Astrocyte Segmentation and Classification

David Carlyn, Lana Frankle, Akhil Kumar Goud Koothal
Kent State University, Computer Science Department

Abstract

Our plugin determines which voxels are part of the astrocyte by calling Connected Threshold Grower from ImageJ. Connected Threshold Grower works by inputting a manually selected seed point and assessing its 26 neighboring voxels against a pre-set intensity threshold. If the neighboring voxel is above that threshold, the seed point is shifted to that voxel, and so on, until all voxels indirectly connected to the original seed point that are above the intensity threshold are segmented as belonging to the astrocyte. After defining the astrocyte, the astrocytic voxel whose six faces are flesh with strings of voxels that extend the furthest in all directions - ie, the centermost voxel - is selected. This determines the voxel threshold - the number of voxels extending from the centermost voxel in all directions before first reaching a background (non-astrocyte) voxel. Our algorithm defines this as the radius of the cell, and all previously segmented voxels that are within that radius are defined as body voxels, all other voxels segmented as part of the cell are defined as process or branch voxels.

Introduction

Our task was to develop an algorithm that could segment astrocytes from each other and from the background of images taken using confocal microscopy, and then to process their features and compile data on the following metrics: total volume, cell body volume, process length, volume, and bifurcation number. This has applications for image analysis in neuroscience research, particularly in assessing the status of astrocytes as healthy versus “activated” in disease models. Existing automated detection of astrocyte morphology is fairly limited relative to algorithms analyzing neurons or oligodendrocytes, which is why this served as an area of interest. Nuclei of both neurons and astrocytes, as well as their nucleoli, can be segmented by a macro developed by a previous student[1], but few algorithms exist to assess cell body volume and process number, length, and branching, and existing algorithms have many flaws.

Methods

Previous Plugin

The previous plugin that was built upon was able to segment the astrocyte as a whole and separate the astrocyte cell body from its branches. It would also run the Analyze Skeleton on the image to receive information about the branches and the bifurcations. The problem with the previous plugin is that it would not function many times, and often resulted in a compiler error. When the plugin did work, there would often be inaccurate results, which will be discussed in the next sections.

Preprocessing Correction

1. Previous Plugin

The previous plugin's method of preprocessing was not suitable for all images, so the code was not built upon, but instead we created our own preprocessing workflow

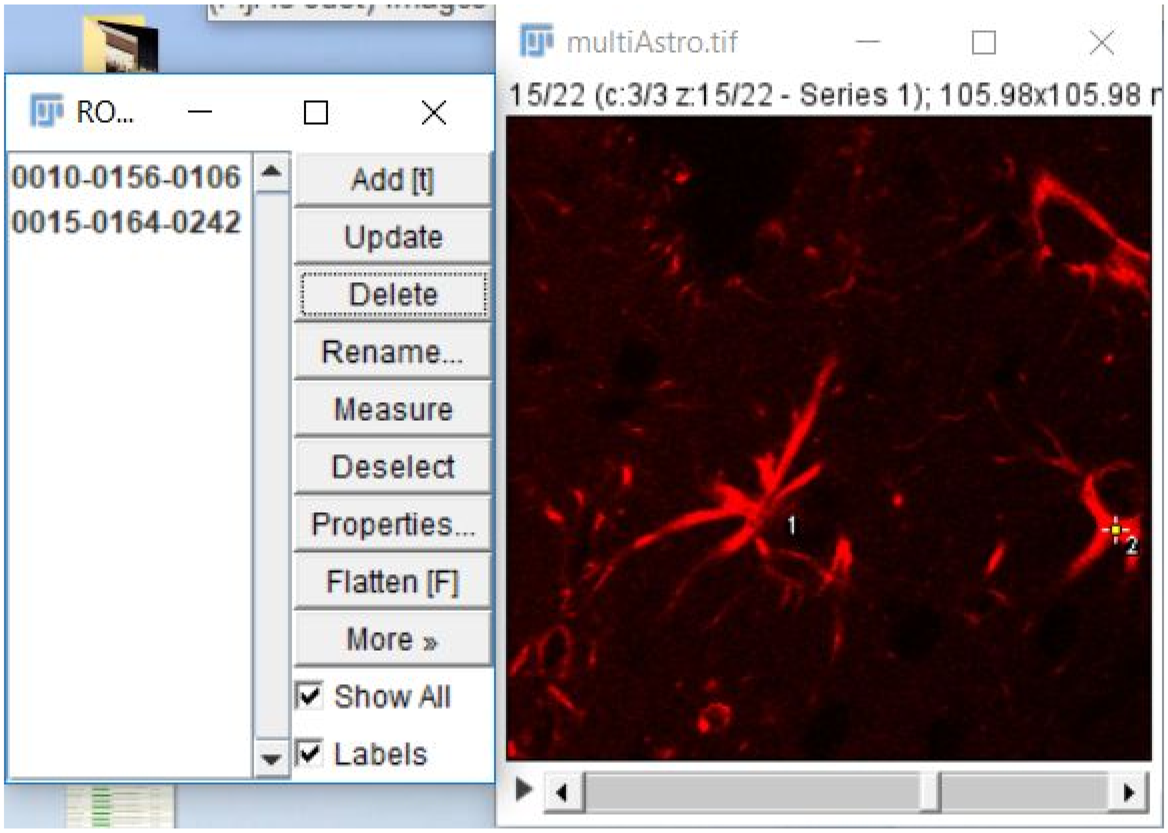
2. The Current Plugin

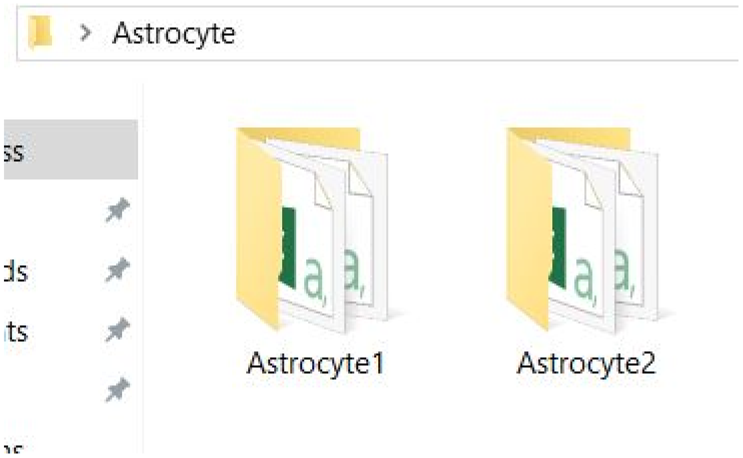
The current plugin scales the image to have an equal sized width, height, and depth. They are set to 1 centimeter for each dimension. The image is then converted to 8-bit to allow other plugins to be applied to the image. Next, the image is segmented using the Connected Threshold Grower which is a part of the 3D Tool Kit. Finally, the Fill Holes plugin is applied to the image to fill in any holes in the image that may have resulted from scanning errors. We are left with a binary image of the astrocyte in question.

Astrocyte Classification

1. Multiple Astrocytes

The previous plugin did not have anything for segmenting and classifying more than one astrocyte at a time. The current plugin makes use of the ROI Manager by looping through the points selected on the current image and performing the preprocessing and classification on each point.





2. Setting The Current Image

In the beginning of the classify method we set a reference to the current binary astrocyte image, and also a reference to the skeletonized version of the same image by running Skeletonize (2D/3D) plugin on it. We then set references to the width, height, and depth of the current astrocyte image.

3. Setting The Threshold

A critical part of classifying an astrocyte is analyzing its parts. There are two main parts that this plugin aims toward: the cell body and its branches. We attempt to define the cell body by first setting a radius threshold for the body. By finding a good radius for the cell body, we can segment out the cell body, which should make finding and analyzing the branches easier. To find this threshold we iterate through every point of the segmented binary image of the astrocyte. When a point is inside of the astrocyte, we then check all of its neighbors within a specified distance. This distance (radius) is considered the threshold, and it starts at one. We check pixels within a cubic region with a radius of that distance, and if all of them reside within that distance we then increment the threshold. This is repeated on the same point until we expand outside of the astrocyte. When this occurs we simply visit another point that hasn't been visited and perform the same process, but we start at the threshold that we ended at. The threshold will only be changed if there is a point that can have a cubic region inside of the astrocyte with a larger radius than the current threshold. At the end of this operation, we will have the threshold that represents the largest radius that some cubic region can have within the astrocyte region.

Note, that the only changes to this algorithm for this project were to constrict it to only checking within the bounds of the image. Before, the plugin would crash if there was an astrocyte near the edge of the image. This algorithm is set within the SetThreshold function.

4. Setting The Body Pixels

Once we have our threshold value, we then iterate through all the points within the image again. For each point that is in the image we check if we can create a cubic region with a radius of the threshold at that point that resides within the astrocyte. If even one pixel is outside of the segmented binary image of the astrocyte, then we ignore that point and move to the next. However, if all the points within the cubic region are within the astrocyte, then we add the current point to an array of points called body_pixels.

Note, this algorithm was also from the previous plugin and was only changed to avoid checking out of the bounds of the image as the plugin would crash because of this. This algorithm is implemented within the SetBodyPixels function.

5. Defining The Cell Body

We now will define and segment the cell body. We will iterate through the array of cell pixels, and for each point in this array we will create this cubic region with the threshold value as the radius. For every point in this cubic region, we set its voxel to a unique value to separate it from the rest of the astrocyte, the branches. For this plugin, we set the value as 100.0.

Note, the previous plugin accomplished this, we just corrected the algorithm to avoid checking out of the bounds of the image. This is accomplish in the ColorBody function.

6. Setting The Branch Voxels

We added a new function to set the intensity of every pixel within the binary image to a unique value. This is done, because we will copy the skeletonized image of the astrocyte on top of the current working image, which is, by default, has all of the voxels of the skeleton set to 255.0. This is the same value the branches had before we set them to a different value, in this case it is 200.0. This is simply done by iterating through the entire image and converting every point that doesn't have a voxel value of 0.0 or 100.0 and setting it to 200.0.

This algorithm is implemented within the SetOtherPixels function

7. Copy On The Skeleton

Here, we simply copy over the skeleton from our skeletonized image onto the current working image of the astrocyte. We only copy points that are not within the cell body. To do this, we iterate through all the points in the skeletonized image and add the points in the skeletonized image to the current working image only if the voxel on the current working image is not equal to 100.0 or 0.0. This will results in an image with the cell body with voxel values of 100.0, branches with voxel values set to 200.0 and with a skeleton with voxel values of 255.0.

Note, this was implemented in the previous plugin. The changes we made were avoiding checking outside of the bounds of the image. This algorithm also wouldn't work without the previous function (SetOtherPixels). So our additional function allowed this algorithm to work. This algorithm is implemented in the AddSkeleton function.

8. Getting Primary Branches

Now we segment each branch from each other. We currently have an image with a cell body with voxel values of 100.0, branches with voxel values of 200.0 and within those branches a skeleton of the branch with voxel values of 255.0. Each primary branch should have a skeleton within it that goes from the cell body to the end of the branch. To separate each branch, we could find all skeleton pixels (value of 255.0) that are immediate neighbors of the cell body, and then store each pixel as a reference to their respective primary branch. To do this we iterate through all the points who has a voxel value of 100.0 (in the cell body) and check the every pixel that touches the current point. Once we find a pixel with a voxel value of 255.0 (a branch skeleton pixel), we store that pixel in an array labeled ‘branch_pixels.’ Each pixel in this array will be a reference to a primary branch.

Note, this algorithm was implemented in the previous plugin; however we added checks for the bounds of the image, and significantly shortened the code to reduce errors and improve readability. We also added a check to avoid duplications. The previous algorithm didn’t check if the same pixel was added to the same array ‘branch_pixels.’ This could happen if the point is next to more than one cell body pixel. So, we created a function to check that the current point that is to be added is not in the array of ‘branch_pixels’ already. This algorithm is implemented in the SetBranches function.

9. Coloring Separate Branches

Now that we have an array of references to each distinct branch, we need to create a set of points for each primary branch. To do this we iterate through each point in the branch_pixels array. For each point, we will assign it a distinct color and give it an accumulator to count the amount of pixels inside of the branch (this will serve as the volume of the branch). Then we will perform a growing algorithm that point. This algorithm will use the point given as a starting point, and grab all of its neighbors and essentially put them in a queue (we use recursion for this), then it will color the voxel at that point with the unique color and increment the accumulator for the volume. We will only color points and increment the volume accumulator if the point’s voxel value doesn’t equal the cell body’s voxel value (100.0), is not currently colored to the unique value, and is not outside of the segmented astrocyte image. Once there are no more pixels to color, then we add the volume accumulator value to an array to access it later.

Note, this algorithm was implemented in the previous plugin; however we made corrections to avoid checking outside of the bounds of the image, significantly reducing the amount of code and simplifying it, and making it usable by having the branch voxel value set to 200.0 in a previous step. The unique color starts at 101.0 and increments by 1.0 for each primary branch. This gives us up to 99 possible colors to give each primary branch. However, the amount of primary branches an astrocyte has should never get anywhere close to this amount. This algorithm is implemented in the ColorBranches function.

10. Deleting Merged Branches

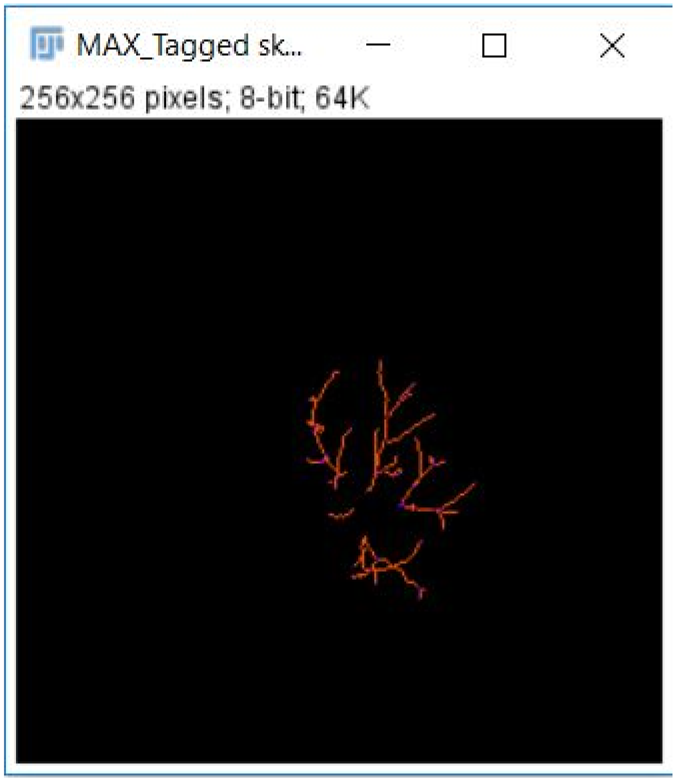
There are some cases where when finding the primary branches, two branches are connected outside of the cell body and may merge during this process. This happens when the cell body hasn’t been defined accurately, this will be discussed in a later section. To temporarily solve this issue and create consistent results, we delete any primary branched pixel that was merged into another. To do this we iterate through all the branch_pixels and delete the reference to the branch pixel if it is not colored with its unique color (because it was colored a different color by another branch in the previous section). Not only is the reference to the branch pixel delete but so are references to the color it had and the size of that branch.

Note that this was partially implement in the previous plugin. It was inconsistent though because it deleted from the arrays as it was moving forward through the array. This caused duplicate branches, because it would create skips in the the for loop. To fix this we switched the direction to iterate backward through the branch_pixels array. This algorithm is implemented in the DeletedMergedBranches function.

11. Saving The Results

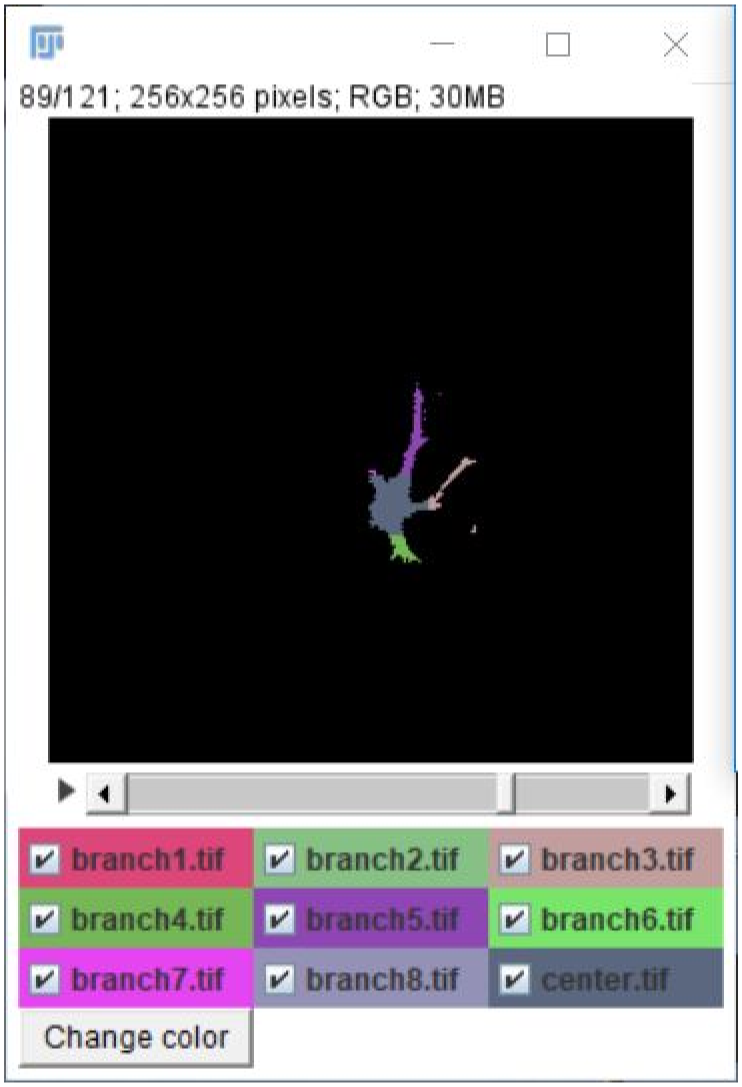
Now that we have all the information, we just need to save it. We create a folder on the Desktop of the user's computer and name it 'Astrocyte.' Inside the folder will be a folder for each astrocyte selected in the image the its information. The user will have to remove everything in the folder before using the plugin again.

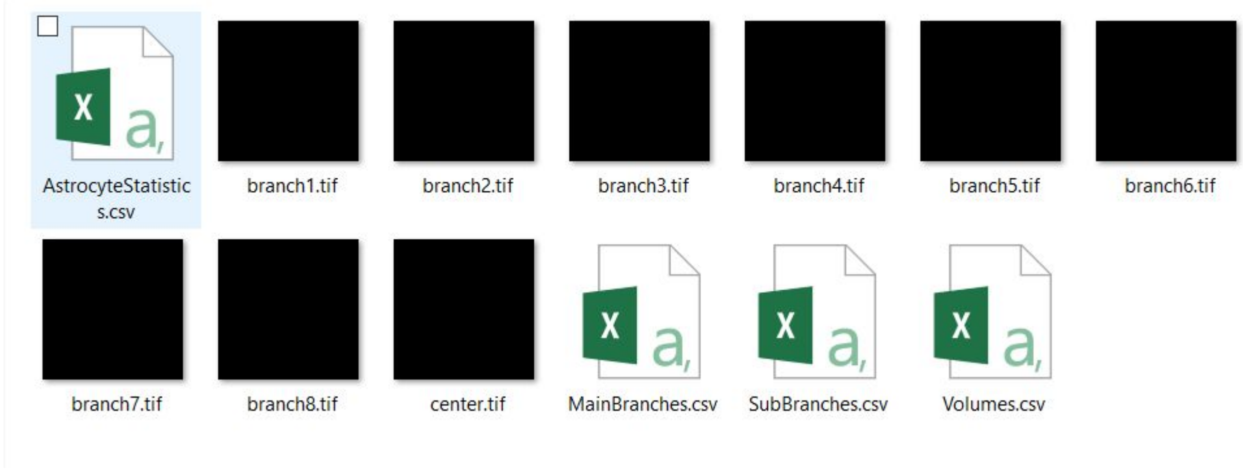
We need to split up the astrocyte cell body and the its branches into different images. First, we loop through all the branch pixels (number of branches) and create an image stack. For each branch pixel, we get the color of the current branch and iterate through the whole image storing pixels (with voxel value of 255.0) that have the same color, and ignore all other pixels in the astrocyte image. We will also remove from the skeletonized image the area where the astrocyte body is. This way we can run analyze skeleton on it to get information on the branches. Then we save the image in a folder for that astrocyte along with the volume of that branch in a table.



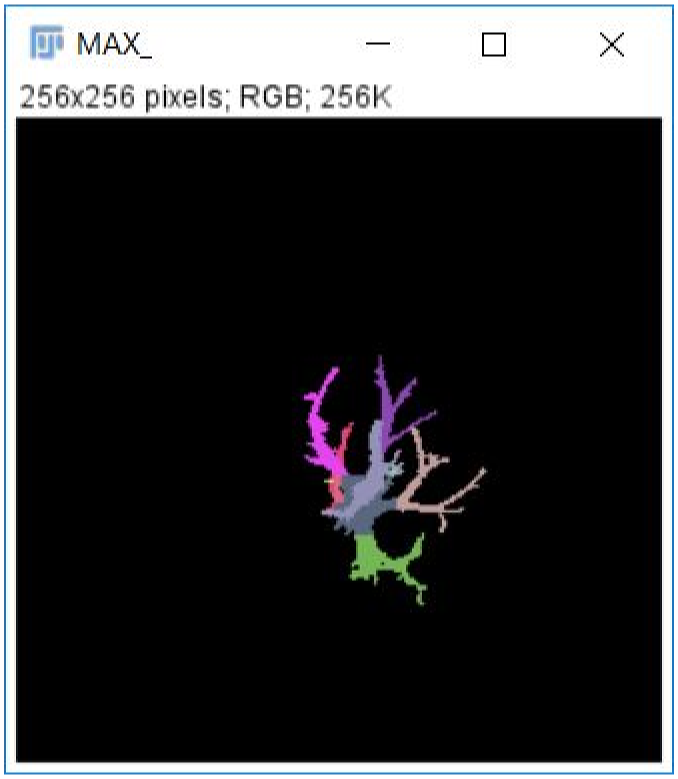
After we save all the separate branches and store their information, we then segment out the astrocyte cell body while tracking its volume. We simply iterate through the entire image until we find pixels with a voxel value of 100.0. When we do we increment a counter to track our volume, but for every other pixel we set its voxel to 0.0. Then we store this new image in the same folder for the current astrocyte and insert its volume in a table.

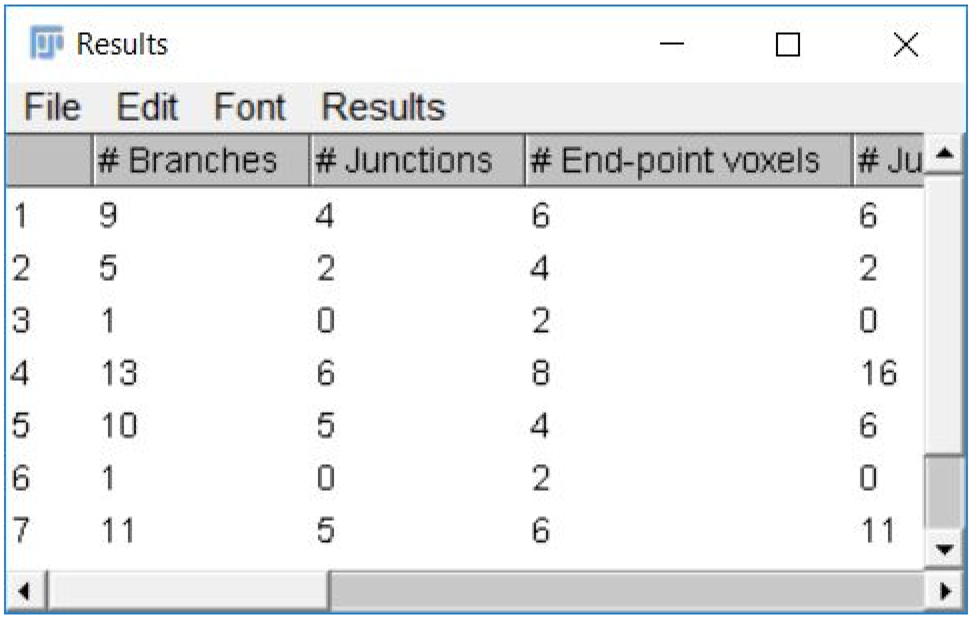
We run the 'Channel Merger' plugin with all the images we stored to display a final image that has each branch and the center of the astrocyte separated for analysis. We run analyze skeleton on the skeletonized image, that we removed the center from (where the astrocyte body would have been), and get the result. We then store the results of all the main branches, sub-branches and the volumes of the individual parts as a .CSV file in the same folder as the astrocytes.


89/121; 256x256 pixels; RGB; 30MB
✓ branch1.tif
✓ branch2.tif
✓ branch3.tif
✓ branch4.tif
✓ branch5.tif
✓ branch6.tif
✓ branch7.tif
✓ branch8.tif
✓ center.tif
Change color

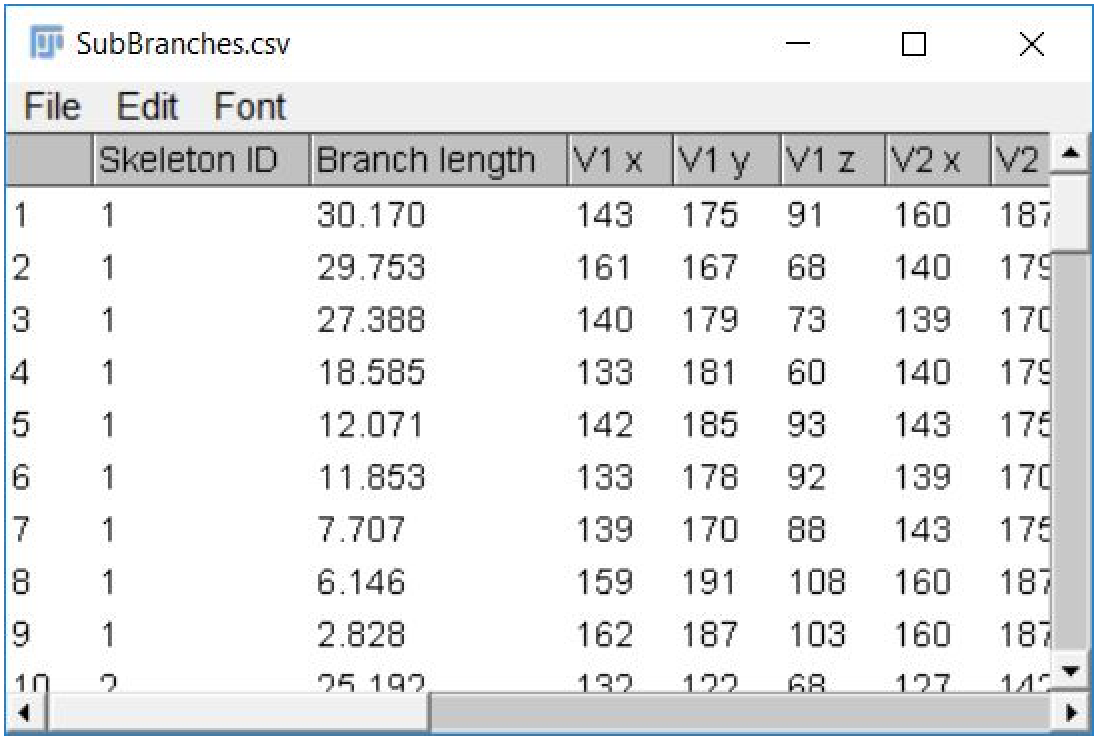

AstrocyteStatistics.csv
branch1.tif
branch2.tif
branch3.tif
branch4.tif
branch5.tif
branch6.tif
branch7.tif
branch8.tif
center.tif
MainBranches.csv
SubBranches.csv
Volumes.csv

The previous plugin did the segmentation of the image, but we obtained and stored the information from these images into results tables and saved them. We also added a way to save the information in an organized structure suitable for multiple astrocytes.

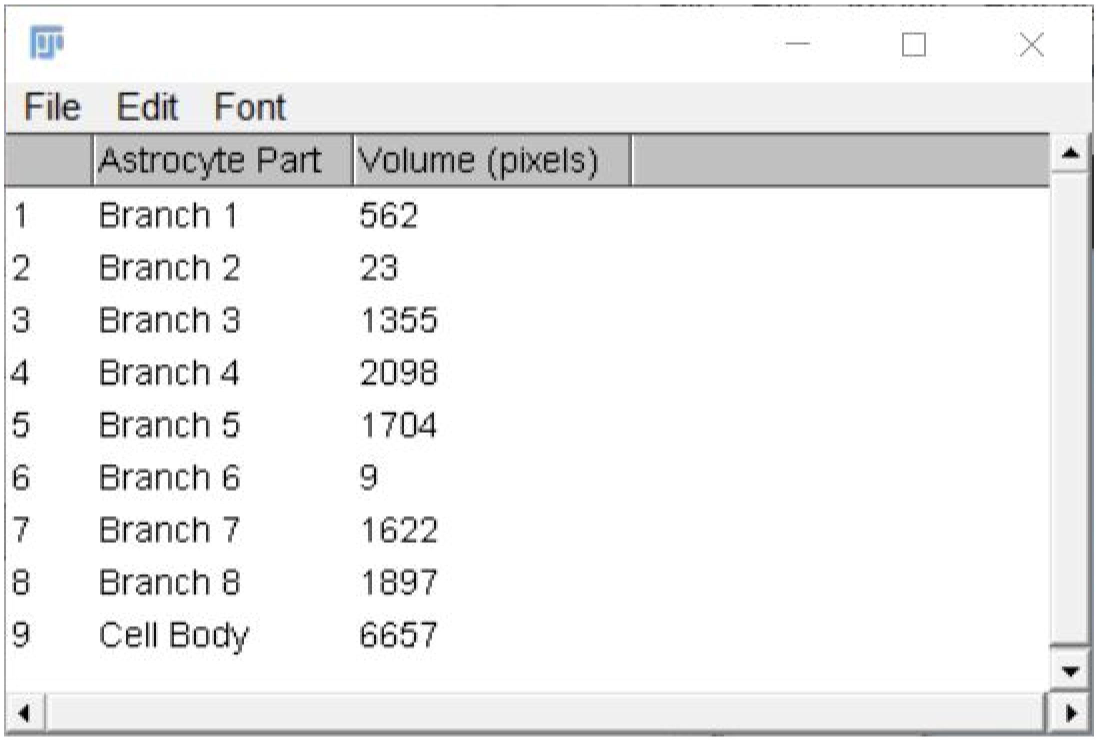




	# Branches	# Junctions	# End-point voxels	# Ju
1	9	4	6	6
2	5	2	4	2
3	1	0	2	0
4	13	6	8	16
5	10	5	4	6
6	1	0	2	0
7	11	5	6	11



	Skeleton ID	Branch length	V1 x	V1 y	V1 z	V2 x	V2
1	1	30.170	143	175	91	160	187
2	1	29.753	161	167	68	140	179
3	1	27.388	140	179	73	139	170
4	1	18.585	133	181	60	140	179
5	1	12.071	142	185	93	143	175
6	1	11.853	133	178	92	139	170
7	1	7.707	139	170	88	143	175
8	1	6.146	159	191	108	160	187
9	1	2.828	162	187	103	160	187
10	2	25.192	132	122	68	127	14



	Astrocyte Part	Volume (pixels)
1	Branch 1	562
2	Branch 2	23
3	Branch 3	1355
4	Branch 4	2098
5	Branch 5	1704
6	Branch 6	9
7	Branch 7	1622
8	Branch 8	1897
9	Cell Body	6657

Code Refactoring

The plugin underwent major refactoring by putting procedures into functions, putting repeated procedures into functions, renaming variables to match their purpose, and simplifying some logic. Some examples include changing ‘image_stack01’ to ‘astrocyte’, and ‘image_stack02’ to ‘skeletonized_image’. Some simplifications are converting a series of function calls with slightly different integer values inputted to a loop that increments the number equivalently. This code refactoring allows for better readability, and opens the plugin to be more adaptable in the future.

Results/Discussion

One issue with our current segmentation algorithm is that because it relies on the assumption of a spherical cell body with a determinate radius, the boundaries where cell body becomes process are sometimes difficult to delineate. As a result, branch pixels are sometimes inaccurately classified as body pixels during coloring. Future directions would involve resolving this issue by reevaluating or tweaking our algorithm, particularly the lines of code responsible for assigning pixels to the cell body. One approach would be assigning different values for the number of pixels separating a given pixel from the background based on which face/neighbor the algorithm grows through. In other words, using a distance transform function to segment an irregular blob instead of a sphere. Only pixels a given number of pixels’ distance away from the nearest background pixel would be counted as body pixels, whereas the remaining pixels would be classified as branch pixels due to the narrow shape of branches causing these pixels to be closer to the nearest background pixel.

Other issues are encountered during preprocessing when objects such as artifacts or microvessels encroach on the boundary of an astrocyte being segmented. In the event that said object has an intensity threshold similar to the astrocyte, it can be picked up and colored along with the astrocyte rather than filtered out as part of the background. To fix this issue, the sensitivity of the thresholds for segmentation may need to be adjusted. Another option would be to filter irregularities in shape that are characteristic of noise or microvessels but not characteristic of astrocytic branches, to ensure that said objects are not analyzed as being part of the cell of interest.

Lastly, one possibly important metric we did not get very much chance to explore is the volume of astrocyte nuclei and the number, size, and distribution of their nucleoli. Images taken for analysis by our plugin are typically three-channel images taken using confocal microscopy, with the channels being represented by DAPI staining for nuclei, GFAP staining for astrocytes, as well as staining for microvessels. Despite this, so far we have been splitting the channels and only making use of one of the three - the channel that shows GFAP staining for astrocytes. One question of interest is whether the size of nuclei and the number, size, and distribution of condensed chromatin (“nucleoli”) within these nuclei correlates to astrocyte size or branch morphology. After making manual measurements to assess whether such a pattern exists, automated comparison of the nuclear channel could be beneficial in piecing together the puzzle of determining astrocyte characteristics and, ultimately, type. Because the DAPI channel picks up nuclei of both astrocytes and neurons, for combined multi-channel analysis it would be necessary to write code that could match the nucleus of an astrocyte to its cell body and processes. This task would be further complicated in images that contain multiple astrocytes close together. However, one potential tool we could use to overlay information from the two channels is image calculator, which is built into Fiji and could combine ROIs from segmented astrocytes with ROIs from segmented nuclei to determine areas of overlap and, ultimately, which nuclei belong to which astrocytes. As far as the nuclear analysis itself, we could make use of the existing plugin mentioned in our introduction, which was also recently developed by a computer science graduate student at Kent State. This macro assesses nuclei and nucleoli volume as well as nucleoli number, and could help us determine, for instance, whether cells with more processes and more bifurcations also have larger nuclei with more nucleoli. Eventually we could interpret such information as providing independent cues about the health or status of a cell, and nuclear characteristics could be assimilated with body and branch characteristics to paint a picture of astrocyte type.

The microvessel channel could also provide potentially useful information, particularly in the analysis of perivascular astrocytes (astrocytes whose feet make contact with microvessels and who modulate blood brain barrier permeability) and in collaboration with the group whose plugin analyzes microvessel vasculature.

In addition to the above adjustments, refinements, and additions to our code, it will be important to assess its accuracy in segmentation by comparing its output to results obtained manually by visually counting astrocyte branches and directly measuring their length and bifurcation number. Manual data collection can be done using either Cell Counter to get raw numbers of primary, secondary, and tertiary branches, or Simple Neurite Tracer to measure process number, branching, and length. While some data has already been collected manually using these methods, we have yet to compare manual results to automated results of segmentation of the same images.

Conclusion

The plugin we developed successfully segments astrocytes from the background and defines their cell bodies as distinct from their processes, and works more efficiently and with fewer bugs than previously designed programs that do similar things. Future directions for our plugin's development include the ones outlined in the aforementioned section (refining cell body definition, segmentation of astrocytes that are touching each other or other objects, assessing bifurcation number, combining our analysis with analysis of nuclei and/or microvessels) as well as gathering more comparison data manually to independently assess the accuracy of our branch segmentation algorithm. Two of our three group members have agreed to continue working on the project after finishing the class, and while we will likely be working on the issue from different sides (raw coding versus integration with biology and neuroscience analysis) we will be collaborating as we continue to explore our options. We have no specific plans to publish our algorithm or its use in a methods paper in a scientific journal, but one option we may explore is presenting its implementation in a poster at the Kent State Neuroscience Symposium in April of 2018.

References

1. Lateef Mahmood Albukhnefis, A. (2016). Nucleus and Nucleolus Segmentation and Analysis. (Unpublished master's thesis). Kent State University, Kent, Ohio.
2. Connected Threshold Grower [Computer software]. (2016). Retrieved from https://sourceforge.net/projects/ij-plugins/files/ij-plugins_toolkit/