University of Nevada
Reno

Semi-Automated Analysis Software for a Novel Biochemistry Assay

A thesis submitted in partial fulfillment of the
requirements for the degree of Master of Science
in Computer Science

by

Joseph M. Vesco

Dr. Frederick C. Harris, Jr., Thesis Advisor

December 2011

THE GRADUATE SCHOOL

We recommend that the thesis
prepared under our supervision by

JOSEPH MICHAEL VESCO

entitled

Semi-Automated Analysis Software For A Novel Biochemistry Assay

be accepted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

Frederick C. Harris, Jr., Ph. D., Advisor

Sergiu M. Dascalu, Ph. D., Committee Member

Josh E. Baker, Ph. D., Graduate School Representative

Marsha H. Read, Ph. D., Associate Dean, Graduate School

December, 2011

Abstract

Some of the work done in the Baker lab at the University of Nevada, Reno has been directed towards the analysis of muscle tissue in a single molecule arrangement. The process is a novel high-throughput single molecule binding assay, or SiMBA for short. This assay is performed by binding the myosin molecules of the muscle tissue to a coverslip and looking at how the fluorescently labeled actin filaments bind to these myosin as imaged with a fluorescence microscope. The conditions are varied and the effects on actin-myosin binding kinetics are observed. In order to analyze the binding times and unbound times a researcher must observe the interactions and manually collect the data. This particular method of data collection is tedious and time consuming thereby making this portion of the experiment the "rate limiting" factor for producing results in a timely manner. As this can take many hours to analyze a one minute long movie an automated or semi-automated solution would be beneficial to this assay. This thesis presents the design and implementation of a semi-automated solution for identifying and tracking a variable number of objects that exhibit a multitude of behavior, and extracting the specific behaviors of motion and stagnation as well as the duration of these behaviors.

Acknowledgments

Thank you to my advisor Dr. Frederick C. Harris, Jr. for your generous support and patience. Thank you to my committee Dr. Sergiu M. Dascalu and Dr. Josh Baker for taking the time to see me through this final event of my degree. And thanks to my colleagues Roger Hoang, Torbjorn Loken, Del Jackson, and Cody White. You are all good people and have helped me immensely during my time here at UNR. I especially want to thank my family for their continued support through out my life. I never would have made it without you.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The Baker Lab project being discussed has been summed up by Del Jackson [6]: *"We have developed a novel high-throughput single molecule binding assay (SiMBA) to measure the affinity of the muscle proteins actin and myosin under varying conditions. Since myosin behaves differently in an ensemble, this assay gives researchers a powerful tool for studying the kinetics at a single molecule level which may give insight into cooperative aspects of muscle dynamics. The data analysis for this assay involves manually observing and counting binding events and unbinding duration. Although the experiment is high-throughput, in that many different conditions can be quickly tested, the analysis is the rate-limiting step, as it involves many hours for a researcher to analyze a single movie. Development of an automated, or even semi-automated, analysis software program would dramatically improve the usefulness of this new experiment."*

The problem was that the only available method to analyze the movies produced was to sit and watch each individual particle and plot the times they moved, did not move and record this as a new event every time this state changes. Figure 1.1 and Figure 1.2 shows two consecutive frames from an assay. From these two images we can see how this is a tedious analysis process and can be a large time constraint of the overall experiment.

These two frames also show two different occurrences that increase the challenge of particle tracking. In Figure 1.3 and Figure 1.4 these same frames have the two incidences highlighted. In red, we see one particle is moving in Figure 1.3 and ends

Figure 1.1: Movie 0.5ugmLS1-2.5nMAct-0.9_MC1.avi, frame 22.

Figure 1.2: Movie 0.5ugmLS1-2.5nMAct-0.9_MC1.avi, frame 23.

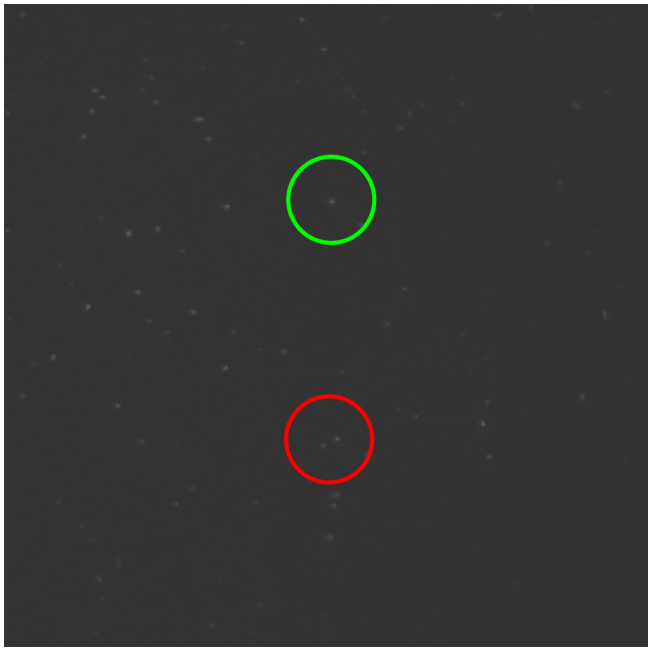

Figure 1.3: Frame 22 with green enclosed object is visible, red is two distinct particles.

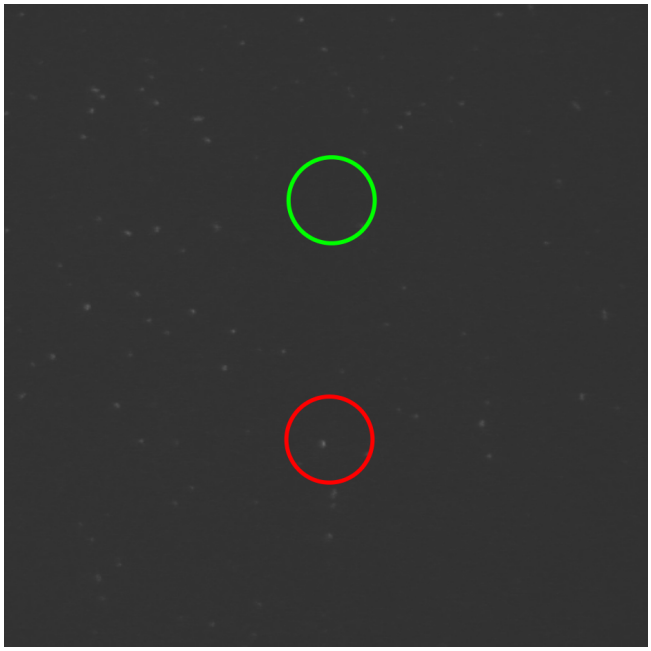

Figure 1.4: Frame 23 where green is no longer visible, red is now two objects seen as one.

up in the same position as a second non moving object in Figure 1.4. To properly track this moving object, one must determine if the moving object is the same object as before when these two objects are later separated. If this is the same object then the duration of the objects state is extended, though if the previously stationary object now in motion a new event would begin and the previous event would have ended. The object marked with green is a moving object that is visible in Figure 1.3 then no longer visible in the next frame as seen in Figure 1.4. Although this particular object does come back in the frame following Figure 1.4, we must ask ourselves how long an object can be missing before being considered to be a different object and again thereby creating a new event.

There have been non specific programs that the lab has pursued to no avail. As these programs were designed to tackle a multitude of problems, they for one lacked the simplicity needed to avoid a steep learning curve and did not give desirable output.

In order to solve this issue of collecting data manually, the software needs to read the raw data, filter the images, process this data to determine all of the objects and events, and then create the needed statistics in a timely and accurate manner.

Designing and developing software to assist this process could unlock a valuable assay for muscle protein research labs across the world. As data analysis is the only limiting step in the experiment, with a successful implementation of an analysis software, this experiment would qualify as a true high-throughput technique.

The rest of this thesis is structured as follows: **Chapter 2** will cover the background and literature review for this work. This will include coverage of the basic laboratory process that generates the movies used for this application. This will be followed by video conversion, basic image processing techniques, and data array manipulation techniques needed in order to have the tools necessary to analyze these movies. Also, a review of programs that may be applicable to this problem and how these compare with the software we have developed will be included. In **Chapter 3** we will present our proposal for how to solve this problem and to generate a readable image and how to use these images to gather the necessary information on each object

present in each frame. **Chapter 4** is a discussion of the software engineering aspect of this program. **Chapter 6** will cover the results of a sample run of SANoBA . In this run I will show how to use the software and how it compares to a manual analysis of the same movie. This will be followed by an actual comparison of the results found by both methods. **Chapter 7** will present our conclusions and go into future work.

Chapter 2

Background and Literature Review

2.1 Problem Description: Biochemical Assay

2.1.1 Overview

An image representing the general muscle model in Figure 2.1 shows one moment of how the components myosin and actin interact (arm and chain) [7]. Figure 2.2 and Figure 2.3 show two positional states of a muscle motor protein myosin, which result in a contracting muscle. It is these states of being bound and unbound that are sought in regards to the SiMBA assay in order to further understand the behaviors and interactions of myosin and actin all the way up to the overall behavior of a working muscle.

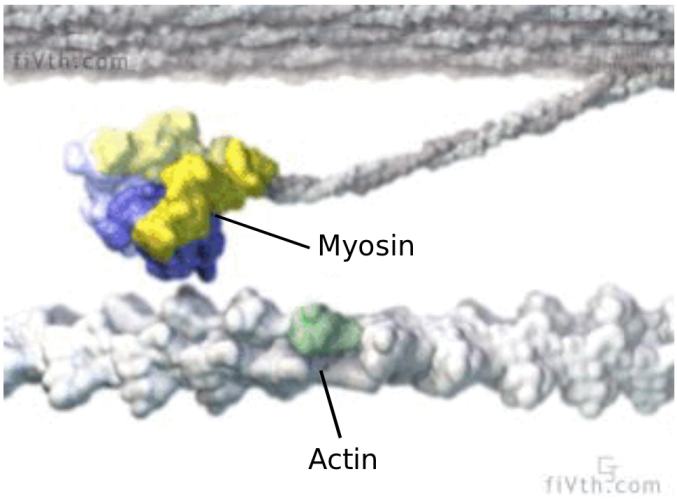

Figure 2.1: A General Muscle Model [7].

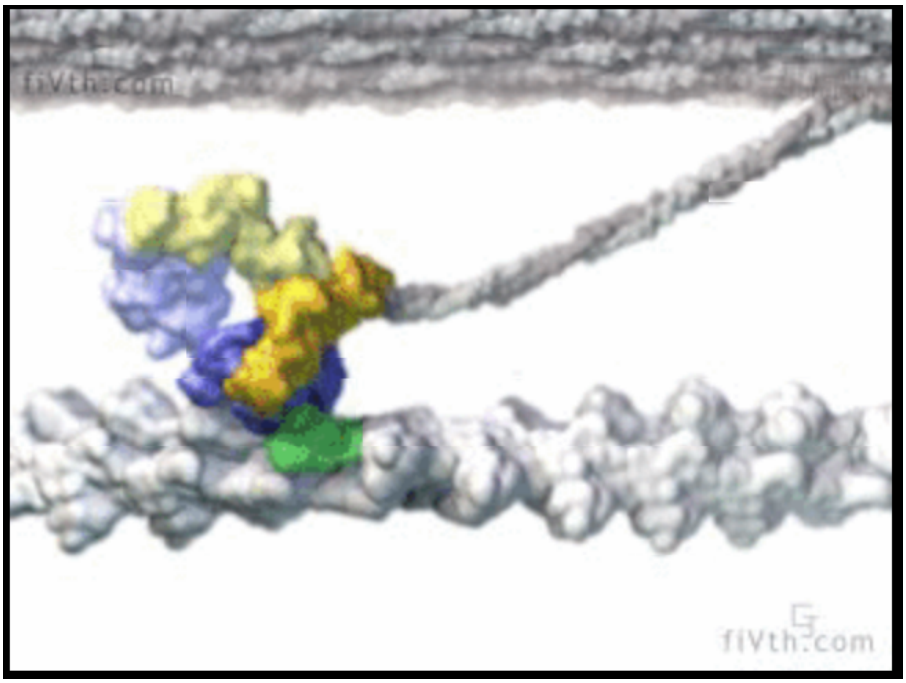

Figure 2.2: Pre-powerstroke state [7].

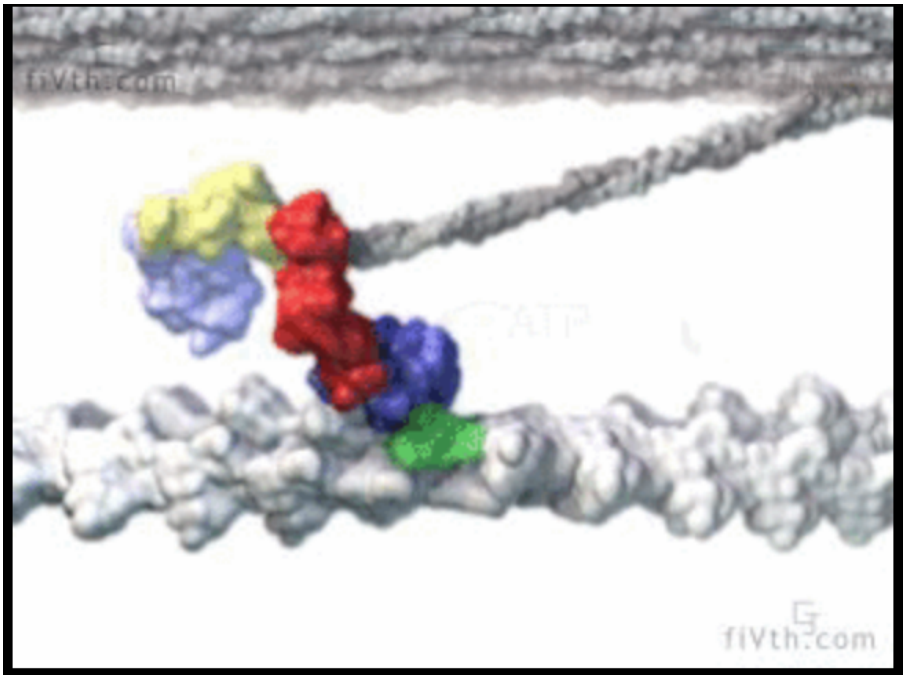

Figure 2.3: Post-powerstroke state [7].

Figure 2.4 is a basic model showing the configuration of myosin and actin in the SiMBA experiment. Visible actin fragments labeled with a fluorophore diffuse in solution with unlabeled (invisible) myosin proteins affixed to a slide surface. The singular myosin molecules are spaced $\approx 1/25 \mu m^2$ [6]. This experiment is viewed through a microscope with a resolution of $54 \mu m$ x $54 \mu m$ (512 pixels x 512 pixels) and an exposure rate of 0.1 (10 frames per second) for 180 seconds. The raw data is collected from a Nikon microscope and Roper charge-coupled device (camera). Images, in the form of movies, are collected with Hamamatsus Simple PCI program [4].

The scrutiny a researcher must perform in the analysis stage of this experiment can last many hours. By developing an automated or semi-automated solution, this time of analysis can be greatly reduced. Certain issues are common that prevent this program from being a fully automated solution. For instance, at uncertain times during a movie, as an object moves from one location to the next, the size of an object from the cameras perspective may increase or decrease significantly. A range for the size is attributed to each object for motion detection, but at times the size may exceed a reasonable range there by making the object ID associated with one particular object to be different than in previous frames. This may be due to at least

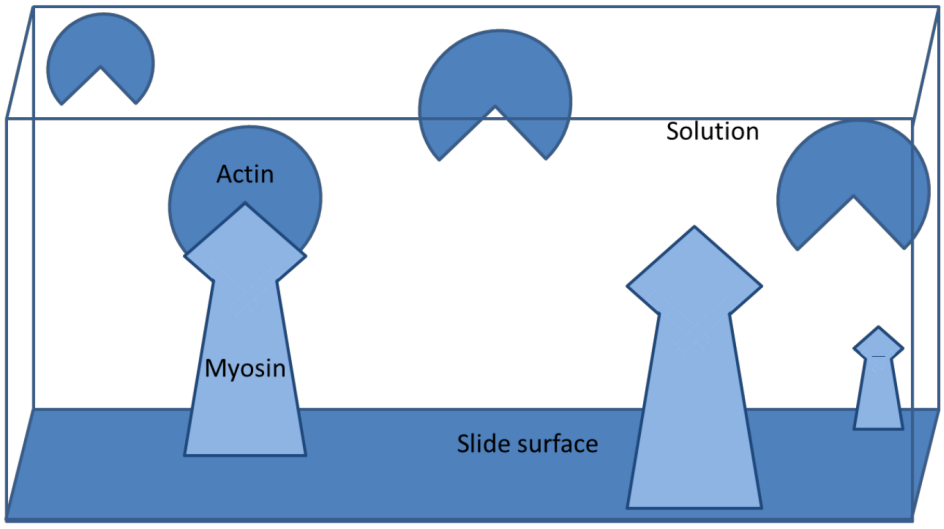

Figure 2.4: Schematic depiction of the experimental components of SiMBA [6].

two situations. One cause is that the camera's focus may shift causing a blurring if not a complete removal of all objects from the frames perspective. Second, the "flow cell slide", as seen in Figure 2.5 has a bit of space in the z-axis (perpendicular to the slide plane) between the main slide and a cover that actin can travel through. Although this space is limited, there is still enough room for the actin to move out of focus and cause a object focus shift.

If the first issue is present for more than 2 frames, all events are ended and new events start when the shift has ended. In the latter case, the same outcome as previously stated occur on an individual basis. This can lead to inaccuracies that can be avoided but are difficult to automate.

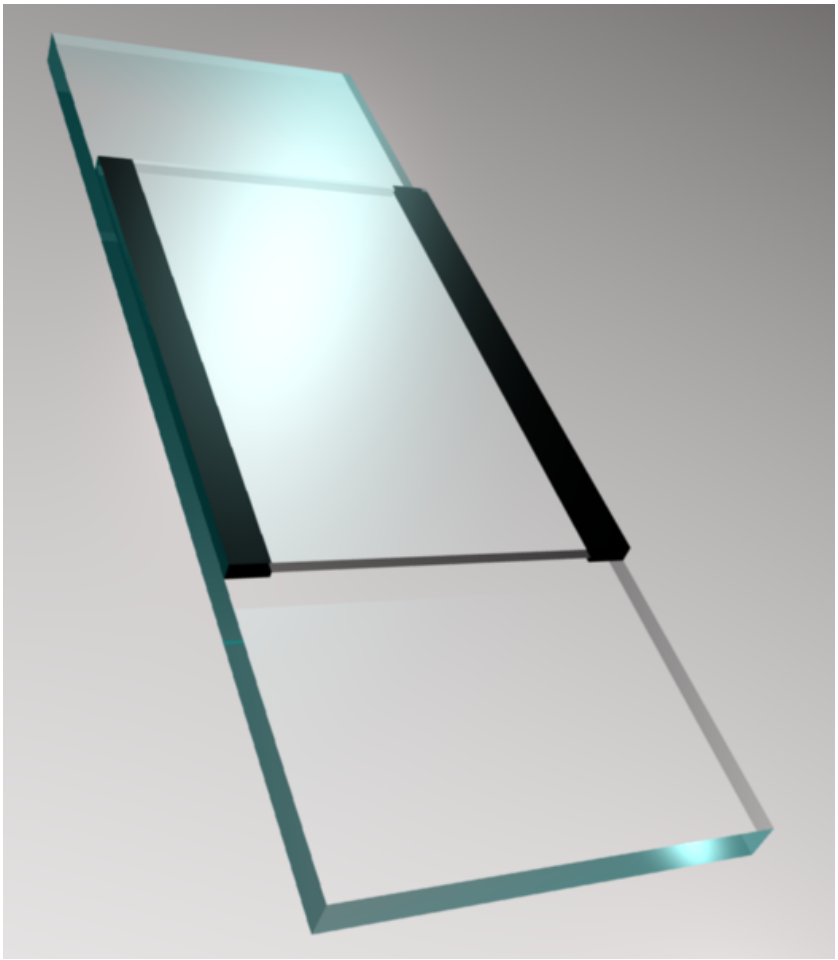

Figure 2.5: Mock up example of a flow cell slide.

2.1.2 Lab Process

The SiMBA experiment begins with the preparation of the muscle samples. The frozen sample proteins and reagents are retrieved (Figure 2.6) then thawed on ice as seen in Figure 2.7.

Figure 2.6: Frozen reagents and proteins are retrieved.

Figure 2.7: The retrieved reagents and proteins are thawed on ice.

Next, the buffers are prepared to ensure a consistent pH and temperature for the proteins Figure 2.8. These buffers used in the SiMBA experiment are identical except for the varying experimental parameter. For example, some of the parameters that may be varied include protein concentration, nucleotide concentration, or inhibitor concentration Figure 2.9.

Figure 2.8: SiMBA buffer preparation.

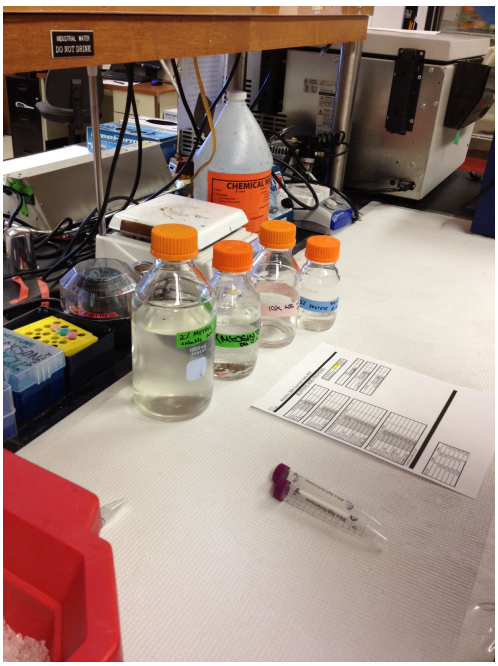

Figure 2.9: SiMBA buffer preparation.

The slides are prepared by thoroughly mixing the buffers in a rotational mixer Figure 2.10. The buffer is then brought to experimental temperature by placing the flow cell slide on a slide warmer before imaging with the microscope Figure 2.11.

Figure 2.10: Rotational mixer.

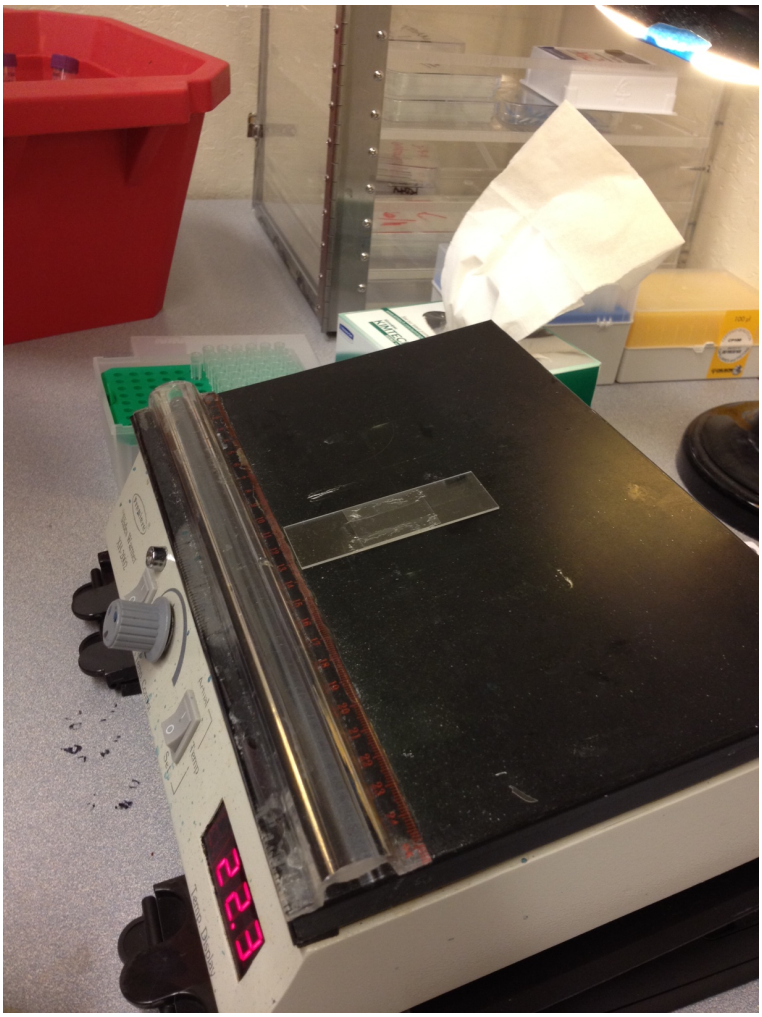

Figure 2.11: Slide warmer.

As seen in Figure 2.12 a Roper 512B CCD is used to image the SiMBA experiments on an Nikon 2000-TE inverted microscope Figure 2.13. Flourescently labeled proteins are excited at 488 nm using total internal reflection fluorescence microscopy. The resulting movie made by the software SimplePCI [4] is in the form of CXD movie file. This same software is then used to convert these raw movies into a TIFF or AVI format.

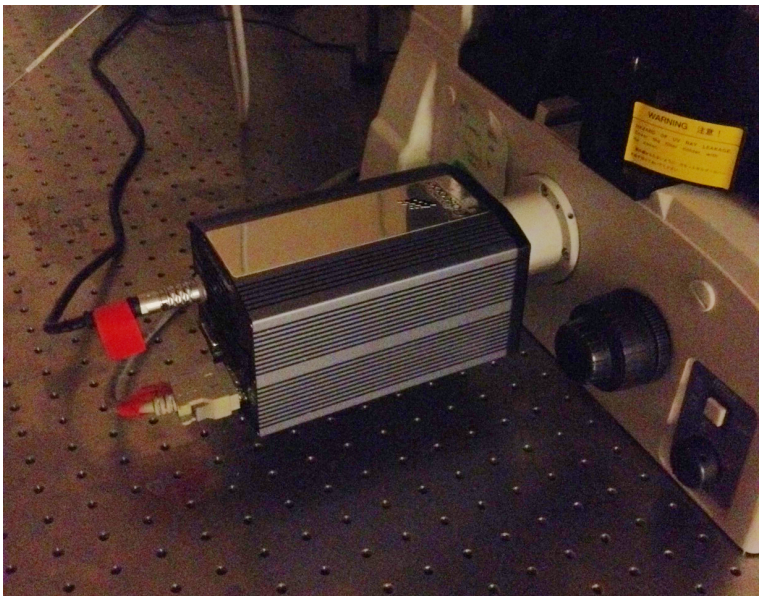

Figure 2.12: SiMBA experiment.

Figure 2.13: SiMBA experiment.

2.1.3 Desired Results

The needed statistical results are time bound (t_{on}) and time unbound (t_{off}). Since the myosin is not visible to the microscope, these binding events are observed by labeled actin particles. The labeling agent used is a fluorescent dye from Alexa [14].

Figure 2.14 shows the schematic representation of t_{on} [6]. A binding event is when an actin particle stays immobile (determined by a user defined sub-pixel threshold) for at least two frames. The duration of this determines the t_{on}.

Figure 2.15 shows the schematic representation of t_{off} [6]. The time it is motion between binding events is the measure of a single t_{off} calculation. Additional statistics that are useful to this research would be the number of binding sites in the field (N_{bs}) and the diffusion coefficient of the actin particles (D_{act}).

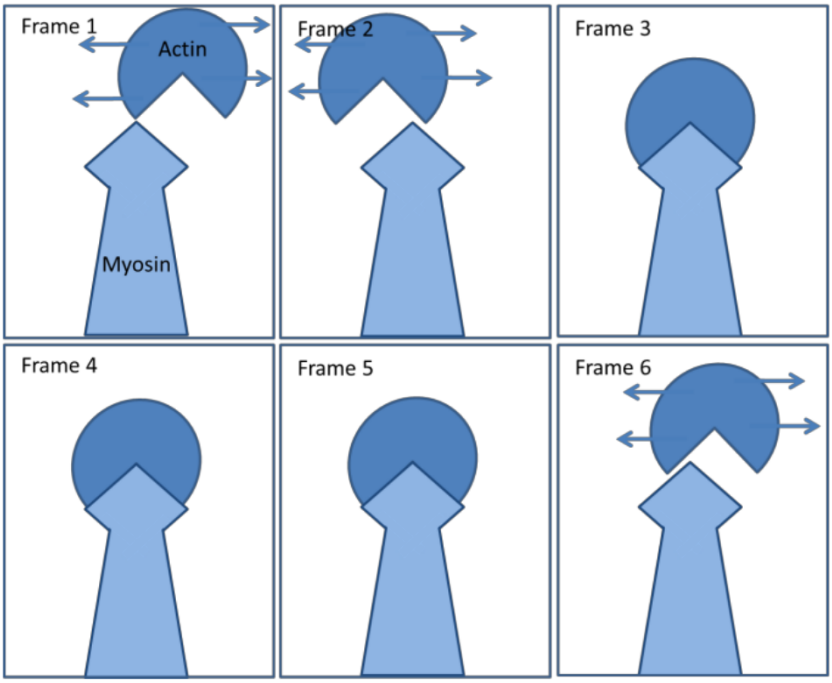

Figure 2.14: Schematic representation of t_{on} binding event [6].

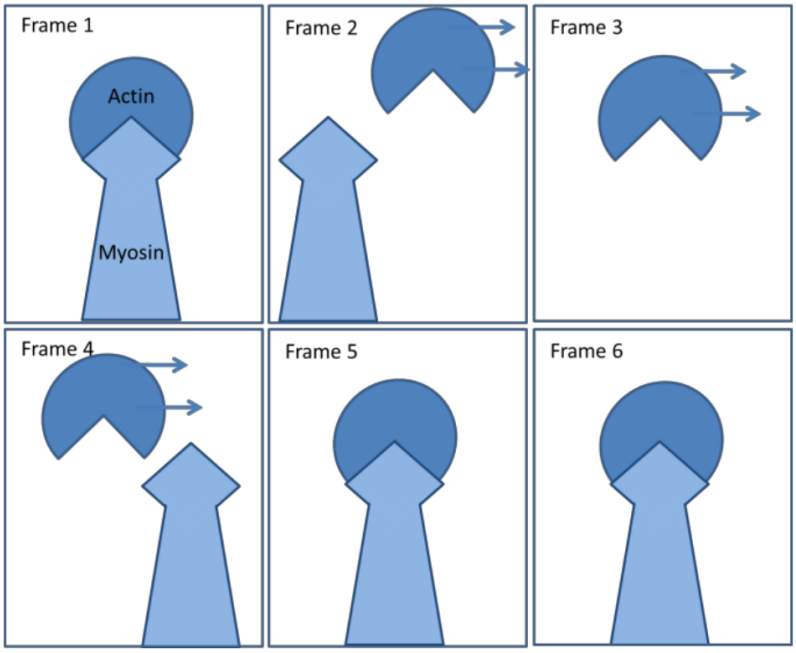

Figure 2.15: Schematic representation of t_{off} binding event [6].

2.2 Image Processing Techniques

Typically in an image processing application there are certain fundamental functions that are needed. These may include but are not limited to erosion, blur, and threshold. Also the imported data is most likely in a movie format. It is sometimes necessary to convert these movies into individual frame images. At this point we will go over the principles of a selected number of functions typically needed for image processing and motion detection.

Video Conversion to an Image Data Array Using the built-in functions of OpenCV, the raw data in an AVI format can be converted to individual frames that are in the format of PGM [3]. The use of the PGM format is to have a portable 2 dimensional array that may be transferred to any operating system in future upgrades. Also, it makes parsing the data easier for custom built processing techniques. The format of PGM (Portable Gray Map) [15] is a 2 dimensional array of values ranging from 0 to 255, where 0 is black and 255 is white.

Threshold Applying the threshold process to image is to determine for each pixel if it has a greyscale value greater than or less than a given value. If this pixels value is less than the threshold value then according to the binary threshold method the pixel is set to 0, which corresponds to black. However, if the pixel is greater than the threshold value it is set to one. There are a variety of threshold methods that include the previously mentioned binary method, truncate, or value to name a few. Some of the threshold methods used in the application OpenCV are shown in Figure 2.16.

Threshold functions are used mainly for two purposes [3]:

- Masking out some pixels that do not belong to a certain range, for example, to extract blobs of certain brightness or color from the image.
- Converting gray scale image to a black-and-white image.

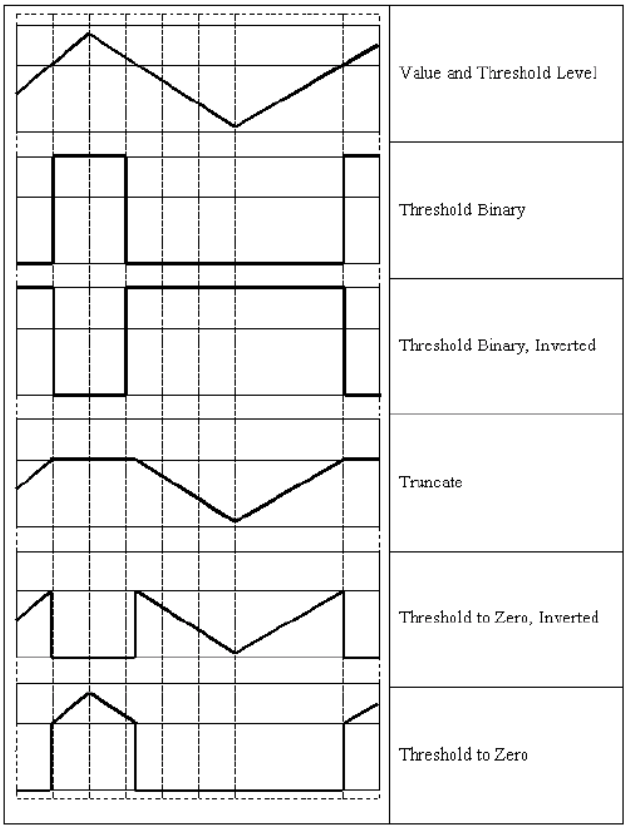

Figure 2.16: Visual representation of the various threshold types [3].

An example that demonstrates both of these purposes starts with the original image seen in Figure 2.17 [9]. After applying a threshold function we see in Figure 2.18 that the brighter parts of the image are pushed up to the value of 1 and the darker parts below the threshold value is set to a value of 0 [8].

Figure 2.17: Original image [9].

Figure 2.18: Image after threshold [8].

Erosion and Dilation Erosion of an image is the process of reducing the size and brightness of an object by taking the neighborhood minimum when passing over each pixel in the image [10]. The concept is easier to understand in terms of a binary image, where parameter pixels are removed from larger objects and smaller object are completely removed [10]. This process can be done a number of times iteratively to achieve the desired results. As we see in Figure 2.19 the processed image had the erosion process incurred and we see that the smaller particles are remove and the larger objects have decreased in size. If a color image is being eroded, each color channel is done independently [3]. In the opposite fashion, dilation of an image increases the size and brightness of objects by taking the maximum of a pixels neighborhood [22]. This is demonstrated in Figure 2.20 where the original binary image is on the left and the right shows the effect of dilation. Here the smaller particles are now substantial and the rest of the objects are larger than in the original image. And as with erosion, this method can be done iteratively and done for each channel of a color image [22].

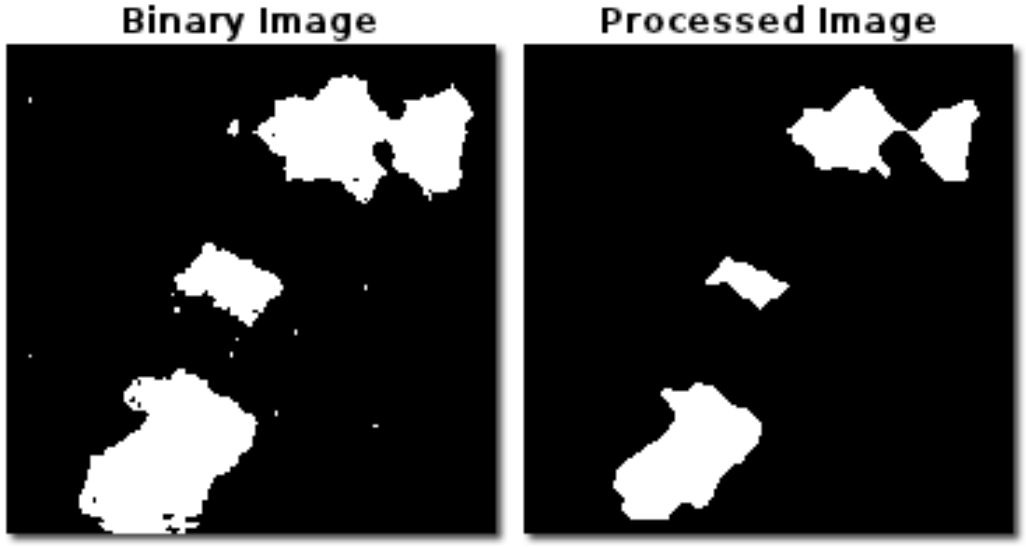

Figure 2.19: Original image and after erosion [17].

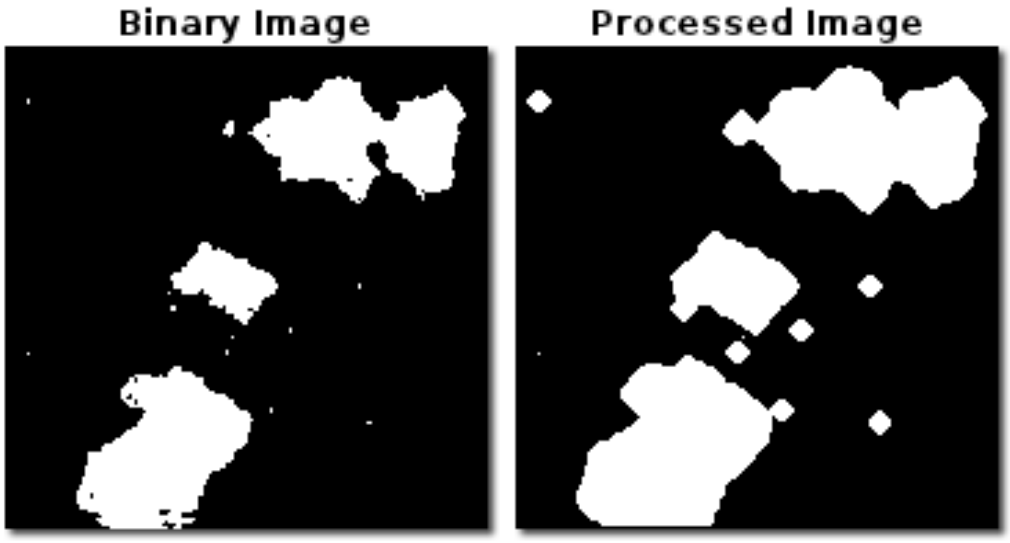

Figure 2.20: Original image and after dilation [17].

Blur Blurring is used to reduce image noise and produce an image with reduced pixelation [11]. There are a variety of blurring methods that include bilateral filtering, median, and Gaussian blur. The equation of the Gaussian parameter is [11]:

$$\sigma = 0.3(\frac{n}{2} - 1) + 0.8 \tag{2.1}$$

$$\text{where n} = \begin{cases} \text{param1 for horizontal} \\ \text{param2 for vertical} \end{cases}$$

The image in Figure 2.21 shows quite a bit of detail with some pixelation. After applying a Gaussian blur to this image we get the resulting image in Figure 2.22 were there is a noticeable amount of lost detail although the major shapes and information are still present.

Searching for Objects When searching a 2 dimensional array of values there are two common methods that may be implemented. The first is a depth-first search algorithm and the second is a breadth-first search. A depth-first search (or DFS) is used to examine a set of data by starting at a designated root node then searching along each branch until backtracking at the end of each branch. The selection of an edge to traverse is determined by choosing an edge starting at the most recently

Figure 2.21: Original image [12].

Figure 2.22: After blurring is applied [12].

reached vertex that has unexplored edges. The previously searched vertices that have unexplored vertices themselves are stored on a stack for backtracking. This allows for a program that is implemented in an iterative or recursive fashion.

For the breath-first search, starting again at a designated root node, for each node each branch is explored that is available at that node. Each child node that is discovered is placed into a queue with the search pattern being that of a "first in-first out" approach. Upon completion of this algorithm, a size value can be returned. The pseudocode for BFS works as follows [20]:

1. Enqueue the root node.
2. Dequeue a node and examine it.
   (a) If the element sought is found in this node, quit the search and return a result.
   (b) Otherwise enqueue any successors (the direct child nodes) that have not yet been discovered.
3. If the queue is empty, every node on the graph has been examined, so quit the search and return "not found".
4. If the queue is not empty, repeat from Step 2.

2.3 Other Products Currently Available

Speckle Tracker J This is a product released as a plug-in for the ImageJ image software [18]. The Baker lab has used it with some positive results. The output of Speckle Tracker J gives the instantaneous velocity of each object. Instantaneous velocity is the velocity of an object as the limit of time goes to zero. In other words, the velocity of an object at any given moment. This software is usable for research in the Baker lab and has gone to shown that there is a demand for researching this type of problem and its solutions.

SimplePCI Simple PCI is a high performance imaging software package with is able to process images at a high speed with low overhead [4]. This product is currently being used by the Baker lab for capturing the video and converting this video to a proper format. It also has some capacity to track objects. Though it is only able to give the constant velocities of each object. This is an average velocity, and for the sake of the Baker lab research, cannot be used as this type of output does not show when an object is bound or not bound just an overall velocity. So it is not very helpful due to the dynamic and quasi-chaotic motion of each object being analyzed.

2.4 We Have A Problem And No Solution Currently Available

So, there is a problem of analyzing the output of the SiMBA experiment in a timely manner. The tools necessary to generate such a method of analysis are at our disposal. We will now discuss the concept of what is needed in our software.

Chapter 3

Semi-Automated Analysis Software for a Novel Biochemistry Assay

We are proposing a novel software package for motion tracking and analysis of the movies produced by the SiMBA experiment. The high level method this program uses can be summed up into six different stages of processing. The data is imported and converted to an image format that is readable by the program. These images are then processed and filtered. Next, the program processes each frame to locate and determine the size of all objects present. This information is then analyzed to determine the events throughout the movie. The fifth stage is the user interaction and control of the software to change any parameters to get the best possible output. The final step is not dealt with in this program but instead is an analysis by the program Origin [5] to determine the overall behavior of the objects in the movie by generating a histogram. A lower level view of this process will now be examined.

Import Data The first goal is to convert the video data into images and import these images into our software package. The This starts with moving the images from the Simple PCI format (CXD) to an AVI file format.

Once the video is in an AVI format we can read it in with OpenCV. The video output format for SimplePCI is CDX. This is SimplePCI's own format and is not currently known to be convertible by a third party software. Thankfully, SimplePCI can convert this format to AVI as this is the only video file format allowed with

OpenCV. The downside of this conversion is the extra time added to get the final results though this step is necessary regardless of the analysis software used. Once the data is converted it can be read in by SANoBA . Using the video conversion functions of OpenCV each frame of the video is converted to the format PGM which is used to allow for an easily manageable set of data arrays.

Image Processing The initial frame images are not ideal due to the aforementioned issues, namely irregular illumination and a low contrast between the background and the objects being analyzed. These frames are filtered in the following manner. The original image is an RGB image which means that the image has three layers of colors. This is not necessary and in fact makes it less efficient in later steps to have multiple color layers. By using a splitting function, the RGB image is converted to a greyscale image.

A sample of the frame image is taken to determine the amount of threshold needed. The amount of threshold ranges from 15 to 100 of the greyscale values (0-255) and is applied using the *Threshold Binary* method [2]. This allows the background to have a value of zero (black) and any pixel attributed to an object will have a value of 255 (white). This threshold is applied and removes any background noise and distinguishes the objects in the frame.

Next, an optional Gaussian blur is introduced to smooth the edges of the objects and to reduce to amount of data loss of the objects themselves when the threshold function is called. This option is on by default, though can be turned off which may be necessary when the density of objects presents many objects that are very close together. If there are many close objects, the Gaussian blur may combine multiple objects into one thereby giving false readings. Lastly, the image is saved as a PGM type image to a folder that is named according to the current blur amount. This folder naming convention allows for non redundant processing if multiple settings are tested and possibly reverted to a previous blur level.

Image Analyzing and Object Coordination After all images are processed, the program moves on to analysis of the image and determining the objects locations in each frame. To analyze each image for the objects, the program uses a recursive algorithm that compares the filtered frame to a second reference image initialized to all white pixels. When searching a filtered frame and a pixel with a value greater than a given threshold is detected, the location surrounding this pixel is recursively searched to check if they are above the threshold as well. For every location that is above the given threshold, a value of (-1) is placed in the corresponding location in the reference image to prevent recounting the pixels repeatedly. When each object is found in its entirety, its size and location are inserted into a data structure. Each frame will have its own array of objects with their location, size, ID number and color. The ID number is assigned to each object of the first frame from 0 to n where n is equal to the final number of objects in the frame. To aid in visual analysis, a color value is also given to each object which is used in the display portion which is discussed later. For each subsequent frame, if a new object is detected, its ID will start at n to avoid multiple objects with the same ID.

Common Image Issues The process could be automated much more efficiently if not for some various issues that arise from time to time. I only discuss a few of these issues that have been present the most in this project. From there I will touch upon each issue a little more in depth.

1. Minor irregularities in image quality.
2. Focus may degrade for a series of frames.
3. Irregular illumination.
4. Moving objects cross over other objects.
5. Low contrast between the background and the objects being analyzed.

1. Minor irregularities in image quality.

The objects in the images are sometimes not well defined from frame to frame. They do appear very distinct in one frame but a multitude of factors may occur, giving rise to the semi-automated portion of this project. Also, image noise may occur and give false readings.

2. Focus may degrade for a series of frames.

The cameras focus will periodically shift causing misinformation into the next frame. For an efficient segue from each frame, these frames could be automatically removed or the user may visually connect each object manually.

3. Irregular illumination.

As an object moves from one location to the next, the size from the cameras perspective may increase or decrease from moderate to more significant. A range for the size is attributed to each object for motion detection, but at times the size can exceed a reasonable range in subsequent frames making the object ID different than previous frames.

4. Moving objects cross over other objects.

The occlusion of objects is a difficult problem when keeping track of objects. This is further compounded if the size of two crossing objects are the same or in a state of flux. At this time the approach taken is to assume if one object is not in motion and it is still not in motion after the occlusion then it is labeled the same before and after the event.

5. Low contrast between the background and the objects being analyzed.

The contrast between background and some objects may not be very high at times leaving some objects to not be accounted for after filtering. The optimal structures to be observed are single actin particles. The more particles that are joined, the brighter the object as seen from the camera. The single particles are not as bright, and if they are not bright enough, they will get lost in filtering or the motion detection may not account for an object properly.

Data Analyzing and Event Coordination The purpose of this step of the process is to analyze the retrieved data and find for each object its time and duration of being bound and unbound and locate all binding sites. For each frame, every object is compared with the previous frame at the current objects location. If an object is present within a user defined radius, the ID and color of the current object is assigned to this new object. If the position of an object in the previous frame is within a smaller radius it is determined to be not moving. If it is outside of this radius is it determined to be in motion. Every object has in its data structure the number of frames it has been present and if it has been in motion or not based on this radius comparison.

User Interaction and Control At this point the user can alter the various thresholds, turn blur on or off or just save the current state of the program and quit. The user interface was designed to gain the best possible analysis of each movie by allowing the user to adjust certain parameters to account for unforeseeable situations. All of the controls and UI will be discussed in further detail in the following chapter.

Final Output and Analysis As stated previously, the desired information at this time is the time bound (t_{on}) and time unbound (t_{off}). Two files are output for each of these event types. In these files are each occurrence and the duration of these occurrences. These numbers are then imported to the software Origin [5] to generate a histogram and for further analysis.

The software engineering and implementation for this software is described in Chapter 4 and Chapter 5, and a walk through of the application and a sample run is presented in Chapter 6.

Chapter 4

Software Design

This chapter describes the design of our software as a image analyzer and motion tracker for use in the experimental assay SiMBA. We describe both the functional and non-functional requirements of the implementation, use-cases, and a description of the classes which makeup the system.

4.1 Requirements Specification

Following standard software engineering guidelines, the main functional [19] and non-functional [16] requirements of SANoBA are presented below.

4.1.1 Functional Requirements

The most important functional requirements of SANoBA are:

F1. The system will convert movie files to image files.
F2. The system will process raw images.
F3. The system will locate objects in the images.
F4. The system will track objects over consecutive images.
F5. The system will disregard objects outside of a set size threshold.
F6. The system will allow size threshold adjustments at run time.
F7. The system will allow motion threshold adjustments at run time.
F8. The system will allow adjustment for amount of time object can be gone.
F9. The system will allow blur adjustments at run time.

F10. The system will store unique sets of frames based on threshold settings.

F11. The system will allow the user to add objects.

F12. The system will allow the user to remove objects.

F13. The system will allow the user to rename objects.

F14. The system will allow the user to navigate movie.

F15. The system will allow the user to animate movie.

F16. The system will allow the user to control environment.

F17. The system will output event data to a file.

4.1.2 Non-Functional Requirements

The most important non-functional requirements of SANoBA are:

N1. The system will run any machine with Windows 7.

N2. The system will be implemented using C++.

N3. The system will use OpenCV.

N4. The system will be easy to learn.

N5. The system will allow keyboard input for functions.

N6. The system will allow input by graphical interface.

N7. The system will allow mouse input for selection and navigation.

N8. The system will process any length of movie allowed by available drive space.

4.2 Use Case Modeling

The functionality of SANoBA has been defined using use cases and scenarios. The functionality of SANoBA is captured in the use case diagram shown in Section 4.2.1 at a high level of abstraction. This was done to help identify the mechanisms through which the user would interact with SANoBA . The use cases are compared to the requirements listed in Section 4.1 using the Requirements Traceability Matrix in Section 4.2.2.

4.2.1 Detailed Use Cases

Presented below are Use Cases for SANoBA . A Use Case Diagram is presented in Figure 4.1.

UC1. Install OpenCV 2.1 - OpenCV is the open source library that contains the necessary functions for this software. This is an easy installation and is contained in the software package.

UC2. Open data at runtime - The user can open valid movie files, saved state files, or navigate through the directory hierarchy.

UC3. Set parameters for analysis - The user may change parameters before opening data or after.

UC4. View details of objects - The details of all objects to be analyzed can be viewed on a frame by frame basis.

UC5. Change available objects at run time - The number of objects analyzed can be added or removed at run time.

UC6. Change available objects identity - The name of objects analyzed can be changed if necessary at run time.

UC7. Navigate processed visual data - Move forward, backward, or to a specific frame of visual data.

UC8. Animate visual data - Frames can be view one at a time or in automatic succession.

UC9. Control speed of animation - The speed of the animation can be controlled to go faster, slower, or stop all together.

UC10. QuickView of last viewed frame - The last framed viewed can be quickly referenced.

UC11. Control UI environment - A grid to aid in observation, name and color labeling of objects, and a help screen can be enabled or disabled individually.

UC12. Save state at program exit - The current state of the program can be saved at anytime or automatically upon exit of program.

UC13. Resume saved state at start up - Selection of a saved state will resume the state of the program to a prior state.

UC14. Output data for further analysis - The output data is then used in generating a histogram.

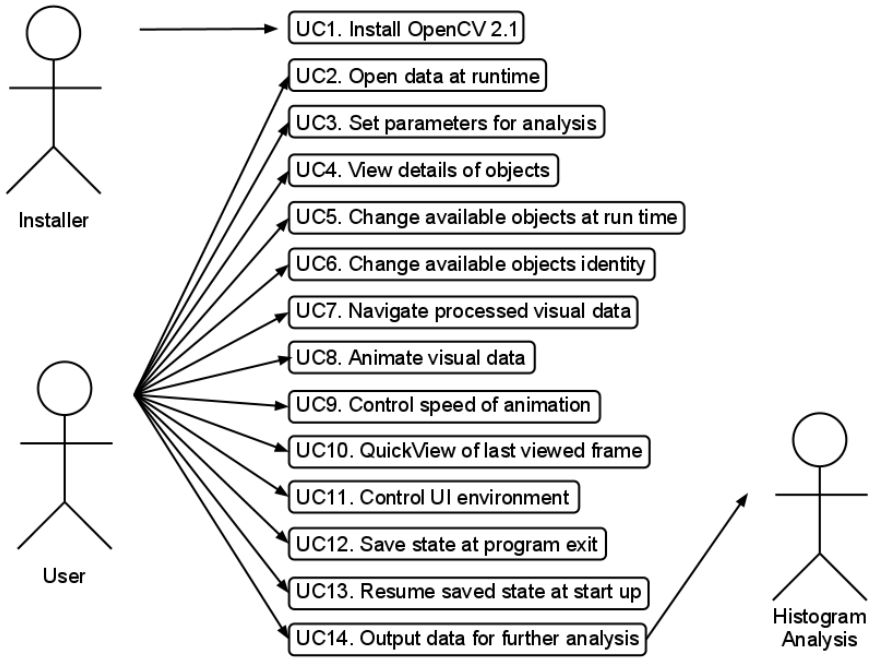

Figure 4.1: Use cases for the data analyzing system.

4.2.2 Traceability Matrix

The Requirements Traceability Matrix detailed below shows how the use cases match up with the requirements listed in Section 4.1.

4.3 Class Diagram

The class diagram for SANoBA is presented as according to the specifications in [1]. Figure 4.3 lists all the classes in SANoBA as well as most of the major functions. Due to size constraints, certain trivial variables and functions have been omitted, as have OpenCV inherited functions.

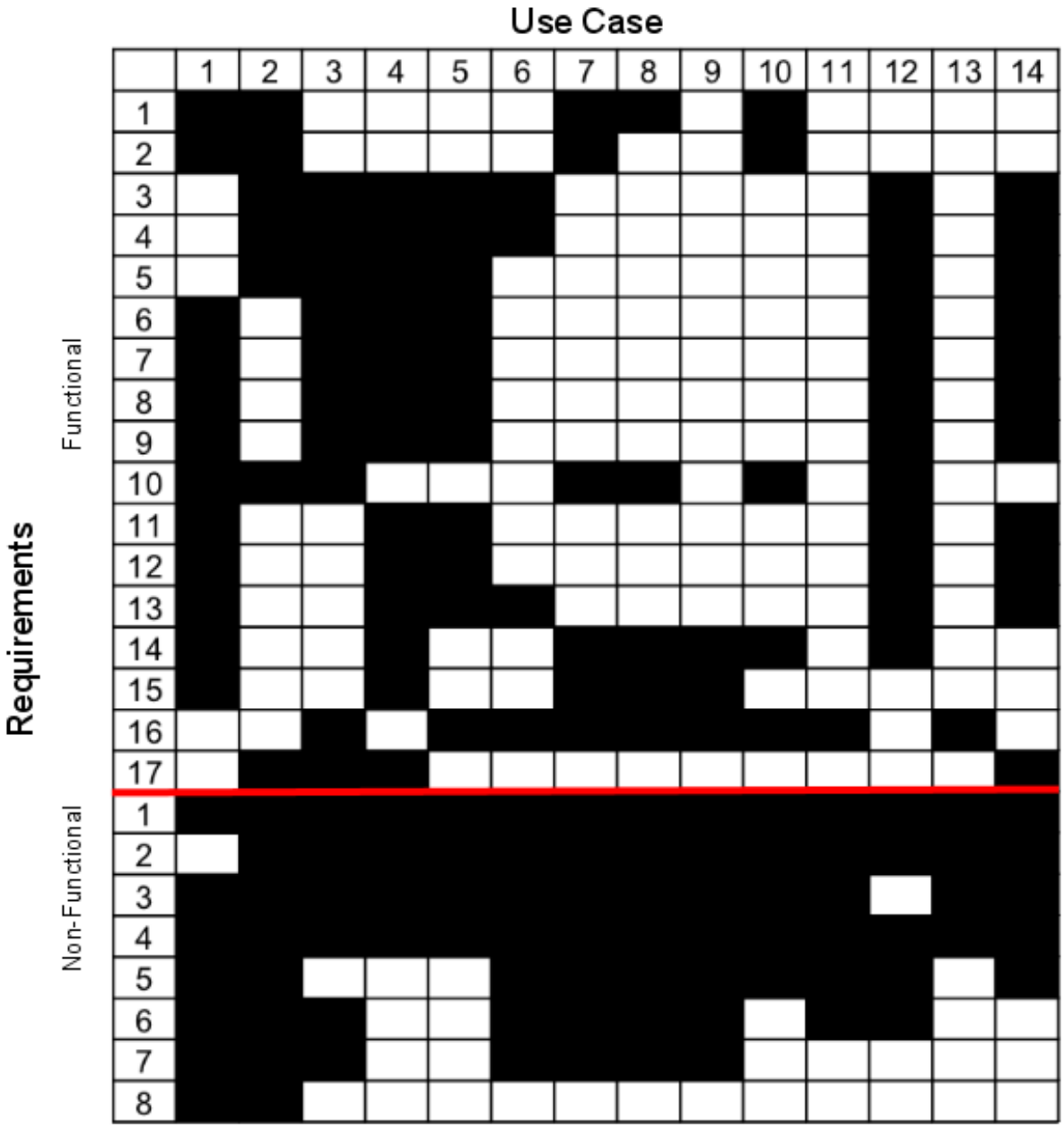

Figure 4.2: Requirements Traceability matrix.

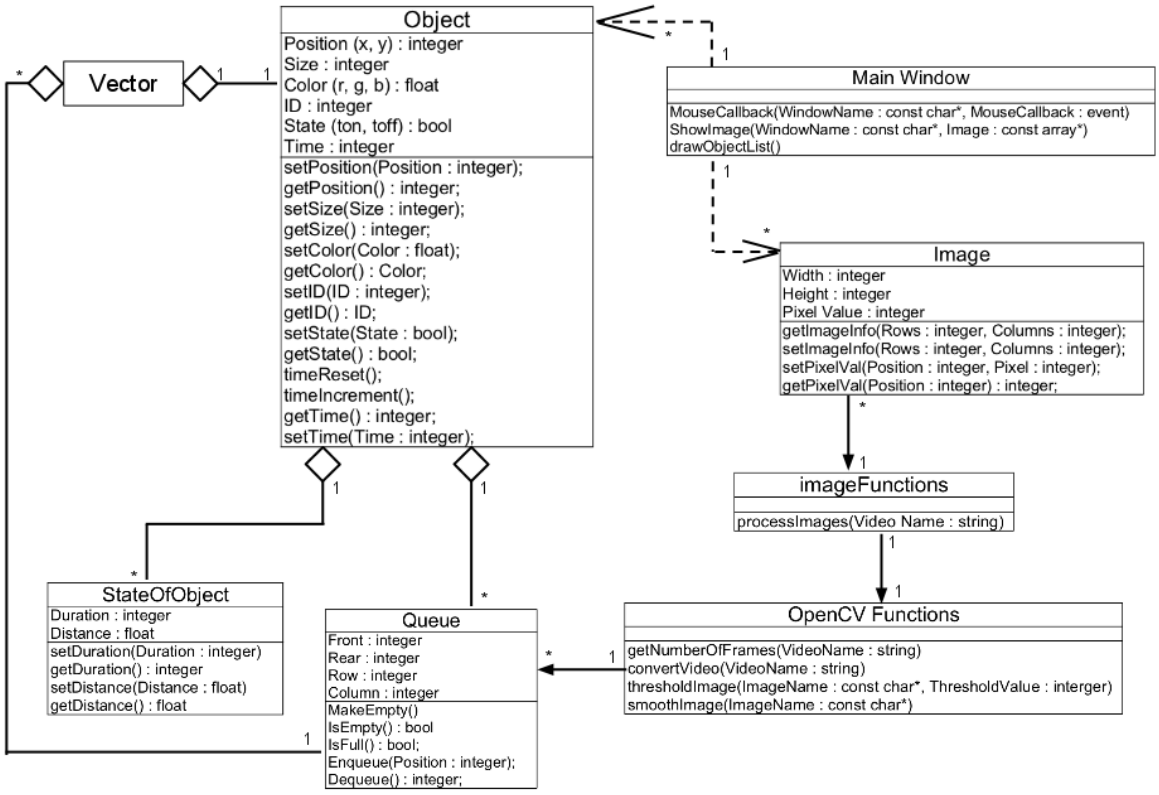

Figure 4.3: Class diagram for SANoBA .

Chapter 5

Software Implementation

5.1 OS Development

Application development began in the Linux environment. This was a necessity to insure development began in a timely manner as a proper windows developer environment was not readily available at the time. The application was able to be ported to the windows environment with few changes and no major issues. With a few checks and a proper initialization, this program could be developed to run in either environment, though it is not necessary at this time and only the windows version is currently needed.

Since the main focus of this project is to help with the analysis in Dr. Baker's lab and the computers that are used for the experiment are windows machines the main OS used for developing is Windows 7. The developing environment used was Visual Studio 2008 and 2010.

5.2 OpenCV and OpenGL

The original intent was to develop the main user interface in OpenGL. This was quickly realized to be a taxing process when combined with the image processing and data analysis procedures. In order to achieve the fastest results possible, a minimal number of processes should be used for the entire program. The idea of using a static image for the user interface could have been used with minimal resources in the OpenGL environment, though the overall program would gain undesired behaviors as

the main tracking processes are built in OpenCV. By having these two libraries in the same program, control must be flipped from the OpenGL processes to the OpenCV processes. This creates an undesirable visual latency for the user and an unnecessary increase in programming time and effort for the developer. As a future addition, a kymograph output will be developed. This is a 3 dimensional representation of all the objects positions over time and is only capable of existing in an OpenGL environment. Since this kymograph is not an analysis of data but a visual representation of previously analyzed data, switching to this visual output is not as taxing to the program. And with the help of some simple functions the program can allow for going back and forth between the two environments with low latency due to the infrequent occurrences of the kymograph visualization.

The OpenCV library assists in opening, importing, and converting the raw movie data into C++ compatible data objects. With few exceptions, the built in functions are very fast and stable making this library a welcome addition for developing this project. OpenCV also has built in functions which simplify the image filtering phase that is done to correct minor irregularities in image quality. It also has a manageable set of features that allow for mouse motion and action detection. This was an essential feature to allow for user interaction as opposed to a program that can only allow parameter input prior to run time with subsequent application closure, adjustment and relaunching.

5.3 User Interface

The idea behind the icon style of the UI is to make this software easy to use. The goal of this software is to have as little involvement on the user side as possible. With this in mind, the amount of customization is relatively limited as compared to competing applications. The entire application is run through the main user interface window with exception of a secondary window that will be discussed shortly. If this UI is resized, the window and mouse coordinates will operate normally. All of the available functions are shown as an image that corresponds to its function. For example, the

“Next frame” button is of an arrow and the function that shows or hides the names of each object is represented by an image of a name tag commonly seen at conventions. There are multiple methods to perform many of these available functions so as to not limit the user to any one method of control. These include the use of the keyboard and mouse navigation that does not involve the icons, both of which will be discussed later. The available functions and how they are used will be discussed in detail in Chapter 6.

The Secondary UI is a window that will pop up when a particular object is to be renamed. This window allows the user to navigate through the movie without losing there current position. The secondary UI will also be discussed further in Chapter 6.

5.4 Keyboard and Mouse Controls

As mentioned earlier, functions can be accessed through the keyboard if so desired. The available keyboard functions can be seen by clicking on the keyboard icon in the lower right corner of the main UI or by pressing the ’k’ key. By using the keys, the user can navigate the movie, open a file, save the current state to mention just a few things.

Aside from clicking the icons, the user can click in the area above the icons to navigate the movie. By right clicking, the movie moves forward one frame. Left clicking moves back one frame. The middle button will start or stop the animation and the corresponding right and left mouse clicks will speed up or slow down the animation speed respectively.

5.5 Troubles Encountered

As with any software engineering project there is bound to be some issues encountered that must be resolved and can take many hours to do so. In this section we will discuss a few of these issues that seemed to have causes the most problems in this project. We have already discussed the issues of combining **OpenGL and OpenCV** but we

wanted to mention it here as many hours were spent debugging and figuring out this problem. In the early stages, **flickering** was a problem as it made the user slightly nauseous when using this program and it helped present the notion that the resources were not being properly used. It was later discovered that there were two main reasons for the flickering. One reason was the loop used to listen for keyboard commands needed to be of short intervals to allow "instantaneous" registration of keyboard presses but also could not be continuous as to diminish the available resources. But the major reason was the use of the frame name for the window displaying the images in OpenCV. By using the name of the frame and then changing frames, the window would be destroyed then a new window would be generated using the new frame name. This ate a lot of resources and was unnecessary. By never changing the name of the window, it was not continuously destroyed and the flickering was resolved.

Memory leaks can spring up time to time if the programmer is not careful with their pointers. But in OpenCV, it may not be apparent right away that memory leaks are occurring until of course in my case an entire movie was processed and half way through viewing the movie the program would crash. This crash was quickly determined to be the result of a memory leak, but finding the source of this leak was another matter. After some time, the source turned out to be from the not deleting the pointers to the images as they were viewed. With a restructuring of the animation and navigation functions, the images were properly destroyed at the proper times and the crash is avoided. **Mouse coordinates** proved to have their issues as well. The scale of the coordinates was not correct when exchanging from OpenCV to the custom image processing techniques that rely on pixel position instead. Also, OpenCV reverses the X and Y values from the common vertical as being Y and horizontal being X.

Irregular illumination sometimes occurs, but is corrected on a frame by frame basis. Originally, the overall movies contrast was set based on the first frame. This gave overexposed frames an improper contrast setting. To alleviate this problem, each frame has a sample taken giving the average contrast of the entire frame. The

contrast of each frame has also been improved by changing from a discrete based adjustment where a specific contrast compensation value given to a frame was based on a a set of ranges to now being a continuous based adjustment thereby increasing the resolution of each frame and giving the converted images an even contrast ratio across all frames of a movie.

5.6 File Handling

File naming is done by using the current time when saved in combination with the amount of blur and the size threshold currently being used. By using these certain values in the file name, the user can find the needed file easier when opening a saved state. The saved **values in the file** are seen in Table 5.1 and Table 5.2.

When processing a new movie or a different blur value the program is able to generate **dynamic folders** that contain the set of images for that blur level. A new folder or image processing does not occur if the movie has been converted at that

Table 5.1: Contents of a saved file.

1	Position threshold.
2	Confirmed blur value.
3	Size threshold.
4	The number of frames an object can be missing.
5	Date and time of save.
6	Video file name.
7	Current frame number.
8	Total number of frames processed.
9	Frame info**(see Table 5.2)**.

Table 5.2: Frame info

1	Number of objects for each frame.
2	Each objects x and y position.
3	Each objects size.
4	Each objects ID.
5	Each objects color in floating RGB.

blur level previously. The name of the folder for these groups of images are named using the movie name and the blur level. If the general “frames” folder that all of these dynamic folders are placed is not available it will be created.

When **opening files** the type of the file is checked and only other folders, AVI and specific TXT are opened. By clicking on a folder the contents of that folder are shown or if the “open” folder icon is clicked or ‘o’ is pressed the info window will go up a directory. If the icon is clicked or ’o’ is pressed in the top parent directory then the open file function will be canceled and the program will resume as normal (unless nothing has been opened yet, then nothing happens). After opening a saved file the IDs and colors for any new objects will start from the highest ID in the file +1 (and the highest color +1).

Chapter 6

Results

6.1 Main User Interface

This chapter covers how someone may interact and utilize the application to ensure proper results. Figure 6.1 presents the main user interface. To run this software on a new computer, certain dependencies are required and are included in the deliverable software package. Since SANoBA was developed in Visual Studio with OpenCV

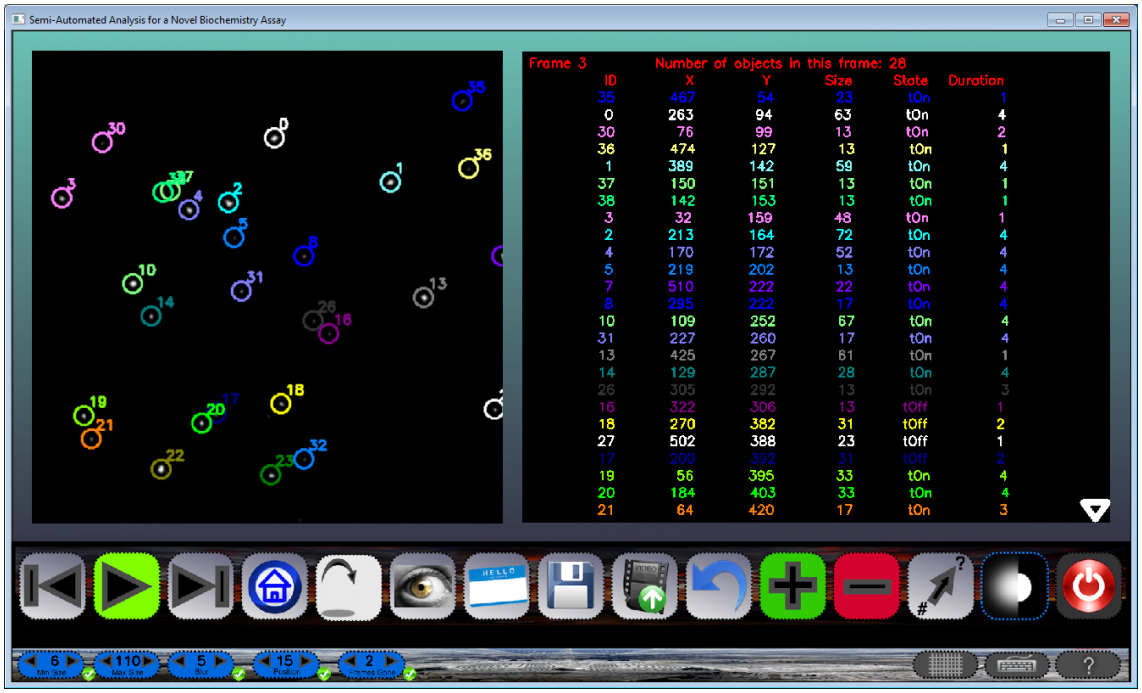

Figure 6.1: Main user interface

Table 6.1: Main User Interface.

1	Objects Window.
2	Red bar that is present at the top of the first frame only.
3	Information window.
4	The current frame number.
5	IDs of object in current frame.
6	X and Y coordinates of each object present in current frame.
7	The size of each object in the current frame.
8	The state of each object in the current frame.
9	How long that object has been in its current state.
10	Arrows only present when more objects than the list can display.
11–25	Program functions **(see Table 6.2)**.
26	Available keyboard functions.
27–31	Threshold settings **(see Table 6.3)**.
32	Show or hide the number grid.
33	Show or hide the available keyboard functions.
34	Show the help screen, any mouse/keyboard action closes this screen.

version 2.1, when installing to a new computer you need to copy the entire deliverable to your computer and install OpenCV 2.1 that is included in the dependencies folder.

Note the violet numbers in Figure 6.2 have been added to help identify certain elements. These elements are listed in Table 6.1 and are described below.

The **Objects Window** (1) is where the processed images are arranged and the user can see how the objects are analyzed and represented through out the entirety of the movie. On the first frame of each movie there is a **red bar** (2) present to aid the user during animation of the movie. The second window in the main UI is the **Information Window** (3) that contains the pertain information of all objects present in the current frame. In this window the **current frame number** (4), **object IDs** (5), **X and Y coordinates** of each object (6), **object size** (7), **object state** (8), **object state duration** (9) and **arrows** (10) when needed. The arrows are only visible to the right of the list of items in the information window if there are more objects or files than this window can hold. The up and down keyboard arrows also scroll through list of objects and files.

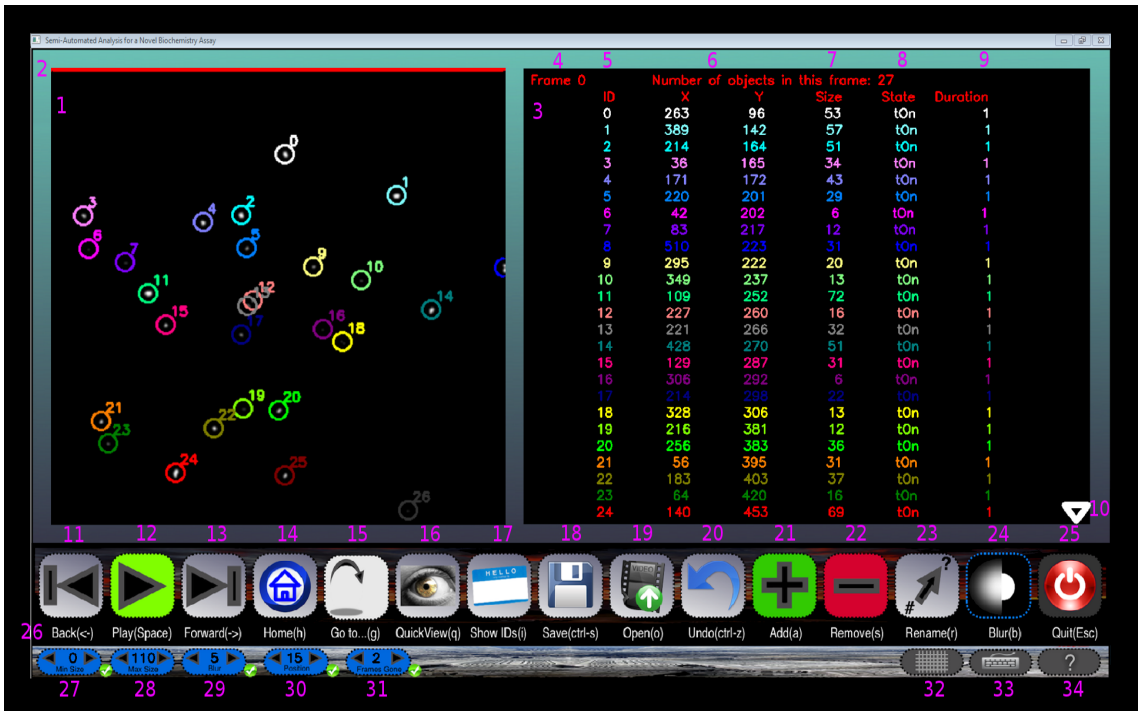

Figure 6.2: Main features in UI numbered.

The program functions (11–25) and threshold settings (27–31) will be covered later. The keyboard functions can be shown or hidden with 26 in Figure 6.2. Numbers 32, 33 and 34 will show or hide the number grid (Figure 6.3), available keyboard functions (Figure 6.4) and help screen (Figure 6.5) respectively. The **grid** helps the user to determine the distance an object may travel by adding a mesh with numbers and letters on the x and y axis. The **keyboard shortcuts** shows the keyboard functionality for quick reference. In the **help screen** all of the functions and settings are shown with a brief description. Any mouse or keyboard action closes this screen while active.

Items 11–16 in Table 6.2 deal with **navigating** through the frame images of a processed movie. When the animation is not on, the user can go from the current frame to either the next or previous frame using 11 and 13 in Figure 6.2, or jump to the beginning or a specific frame. The frames can be set to automatically progress by using 12 in Figure 6.2 with the speed being controlled by 11 and 13 to decrease

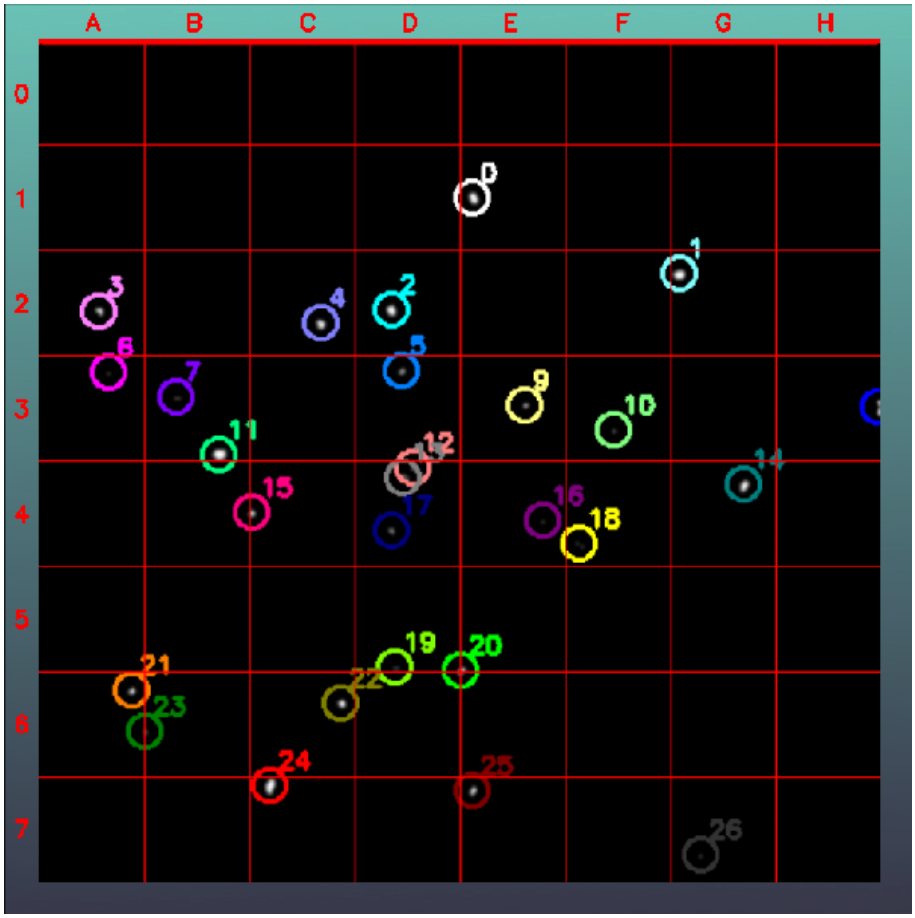

Figure 6.3: Numbered grid.

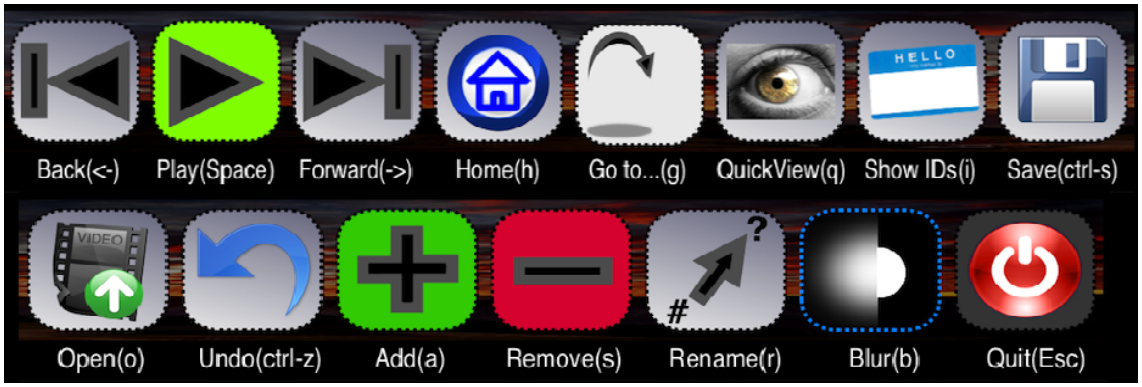

Figure 6.4: Keyboard Shortcuts.

Table 6.2: Program Functions.

11	Go back one frame or slow down animation speed.
12	Start or stop animation.
13	Go forward one frame or speed up animation speed.
14	Go to the first frame of movie.
15	Go to a specific frame number.
16	See the last frame viewed, not available if jumped frames.
17	Show or hide ID numbers and or circles around objects.
18	Save current state of program and return later if necessary.
19	Open a new movie or a saved file.
20	Undo, currently unavailable (future work will be for remove and rename).
21	Add an object (will not work if object is smaller than threshold).
22	Remove object, if multiple objects selected user can choose which to remove.
23	Rename an object (Activates secondary window).
24	Turn blur/smoothing on or off.
25	Quit program.

Figure 6.5: Help Screen with all functions.

or increase the animation speed respectively. To quickly go to the first frame of the movie, press the "Home" icon (14 in Figure 6.2). If the user needs to go to a specific frame, when "Go to..." (15) is active, the user can type in the frame number and press enter to quickly go to that frame. As seen in Figure 6.6 and Figure 6.7 the "Go to..." is pretty straight forward. The frame to jump to can be erased with the backspace and is only confirmed with the "enter" key. The QuickView button (16 in Figure 6.2) is mainly to see the previous frame as the user views the movie frame by frame for easy reference. However it really allows the user to view the last seen frame. So if the current frame is 8 and the last frame viewed was frame 9, the QuickView function would show 9 not 7. This is only available for consecutive frames and does not hold if the user jumped frames.

By default, the **IDs** of all objects are present as well as an encompassing circle as seen in Figure 6.8. This can be changed so that the IDs are not present but the circles are or both can be removed entirely. The IDs can be hidden without any penalties

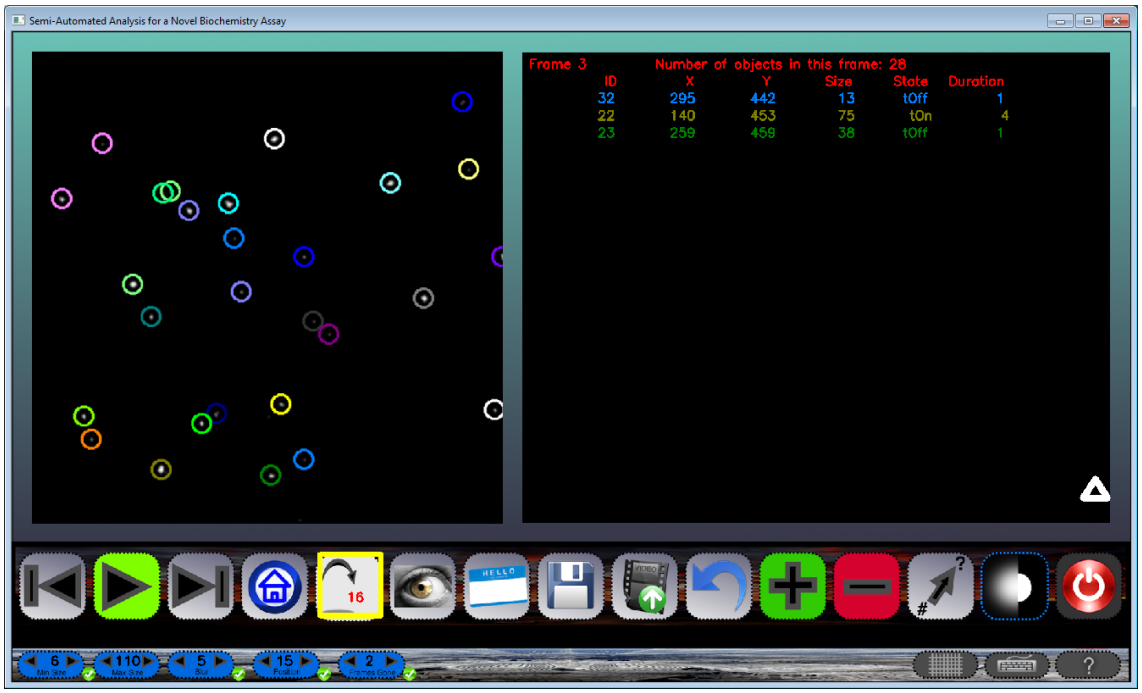

Figure 6.6: Go to a specific frame.

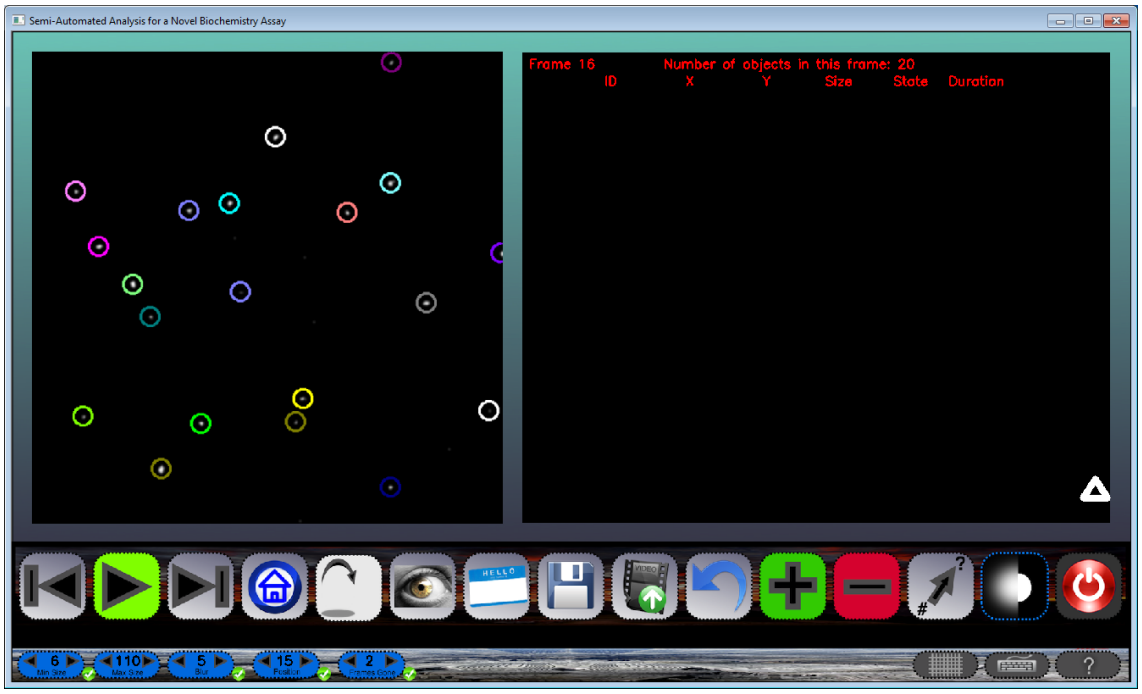

Figure 6.7: Go to a specific frame.

on analysis as seen in Figure 6.9. To see the objects clearly like in Figure 6.10 it is helpful to remove all augmented artifacts from the viewer.

If the current state of the program needs to be saved, the icon **save** stores the current state of the program. All objects removed, added or renamed as well as the current frame and threshold settings will be as they are when a saved file is opened. The "saved" files are stored in a folder rightly named "saved" and can be opened any time the user desires.

The user can traverse the folder system to **open** any file or movie needed Figure 6.11. The allowed files at this time are AVI movie files and TXT files in the format of SANoBA and folders with in the parent directory of the program. After opening saved file, IDs and color of new objects start from +1 of highest color and ID in file so that if a object is added then it will start from that point so as to not repeat any IDs.

To **add** an object first select the green plus symbol (see highlighted in Fig-

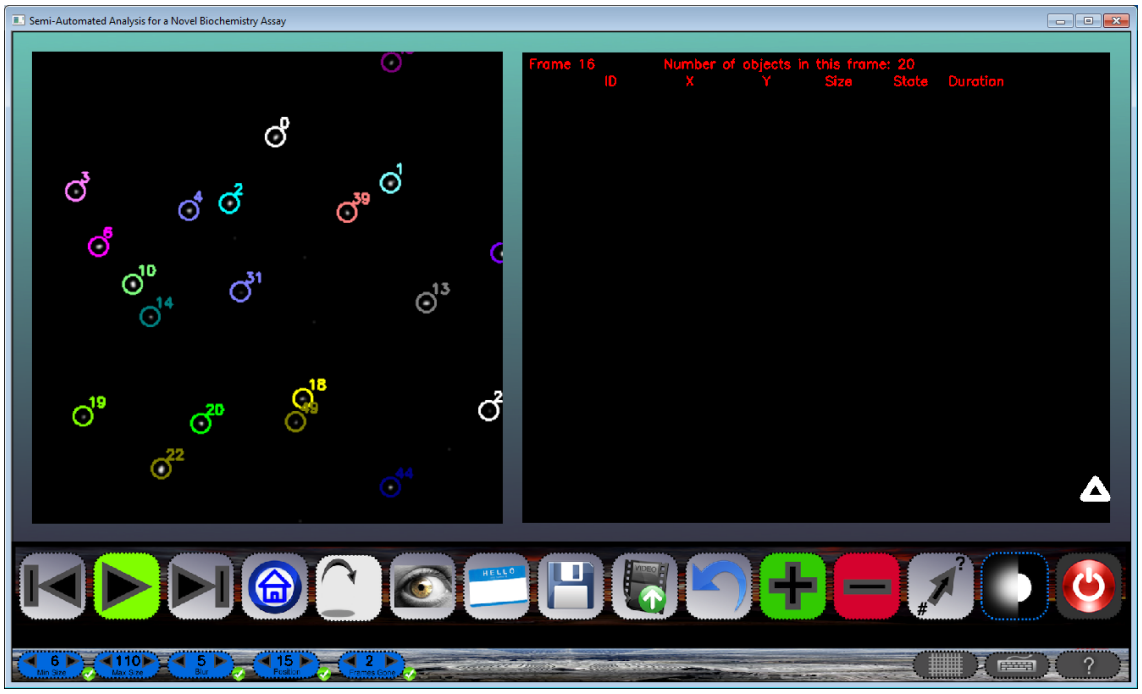

Figure 6.8: Default view showing IDs and location circles.

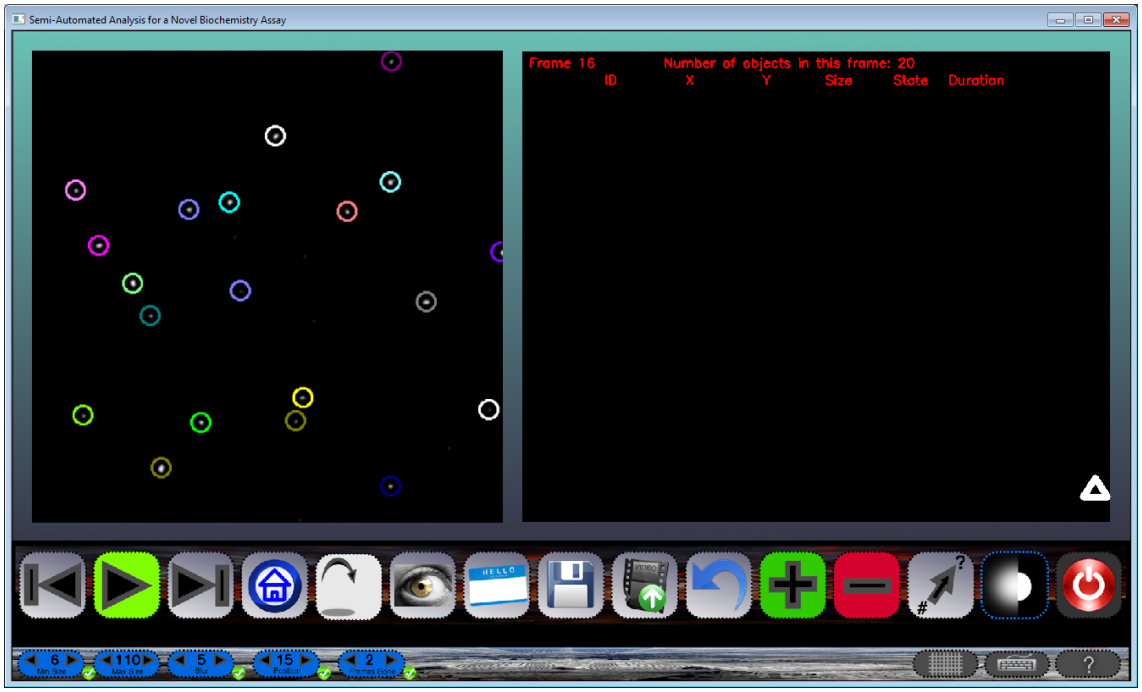

Figure 6.9: IDs removed with location circles still visible.

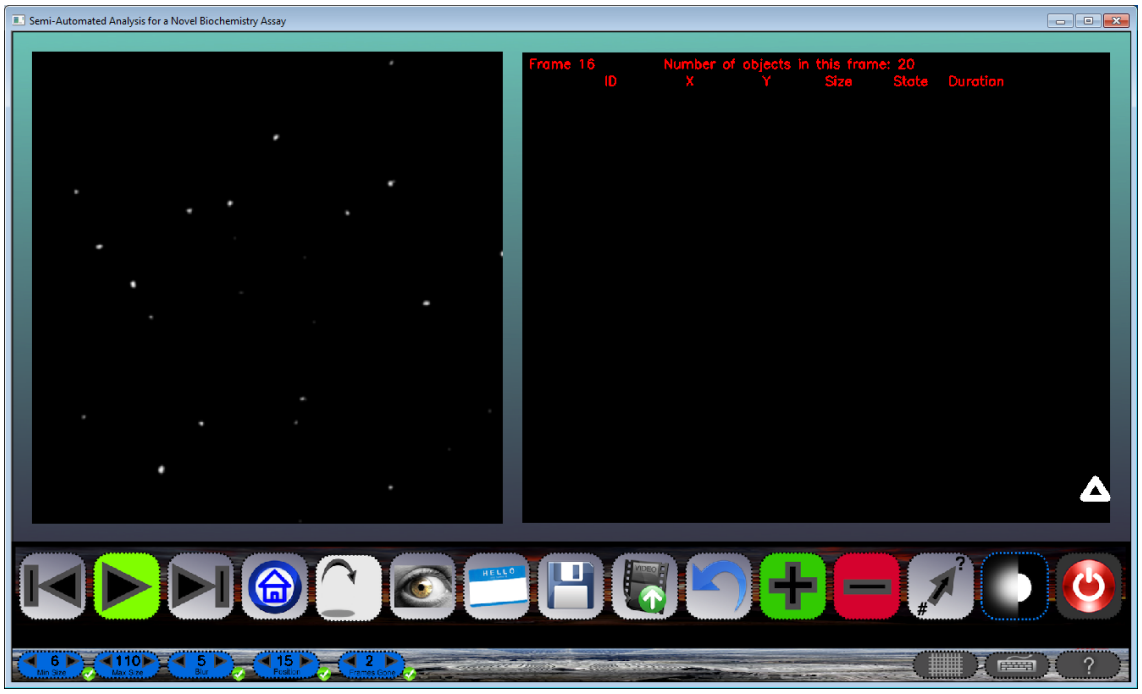

Figure 6.10: IDs and circles removed.

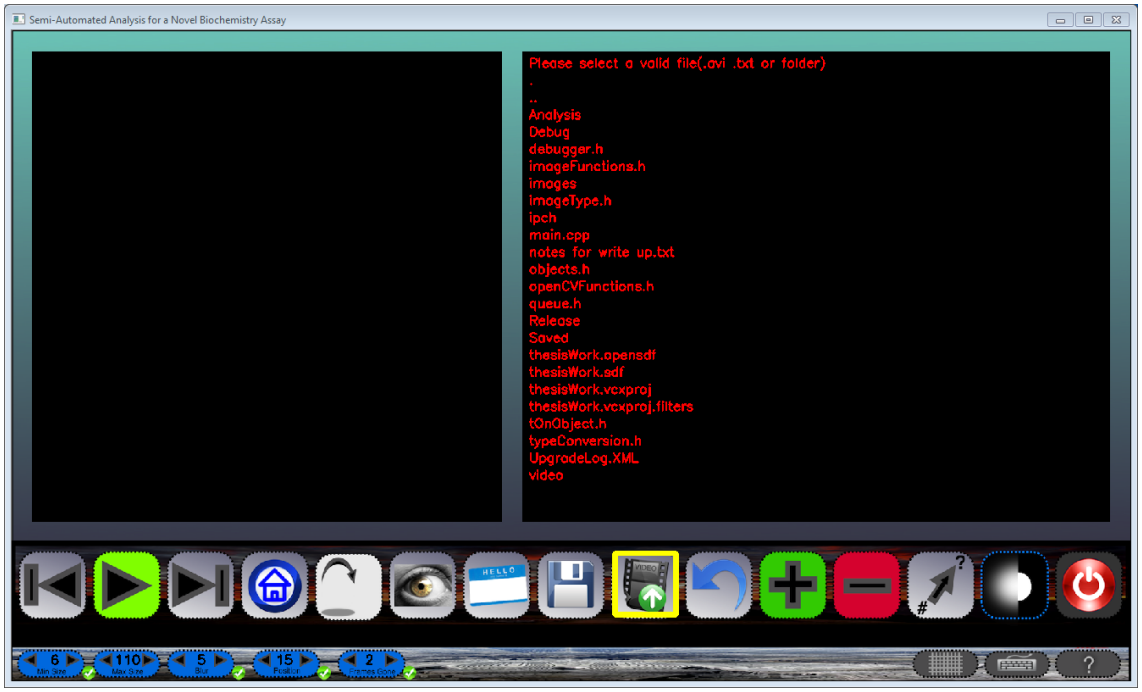

Figure 6.11: Folder system to be navigated.

ure 6.12). It is hard to see the object in Figure 6.12 just under object 18, but it was deleted by accident by the user and they wish to add it back to the object list. After selecting the add option, click and drag with the mouse to encompass the object to be added (Figure 6.13). If the object being added has a size less than the threshold set, it will not be added. If multiple objects are selected and they both are within the threshold limits, they will be added and as separate objects. At this time, the object(s) added are only added in the current frame (Figure 6.14). A future update would be to have the option to add the same object in future frames or just current.

Removing an object allows the user to **remove** any unnecessary objects from being analyzed we will see in Figure 6.15. Removing an object is done with a mouse selection box that occurs with a click and drag method. A selectable list is present in the "Information Window" if more than one object is selected. The user can select from this list the object to be removed Figure 6.16. As you can see in Figure 6.17

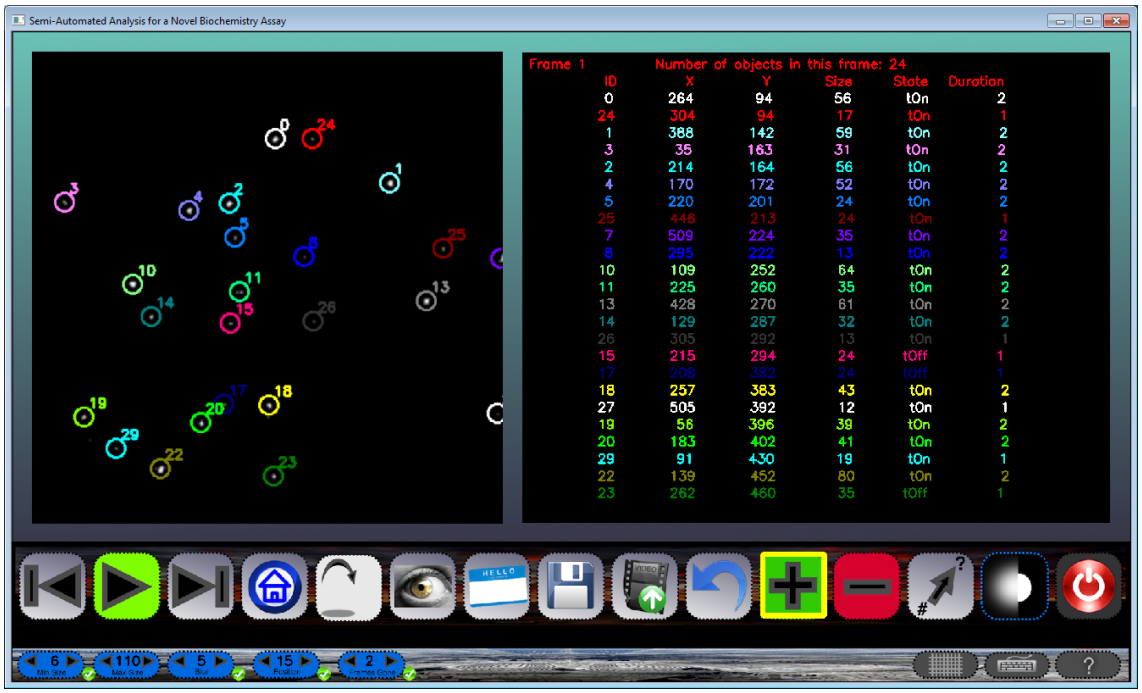

Figure 6.12: Adding an object.

Figure 6.13: Adding an object.

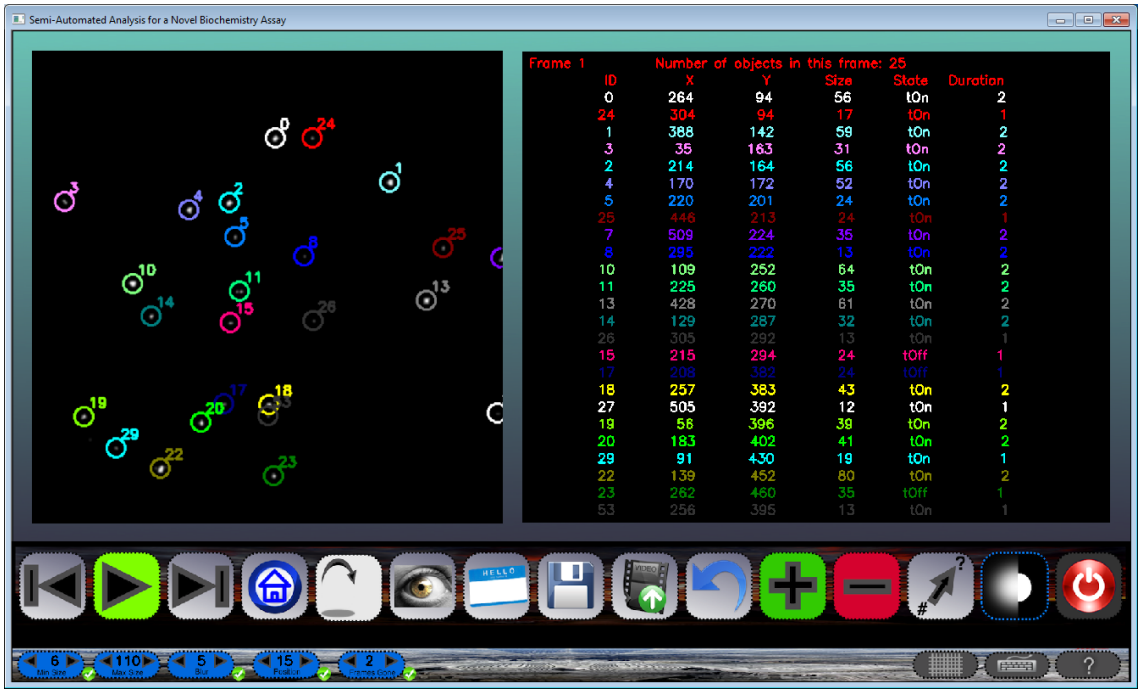

Figure 6.14: New object is added to list.

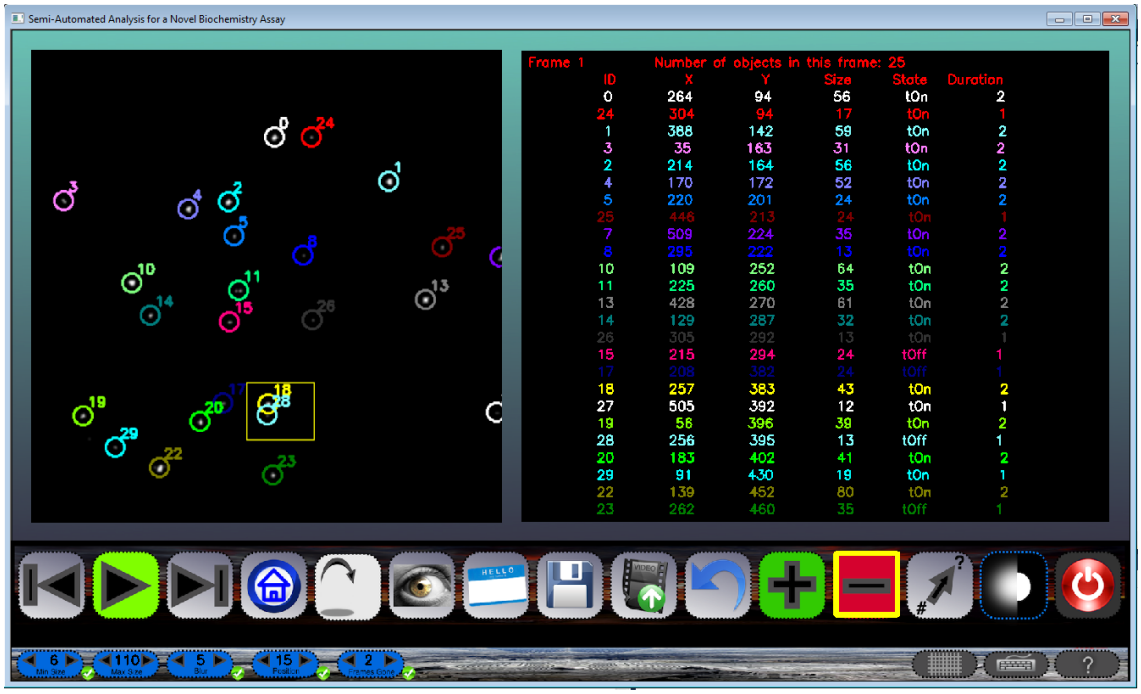

Figure 6.15: Remove function is selected as are multiple objects.

the object labeled 28 is removed from both the object window and the information window. When an object is removed, all objects with that ID are removed from future not previous frames and is also removed from any further analysis unless it is manually added back or the movie is opened as a new movie not from a saved session. If no object is detected in the mouse selection box, the mode remains active until the user selects 1 or more objects or selects a mode (same or otherwise) or presses the escape key.

As we will see from this next example, there may come a time when the user may need to **rename** an object. The remove function start off similarly to the add function in that is uses a mouse selection box. Also, if multiple items are selected a list of items are displayed in the "Information Window". When an ID is changed, IDs for all future frames containing same ID are updated as well, though the search stops when a frame without that ID present is happened upon. When changing an ID, a check is done of that ID in the current frame so that two objects in same frame cannot have

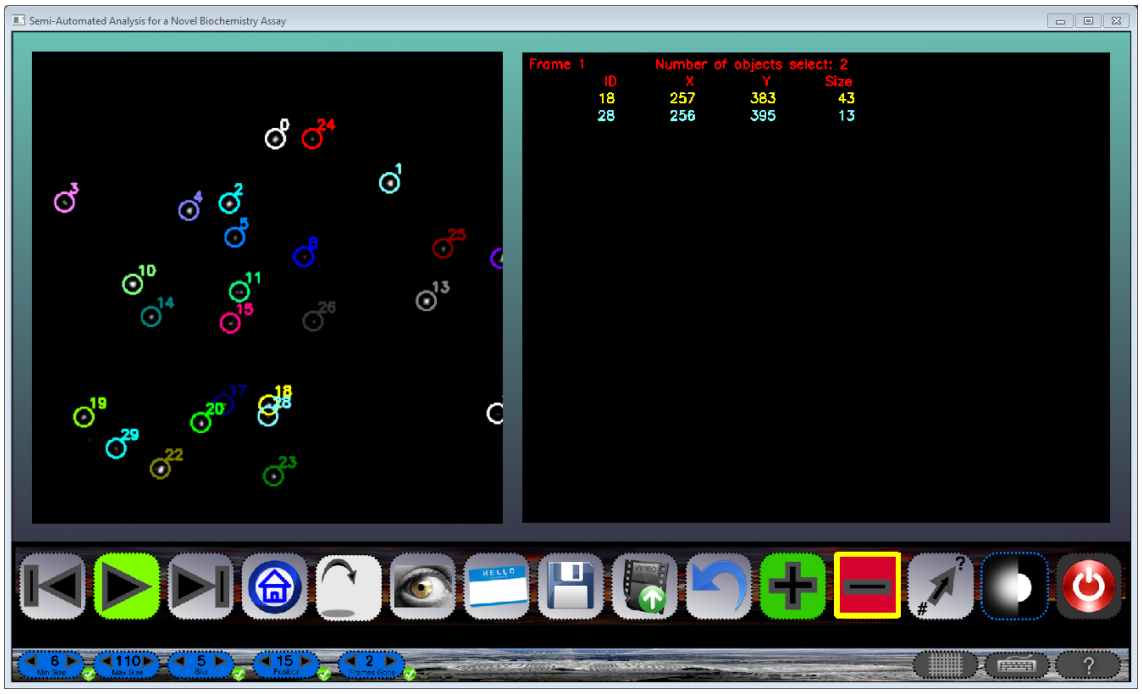

Figure 6.16: List of the selected objects.

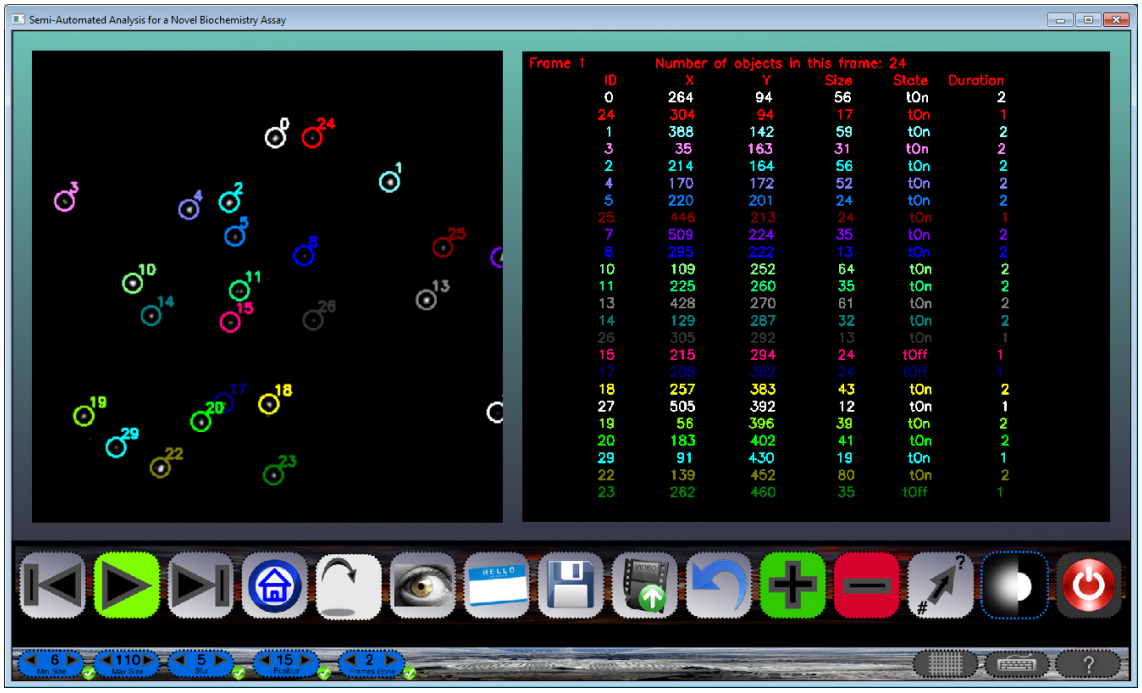

Figure 6.17: Updated UI with 1 object removed.

same ID. The ID is found be the user moving to different frames(forward/backward) in the secondary window that will be discussed later. Now an example is created to show how the renaming function operates.

As seen in Figure 6.18 there is an object with the ID of **9**. Also in this figure it is seen that the number of frames an object can be gone until it is considered gone is set to 2 frames. In Figure 6.19 it is four frames later and we see the return of this object yet now it is labeled with the ID of 54. When the "Rename" function is active, the user may select an object Figure 6.20. The same resulting list will occur if multiple objects are selected. Once an object is selected for renaming the secondary user interface (this will be discussed later) is presented Figure 6.21 and will automatically be on one frame previous from the current frame. Using the navigation controls, the user can go to the frame that the object with the desired ID is present. Once the desired ID is found, the user can enter the number as seen in Figure 6.21 and press enter. Lastly, it is seen in Figure 6.22 that the desired ID is present to the object in question and all analysis will follow suit to this update.

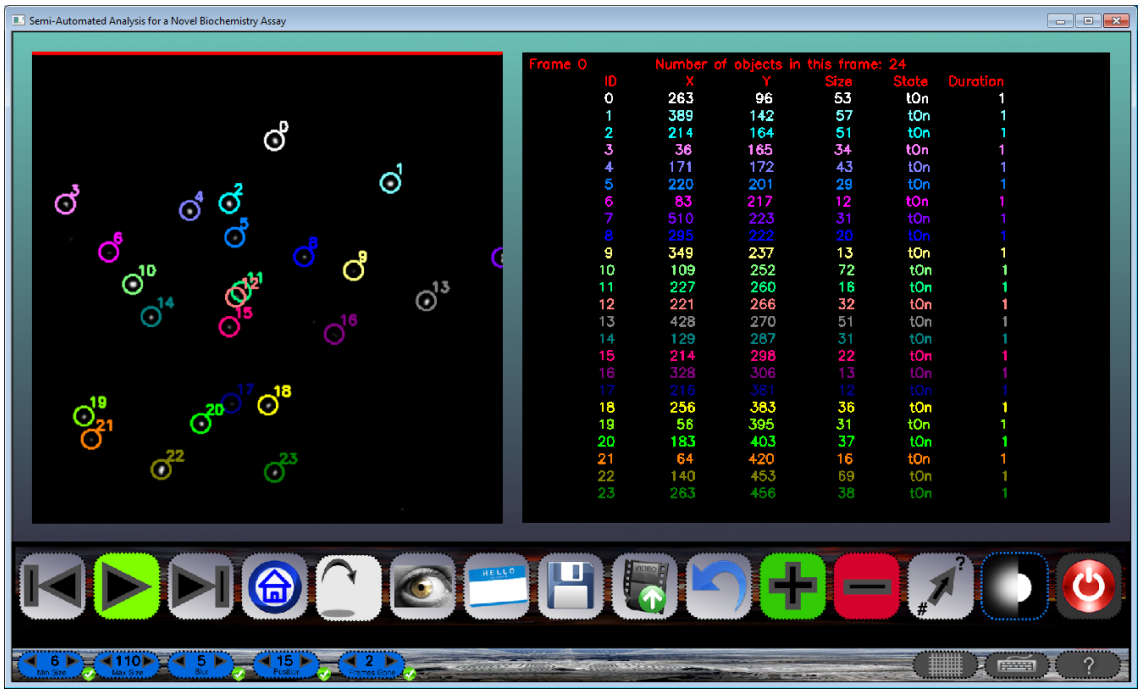

Figure 6.18: First frame and we see object 9 in yellow.

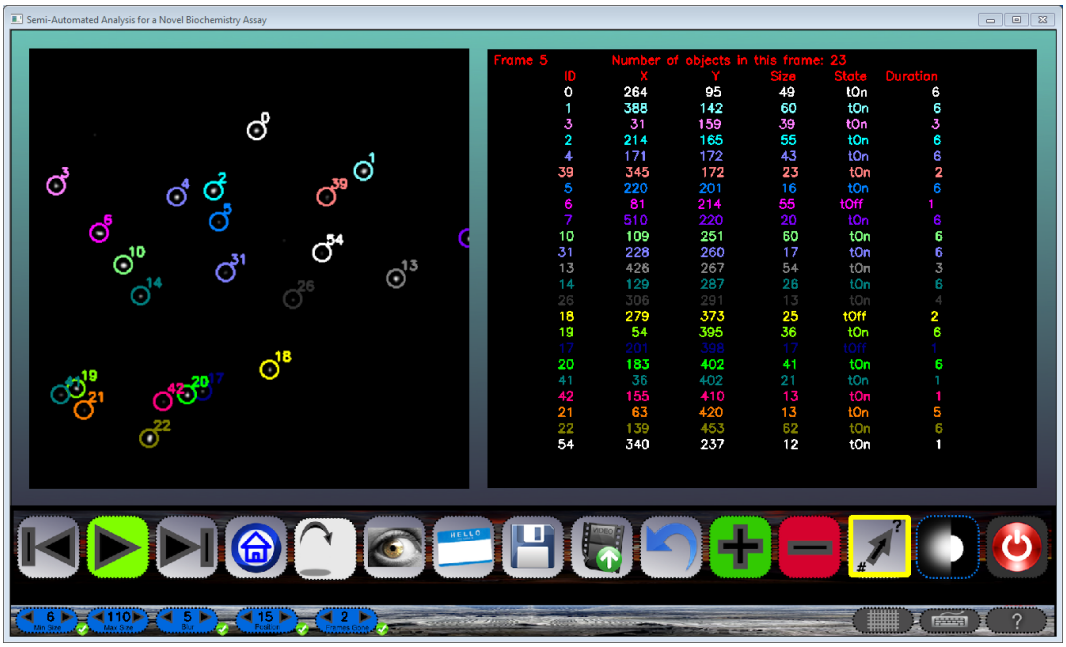

Figure 6.19: In frame 5, object 9 returns with a different ID.

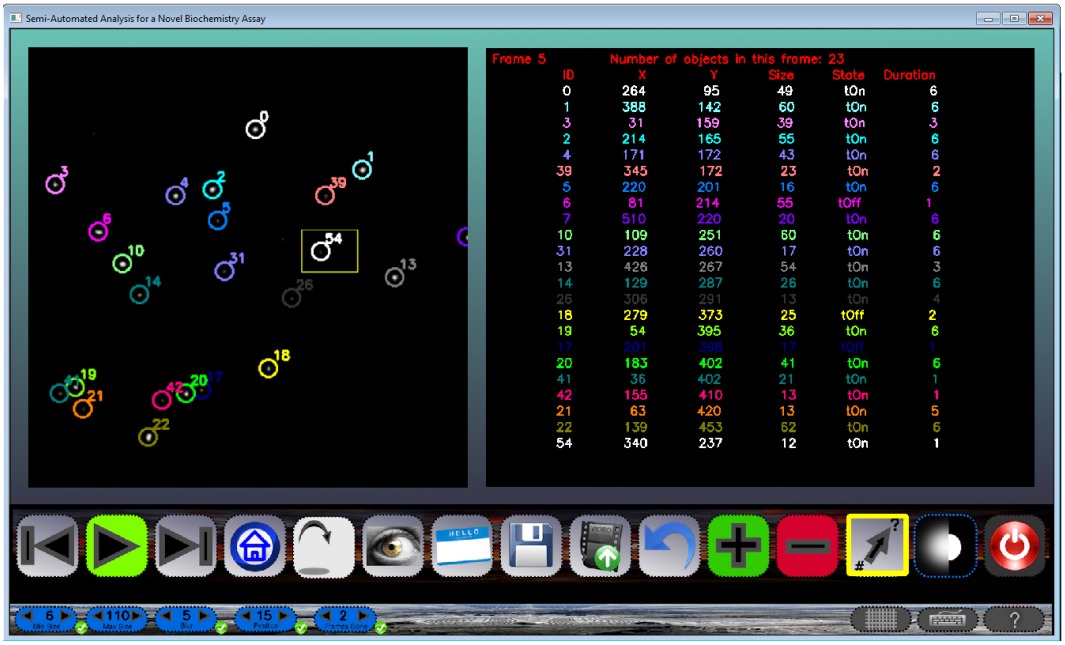

Figure 6.20: Only one object is selected so no list is present.

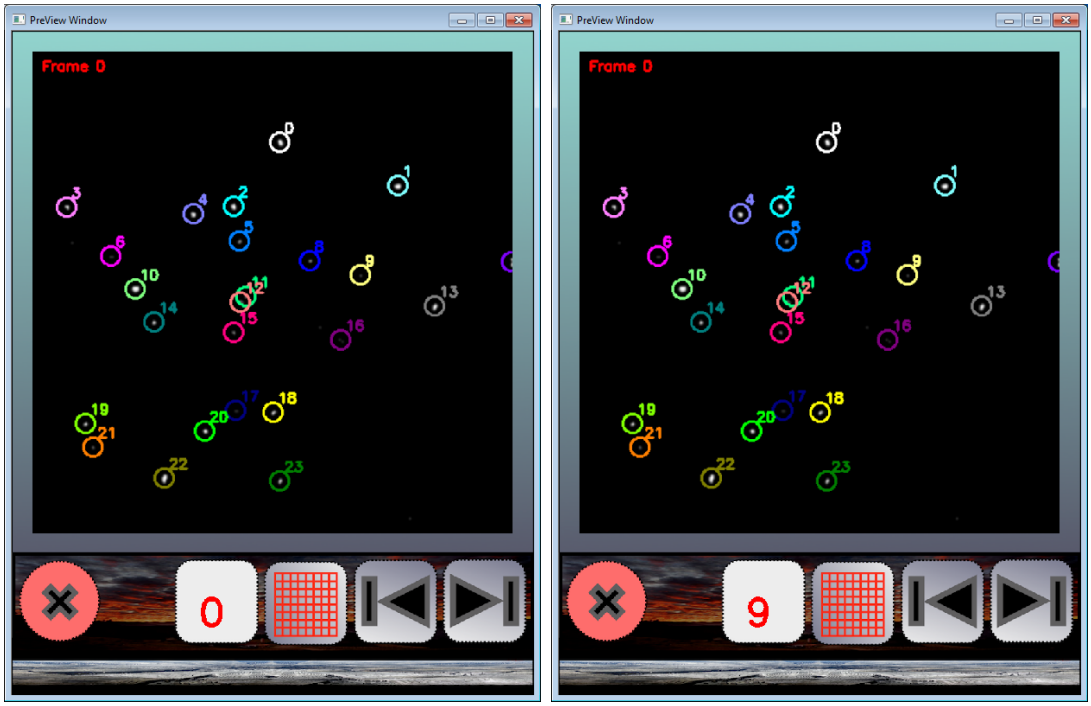

Figure 6.21: Secondary User Interface. On the right the desired ID is entered.

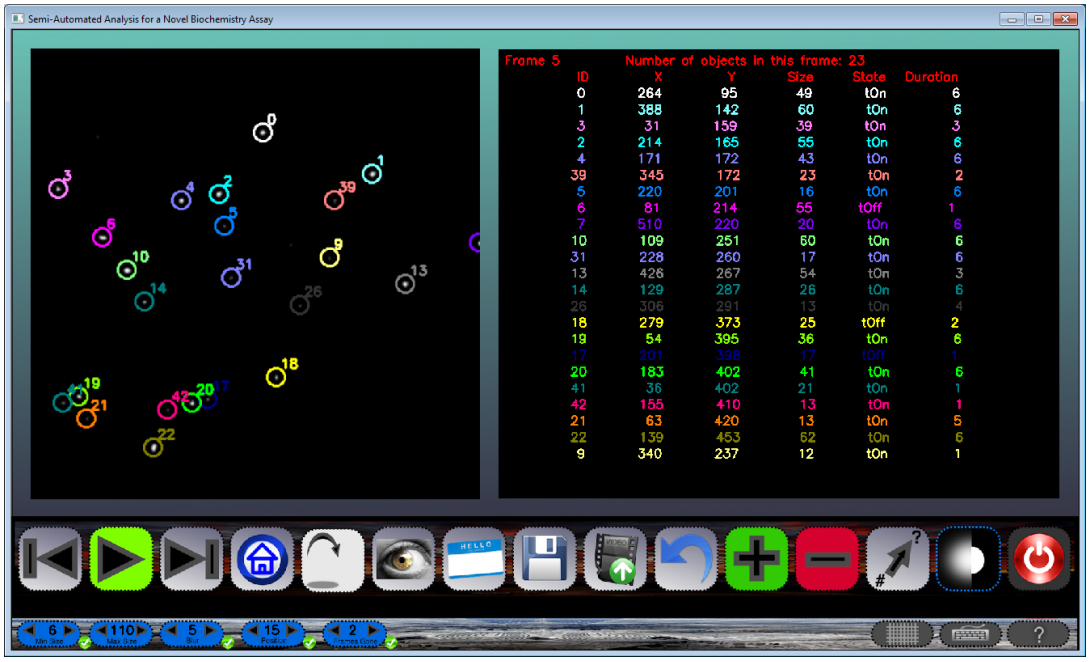

Figure 6.22: Updated object.

The **blur level** is set to 5 by default Figure 6.23. However, this setting may not be ideal for the current situation and the user may change this value to a higher or lower value. When the confirm button next to the blur value all of the slides in the movie are reprocessed and displayed as seen in Figure 6.24. Also, the algorithm is redone for the new set of imported images giving different results. This is due to the fact that some objects may disappear if too much blur is used and some may even split into multiple objects if not enough is used. Each blur value has its own images stored for future reference so that they do not have to be reprocessed. The blur feature can also be turn off if necessary like in Figure 6.25.

There are a number of threshold settings as seen in Table 6.3 that are available for the user. The **size thresholds** (27 & 28) allow the user to remove any objects that are unwanted due to their size. Objects that are too small may be noise that was not removed in the initial process phase. Larger objects, at least in this experiment, are considered to be actin that has not been broken down enough and yield results

Figure 6.23: Default blur value of 5.

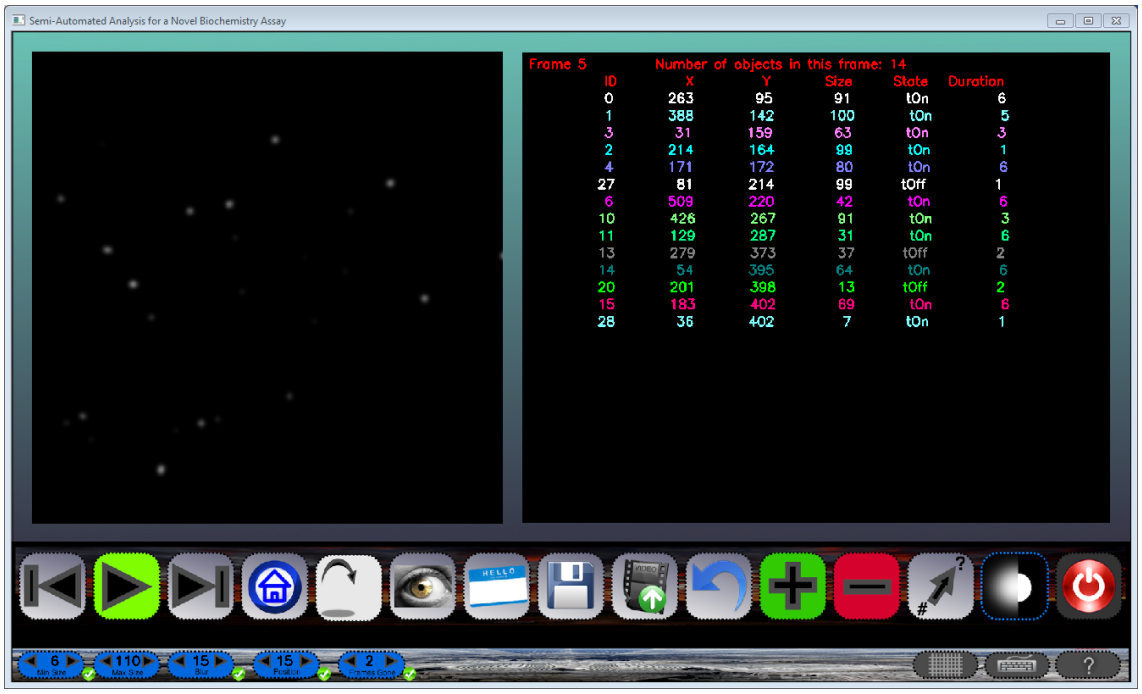

Figure 6.24: New blur value of 15.

Figure 6.25: Blur is turned off.

Table 6.3: Threshold Parameters.

27	Adjust the minimum object size.
28	Adjust the maximum object size.
29	Adjust the amount of blur.
30	Adjust how far an object moves before considered to be moving.
31	Adjust how many frames an object is gone before considered to be gone.

that are not needed at this time. When the user adjusts the size threshold, the objects that are no longer labeled are still involved in the analysis until the confirm button is pressed and the vector is repopulated. These values can be set before opening a movie or after. The **blur** (29) can be adjusted to give better results based on the current movie situation. Though it must be noted that as the blur increases so does the size of all objects and the potential to lose smaller objects increases as well. Part of the algorithm to track a moving object is to determine if in a certain **radius of motion** (30) the object is still present. In other words, how far an object moves before considered to be moving. For faster moving objects, this radius needs to increase. The objects in this experiment are not fast relatively speaking. This radius can be adjusted for slower objects or even faster objects if deemed necessary. Also, due to the many factors such as camera focus shift, irregular illumination and image processing the objects may appear to be moving but are in fact immobile. The final threshold setting is to adjust how many **frames an object is gone** (31) before considered to be gone. This can help if there are multiple consecutive camera focus shifted frames or an object is gone for too long to be accurately labeled with a previous ID.

Beyond the icons of the main UI the user can use the common **IO devices** to perform the available functions and navigation. As seen in Table 6.4, by clicking in the window above the icon area, one can navigate through the frames if so desired. In Table 6.5 all of the previously described functions are available to the keyboard enthusiasts. The shortcuts are mostly single key strokes with notable exceptions. For most users it is ingrained that to save a document the standard is the good ol' ctrl-s. The same goes for most undo's in the world to be ctrl-z (though it is set up, the

function is not available at this time).

Table 6.4: Mouse Controls.

Right click non icon part of window	Next frame/increase speed.
Left click non icon part of window	Previous frame/decrease speed.
Middle click over non icon part of window	Play/stop animation.

Table 6.5: Keyboard Controls.

b	Smoothing on/off.
<-	Previous frame/decrease animation speed.
->	Next frame/increase animation speed.
esc	Quit program or active function.
o	Open file (saved txt or new avi).
ctrl+s	Save document .
space	Start/stop animation .
i	Show/hide id numbers and object boxes.
a	Add object to vector.
s	Remove object from vector.
h	Go to first frame (number 0).
g	Go to specific frame (enter frame with keyboard).
q	QuickView of frame last on if different by +/- 1.
ctrl+z	Undo (NOT DONE YET).
r	Rename an object with an ID from previous or other frame.
k	Show/hide keyboard.
g	Show/hide grid and numbering.
? or /	Show help, any mouse click or key press will close help screen.

6.2 Secondary UI

The secondary UI behaves similarly to the main UI though with a limited set of functions. In Table 6.6 the functions are straight forward in regards to the main UI functions except for “37”. This icon shows the ID to be transferred to the selected object in the current frame of the main UI. This can be erased with the backspace and is only confirmed with the “enter” key. After the ID is keyed and enter is pressed,

the secondary UI window will close returning the user to the main UI and the selected object is now renamed to the entered value as long as it is not a repeat of an ID in this current frame in the main UI.

Table 6.6: Secondary UI Functions.

36	Close Secondary UI Window.
37	Entered value of new ID showed here.
38	Show or hide the grid.
39	Go back one frame.
40	Go forward one frame.

Figure 6.26: PreView Window Interface.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

A method of observing the behavior of muscle tissue in a single molecule configuration has been developed at the Baker lab at the University of Nevada, Reno. The process is a novel high-throughput single molecule binding assay, or SiMBA for short. This assay is performed by placing the myosin molecule of the muscle tissue on a coverslip and looking at how the fluorescently labeled actin filaments bind to these myosin as imaged with a fluorescence microscope. The conditions are varied for each experiment and the response to these changes of the actin-myosin binding kinetics are observed. In order to analyze the binding times and unbound times a researcher must observe the interactions and manually collect the data. The analysis of this experiment is very time consuming for the researcher. By automating the analysis, this method can be further improved to gain a better knowledge of muscle dynamics.

The work present in this thesis describes the design and implementation of a semi-automated solution for identifying and tracking a variable number of objects that exhibit a multitude of behavior, and extracting the specific behaviors of motion and stagnation as well as the duration of these behaviors. The end result is to extract the time spent in motion and when bound as well as the duration of these behaviors. In its infancy, the software has many hurdles to overcome, yet the end results will be highly beneficial to this research.

This software converts and processes the raw data extracted from the SiMBA

experiment. The user has the ability to manipulate the settings if necessary to achieve the best possible results. When the best results are found, the researcher may now take the output of this software and have analysis software determine the necessary histograms for behavior analysis. This last step is necessary for all the competing software as it is a specialized analysis that is beyond the scope of this part of the analysis.

7.2 Future Work

OpenCV GPU The development of an open source version of OpenCV running on the graphics processor has been in the works recently. It is however not viable to this software at this time. In future updates taking advantage of parallelization on a GPU can only benefit the user and getting the results in an even more time sensitive manner. The main portion of the software that would utilize this parallelization would be the processing and analysis of the images as these are the time limiting factors of the current software. The projected time reduction of a full movie would theoretically run at least 5-10 times faster [13]. Even though this is moving towards a better solution than previous methods, by efficiently using parallelization, the time of analysis could approach a speed that would allow for streaming video with a frame rate of 20-30 frames per second. Compared to manual analysis (12 hours) this is a huge advantage, but a competent GPGPU solution will have to remain a dream for future updates for the time being.

Kymograph Image A feature that I was hoping to get included is the ability to see a 3D model of all objects over the entire duration of a movie. A kymograph is described as a graphical representation of spacial position over time, where the spatial axis represents time [21]. It basically is a stacking of the frames to see the objects positions over time and also to see how much the objects change in size. And example image to get a idea of this feature is seen in Figure 7.1. The main difference would be that the SANoBA version would have multiple colored strains as opposed to just

green as in the figure. This would be an interactive 3D model in that it could be

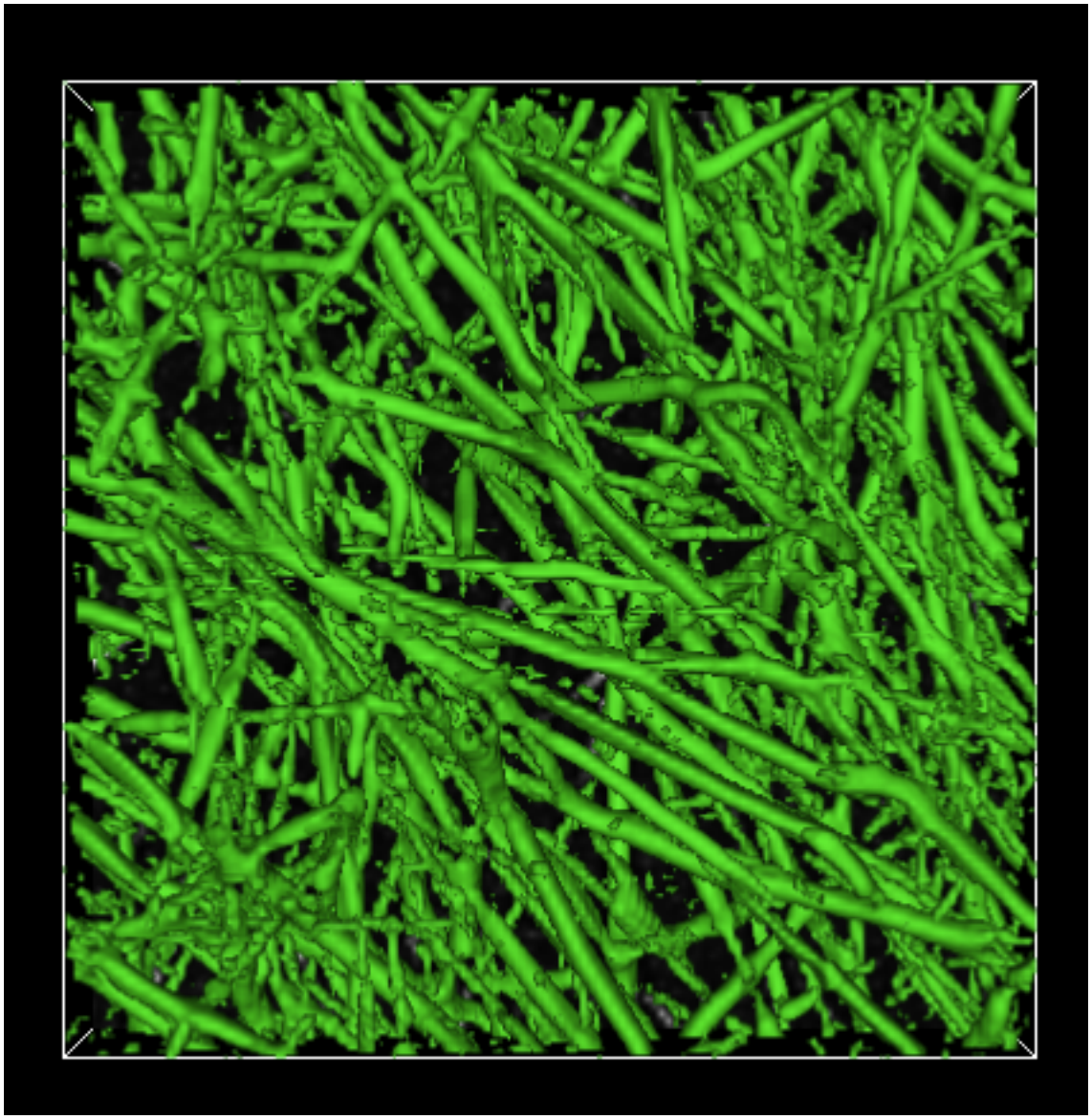

Figure 7.1: Basic representation of a kymograph.

rotated and scaled. Also, it would possibly have the ability to view the history of individual objects by removing all others aside from a particular object.

Add Function Expansion A future update would be to have the option to add the same object in future frames or just current. Currently, the user must add the object one frame at a time and can be tedious though it is not often to occur.

Undo Function As stated previously, the undo function is not currently functional. The most needed functions to have an undo is the removing and renaming of an object. These can occur to the wrong object relatively easy at this point and major headaches can be avoided with this implementation. The proposed method is to have a running queue of these actions and keep the frame of occurrence, the objects location as well as its new and old ID. By using the familiar ctrl-z, this queue would pop the top event from the queue and rename the object to the previous ID.

Status Bar When opening any file in any program, if it is less than instantaneous, a status bar is absolutely necessary. Multiple attempts have been made for this feature, yet it has been determined that a child process needs to be created to update a counter of sorts. In previous attempts, as soon as the image processing began the updates made to the status bar functions were "unheard" until completion of the processing function. With the use of a forking procedure, the status can be updated with negligible overhead to the image processing procedure.

Recent Projects Listing Also useful to a user of any software is a list of recent projects that are selectable. As any one knows who have used any type of project orientated software over a period of time, it is absolutely necessary to reduce the amount of searching needed to open a file, especially one that has been opened previously.

Object History This is a listing that would give a numerical version of the kymograph. This could be accessed in the information window and would give the history of each object in the current frame image. By history, it could show the frames that each object is present. This could be expanded by clicking on a particular object and get the size and location of that object in each frame it is present. This would expand the removing an object feature by allowing the removal of an object that is only present in one frame for instance.

Multiple Operating Systems The more operating systems a program can run on the better. For future updates my goal for this software is to ensure a proper run time environment for Windows XP and 7 as well as Ubuntu minimally.

In Program Conversions With the hope that SimplePCI releases a plug-in to allow third party programs to convert their CXD format to other formats will aid in reducing the time a user needs for a complete analysis.

Zoom Function Though it has been deemed unnecessary at this time, a useful feature would be the ability to zoom in on the object images to gain a better perspective of the occurrences in the movies and analysis. A down side to this endeavor is that OpenCV does not currently support the mouse wheel, though keyboard functionality would remedy this situation. Additions to the information window would include the zoom level and position of the zoom window in relation to the image window as a whole.

Adaptability The output and analysis can be adapted to other problems in the near future as simple plug-ins. For example, if someone wanted to track an asteroid moving across the sky, SANoBA can track it but the final output would have different data. Instead of outputting just the duration of states, there could be an output of velocities, past and future trajectories, though the development of a plug-in for future trajectories is far beyond my expertise at this time.

Motion Analysis The constant and instantaneous velocities can be derived as long as the frame size is known and how many frames per second there are in a particular movie. With this, the angular acceleration and uniform acceleration could even be extrapolated as well. For this experiment, the size of the frame is constant as are the frames per second. If ever needed, this analysis could be added without much difficulty and could further increase the value of this software.

Bibliography

[1] J. Arlow and I. Neustadt. *Unified Process: Practical Object-Oriented Analysis Design*. Addison-Wesley, 2nd edition, 2005.

[2] S. Bouakaz. Image Processing and Analysis Reference. `http://www710.univ-lyon1.fr/~bouakaz/OpenCV-0.9.5/docs/ref/` (Accessed November 17, 2011).

[3] G. Bradski and A. Kaehler. *Learning OpenCV: Computer Vision with the OpenCV Library*. O'Reilly Media, 1st edition, October 2008.

[4] H. Corp. SimplePCI - Software for Image Acquisition and Analysis. `http://hcimage.com/simplepci.htm` (Accessed November 2, 2011).

[5] O. Corp. Origin - Data Analysis and Graphing Software. `http://www.originlab.com/` (Accessed November 2, 2011).

[6] D. Jackson. GPGPU Proposal for Automated Analysis Software for A Novel Biochemistry Assay. http://www.cse.unr.edu/ jvesco/simbaProposal.pdf.

[7] G. Johnson. Animated model for myosin-based motility. `http://valelab.ucsf.edu/images/movies/` (Accessed November 2, 2011).

[8] A. Krizhanovsky. Pavlovsk Railing of bridge Yellow palace Winter bw threshold.jpg. `http://en.wikipedia.org/wiki/File:Pavlovsk_Railing_of_bridge_Yellow_palace_Winter_bw_threshold.jpg` (Accessed December 9, 2011).

[9] A. Krizhanovsky. Pavlovsk Railing of bridge Yellow palace Winter.jpg. `http://en.wikipedia.org/wiki/File:Pavlovsk_Railing_of_bridge_Yellow_palace_Winter.jpg` (Accessed December 9, 2011).

[10] W. Landsman. Eroding and Dilating Image Objects. `idlastro.gsfc.nasa.gov/idl_html_help/Eroding_and_Dilating_Image_Objects.html` (Accessed October 24, 2011).

[11] W. Landsman. Smoothing an Image. `http://idlastro.gsfc.nasa.gov/idl_html_help/Smoothing_an_Image.html` (Accessed October 24, 2011).

[12] R. Manduchi. Bilateral Filtering for Gray and Color Images. `http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/MANDUCHI1/Bilateral_Filtering.html` (Accessed December 9, 2011).

[13] Nvidia Corp. Nvidia adds gpu acceleration for opencv application development. http://pressroom.nvidia.com/ (Accessed November 27, 2011).

[14] N. Panchuk-Voloshina, R. Haugland, J. Bishop-Stewart, M. Bhalgat, P. Millard, F. Mao, W. Leung, and R. Haugland. Alexa dyes, a series of new fluorescent dyes that yield exceptionally bright, photostable conjugates. *Journal of Histochemistry Cytochemistry*, 47(9):1179–1188, 1999.

[15] J. Poskanzer. pgm. http://netpbm.sourceforge.net/doc/pgm.html (Accessed October 24, 2011).

[16] R. S. Pressman. *Software Engineering: A Practitioner's Approach*. McGraw-Hill, 7th edition, 2010.

[17] J. Russ. Iterative Morphological Operations. http://molecular.magnet.fsu.edu/primer/java/digitalimaging/russ/erosiondilation/index.html (Accessed December 9, 2011).

[18] M. Smith, E. Karatekin, A. Gohlke, H. Mizuno, N. Watanabe, and D. Vavylonis. Speckle trackerj. http://athena.physics.lehigh.edu/speckletrackerj (Accessed November 3, 2011).

[19] I. Sommerville. *Software Engineering*. Addison-Wesley, 9th edition, 2010.

[20] T. F. E. Wikipedia. Breadth-first search. http://en.wikipedia.org/wiki/Breadth-first_search (Accessed October 24, 2011).

[21] Wikipedia, The Free Encyclopedia. Kymograph. http://en.wikipedia.org/wiki/Kymograph (Accessed November 27, 2011).

[22] I. Willowgarage. Image Filtering. http://opencv.willowgarage.com/documentation/image_filtering.html#dilate (Accessed December 9, 2011).