IMPROVING THE PERFORMANCE OF *C.ELEGANS ROUNDWORM TRACKING AND SEGMENTATION*

ADELINA VOUKADINOVA

University of Illinois at Chicago
avouka2@uic.edu

TEJUS PRASAD

Illinois Institute of Technology
tprasad@hawk.iit.edu

IOAN RAICU

Illinois Institute of Technology
iraicu@cs.iit.edu

DANIELA RAICU

DePaul University
draicu@cs.depaul.edu

Caenorhabditis Elegans is a type of roundworm that is being genetically modified and recorded for analysis of their simple neural network. The worms are recorded using a 640x480 camera at 30 fps. The segmentation code which extracts the worm's centroid location is being improved for time performance in order to be fast enough to process images at a higher resolution of 4K in real time. Time performance is being improved by translating the C++ code to CUDA. By running the code on CUDA, we are able to utilize the GPUs of the computer and parallelize segmentation. This significantly improved the speed of the segmentation. Our work focused on translating certain functions—contouring, thresholding, blurring—in the Open CV library into CUDA. Future work of this project involves running the code on a distributive system such as Spark or Hadoop and having the segmentation code detect the head, tail, and medial axis of the worm.

1.

1. Introduction

1. C.Elegans Background Information

The non-parasitic soil nematode, *Caenorhabditis elegans,* is part of the Animalia kingdom and is safely and commonly used for experimentation in laboratories. The size of the adult worm is small, approximately 1 mm in length and the most distance it can cover in a second is approximately 0.5 mm. *C.elegans* is also transparent, which allows for visualization of cells and neural networks under proper magnification conditions. Since the worm lives for about 2-3 weeks in normal environmental conditions, it makes for useful, rapid experimentation in the biology community. The wild type worm contains a constant number and position of cells, 959 to be exact, making it easy to track and follow cell mutations and development. The worms also have a simple neural network of 302 neurons and approximately 7000 synaptic connections [1]. Genetic engineers are able to utilize *C.elegans* to determine how specific mutants behave when their genome is altered. Since their cell count and position doesn't change, it makes it easier to alter the worm's genome. *C.elegans's* diet is simple, as they feed on common bacteria such as *E. coli.* The worms have two sexes, hermaphrodite and male. The hermaphrodites can either mate with males or self-fertilize to make an offspring, but males are unable to self-fertilize, making them infrequent in cultivated worms [4] [7]. For this specific experiment, the worms were genetically modified.

2. Tracking Background Information

Tracking of the worm serves the purpose of producing images of the worm at each position in time. The camera we have been working with captures the worm's position at 30 fps. Tracking output is a collection of grayscale jpg images that are then processed through the segmentation code and analyzed. The tracking is not only the software that captures the worm images, but it's also the housing of the worm. The worm must be placed on an agar plate that could house food for the worm if need be. The base of the housing must not capture any movement noise, since that would produce noisy images. The camera must also be able to move around on a X-Y translational stage to follow the worm as it moves on the agar plate.

3. Segmentation Background Information

The segmentation code works on any type of *C.elegans* worm so long as each agar plate holds only one worm. The segmentation of the worm must accurately and efficiently locate the worm. Various methods exist for locating the worm. Some of the more obvious ones are using the head or tail, depending on which end is at the front of the displacement vector. One of the issues with this method is that the location changes drastically if the worm begins moving backwards, causing the location of the worm to jump its length even though it obviously hasn't moved that far. Another viable option for determining the position of the worm is by using its centroid. Computing the centroid begins with isolating the worm by creating a rectangular box around the area it spans. From there, the length and width of each side of the rectangle are divided by half, and a cross section is created between the 4 values. The point of intersection becomes the centroid of the worm, and it might not always lie on the worm itself.

2. Materials and Methods

2.1 Hardware

The hardware components include the base of the tracker with its camera, the housing of the worms, and the computer that facilitates the execution of the tracking system. The base of the tracker consists of all the fundamental tracking components necessary for recording the worm. It includes a X-Y translational stage, a camera, two stepper motors, and a motor control board. The two step motors and stage are controlled by an EiBotBoard. A new 4K camera has been purchased, but the housing for the new camera has to be rebuilt, so we have not been able to utilize the new camera for recording yet. The current tracker is capable of recording in 1280x960, 1280x720, and 640x480; we are currently recording in 640x480 resolution.

The housing is responsible for encasing the worm while the tracker is recording its movements. It consists of a 150 mm agar plate, which is isolated from the base to avoid any vibrations from interfering with the recording of the worm and/or its behavior. The worm is placed on the agar plate and recorded from underneath. The computer, a dedicated Intel NUC small form factor PC, is used for executing the tracking software and controlling the base's translational stage movements [9].

2.2 Software

There are two components to the software used for tracking when recording the worm or when segmenting the recorded images to retrieve information from the worms' movements.

2.2.1 Worm Tracking

The current tracking code is using SarXos Webcam Capture API to control the camera [3]. Once the resolution of the camera is set up, various parameters must be set. Segmentation window size, minimal size of the component to be considered a possible worm, motor pixel movement per step in the X direction, motor pixel movement per step in the Y direction, move decision confidence distance, and move decision boundary pixel are all the necessary parameters.

The new tracking software produces an output log text file with four columns: frame number, time, centroid X value, and centroid Y value. The older software produces a four column output log text file with the frame number, centroid X value, centroid Y value, and area size of the bounded worm. One of the main reasons the output file contains the centroid is to facilitate segmentation by determining the relative location of the centroid.

2.2.2 Worm Segmentation

Our worm segmentation code has undergone various modifications since first written in Java. Since then it has been rewritten in C++ to improve time performance, and it's been translated to CUDA [2]. Using CUDA, the code can run on the GPU (graphical processing unit) instead of the CPU (central processing unit), which increased time performance. The CPU holds a few cores with a lot of cache memory, whereas the GPU has thousands of cores, making it easy to handle thousands of threads. This means that you can run the same code on thousands of cores vs a handful, which increases time performance by a large factor [6].

One of the main reasons work was being done to improve time performance of the worm segmentation is due to the resolution of our camera being increased. The previous camera was 640x480 resolution, which isn't a high enough resolution. It is difficult to differentiate the head and tail of the worm when the resolution is too low. Another issue

with lower resolution cameras is the translational stage moves more and produces output images with greater precipitate or inaccurate images of the worm. Precipitate arises when water droplets form on the agar plate, or there is noise from not cleaning the plate properly. Improving time performance of the segmentation code is key for when the camera resolution is increased to 4K, so information can accurately be retrieved from the collected data. The camera records the worm at 30 fps, so the code needs to process the frames at 30 fps or more. This way, segmentation of the worm can occur in real time with the tracking.

Segmentation works by taking the grayscale images produced by the camera, and loading and processing them sequentially. The code goes through every single image and produces an output log file of the image frame number, the x value of the centroid, the y value of the centroid, and the total area of the bounded rectangle around the worm. Once an image is loaded and passed through our function, a box filter is applied to blur out the image. Once that is done, the image is made binary to isolate the worm and make it easier to manipulate. The worm is then bounded by a rectangular box, with its edges on the worm's furthest edge from each side. To compute the centroid and return it, the length and width of the rectangle are divided in half, and then

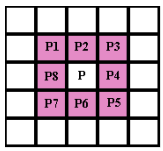
$$(C_x, C_y) = \left(\frac{\sum_{i=1}^{M} x_i}{M}, \frac{\sum_{i=1}^{M} y_i}{M}\right)$$

added to each edge accordingly [10]. The centroid of the worm is the average of the x and y coordinates of all M pixels on the worm body using the following formula [9]:

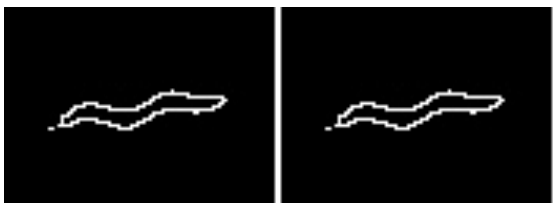
Originally, the segmentation code was written in Java, and then rewritten in Open CV/C++ to improve the time performance. The code runs on an AWS cloud EC2 instance. The time performance at 640x480 resolution improved approximately 6.67x. Open CV had many of the image processing functions already implemented in its library. The box blur filter, thresholding, and the contour functions were called directly from the Open CV library instead of being manually implemented.

In C++ at 4K resolution, the code executes approximately 3.95x faster. Further work was done to increase the speed by running the segmentation on the GPU. Running a process on the GPU increases its time performance since the functions can be parallelized by utilizing thousands of cores instead of using the cores on a CPU. In order for the Open CV/C++ code to run on the GPU, it had to be rewritten in CUDA. CUDA is a language compatible with Open CV, and it parallelizes a code's functions by running them on the GPU.

An issue that was encountered while translating the code from C++ to CUDA was that the Open CV library contains a findContour() function that does not exist in CUDA's library. A new contour algorithm was implemented that correctly returned the contours of the worm. Thresholding and blurring were also rewritten to run on CUDA. This was essential because the code in CUDA cannot properly locate the centroid without a bounding rectangle around the worm, and the bounding box is drawn according to the contour, blurring, and thresholding. The previous findContour() function produced a vector of vectors, where each vector held a list of vector points on the image where a contour exists. The code assumes that the largest contour vector is the worm. The output of the contour vectors from findContour() are in the form [x, y], where the x is the x value and the y is the y value of the pixel location on the image.



Our algorithm works on a basic principle similar to a Moore-Neighbor Tracing, but more straight-forward [5]. The algorithm goes through every row and column of the image and analyzes each pixel value independently of all the others. At each pixel, it processes its neighboring 8 pixels and compares if any of the pixels are white. Our binary image has the worm filled in black on a white image. If any of the neighboring pixels our white, it labels the current pixel location as a contour. All of the contours are stored in a vector. When we compared the visual output of findContour() with our function, we got identical results. The only difference is the empirical output of the vector. The findContour() function produces a list of values which are then connected as vectors; our function outputs every single contour value as an individual point.



3. Future Work

Since our code is running on the GPU in CUDA, the next step is transferring it to a distributive system such as Spark. Spark works on the principal of parallelizing real-time image processing and analysis by distributing data among thousands of nodes. Spark is compatible with various languages such as Java, Python, and R which will make is simpler for us

to run segmentation on [8]. Other potential work for the future is head and tail differentiation, along with determining the medial axis of the worm. Once we are able to utilize the 4K camera, we can better differentiate the head and tail of the worm to be able to know whether the worm is going forwards or backwards. Usually the head is slightly lighter and rounder than the tail, so the new camera will make our predictions more accurate. The medial axis is the "skeleton" of the worm, and is the line running through the middle point of the entire worm.

4. Acknowledgements

Work was partially supported by award 1461260 (REU) of the National Science Foundation. This work was also partially supported by DePaul University, Illinois Institute of Technology, and Rosalind Franklin University.

5. References

1. "A Short History of C. Elegans Research." *WormClassroom.* Laboratory for Optical and Computational Instrumentation, University of Wisconsin-Madison. Web.
2. "CUDA." *OpenCV.* Itseez, 2016. Web.
3. "Webcam Capture API." *Webcam Capture.* SarXos. Web.
4. Eisenmann, D. M., Wnt signaling (June 25, 2005), *WormBook*, ed. The *C. elegans* Research Community, WormBook, doi/10.1895/wormbook.1.7.1, http://www.wormbook.org.
5. Ghuneim, Abeer George. "Contour Tracing Algorithms." *Contour Tracing.* School of Computer Science, McGill University, 2000. Web.
6. Krewell, Kevin. "What's the Difference Between a CPU and a GPU?" *NVIDIA.* NVIDIA. Web. 16 Dec. 2009.
7. Lints, R. and Hall, D.H. 2009. Male introduction. In *WormAtlas*. doi:10.3908/wormatlas.2.1
8. Mader, Kevin. "Scaling Up Fast: Real-time Image Processing and Analytics Using Spark." *Spark Summit 2014.* Apache Spark, 07 May 2014. Web.
9. Moy K, Li W, Tran HP, Simonis V, Story E, Brandon C, et al. (2015) Computational Methods for Tracking, Quantitative Assessment, and Visualization of C. elegans Locomotory Behavior. PLoS ONE 10 (12): e0145870. doi/10.1371/journal.pone.0145870
10. Shirodkar, A (2016) WormSegmenter source code [Source code], http://cloud.aditya11.com/index.php/apps/files/