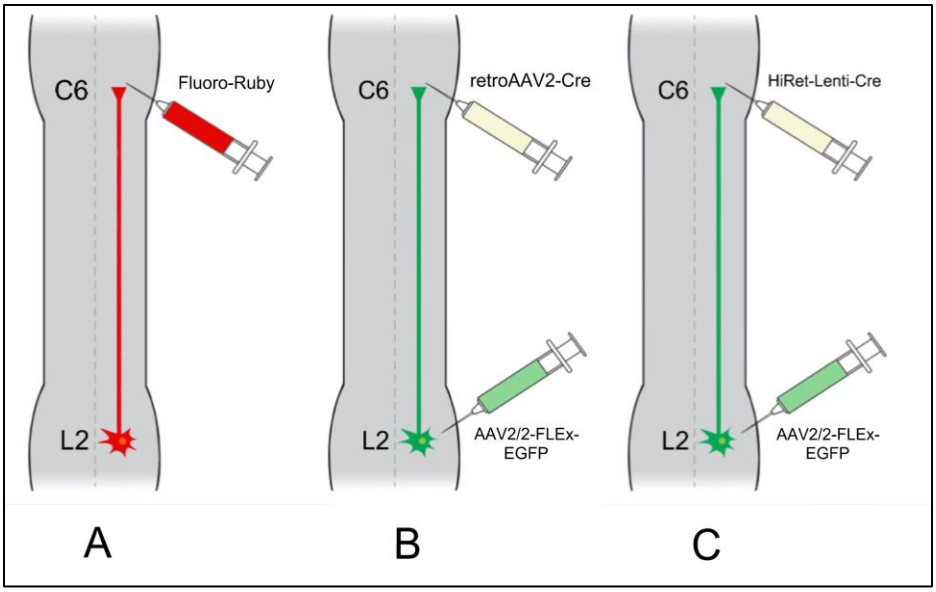

FIGURE 2 - Injections for the three different tracing techniques:

A) Fluoro-Ruby B) retro-AAV C) Hi-Ret

For all animals, the incision site(s) was/were sutured in layers and the wound closed with surgical staples. Buprenorphine (0.1 mg/kg, 0.37 ml, subcutaneous) was provided every 12 hours for the first 48 hours post-surgery for pain management, and gentamicin (20 mg/kg, 0.23 ml, s.c.) was administered once daily for 7 days. Saline solution was administered every 12 hours for the first 48 hours post-surgery and then once a day for the next 4 days for hydration. All animals recovered voluntary bladder control within 24 hours post-surgery.

C. Tissue Processing and Imaging

Two weeks following FR injections, and three weeks following viral injections, animals were sacrificed with an overdose of ketamine/xylazine/acepromazine, then transcardially perfused with 0.1 M phosphate-buffered saline (PBS) (pH 7.4) followed by 4%

paraformaldehyde (PFA) in PBS. Spinal cords were dissected, post-fixed in PFA for 1 hour, transferred to 30% sucrose, and stored at 4 °C. Spinal segments L1-3 were dissected, embedded in tissue freezing medium, cryosectioned at 30 µm, slide mounted, and stored at −20 °C. Slides were cover-slipped with Fluoromount (Company, City, ST) and air dried overnight.

Images of spinal cord sections were captured using a Nikon (Mellville, NY) TiE 300 inverted microscope with the 10× objective and TxRed filter settings for FR labeling and GFP settings for viral labeling. Every other tissue section with labeled cell bodies was imaged to prevent double counting of neurons. The number of labeled cells was manually counted, and Inkscape (https://inkscape.org), a free graphics software, was used to overlay a lamina map to record which spinal lamina the labeled neurons were found. The number of labeled neurons and location within the laminae was recorded in Excel. Along with manual counting, the number of cells in each imaged section was run through the custom MATLAB program.

D. MATLAB Programming

Using MATLAB, a program was developed to automatically detect and count labeled cells. Image processing functions and MATLAB's AppDesigner were utilized to create a functional and user-friendly application for automatic cell counting. The various functions of the program are listed in **Table 1** below.

TABLE I

LIST AND DESCRIPTION OF MATLAB FUNCTIONS

MATLAB Function	Brief Description
startupFcn	Application runs and is maximized to full screen
BrowseImageButtonPushed	Enables user to select desired image using the file selector, displays selected image within axes and file name in adjacent edit box
TraceROIButtonPushed	Enables user to trace region of interest within selected image, new traced image replaces selected image
ColorThresholdSliderValueChanged	Pixels of selected image are thresholded based upon chosen value, image is converted to binary, thresholded image is displayed in axes above slider
ManualProcessingButtonPushed	Provides user with option to trace image artifacts and eliminate them from thresholded image, check box is filled if button is pushed
SelectLaminaButtonPushed	Enables user to select lamina overlay, displays file name in adjacent edit box
DisplayOverlayButtonPushed	Figure window appears displaying original image with selected lamina overlay
CountButtonPushed	Number of cells is automatically counted, figure window displayed with two axes- original image with lamina overlay on the left, thresholded image with boundaries traced around the counted cells on the right
DirectionsButtonPushed	Figure window appears with directions detailing application use

Properties within the application programming contain object data and are stored and called throughout the different functions. These properties including a brief description are listed in **Table II** below.

TABLE II

LIST AND DESCRIPTION OF MATLAB PROPERTIES

Property	Brief Description
Image	Image data corresponding to image selected by user
full_img	Image object created from display of selected image
ROI	Image data corresponding to traced image
thresholdedImage	Image data corresponding to image after thresholding
full_threshold	Image object created from display of thresholded image
processedROI	Image data corresponding to image after manual processing
overlay	Image data corresponding to selected lamina overlay

The various functions and properties of the custom MATLAB application will be further discussed by utilizing an example image from the chemical tracing vs. dual-viral tracing experiment. This image is a lumbar section from an animal injected with the retro-AAV2-Cre virus at C5/6 and AAV2-FLEx-EGFP at L2/3.

i. Startup Function and Application Interface

When the application is first opened, it is programmed to become full screen automatically, without the user having to manually maximize the application window. The application interface is shown in **Figure 3** below.

FIGURE 3 – Screenshot of Application Interface

ii. Browse Image and Trace ROI Functions

The first interactive function of the application is the ‘BrowseImageButtonPushed’ function. When the button labeled “Browse Image” is pushed, the file selector is displayed, allowing the user to select an image of a spinal cord section. The file selector enables 8-bit images of file type ‘png’, ‘jpeg’, ‘bmp,’ and ‘tif’ to be selected. The selected image is then displayed within the axes under the label “Original Image,” and the image data are stored as the property ‘Image.’ The image object is stored as the property ‘full_img.’ The name of the file is also displayed in the ‘Edit Text’ box next to the “Browse Image” button. The displayed image within the application interface is shown below in **Figure 4**.

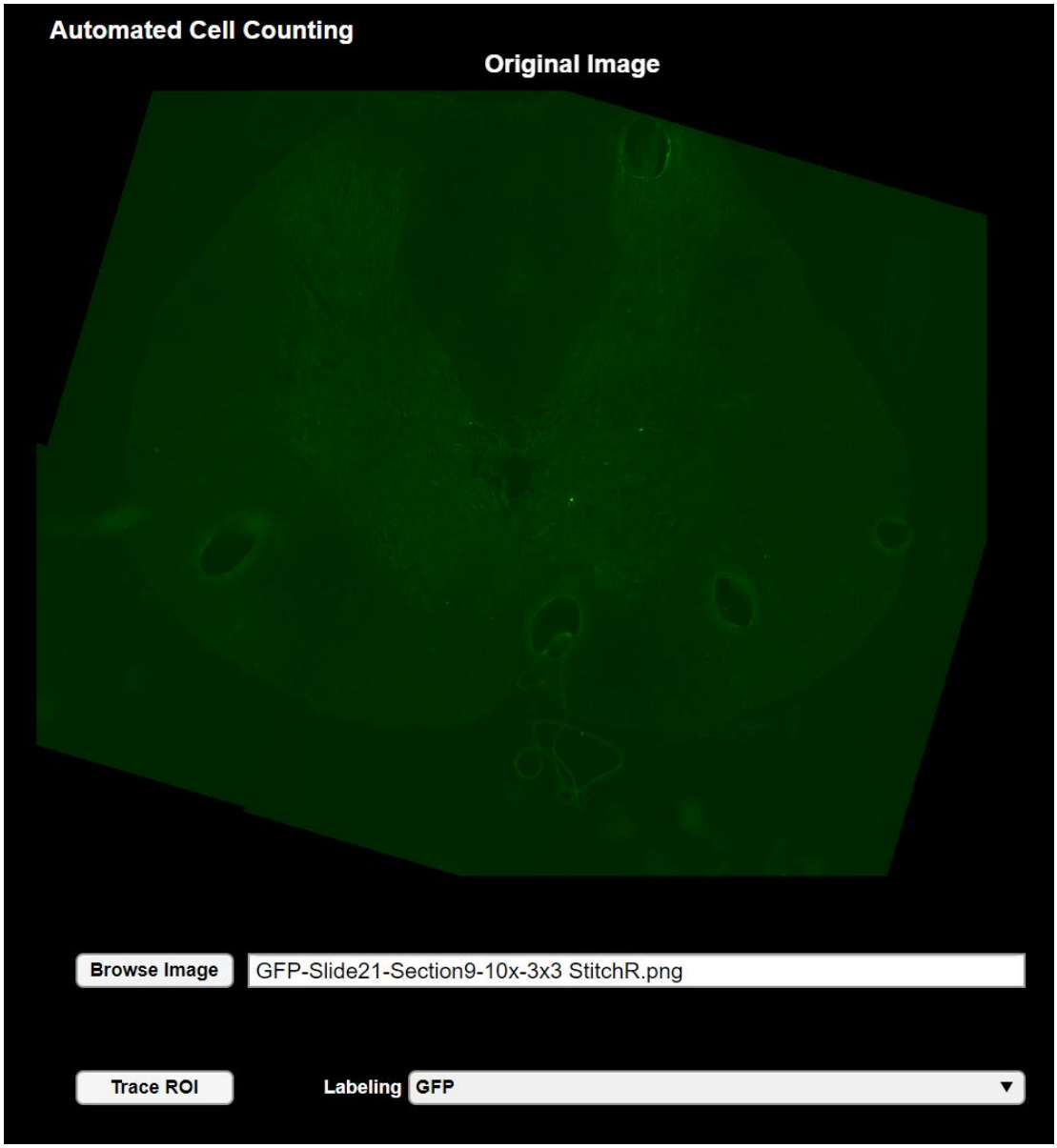

FIGURE 4 – Selected Image and File Name Displayed

The next function is the 'TraceROIButtonPushed' function. When the "Trace ROI" button is pushed, a separate figure window with the selected image is displayed. The user can then trace the region of interest in which labeled cells should be counted. For the images in this tracing study, the right side of the gray matter should be outlined because the injections were unilateral. Once the ROI has been traced, each pixel outside of that outline is programmed to turn

black, eliminating extraneous background. The new image with eliminated background is displayed within the separate figure window, and the original image in the axis below the "Original Image" label is replaced with the new image. The new traced image data are stored as the property 'ROI.' Next to the "Trace ROI" button, there is a drop-down list used to select the type of labeling being analyzed. The two options on the list are 'GFP' and 'FluoroRuby.' The selection will determine how the color thresholding is performed in the next step. The interface with the new traced image and the drop-down list options displayed is shown in **Figure 5** below.

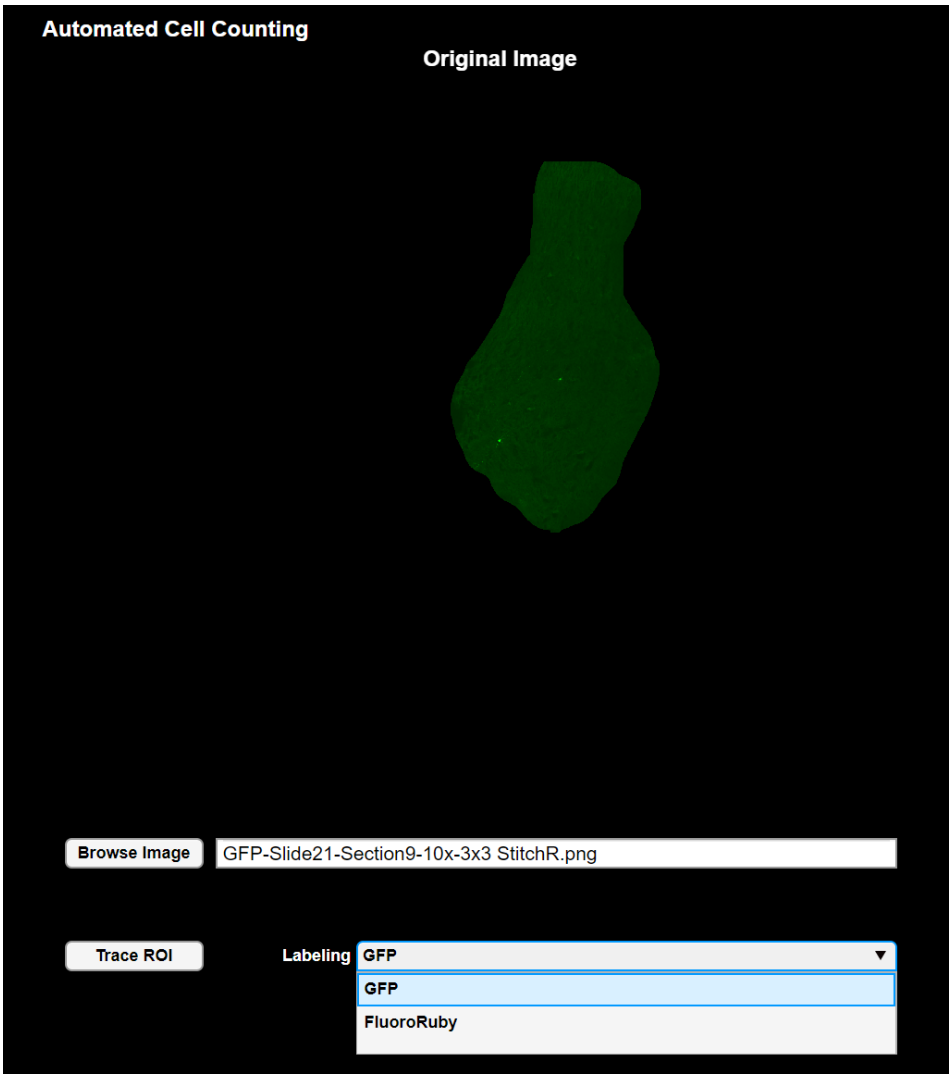

FIGURE 5 – Screenshot with Traced Image and Drop-Down List Displayed

iii. Color Thresholding and Manual Processing Functions

The image must be further processed before cells can be automatically detected and counted. The 'ColorThresholdSliderValueChanged' function enables the user to eliminate the remaining background, leaving only the labeled cells. The traced image is originally stored as an RGB, or "truecolor" image within the property 'ROI.' RGB image data includes *m*-by-*n*-by-3 data array that defines red, green, and blue color components of each individual pixel. When the value of the color threshold slider is changed, the program first extracts the image pixels within the color channel that correspond to the labeling selected using the drop-down list. If 'GFP' is selected, only the pixels within the green channel are extracted. If 'FluoroRuby' is selected, only the pixels within the red channel are extracted.

When the pixels are extracted, the RGB image is automatically converted to grayscale, and the values of the pixels are converted based upon the selected threshold value. The threshold value is determined by the position of the slider labeled "Color Threshold." The color threshold slider values range from 0-255. A pixel value of 0 is equivalent to the color black in a grayscale image, while 255 is equivalent to white. Every pixel value below the threshold value is converted to a pixel value of 0, so that the pixels are converted to black. Finally, the thresholded image is converted to binary so that every pixel with a non-zero value is converted to a value of 1. The binary image data is then stored within the property 'thresholdedImage' and displayed within the second axis placed above the color threshold slider as shown below in **Figure 6**. The image object is stored as the property 'full_threshold.'

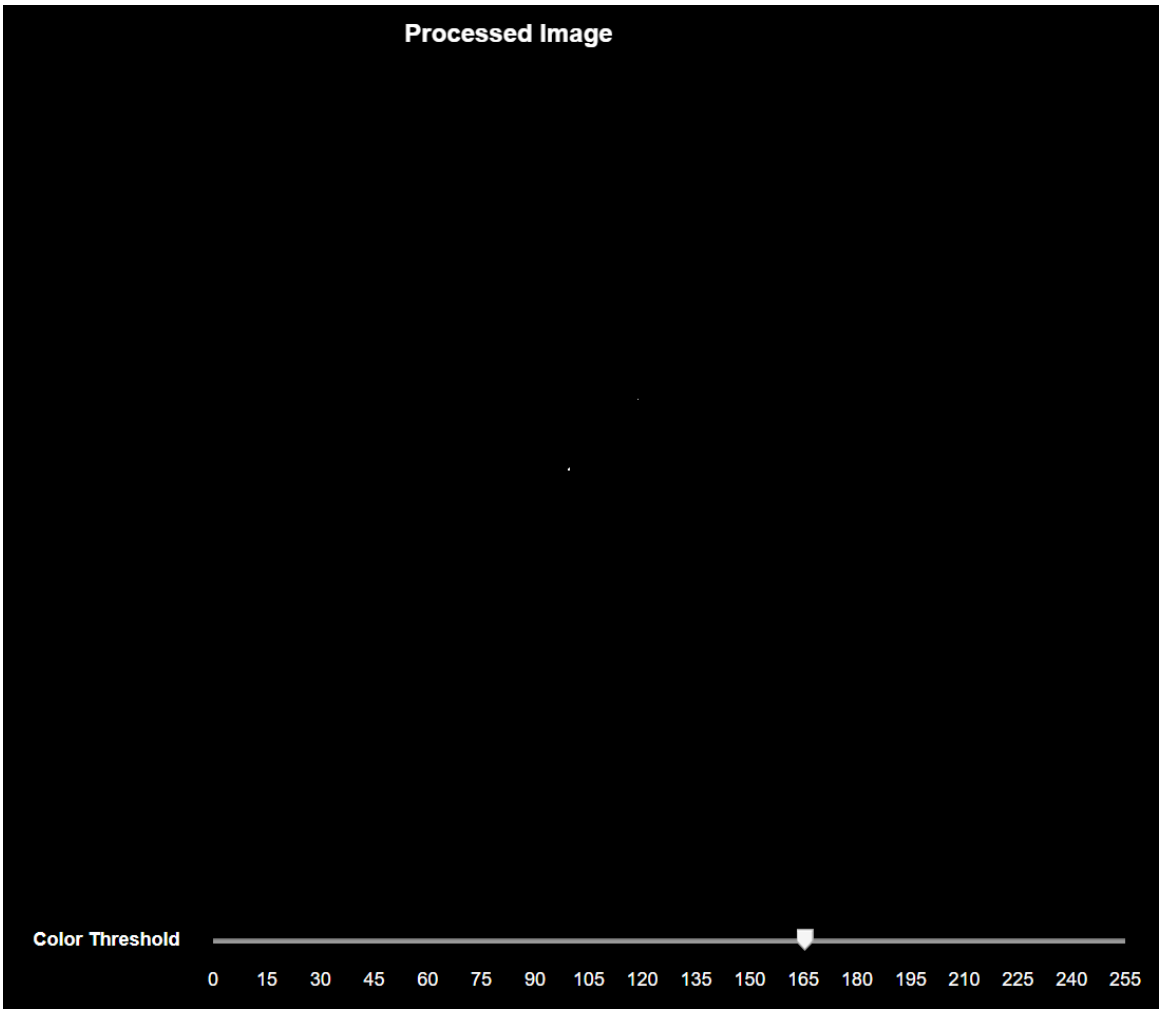

FIGURE 6 – Screenshot of Binary Image

After the traced image has been thresholded and converted to binary, the 'ManualProcessingButtonPushed' function can be utilized to remove artifacts in the image that could be mistaken for labeled cells. When the manual processing button is pushed, a separate figure window appears containing the thresholded image. The user can then trace around the artifact, and the pixels inside the outlined region will be converted to black, with a pixel value of 0. The image with the blacked-out region will then replace the thresholded image in the second axis above the color threshold slider. If an image has been manually processed, the image data

for the processed image is stored in the property ‘processedROI,’ and the check box adjacent to the manual processing button becomes filled. In order to display the functionality of the manual processing button, an object that is not a clear artifact has been outlined to be removed from the image. The figure window with the outlined region is shown in **Figure 7** below.

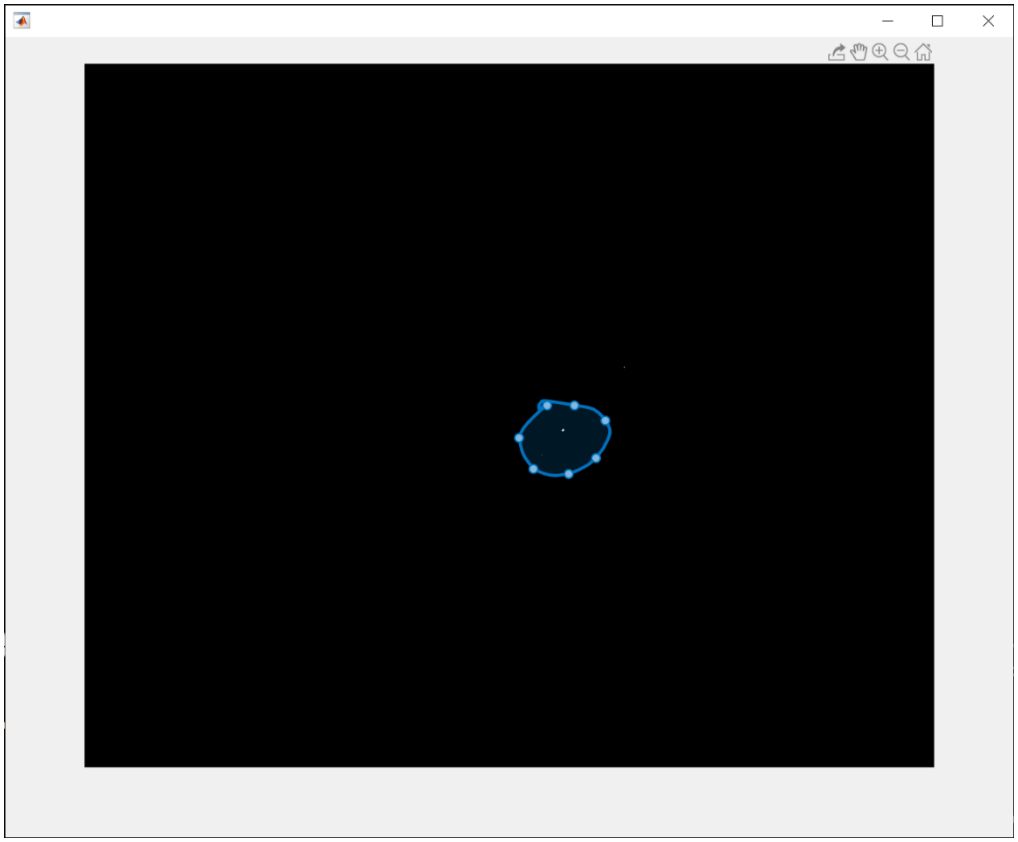

FIGURE 7 – Screenshot of Manual Processing Figure Window

iv. Lamina Overlay and Count Functions

The next function is the ‘SelectLaminaButtonPushed’ function. When the button labeled “Select Lamina” is pushed, the file selector is displayed, allowing the user to select a lamina overlay image. The file selector enables 8-bit images of file type ‘png’, ‘jpeg’, ‘bmp,’ and ‘tif’ to be selected. Once the lamina overlay has been selected, the file name is displayed in the edit field

adjacent to the “Select Lamina” button. The image data of the lamina overlay are stored as the property ‘overlay.’ Any lamina image can be uploaded to the program, but will only work as overlay if the background of the image has been removed previously. Lamina images ready to be used currently include laminae T12 – L4.

The ‘DisplayOverlayButtonPushed’ function enables the user to view the selected lamina overlaid onto the original spinal cord section image. When the “Display Overlay” button is pushed, a figure window appears displaying the original image with a lamina overlay, positioned according to the size and coordinate values specified by the user. The numeric edit fields pictured in **Figure 8** allow the user to specify the “Width” and “Height” of the lamina overlay, the coordinate position of the overlay within the x and y axes, and the rotation of the overlay in degrees. The width and height correspond to the row and column dimensions of the overlaid image.

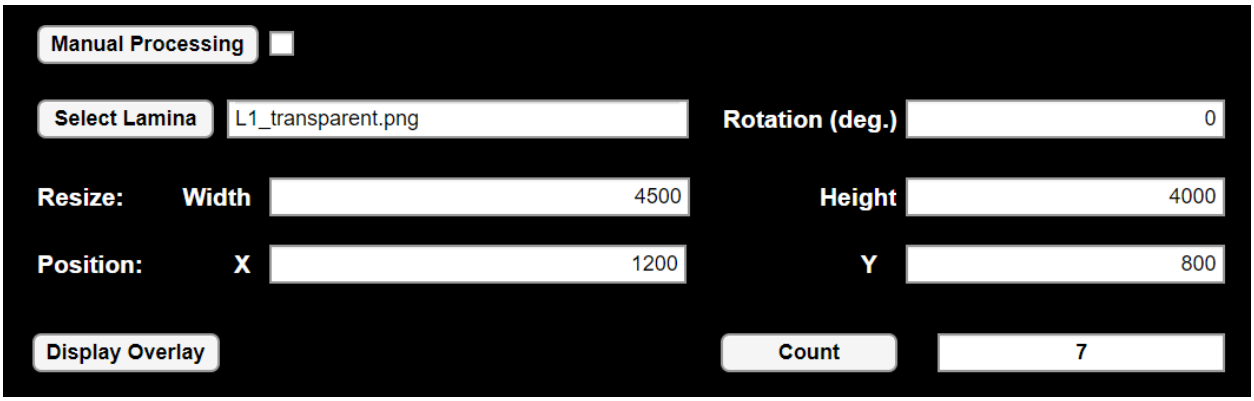

FIGURE 8 – Screenshot of Resize and Position Edit Fields

Once the dimensions and position of the lamina overlay have been specified, the original image with a lamina overlay is displayed, as shown in **Figure 9.** The size and coordinates of the lamina overlay can be updated, and the image re-displayed as needed.

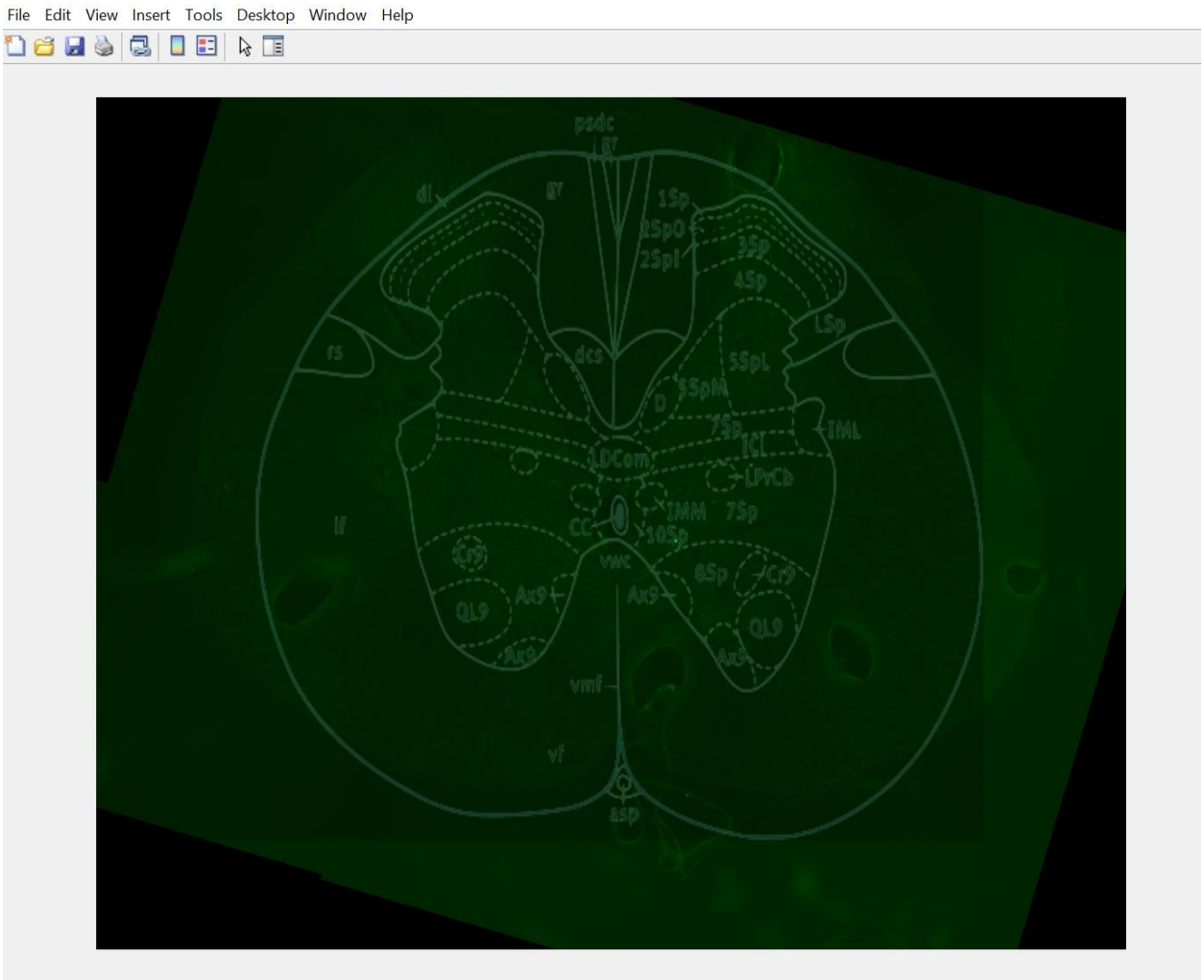

FIGURE 9 – Original Image with Lamina Overlay

The ‘CountButtonPushed’ function automatically detects and counts labeled cells based on the thresholded binary image. When the “Count” button is pushed, the application is programmed to utilize image region properties and MATLAB’s ‘ncount’ function to only count regions that have a pixel area greater than 100 and an eccentricity less than 0.97. Pixel area is the actual number of pixels within a region, while eccentricity is the ratio of the distance between the foci of the ellipse and its major axis length. Eccentricity values range from 0 to 1, 0 being a perfect circle and 1 being a line segment. The eccentricity threshold value was included in the

count function so that dendrites without a visible cell body are not miscounted as labeled cells, because dendrite shape resembles a line segment. The threshold values for pixel area and eccentricity were determined after 10 test images were run through the application, and the number and location of automatically counted cells was directly compared to the number and location of manually counted cells for each image. The number of counted cells is displayed in the numeric edit field adjacent to the "Count" button. An *elseif* expression is programmed within the 'CountButtonPushed' function so that the data stored in the 'thresholdedImage' property is called when the check box adjacent to the manual processing button is not filled, indicating that the image has not been manually processed. If a check appears in the box, indicating an image has been manually processed, the image data stored in the 'processedROI' will be called and counted.

After the number of regions, or cells, has been counted, MATLAB's 'bwboundaries' function is utilized to trace the exterior boundaries of the counted cells. A *for* loop is utilized so that for every region in the image, the properties are called to determine the pixel area and eccentricity, and then the regions that have an area >100 pixels and an eccentricity < 0.97 are both counted and traced with a red boundary outline. A new figure is then displayed with two subplots. The subplot on the right of the figure displays the thresholded (or thresholded and manually processed) image with the image regions, or labeled cells, outlined. The subplot on the left of the figure displays the original spinal section image with the lamina overlay, as is displayed when the "Display Overlay" button is pushed. As shown in **Figure 10**, the user can zoom in on either subplot to determine in which lamina of the spinal cord the cells are located.

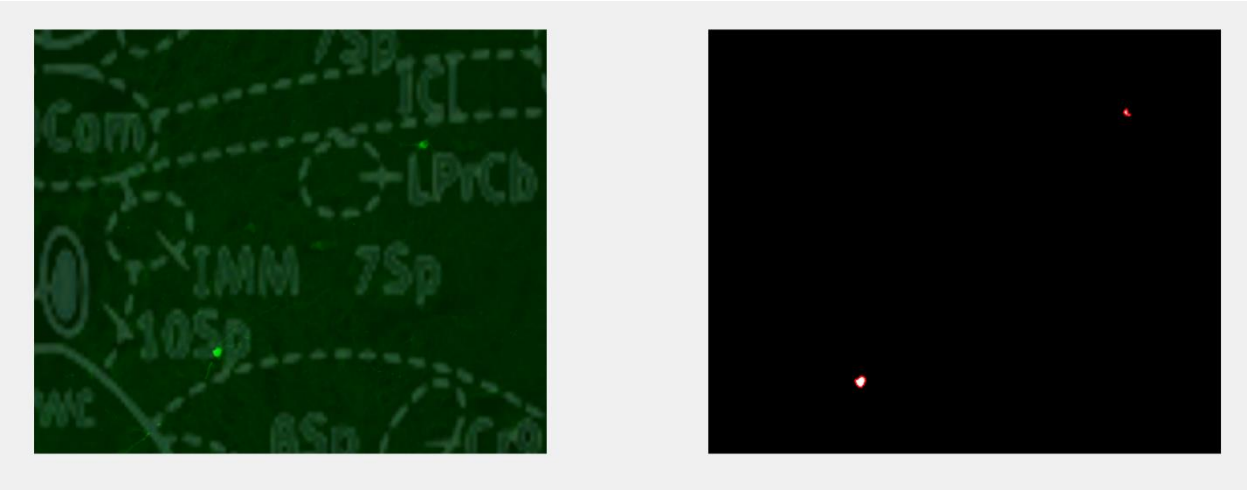

FIGURE 10 – Lamina Overlay and Image Regions with Boundaries

Finally, the ‘DirectionsButtonPushed’ function enables the user to view directions detailing application use. When button labeled “Directions” is pushed, a figure appears with an information icon and detailed directions for reference, as shown in **Figure 11** below.

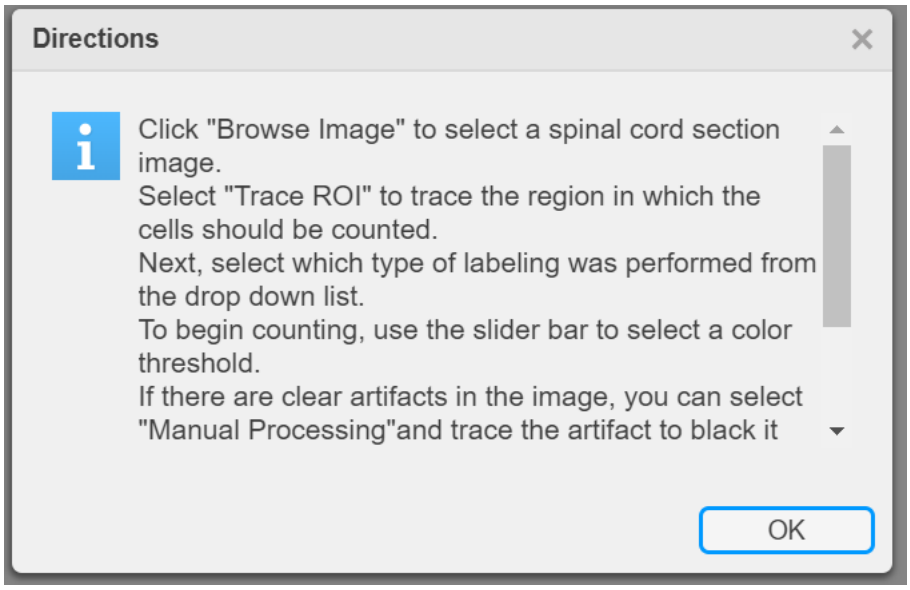

FIGURE 11 – Screenshot of Directions Pop-up Window

To validate the accuracy of the custom MATLAB application, every image that was manually counted was also run through the application. The number and location of labeled cells for each image analyzed using the application was recorded in Excel and then directly compared to the number and location of labeled cells from manual counting.

III. Results

A. Evaluating Efficiency of Chemical and Viral Tracing

To directly compare the efficiency of chemical tracers versus dual-viral systems for labeling LAPNs, both the total number of neurons from each group and the number of neurons per section for each group were compared. One animal from the HiRet-Lenti-Cre virus group was removed from analysis, as labeling at the lumbar injection site of this animal only spanned one-half the rostral-caudal distance seen in all other viral labeled animals, it is likely that one of the lumbar injection sites was missed. The statistical differences seen between groups were not altered by removing this animal from analysis.

An analysis of variance (ANOVA) and Tukey post-hoc test were performed using the program 'R' to compare the total number of LAPNs labeled from each group (group mean ± standard deviation; retroAAV: 31.5 ± 10.15, HiRet: 125.33 ± 45.796, FR: 135.5 ± 52.29). The total number of neurons labeled was significantly lower in the retro-AAV-Cre virus group compared to both the HiRet-Lenti-Cre virus group and the FR group (retroAAV vs. HiRet: $p = .000507$, retroAAV vs. FR: $p = .000116$). The mean number of neurons was not significantly different between the HiRet-Lenti-Cre virus group and the FR group (HiRet vs. FR: $p = .953487$). These comparisons are displayed in **Figure 12** below.

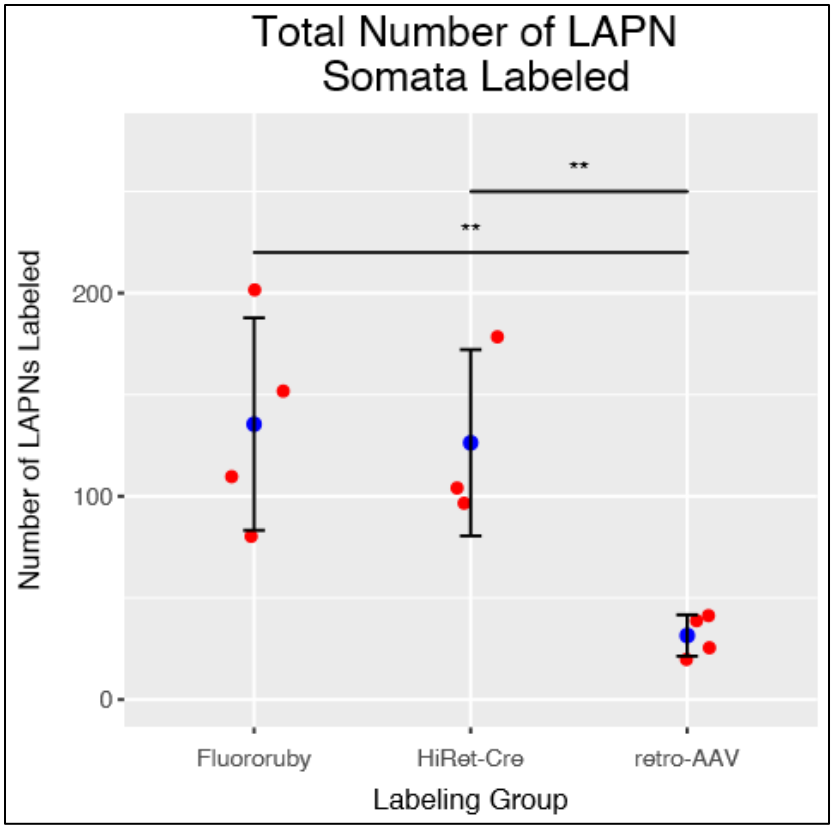

FIGURE 12 – Total LAPNs labeled by labeling group. Blue dots represent group means, red dots represent data from individual animals, and error bars represent ±1 SD, **p<.01

An ANOVA and Tukey post-hoc test were also performed using the program 'R' to compare the number of neurons labeled per section for each group (group mean ± standard deviation; retroAAV: 0.5025 ± 0.134, HiRet: 2.35 ± 0.466, FR: 2.117 ± 0.764). The mean number of neurons per section was significantly lower in the retro-AAV-Cre virus group compared to both the HiRet-Lenti-Cre virus group and the FR group (retroAAV vs. HiRet: p = .0001, retroAAV vs. FR: p = .0001). The mean number of neurons per section was not significantly different between the HiRet-Lenti-Cre virus group and the FR group (HiRet vs. FR: p = .832). These comparisons are displayed in **Figure 13** below.

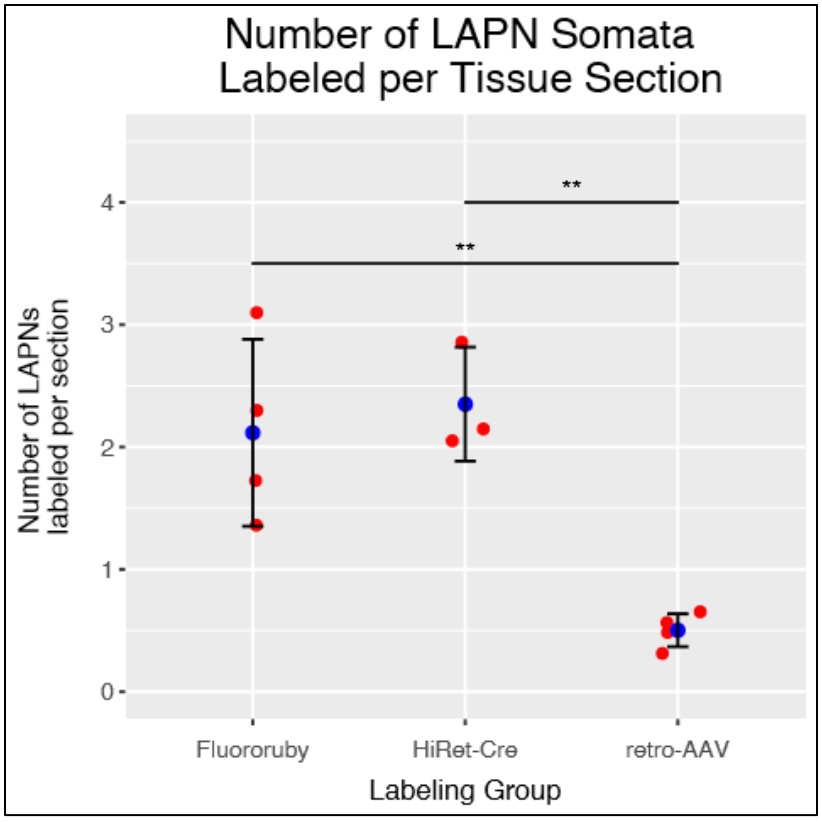

FIGURE 13 – LAPNs labeled per spinal cord tissue section by labeling group. Blue dots represent group means, red dots represent data from individual animals, and error bars represent ± 1 SD, **p<.01

B. MATLAB Application Validation

All images that were manually counted were also run through the custom MATLAB application to determine the accuracy of the program. Difference scores were calculated by subtracting the number of cells counted by the MATLAB program from the number of cells counted manually. An analysis of variance (ANOVA) was performed using the program 'R' to compare the difference scores between groups, and determine if there was greater error in any of the groups (mean of difference scores ± standard deviation; retroAAV: -1.75 ± 6.95, HiRet: -0.5

± 13.18, FR: 2.75 ± 12.84). The number of LAPNs counted manually was not significantly different from the number of LAPNs counted by the MATLAB application in any labeling group (ANOVA p = .848) . An animal-by-animal comparison of counting methods is shown in **Figure 14** below.

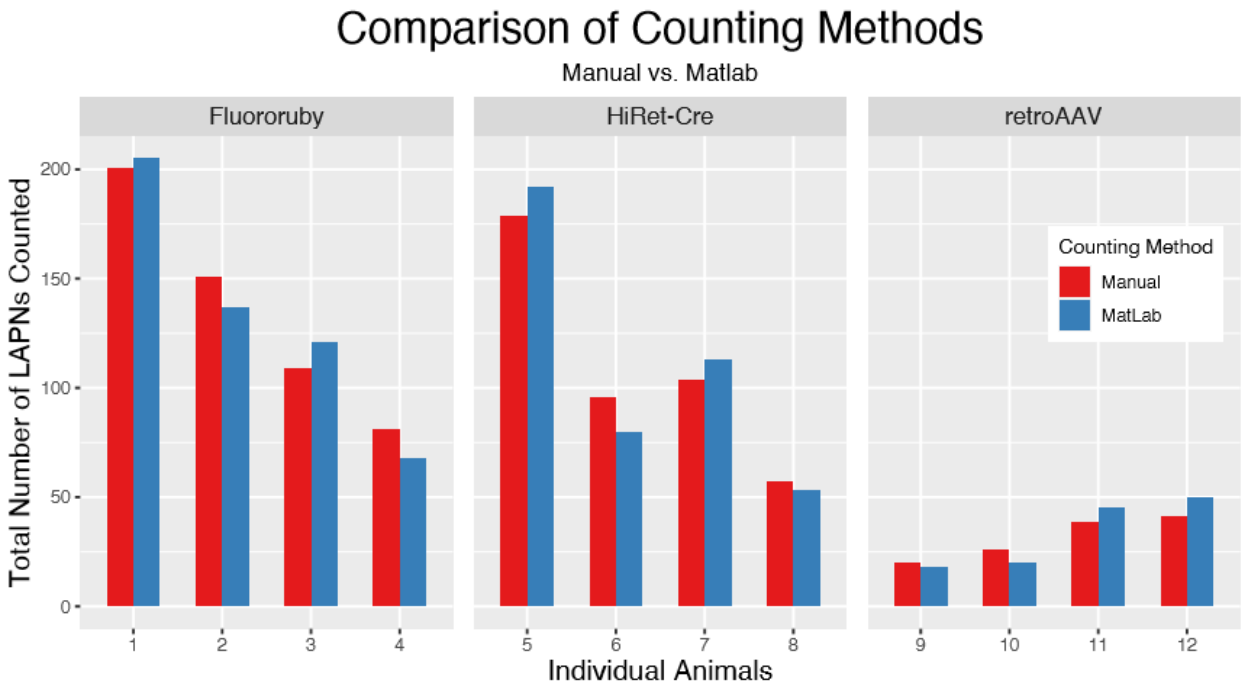

FIGURE 14 – Comparison of manual counting and MATLAB counting of LAPNs for each animal.

The percent changes between counting methods were also calculated for each animal and are shown in **Table III**. Negative percentages indicate a decrease in the number of LAPNs counted by the MATLAB application, and positive percentages indicate an increase in the number of LAPNs counted by the MATLAB application.

TABLE III

PERCENT CHANGES BETWEEN COUNTING METHODS

Individual Percent Changes Between Counting Methods		
Animal Number	**Labeling Group**	**Percent Change from Manual to MatLab Counting**
1	Fluororuby	1.99%
2	Fluororuby	-9.27%
3	Fluororuby	11.01%
4	Fluororuby	-16.05%
5	HiRet-Cre	7.26%
6	HiRet-Cre	-16.67%
7	HiRet-Cre	8.65%
8	HiRet-Cre	-7.02%
9	retroAAV	-10.00%
10	retroAAV	-23.08%
11	retroAAV	15.38%
12	retroAAV	21.95%

The relationship between the two counting methods was directly analyzed using a correlation, as shown in **Figure 15** (Pearson R value = 0.99; $p = 3.8 \times 10^{-9}$). There was a strong relationship between manual counting and MATLAB counting, as well as a statistically significant correlation.

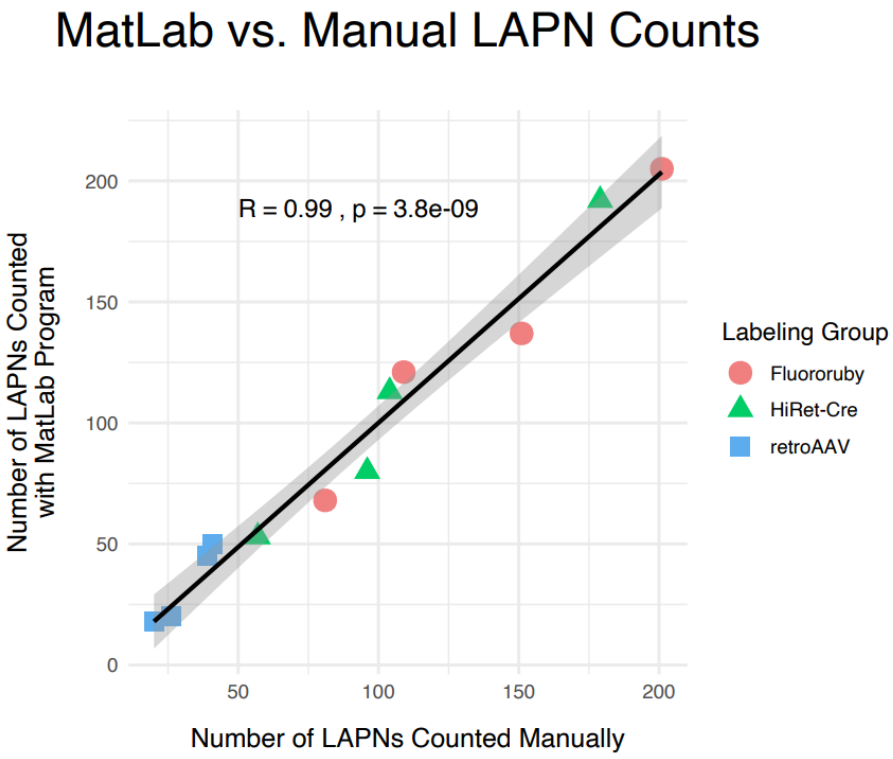

FIGURE 15 – Correlation between manual and MATLAB counting methods. Black line indicates trend line, and light gray represents 95% confidence interval.

III. Discussion

Based on preliminary work and previous literature, we hypothesized that 1) the HiRet-Lenti group would label and identify greater numbers of LAPNs than retro-AAV group, 2) the HiRet-Lenti group would provide greater specificity than FR, and 3) FR would label more neurons than either dual-viral labeling group. As expected, both the HiRet-Lenti and FluoroRuby groups labeled significantly greater numbers of LAPNs than the retro-AAV group. These results show that despite the retro-AAV being a robust tool for tracing cortical neurons and their projections[16], retro-AAV is inefficient for labeling propriospinal neurons such as LAPNs. However, dual-viral labeling utilizing the HiRet-lentiviruses, such as the HiRet-Cre used here, are a more robust and reliable means of labeling propriospinal neurons than retro-AAV.

Unexpectedly, there was no difference between the number of LAPNs labeled between the FluoroRuby and HiRet-Lenti groups. This is likely due to the similar rostral-caudal spread of the volumes/doses of FluoroRuby and the viruses that were injected. During preliminary work, a 0.25 µl bolus of FluoroRuby was injected ipsilaterally into the C5/6 intermediate spinal gray matter of one animal. The animal was sacrificed after one week after the injection, and the spinal cord was dissected. The C5/6 segment was then cryosectioned longitudinally and slide mounted. Microscopy images acquired using the Nikon TiE 300 inverted microscope revealed that the volume of FluoroRuby injected spread rostral-caudally one spinal segment at the injection site, approximately 1.1mm. This is an equivalent rostral-caudal spread to the spread of the of the viral doses injected. For consistency between the FluoroRuby and viral tracing groups, 0.25 µl of FluoroRuby was injected in the chemical tracing group (n = 4).

Although HiRet-Lenti labeled the same number of neurons as FR, dual-viral tracing utilizing the HiRet-lentivirus provides more specificity than traditional chemical tracing with FR. Previous work from our lab compared the laminar distribution of LAPNs using dual-viral labeling with HiRet-Lenti-Cre versus chemical tracing with cholera toxin b (CTB) (**Figure 16 A&B**). Heat maps and contour plots were generated using a different custom MATLAB program to show the laminar distribution of LAPNs (**Figure 16 C-F**). The laminar distributions were similar between the two groups, however virus labeling was more specific. Most of the neurons labeled by HiRet-Lenti were in lamina 6,7,8, while the distribution of neurons labeled by CTB varied between laminas 5,6,7, and 8 (**Figure 16 G**).

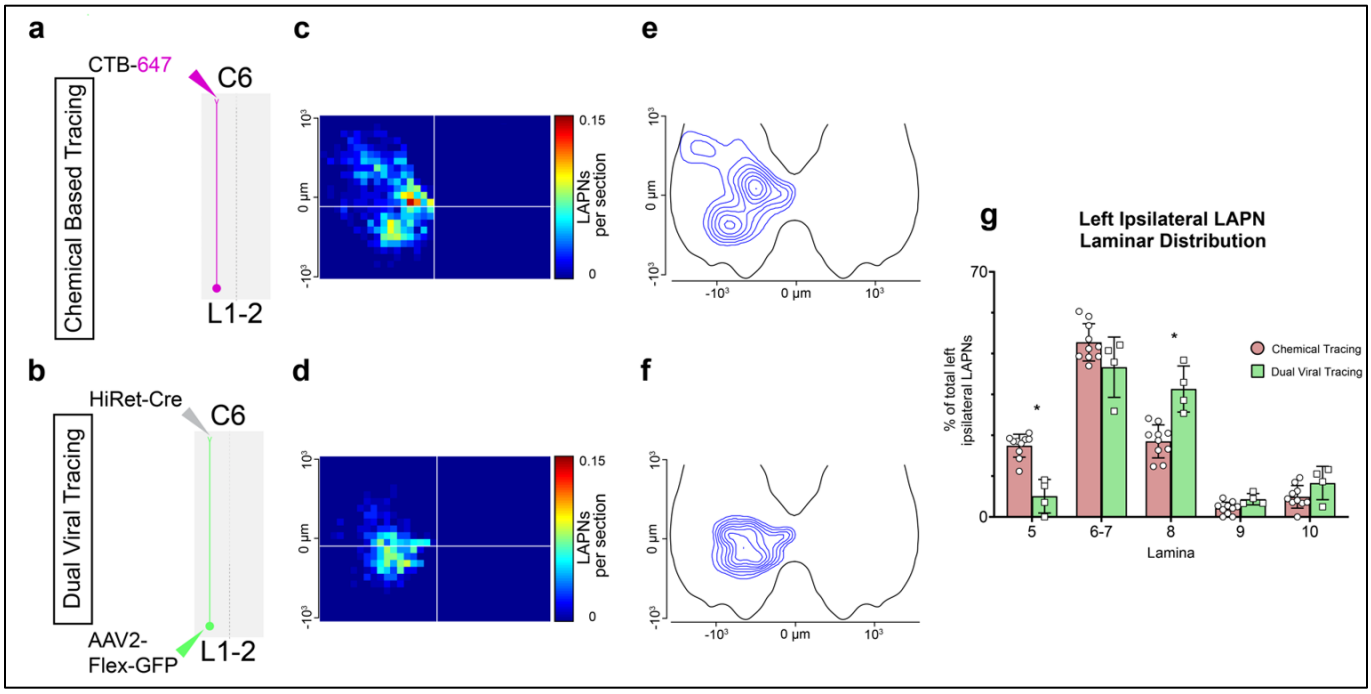

FIGURE 16 – a. CTB injection. b. Dual-virus injection of HiRet-Lenti-Cre and AAV2. c, d. MATLAB generated heat maps. e, f. MATLAB generated contour plots. g. Laminar distribution of LAPNs.

The similar numbers of LAPNs labeled by HiRet-Lenti and FR seen here is likely due to similar rostral-caudal spread of FR and viruses at the injection site(s). However, the HiRet-Lenti

dual-viral system provides a greater specificity of labeling, as the expression of EGFP is dependent on the presence of both injected viruses, while FR can be taken up by fibers of passage in addition to the axon terminals of the target neuronal population.[12,13] Additionally, LAPNs labeled by HiRet-Lenti were brighter, more prominent, and typically easier to identify than LAPNs labeled by FR. Thus, despite the number of LAPNs labeled being similar, the dual-viral labeling utilizing HiRet-lentiviruses is preferred due to greater specificity and more prominent labeling. Although chemical tracers such as FR or CTB are less specific than the dual-viral systems, chemical tracers can be useful for studies that combine axonal tract tracing with electrophysiological recording, such as in facial nerves.[18]

We also found that dual-viral systems utilizing retro-AAVs, such as the retro-AAV2-Cre virus used here, do not provide robust labeling when tracing long propriospinal neurons in rats. This is somewhat expected as the goal of the directed evolution of the retro-AAV was to infect the axon terminals of corticopontine neurons in mice and label the somata of these corticopontine neurons.[16] The low number of LAPNs labeled by the dual-viral system utilizing the retro-AAV is likely attributed to either low infectivity of propriospinal axon terminals and/or poor retrograde transport of the virus. LAPN axons in rats are 7 cm long, which is approximately 10 times longer than of the mouse corticopontine axons (6-7mm long) that the retro-AAV was developed to target.[16] This finding emphasizes the need to empirically test, characterize, and optimize individual viruses used for labeling a neuronal population of interest. However, the retro-AAV is still an effective tracing tool when targeting cortical neurons with short projections, such as corticopontine neurons.[16] Based on our findings, dual-viral systems utilizing a HiRet-Lentivirus at the level of the axon terminals and AAV2 containing a FLEx switch at the level of the cell

bodies confers robust, specific, and prominent labeling compared to other tracing methods when targeting propriospinal neurons such as LAPNs.

For MATLAB application validation, each image that was manually counted was run through the custom MATLAB application. The number of LAPNs counted manually did not differ from the number of LAPNs counted by the MATLAB application between labeling groups. The significant correlation between manual counting and MATLAB counting also indicated a strong relationship between methods, and further emphasizes the accuracy of the MATLAB application. The calculated percentage changes per animal were highest for the retro-AAV group, which is to be expected as the overall number of LAPNs counted was the lowest for that group. Based on these results, the custom MATLAB application accurately determined the number of ipsilateral cell bodies labeled by each of the tracing techniques analyzed in this experiment. The interactive application is also free and user-friendly, allowing the user to navigate through a large number of images and overlay different laminae with ease. Overall, the program provides an automatic, efficient, and unbiased method of counting cells in spinal cord tissue sections.

REFERENCES

1. “Spinal Cord Injury Facts and Figures at a Glance.” *National Spinal Cord Injury Statistical Center*, The University of Alabama at Birmingham, 2020, www.nscisc.uab.edu/.

2. Kjell, J., & Olson, L. (2016). Rat models of spinal cord injury: from pathology to potential therapies. *Disease models & mechanisms*, *9*(10), 1125–1137. https://doi.org/10.1242/dmm.025833

3. Pocratsky, A. M., Burke, D. A., Morehouse, J. R., Beare, J. E., Riegler, A. S., Tsoulfas, P., States, G., Whittemore, S. R., & Magnuson, D. (2017). Reversible silencing of lumbar spinal interneurons unmasks a task-specific network for securing hindlimb alternation. *Nature communications*, *8*(1), 1963. https://doi.org/10.1038/s41467-017-02033-x

4. Kiehn O. Locomotor circuits in the mammalian spinal cord. *Annu Rev Neurosci*. 2006;29:279-306. doi:10.1146/annurev.neuro.29.051605.112910

5. Juvin L, Simmers J, Morin D. Propriospinal circuitry underlying interlimb coordination in mammalian quadrupedal locomotion. *J Neurosci*. 2005;25(25):6025-6035. doi:10.1523/JNEUROSCI.0696-05.2005

6. Laliberte AM, Goltash S, Lalonde NR, Bui TV. Propriospinal Neurons: Essential Elements of Locomotor Control in the Intact and Possibly the Injured Spinal Cord. *Front* Cell Neurosci. 2019;13:512. Published 2019 Nov 12. doi:10.3389/fncel.2019.00512

7. Reed, W. R., Shum-Siu, A., Onifer, S. M., & Magnuson, D. S. (2006). Inter-enlargement pathways in the ventrolateral funiculus of the adult rat spinal cord. *Neuroscience*, *142*(4), 1195–1207. https://doi.org/10.1016/j.neuroscience.2006.07.017

8. Courtine, G., Song, B., Roy, R. R., Zhong, H., Herrmann, J. E., Ao, Y., Qi, J., Edgerton, V. R., & Sofroniew, M. V. (2008). Recovery of supraspinal control of stepping via indirect propriospinal relay connections after spinal cord injury. *Nature medicine*, *14*(1), 69–74. https://doi.org/10.1038/nm1682

9. Kinoshita M, Matsui R, Kato S, et al. Genetic dissection of the circuit for hand dexterity in primates. *Nature*. 2012;487(7406):235-238. doi:10.1038/nature11206

10. Nassi JJ, Cepko CL, Born RT, Beier KT. Neuroanatomy goes viral!. *Front*

Neuroanat. 2015;9:80. Published 2015 Jul 1. doi:10.3389/fnana.2015.00080

11. Kumar TR, Larson M, Wang H, McDermott J, Bronshteyn I. Transgenic mouse technology: principles and methods. Methods Mol Biol. 2009;590:335-362. doi:10.1007/978-1-60327-378-7_22

12. Dado, R. J., Burstein, R., Cliffer, K. D., & Giesler Jr, G. J. (1990). Evidence that Fluoro-Gold can be transported avidly through fibers of passage. *Brain research*, 533(2), 329-333.

13. Chen, S., & Aston-Jones, G. (1995). Evidence that cholera toxin B subunit (CTb) can be avidly taken up and transported by fibers of passage. *Brain research*, 674(1), 107-111.

14. Atasoy D, Aponte Y, Su HH, Sternson SM. A FLEX switch targets Channelrhodopsin-2 to multiple cell types for imaging and long-range circuit mapping. *J Neurosci*. 2008;28(28):7025-7030. doi:10.1523/JNEUROSCI.1954-08.2008

15. Kato S, Kobayashi K, Inoue K, et al. A lentiviral strategy for highly efficient retrograde gene transfer by pseudotyping with fusion envelope glycoprotein. *Hum Gene Ther*. 2011;22(2):197-206. doi:10.1089/hum.2009.179

16. Tervo DG, Hwang BY, Viswanathan S, et al. A Designer AAV Variant Permits Efficient Retrograde Access to Projection Neurons. *Neuron*. 2016;92(2):372-382. doi:10.1016/j.neuron.2016.09.021

17. Schofield BR. Retrograde axonal tracing with fluorescent markers. Curr Protoc Neurosci. 2008;Chapter 1:. doi:10.1002/0471142301.ns0117s43

APPENDIX I

MATLAB Programming

```
classdef MEng_AppDesigner < matlab.apps.AppBase
    % Properties that correspond to app components
    properties (Access = public)
        UIFigure                    matlab.ui.Figure
        UIAxes_2                    matlab.ui.control.UIAxes
        BrowseImageButton           matlab.ui.control.Button
        FileName                    matlab.ui.control.EditField
        DirectionsButton            matlab.ui.control.Button
        TraceROIButton              matlab.ui.control.Button
        ColorThresholdLabel         matlab.ui.control.Label
        ColorThresholdSlider        matlab.ui.control.Slider
        CountButton                 matlab.ui.control.Button
        DropDown                    matlab.ui.control.DropDown
        UIAxes                      matlab.ui.control.UIAxes
        AutomatedCellCountingLabel  matlab.ui.control.Label
        EditField                   matlab.ui.control.NumericEditField
        ManualProcessingButton      matlab.ui.control.Button
        CheckBox                    matlab.ui.control.CheckBox
        OriginalImageLabel          matlab.ui.control.Label
        ProcessedImageLabel         matlab.ui.control.Label
        Labeling                    matlab.ui.control.Label
        SelectLaminaButton          matlab.ui.control.Button
        EditField_2                 matlab.ui.control.EditField
        WidthEditFieldLabel         matlab.ui.control.Label
        WidthEditField              matlab.ui.control.NumericEditField
        HeightEditFieldLabel        matlab.ui.control.Label
        HeightEditField             matlab.ui.control.NumericEditField
        ResizeLabel                 matlab.ui.control.Label
        XEditField                  matlab.ui.control.NumericEditField
        XEditFieldLabel             matlab.ui.control.Label
        YEditField                  matlab.ui.control.NumericEditField
        YEditFieldLabel             matlab.ui.control.Label
        PositionLabel               matlab.ui.control.Label
        DisplayOverlayButton        matlab.ui.control.Button
        RotationdegEditFieldLabel   matlab.ui.control.Label
        RotationdegEditField        matlab.ui.control.NumericEditField
    end
    properties (Access = private)
        Image %Browsed image
        full_img %img before tracing ROI
        ROI %image after tracing
        folder %folder with lamina overlay images
        thresholdedImage %image after being thresholded
        full_threshold %image before processing
        processedROI %image after processing
        overlay %browsed lamina overlay
    end
```

```
    % Callbacks that handle component events
    methods (Access = private)
        % Code that executes after component creation
        function startupFcn(app)
            %starts the app at fullscreen automatically
            drawnow;
            app.UIFigure.WindowState = 'maximized';
        end
        % Button pushed function: BrowseImageButton
        function BrowseImageButtonPushed(app, event)
            [filename, pathname] = uigetfile({'*.png';'*.jpg';'*.bmp';
'*.tif'},'File Selector');
            select_image = strcat(pathname, filename);
            I = imread(select_image);
            img = imshow(I,'Parent',app.UIAxes_2);
            app.Image = I;
            app.full_img = img;
            assignin('base','filename',filename);
            assignin('base','pathname',pathname);
            %display pathname and filename
            app.FileName.Value = filename
        end
        % Button pushed function: DirectionsButton
        function DirectionsButtonPushed(app, event)
            % Directions on how to use app
            f = uifigure;
            message = sprintf(['Click "Browse Image" to select a spinal cord
section image.' ...
                '\nSelect "Trace ROI" to trace the region in which the cells
should be counted.' ...
                '\nNext, select which type of labeling was performed from the
drop down list.' ...
                '\nTo begin counting, use the slider bar to select a color
threshold.' ...
                '\nIf there are clear artifacts in the image, you can select
"Manual Processing"' ...
                'and trace the artifact to black it out.\nWhen you are ready to
overlay lamina,' ...
                'click "Select Lamina" to choose the overlay. \nFinally, click
"Count" to automatically ' ...
                'count the labeled cells and display the lamina overlay.']);
            uialert(f,message,'Directions','Icon','info');
        end
        % Button pushed function: TraceROIButton
        function TraceROIButtonPushed(app, event)
            %drawing freehand ROI
            imshow(app.Image)
            h = drawfreehand
            h.FaceAlpha = 0;
```

```
    h.FaceSelectable = false;
    %blacking out non-ROI portion of image
    BW = createMask(h,app.full_img);
    BW(:,:,2) = BW;
    BW(:,:,3) = BW(:,:,1);
    ROI = app.Image;
    ROI(BW == 0) = 0;
    %showing ROI in App Axes
    imshow(ROI);
    imshow(ROI,'Parent',app.UIAxes_2);
    imwrite(ROI,'tracedImage.png');
    traced_ROI = imread('tracedImage.png');
    app.ROI = traced_ROI;

end
% Button pushed function: CountButton
function CountButtonPushed(app, event)
    value = app.EditField_2.Value;
    figure;
    subplot(1,2,1)
    imshow(app.Image)
    overlay = imread(app.overlay);
    height = app.HeightEditField.Value;
    width = app.WidthEditField.Value;
    xval = app.XEditField.Value;
    yval = app.YEditField.Value;
    rot = app.RotationdegEditField.Value;
    overlay = imresize(overlay, [height width]);
    overlay2 = imrotate(overlay, rot,'crop');
    hold on
    J = imtranslate(overlay2,[xval, yval],'OutputView','full');
    f3 = imshow(J)
    set(f3,'AlphaData',0.2);
  if app.CheckBox.Value == 1
    subplot(1,2,2)
    imdata_threshold = app.processedROI;
    imshow(imdata_threshold);
    %Traces region boundaries
    [B,L] = bwboundaries(imdata_threshold);
    hold on;
    ncount = 0;
    stats2 = regionprops(L,'Area');
    for k=1:length(B)
        boundary = B{k};
        obj_area2 = stats2(k).Area;
        if obj_area2 > 200
            plot(boundary(:,2),boundary(:,1),'r','LineWidth',1);
            ncount = ncount + 1
        end
    end
    app.EditField.Value = ncount
  elseif app.CheckBox.Value == 0
```

```
    subplot(1,2,2)
    imdata_threshold = app.thresholdedImage;
    imshow(imdata_threshold);
    %Traces region boundaries
    [B,L] = bwboundaries(imdata_threshold);
    hold on;
    ncount = 0;
    stats2 = regionprops(L,'Area');
    stats3 = regionprops(L,'Eccentricity');
for k=1:length(B)
    boundary = B{k};
    obj_area2 = stats2(k).Area;
    obj_area3 = stats3(k).Eccentricity;
    if obj_area2 > 100 & obj_area3 < .97
       plot(boundary(:,2),boundary(:,1),'r','LineWidth',1);
       ncount = ncount + 1
    end
end
    app.EditField.Value = ncount
  end
end
% Value changed function: ColorThresholdSlider
function ColorThresholdSliderValueChanged(app, event)
    value = app.ColorThresholdSlider.Value;
    if strcmp(app.DropDown.Value,'GFP')
    %Extracting image in green channel
    imdata = app.ROI;
    imdata_green = imdata(:,:,2);
    %Changing every gray level value less than selected value to black
    imdata_green(find(imdata_green<value)) = 0;
    full_threshold = imshow(imdata_green,'Parent',app.UIAxes);
    imwrite(imdata_green,'thresholdedImage.png');
    thresholded_image = imread('thresholdedImage.png');
    app.thresholdedImage = thresholded_image;
    app.full_threshold = full_threshold;
    elseif strcmp(app.DropDown.Value,'FluoroRuby')
    %Extracting image in green channel
    imdata = app.ROI;
    imdata_green = imdata(:,:,1);
    %Changing every gray level value less than selected value to black
    imdata_green(find(imdata_green<value)) = 0;
    full_threshold = imshow(imdata_green,'Parent',app.UIAxes);
    imwrite(imdata_green,'thresholdedImage.png');
    thresholded_image = imread('thresholdedImage.png');
    app.thresholdedImage = thresholded_image;
    app.full_threshold = full_threshold;
    end
end
% Button pushed function: ManualProcessingButton
function ManualProcessingButtonPushed(app, event)
    %addressing the problem of double counting cells
    %blacking out ROI portion of the image
```

```
            ax = axes('Parent', uifigure)
            imshow(app.thresholdedImage,'Parent',ax)

            %drawing freehand ROI
            h = drawfreehand(ax)
            h.FaceAlpha = 0;
            h.FaceSelectable = false;
            %blacking out non-ROI portion of image
            BW = createMask(h,app.full_threshold);
            insideMasked = app.thresholdedImage;
            insideMasked(BW) = 0;
            %showing ROI in App Axes
            imshow(insideMasked,'Parent',ax);
            imshow(insideMasked,'Parent',app.UIAxes);
            imwrite(insideMasked,'processedImage.png');
            processed_ROI = imread('processedImage.png');
            app.processedROI = processed_ROI;
            %app.processedROI = app.thresholdedImage
            app.CheckBox.Value = 1
        end
        % Button pushed function: SelectLaminaButton
        function SelectLaminaButtonPushed(app, event)
            [filename, pathname] =
uigetfile({'*.png';'*.jpg';'*.bmp';'*.tif'},'File Selector');
            select_lamina = strcat(pathname, filename);
            app.overlay = select_lamina;
            app.EditField_2.Value = filename
        end
        % Button pushed function: DisplayOverlayButton
        function DisplayOverlayButtonPushed(app, event)
            figure
            imshow(app.Image)
            overlay = imread(app.overlay);
            height = app.HeightEditField.Value;
            width = app.WidthEditField.Value;
            xval = app.XEditField.Value;
            yval = app.YEditField.Value;
            rot = app.RotationdegEditField.Value;
            overlay = imresize(overlay, [height width]);
            overlay2 = imrotate(overlay, rot,'crop');
            hold on
            J = imtranslate(overlay2,[xval, yval],'OutputView','full');
            f3 = imshow(J)
            set(f3,'AlphaData',0.2);
        end
    end
    % Component initialization
    methods (Access = private)
        % Create UIFigure and components
        function createComponents(app)
            % Create UIFigure and hide until all components are created
            app.UIFigure = uifigure('Visible', 'off');
```

```
            app.UIFigure.Color = [0 0 0];
            app.UIFigure.Position = [100 100 796 654];
            app.UIFigure.Name = 'UI Figure';
            % Create UIAxes_2
            app.UIAxes_2 = uiaxes(app.UIFigure);
            title(app.UIAxes_2, '')
            xlabel(app.UIAxes_2, '')
            ylabel(app.UIAxes_2, '')
            app.UIAxes_2.Box = 'on';
            app.UIAxes_2.XTick = [];
            app.UIAxes_2.YTick = [];
            app.UIAxes_2.BackgroundColor = [0 0 0];
            app.UIAxes_2.Position = [24 321 366 281];
            % Create BrowseImageButton
            app.BrowseImageButton = uibutton(app.UIFigure, 'push');
            app.BrowseImageButton.ButtonPushedFcn = createCallbackFcn(app,
@BrowseImageButtonPushed, true);
            app.BrowseImageButton.FontWeight = 'bold';
            app.BrowseImageButton.Position = [54 271 100 22];
            app.BrowseImageButton.Text = 'Browse Image';
            % Create FileName
            app.FileName = uieditfield(app.UIFigure, 'text');
            app.FileName.FontSize = 14;
            app.FileName.Position = [163 271 169 22];
            % Create DirectionsButton
            app.DirectionsButton = uibutton(app.UIFigure, 'push');
            app.DirectionsButton.ButtonPushedFcn = createCallbackFcn(app,
@DirectionsButtonPushed, true);
            app.DirectionsButton.FontWeight = 'bold';
            app.DirectionsButton.Position = [54 23 100 22];
            app.DirectionsButton.Text = {'Directions'; ''};
            % Create TraceROIButton
            app.TraceROIButton = uibutton(app.UIFigure, 'push');
            app.TraceROIButton.ButtonPushedFcn = createCallbackFcn(app,
@TraceROIButtonPushed, true);
            app.TraceROIButton.FontWeight = 'bold';
            app.TraceROIButton.Position = [54 197 100 22];
            app.TraceROIButton.Text = 'Trace ROI';
            % Create ColorThresholdLabel
            app.ColorThresholdLabel = uilabel(app.UIFigure);
            app.ColorThresholdLabel.HorizontalAlignment = 'right';
            app.ColorThresholdLabel.FontWeight = 'bold';
            app.ColorThresholdLabel.FontColor = [1 1 1];
            app.ColorThresholdLabel.Position = [402 271 98 22];
            app.ColorThresholdLabel.Text = {'Color Threshold'; ''};
            % Create ColorThresholdSlider
            app.ColorThresholdSlider = uislider(app.UIFigure);
            app.ColorThresholdSlider.Limits = [0 255];
            app.ColorThresholdSlider.ValueChangedFcn = createCallbackFcn(app,
@ColorThresholdSliderValueChanged, true);
            app.ColorThresholdSlider.FontColor = [1 1 1];
            app.ColorThresholdSlider.Position = [521 280 254 3];
```

```
            % Create CountButton
            app.CountButton = uibutton(app.UIFigure, 'push');
            app.CountButton.ButtonPushedFcn = createCallbackFcn(app,
@CountButtonPushed, true);
            app.CountButton.FontWeight = 'bold';
            app.CountButton.Position = [609 23 100 22];
            app.CountButton.Text = 'Count';
            % Create DropDown
            app.DropDown = uidropdown(app.UIFigure);
            app.DropDown.Items = {'GFP', 'FluoroRuby', ''};
            app.DropDown.FontWeight = 'bold';
            app.DropDown.BackgroundColor = [0.9412 0.9412 0.9412];
            app.DropDown.Position = [285 197 47 22];
            app.DropDown.Value = 'GFP';
            % Create UIAxes
            app.UIAxes = uiaxes(app.UIFigure);
            title(app.UIAxes, '')
            xlabel(app.UIAxes, '')
            ylabel(app.UIAxes, '')
            app.UIAxes.Box = 'on';
            app.UIAxes.XTick = [];
            app.UIAxes.YTick = [];
            app.UIAxes.TitleFontWeight = 'normal';
            app.UIAxes.BackgroundColor = [0 0 0];
            app.UIAxes.Position = [402 321 366 281];
            % Create AutomatedCellCountingLabel
            app.AutomatedCellCountingLabel = uilabel(app.UIFigure);
            app.AutomatedCellCountingLabel.HorizontalAlignment = 'center';
            app.AutomatedCellCountingLabel.FontSize = 16;
            app.AutomatedCellCountingLabel.FontWeight = 'bold';
            app.AutomatedCellCountingLabel.FontColor = [1 1 1];
            app.AutomatedCellCountingLabel.Position = [35 622 198 22];
            app.AutomatedCellCountingLabel.Text = 'Automated Cell Counting';
            % Create EditField
            app.EditField = uieditfield(app.UIFigure, 'numeric');
            app.EditField.HorizontalAlignment = 'center';
            app.EditField.FontWeight = 'bold';
            app.EditField.Position = [732 23 31 22];
            % Create ManualProcessingButton
            app.ManualProcessingButton = uibutton(app.UIFigure, 'push');
            app.ManualProcessingButton.ButtonPushedFcn = createCallbackFcn(app,
@ManualProcessingButtonPushed, true);
            app.ManualProcessingButton.FontWeight = 'bold';
            app.ManualProcessingButton.Position = [413 197 125 22];
            app.ManualProcessingButton.Text = 'Manual Processing';
            % Create CheckBox
            app.CheckBox = uicheckbox(app.UIFigure);
            app.CheckBox.Text = '';
            app.CheckBox.Position = [544 192 46 33];
            % Create OriginalImageLabel
            app.OriginalImageLabel = uilabel(app.UIFigure);
            app.OriginalImageLabel.HorizontalAlignment = 'center';
```

```
            app.OriginalImageLabel.FontSize = 16;
            app.OriginalImageLabel.FontWeight = 'bold';
            app.OriginalImageLabel.FontColor = [1 1 1];
            app.OriginalImageLabel.Position = [149 601 116 22];
            app.OriginalImageLabel.Text = 'Original Image';
            % Create ProcessedImageLabel
            app.ProcessedImageLabel = uilabel(app.UIFigure);
            app.ProcessedImageLabel.HorizontalAlignment = 'center';
            app.ProcessedImageLabel.FontSize = 16;
            app.ProcessedImageLabel.FontWeight = 'bold';
            app.ProcessedImageLabel.FontColor = [1 1 1];
            app.ProcessedImageLabel.Position = [544 601 137 22];
            app.ProcessedImageLabel.Text = 'Processed Image';
            % Create Labeling
            app.Labeling = uilabel(app.UIFigure);
            app.Labeling.FontWeight = 'bold';
            app.Labeling.FontColor = [1 1 1];
            app.Labeling.Position = [232 197 108 22];
            app.Labeling.Text = 'Labeling';
            % Create SelectLaminaButton
            app.SelectLaminaButton = uibutton(app.UIFigure, 'push');
            app.SelectLaminaButton.ButtonPushedFcn = createCallbackFcn(app,
@SelectLaminaButtonPushed, true);
            app.SelectLaminaButton.FontWeight = 'bold';
            app.SelectLaminaButton.Position = [413 155 100 22];
            app.SelectLaminaButton.Text = 'Select Lamina';
            % Create EditField_2
            app.EditField_2 = uieditfield(app.UIFigure, 'text');
            app.EditField_2.Position = [520 155 71 22];
            % Create WidthEditFieldLabel
            app.WidthEditFieldLabel = uilabel(app.UIFigure);
            app.WidthEditFieldLabel.HorizontalAlignment = 'right';
            app.WidthEditFieldLabel.FontSize = 14;
            app.WidthEditFieldLabel.FontWeight = 'bold';
            app.WidthEditFieldLabel.FontColor = [1 1 1];
            app.WidthEditFieldLabel.Position = [490 111 44 22];
            app.WidthEditFieldLabel.Text = 'Width';
            % Create WidthEditField
            app.WidthEditField = uieditfield(app.UIFigure, 'numeric');
            app.WidthEditField.Position = [544 111 48 22];
            % Create HeightEditFieldLabel
            app.HeightEditFieldLabel = uilabel(app.UIFigure);
            app.HeightEditFieldLabel.HorizontalAlignment = 'right';
            app.HeightEditFieldLabel.FontSize = 14;
            app.HeightEditFieldLabel.FontWeight = 'bold';
            app.HeightEditFieldLabel.FontColor = [1 1 1];
            app.HeightEditFieldLabel.Position = [660 111 49 22];
            app.HeightEditFieldLabel.Text = 'Height';
            % Create HeightEditField
            app.HeightEditField = uieditfield(app.UIFigure, 'numeric');
            app.HeightEditField.Position = [714 111 49 22];
            % Create ResizeLabel
```

```
            app.ResizeLabel = uilabel(app.UIFigure);
            app.ResizeLabel.FontSize = 14;
            app.ResizeLabel.FontWeight = 'bold';
            app.ResizeLabel.FontColor = [1 1 1];
            app.ResizeLabel.Position = [413 111 54 22];
            app.ResizeLabel.Text = {'Resize:'; ''};
            % Create XEditField
            app.XEditField = uieditfield(app.UIFigure, 'numeric');
            app.XEditField.Position = [544 73 46 22];
            % Create XEditFieldLabel
            app.XEditFieldLabel = uilabel(app.UIFigure);
            app.XEditFieldLabel.HorizontalAlignment = 'right';
            app.XEditFieldLabel.FontSize = 14;
            app.XEditFieldLabel.FontWeight = 'bold';
            app.XEditFieldLabel.FontColor = [1 1 1];
            app.XEditFieldLabel.Position = [509 73 25 22];
            app.XEditFieldLabel.Text = 'X';
            % Create YEditField
            app.YEditField = uieditfield(app.UIFigure, 'numeric');
            app.YEditField.Position = [714 73 49 22];
            % Create YEditFieldLabel
            app.YEditFieldLabel = uilabel(app.UIFigure);
            app.YEditFieldLabel.HorizontalAlignment = 'right';
            app.YEditFieldLabel.FontSize = 14;
            app.YEditFieldLabel.FontWeight = 'bold';
            app.YEditFieldLabel.FontColor = [1 1 1];
            app.YEditFieldLabel.Position = [673 73 25 22];
            app.YEditFieldLabel.Text = 'Y';
            % Create PositionLabel
            app.PositionLabel = uilabel(app.UIFigure);
            app.PositionLabel.FontSize = 14;
            app.PositionLabel.FontWeight = 'bold';
            app.PositionLabel.FontColor = [1 1 1];
            app.PositionLabel.Position = [413 73 65 22];
            app.PositionLabel.Text = 'Position:';
            % Create DisplayOverlayButton
            app.DisplayOverlayButton = uibutton(app.UIFigure, 'push');
            app.DisplayOverlayButton.ButtonPushedFcn = createCallbackFcn(app,
@DisplayOverlayButtonPushed, true);
            app.DisplayOverlayButton.FontWeight = 'bold';
            app.DisplayOverlayButton.Position = [410 23 106 22];
            app.DisplayOverlayButton.Text = 'Display Overlay';
            % Create RotationdegEditFieldLabel
            app.RotationdegEditFieldLabel = uilabel(app.UIFigure);
            app.RotationdegEditFieldLabel.HorizontalAlignment = 'right';
            app.RotationdegEditFieldLabel.FontSize = 14;
            app.RotationdegEditFieldLabel.FontWeight = 'bold';
            app.RotationdegEditFieldLabel.FontColor = [1 1 1];
            app.RotationdegEditFieldLabel.Position = [605 155 104 22];
            app.RotationdegEditFieldLabel.Text = 'Rotation (deg.)';
            % Create RotationdegEditField
            app.RotationdegEditField = uieditfield(app.UIFigure, 'numeric');
```

```
            app.RotationdegEditField.Position = [714 155 49 22];


            % Show the figure after all components are created
            app.UIFigure.Visible = 'on';
        end
    end
    % App creation and deletion
    methods (Access = public)
        % Construct app
        function app = MEng_AppDesigner
            % Create UIFigure and components
            createComponents(app)
            % Register the app with App Designer
            registerApp(app, app.UIFigure)
            % Execute the startup function
            runStartupFcn(app, @startupFcn)
            if nargout == 0
                clear app
            end
        end
        % Code that executes before app deletion
        function delete(app)
            % Delete UIFigure when app is deleted
            delete(app.UIFigure)
        end
    end
end
```