ZEBRAFISH VIDEO ANALYSIS SYSTEM FOR HIGH-THROUGHPUT DRUG ASSAY

by

Douglas W. Todd

Copyright © Douglas W. Todd 2016

A Thesis Submitted to the Faculty of the

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

In Partial Fulfillment of the Requirements
For the Degree of

MASTER OF SCIENCE

In the Graduate College

THE UNIVERSITY OF ARIZONA

2016
STATEMENT BY AUTHOR

The thesis titled *Zebrafish Video Analysis System for High-Throughput Drug Assay* prepared by Douglas W. Todd has been submitted in partial fulfillment of requirements for a masters degree at the University of Arizona and is deposited in the University Library to be made available to borrowers under rules of the Library.

Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgment of the source is made. Requests for permission for extended quotation from or reproduction of this manuscript in whole or in part may be granted by the copyright holder.

SIGNED: Douglas W. Todd

APPROVAL BY THESIS DIRECTOR

This thesis has been approved on the date shown below:

______________________________
Jeffrey J. Rodríguez
Associate Professor of
Electrical and Computer Engineering

May 26, 2016
Date
I would like to thank Dr. Jacob for the opportunity to work on a project that has the potential to provide real benefit to people, and Dr. Rodríguez for his support and guidance on this project. Thanks to my lab mates Rohit Philip and Sundaresh Ram for their contributions of ideas. Thank you to Mr. Michael Casey for his help and support. Thanks to the Writing Skills Improvement Program and Heather Severson. Thanks to the committee members Dr. Powers and Dr. Tharp. I would like to thank my wife Maria who has always supported my education with her love, encouragement, patience, and support. Finally, I would like to thank Dr. Ditzler for his advice and support.
To my wife, Maria,
whose love and encouragement
made this possible
TABLE OF CONTENTS

LIST OF FIGURES .................................................. 6
LIST OF TABLES .................................................... 7
ABSTRACT ............................................................. 8
1 INTRODUCTION ................................................. 9
2 HARDWARE DEVELOPMENT ................................. 12
3 SYSTEM TEST AND CONTROL SOFTWARE ............. 34
4 VIDEO PROCESSING SOFTWARE ......................... 54
5 ANALYSIS AND PERFORMANCE ........................... 71
6 CONCLUSIONS AND FUTURE WORK .................... 84
REFERENCES ...................................................... 89
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Cabinet enclosing the behavioral assay apparatus.</td>
<td>17</td>
</tr>
<tr>
<td>2.2</td>
<td>Tank assembly.</td>
<td>18</td>
</tr>
<tr>
<td>2.3</td>
<td>Swimming lanes.</td>
<td>20</td>
</tr>
<tr>
<td>2.4</td>
<td>Swimming chambers.</td>
<td>21</td>
</tr>
<tr>
<td>2.5</td>
<td>Valves studied to get proper flow rates.</td>
<td>25</td>
</tr>
<tr>
<td>2.6</td>
<td>Alternative lens for the Raspberry Pi imaging system.</td>
<td>28</td>
</tr>
<tr>
<td>2.7</td>
<td>Raspberry Pi NoIR camera with adjusted lens.</td>
<td>29</td>
</tr>
<tr>
<td>2.8</td>
<td>Error in apparent fish location due to fish swimming height.</td>
<td>30</td>
</tr>
<tr>
<td>2.9</td>
<td>Distortion artifacts due to change of fish height in water.</td>
<td>31</td>
</tr>
<tr>
<td>2.10</td>
<td>Square object (4 mm x 4 mm) projected from bottom surface to top surface of swimming chamber.</td>
<td>32</td>
</tr>
<tr>
<td>2.11</td>
<td>Fish direction ambiguity when swimming directly towards camera.</td>
<td>33</td>
</tr>
<tr>
<td>3.1</td>
<td>High-level software and hardware system diagram.</td>
<td>35</td>
</tr>
<tr>
<td>3.2</td>
<td>Timing control state machine.</td>
<td>40</td>
</tr>
<tr>
<td>3.3</td>
<td>Test data acquisition time-line.</td>
<td>41</td>
</tr>
<tr>
<td>3.4</td>
<td>Example configuration tables.</td>
<td>45</td>
</tr>
<tr>
<td>3.5</td>
<td>Database containing timing information for tests.</td>
<td>47</td>
</tr>
<tr>
<td>4.1</td>
<td>Head-tail ambiguity</td>
<td>60</td>
</tr>
<tr>
<td>4.2</td>
<td>Rheotaxis (RI) illustration.</td>
<td>62</td>
</tr>
<tr>
<td>4.3</td>
<td>Detected fish and detection mask.</td>
<td>63</td>
</tr>
<tr>
<td>4.4</td>
<td>Track formation and motion.</td>
<td>68</td>
</tr>
<tr>
<td>5.1</td>
<td>Histogram of human to human variances.</td>
<td>78</td>
</tr>
<tr>
<td>5.2</td>
<td>Histogram of angular deviations of various detectors with respect to averaged human detections.</td>
<td>79</td>
</tr>
<tr>
<td>5.3</td>
<td>Precision and recall plot of different detectors.</td>
<td>80</td>
</tr>
<tr>
<td>5.4</td>
<td>Rheotaxis by epoch. Human versus machine.</td>
<td>82</td>
</tr>
</tbody>
</table>
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Video bandwidth verses Trial time start offset</td>
<td>14</td>
</tr>
<tr>
<td>5.1</td>
<td>Processing times on control computer</td>
<td>72</td>
</tr>
<tr>
<td>5.2</td>
<td>Processing times on 48 core computer</td>
<td>74</td>
</tr>
<tr>
<td>5.3</td>
<td>Measured network throughput</td>
<td>76</td>
</tr>
<tr>
<td>5.4</td>
<td>Best detections MAD scores</td>
<td>81</td>
</tr>
<tr>
<td>5.5</td>
<td>Best detections F-score</td>
<td>81</td>
</tr>
</tbody>
</table>
ABSTRACT

Zebrafish swimming behavior is used in a new, automated drug assay system as a biomarker to measure drug efficiency to prevent or restore hearing loss. This system records video of zebrafish larvae under infrared lighting using Raspberry Pi cameras and measures fish swimming behavior. This automated system significantly reduces the operator time required to process experiments in parallel. Multiple tanks, each consisting of sixteen experiments are operated in parallel. Once a set of experiments starts, all data transfer and processing operations are automatic. A web interface allows the operator to configure, monitor and control the experiments and review reports. Ethernet connects the various hardware components, allowing loose coupling of the distributed software used to schedule and run the experiments. The operator can configure the data processing to be done on the local computer or offloaded to a high-performance computer cluster to achieve even higher throughput. Computationally efficient image processing algorithms provided automated zebrafish detection and motion analysis. Quantitative assessment of error in the position and orientation of the detected fish uses manual data analysis by human observers as the reference. The system error in orientation and position is comparable to human inter-operator error.
CHAPTER 1

INTRODUCTION

Hearing loss is one of the most common disabilities. It affects approximately one in 1000 children [1]. About 42 to 47 percent of the population have a hearing impairment by the age of 65 [2]. Ototoxic drugs (chemotherapy treatments) lead to the loss of hearing in approximately 55 percent of patients [3]. These drugs, aging, and noise cause similar damage to the inner ear cochlear hair cells [4, 5]. Also, oxidative stress and free radicals cause damage to these cells [3, 6]. Currently, there are no FDA-approved drugs designed for the treatment of hearing loss. This thesis has been motivated by creating a process and instrumentation which increases the throughput in evaluating the efficacy of drug combinations which treat hearing loss.

Various researchers have used zebrafish as a biomarker to study disease. The hair cells along the zebrafish lateral line have a similar biology to the hair cells in the human inner ear [7]. In 2007, Gosseling [8] used zebrafish as a model to study cancer susceptibility. In 2010, Ou published a method of drug screening using observations of hair cell damage along the lateral line of zebrafish [9]. In 2012, Suli [10] demonstrated that drug-induced fish swimming rheotaxis behavior could be related to loss of hair cells along the zebrafish’s lateral line; their experiments were conducted in infrared (IR) lighting conditions. (Infrared lighting prohibits the fish from orientating using lighting queues.) Drug screening methods were devised to find drugs that protect against hair cell damage induced by cisplatin (a chemotherapy drug) [11]. In 2015 Niihori, demonstrated a proof-of-concept system for drug toxicity analysis using zebrafish rheotaxis behavior as a biomarker [12]. These studies showed that zebrafish are an effective proxy for studying cell damage related to human hearing loss.

The prior systems lacked automation and computer resources necessary to per-
form high throughput analysis of zebrafish swimming behavior to determine drug efficacy on a large number of drugs. The data processing in the Suli system was semi-automated. The camera took two sets of ten images, which were then processed using ImageJ [13]. Niihori’s system required a human operator to locate the fish manually within each frame and then manually estimate the orientation of each fish. Additionally, low-resolution video processing (720p) led to ambiguities in fish position and orientation, and different operators would have different results.

The system described in this thesis overcomes the limitations of the prior work and is a fully automated, high throughput, zebrafish behavioral assay system that allows the toxicity of a multitude of drugs to be analyzed in parallel. The key elements of our system are as follows. 1) A highly automated system eases operator workload and fully manages tests from data collection through the reporting of the fish swimming behavior analysis. 2) Multiple experiments are conducted concurrently. The system is easily expandable to facilitate additional experiments. 3) Flexibility in the data analysis architecture allows different video processing techniques. 4) Data processing steps are loosely coupled, easing system migration and upgrades. 5) A novel video background normalization algorithm normalizes video frame intensity over time and space, which allows simple thresholding that significantly speeds up fish detection tasks, allowing all video frames to be processed using inexpensive hardware.

Our system comprises a set of cabinets containing tanks, where each tank contains multiple swimming lanes. Each swimming lane has a Raspberry Pi camera, which records fish behavior; these recordings are automatically sent to data processing computers where the videos are analyzed to determine the number of fish swimming upstream within thirty degrees of the oncoming flow of water. This automation significantly reduces operator workload. Tanks and other test hardware are connected through a high-speed network and are controlled by processes running on a control computer. The network allows us to scale the system to the desired testing throughput. Various data analysis options are available offering
a time and complexity trade-off for the analysis of the video data. Additionally, multiple parallel data processing resources can be connected to the high-speed network, minimizing the amount of video data transferred to external computers. The high-speed network and scalability of data processing resources allow us to increase significantly the number of parallel experiments conducted. Niihori’s data analysis was performed by hand; this analysis took over four hours to accomplish for one video. Our computer-based analysis system can process more data per video with higher resolution in less time, the automated system is at least 60 times faster. Our background normalization technique enables us to process every video frame efficiently. This processing enables us not only to determine fish orientation as previous researchers have done, but also fish motion, which allows us to determine new behavioral metrics based on the motion.

The system described in this thesis is highly scalable and overcomes the limitations of resolution, angular accuracy and inter-operator error found in the previous systems through fully automated processing and analysis; its network-centric architecture is easily expanded to allow multiple simultaneous experiments and alternate analysis techniques using different computational hardware configurations. The system architecture allows expansion to accommodate the testing of hundreds to thousands of drugs per year.

Performance accuracy of critical parameters has been measured and found to be as accurate as that obtained by human operators. Multiple observers measured zebrafish orientation. From these measurements, inter-operator errors were estimated, and the circular mean of the measurements was taken as ground truth. The performance of the automated system was compared against the ground truth, and the system to ground truth error was determined to be within the human inter-operator error.

Subsequent chapters of this thesis describe the system hardware, software control, automated detection of zebrafish, motion estimation, network throughput, and measurement results. The final chapter suggests future work.
CHAPTER 2

HARDWARE DEVELOPMENT

The major hardware components of the zebrafish analysis system can be grouped into two classes. One class of hardware consists of control and data management, and the other is the physical test system. Water flow rates through the system are set using valves, and the stability of these flow rates as a function of water pressure is analyzed. Minimizing optical distortion is a major focus of the physical hardware, so two optical studies were performed. The first study measured distortion caused by camera imaging, while the second study looked at measurement distortion artifacts when the three-dimensional fish were projected onto a two-dimensional image plane.

Network and Control Hardware

Two Ethernet networks are the foundation of the control system. The control computer, data processing resources, Raspberry Pis and power distribution units (PDUs), which control power to the pumps and lights, are deployed on top of these networks. The system has a public facing network whose responsibilities are to provide a path for web-based user interfaces, possible connection to outside resources for data processing, and a facility for remote maintenance. Likewise, the system’s private network provides high-speed data and control paths whose primary purpose is to send commands to the various system components (e.g., the Raspberry Pis, PDUs, and possibly data processing computers) and transfer video from multiple cameras simultaneously to a storage device. The constraints of the system are mostly determined by the throughput of these two networks. This network-based topology is highly scalable; data processing components can be added to the system to accommodate an increase in the number of swimming lanes or an increase in the data analysis complexity.
The control computer has two independent network interfaces. These interfaces attach to both networks, and manage communications on them. The control computer interacts with the operator by generating web pages where the operator enters configuration and control information, and multiple users may interact with the system concurrently. The system may be configured to use external resources for data analysis, where video data and commands to the external resources will be issued over the public-facing network. System software running within the control computer generates commands to the other components over the private high-speed gigabit Ethernet network. The control computer responds to messages from the web-based user interface, while data processing and other system components respond to messages issued by the control computer. Normally, the control computer is equipped with a disk drive, which collects and buffers the video data from the Raspberry Pis. However, it is possible for the video collection and data processing to be distributed throughout the system if Ethernet capacity on the private network is exceeded. The specifics of the system software are described in Chapter 3.

Our system is currently configured so that each trial (defined as a set of experiments associated with a single cabinet sharing a common lighting system and pump system, but which typically includes multiple swimming lanes and multiple cameras) transmits 16 concurrent video streams with one high-definition video stream from each camera. Each stream occupies a network bandwidth of 17 megabits per second, for a total of 272 megabits per second for the aggregated network traffic for each trial. The recording time for each trial is 5 minutes, giving a total of 0.638 gigabytes of storage for each video stream or 10.2 gigabytes of video data for the trial. Currently, trials take a minimum of 35 minutes (including time for fish to become acclimated to the environment), with the video data generated during the last 5 minutes.

The peak network capacity at any point along the network is limited to 1 gigabit per second. It is conceivable that this bandwidth limitation could be limiting if
Table 2.1: Operator test start time determines the amount of bandwidth used to move data through the network. Operators stagger trial start times. We recommend that the start times are staggered by at least three minutes.

<table>
<thead>
<tr>
<th>Offset (sec)</th>
<th>Peak BW (Mbps)</th>
<th>Avg BW</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>102</td>
<td>816</td>
<td>816</td>
<td>Max Capacity, 1 min, 42 sec</td>
</tr>
<tr>
<td>150</td>
<td>544</td>
<td>544</td>
<td>2 Videos in flight</td>
</tr>
<tr>
<td>180</td>
<td>544</td>
<td>450</td>
<td>Recommended staggering (3 Min)</td>
</tr>
</tbody>
</table>

A large number of trials (greater than 24) are conducted concurrently. To handle such a scenario, the system supports two alternative topologies to support even higher video throughput, and three topologies to augment the data processing capability to support more complex video analysis. The number of simultaneous tests performed is a function of network bandwidth and percentage of test time that video is being recorded. This bandwidth and recorded video topic will be discussed further below.

The simplest model of network configuration directs all video traffic to the control computer. Gigabit Ethernet bandwidth allows a maximum of video transmission from three simultaneous trials to be conducted concurrently, however the instructions that we give to the operators limit the number of concurrent video transfers to two. We do this to give the system some margin, the system will still function properly if operators make mistakes. Operator instructions are simple, wait three minutes before starting another trial. A wait time of one minute and 42 seconds would allow three concurrent tests to be in transit at the same time. Operators are instructed to stagger the tests by at least three minutes. Each test is five minutes long, so this staggering ensures that the first test has completed its video transfer before the third test starts, limiting the bandwidth through the system to 544 megabits per second. This strategy supports 14 concurrent trials. Table 2.1 shows the effects of offsetting the operator start times, peak bandwidth and required bandwidth.
Given that the operators need to re-provision the tanks with new fish for the next test cycle, and that the overall test cycle approaches an hour, with 5 minutes of video recorded, a reduced Ethernet bandwidth of 544 megabits per second allows 24 concurrent trials to be supported per hour (12 intervals of 5 minutes per interval times 2 tanks recording per interval). This model has two variations for data processing. In the first variant, the control computer analyzes the data. In the second variant, the control computer sends data over the public-facing network for analysis by a computer cluster. The maximum sustained bandwidth required for this public-facing connection is 544 megabits per second (See “Raspberry Pi Cameras” below). The control computer sends video to the cluster which performs analysis and sends the results back to the control computer. The second variant for the network topology localizes the video traffic to the private network. In this topology, video data is directed to computers distributed within the test system. Each set of sixteen Raspberry Pi computers is attached to an Ethernet switch on the private network. In this case, we add a local data processing computer to the configuration, and the system software instructs the Raspberry Pis to deliver their video content to the closest computer within the cluster rather than to the control computer. Video traffic is localized to the switches. The number of possible concurrent trials is effectively unlimited.

The control computer sends messages over the private network to various system components (e.g., Raspberry Pi cameras, power distribution units (PDU), and possibly data processing computers). The system allows for different numbers of cameras, lights and pumps to be grouped together and managed as one device in a trial. These devices are physically co-located with and connected to an Ethernet switch within the cabinet. During a trial, programs running on the control computer manage each of these physical groupings as a set. The private network is physically formed by daisy-chaining the Ethernet switches from multiple cabinets together.

The flexibility created by these two networks allows a mixing and matching of
hardware resources, which can be fine-tuned to the system requirements as data analysis needs change; for example, it is possible to run data analysis software on a cluster of computers located on the public network provided that the public network has sufficient bandwidth, in this case the public facing network should be gigabit Ethernet. A database is used which defines the network topology; this database contains routing information identifying how video traffic is routed in the system, and will be described in more detail in the configuration section of Chapter 3.

Cabinets

System hardware is distributed among cabinets. Multiple cabinets are daisy-chained together using Ethernet cabling, which creates the private Ethernet network managed by the control computer. Cabinets house the equipment used to video record the fish, manage the water flow, and provide a light-tight housing which eliminates the fishes’ ability to orient themselves using visible lighting cues. Cabinets are divided into four major sections: The first (top) section houses four 940 nm infrared floodlights used to illuminate the fish. The second section has an opening, which allows the operator to change the fish used in the experiments. The third section contains the tank and pump. The lowest section contains the PDU, network switch, and Raspberry Pi power supplies. Ethernet and power cabling runs through access holes between the different sections. The overall measurements for the cabinet are 22 inches deep, 48 inches wide, and 52 inches high and is constructed from half-inch high-density polyethylene (HDPE). Major concerns with the cabinets include the potential for trapping heat within them, and providing correct lighting for video recording the fish. The cabinet is shown in Figure 2.1.

There’s a potential issue with heating within the cabinet. The target temperature for testing the fish is 28.5°±0.5° C; the normal room temperature is 26.3° C. HDPE is a thermal insulator, and the power consumed within the cabinet approaches 160 W during a trial. The four IR lights at the top of the cabinet produce 100 W when operating; the Raspberry Pis produce 26 watts when idle, and
46 watts when active; the pump produces 20 watts when active; and the network switch and PDU produce approximately 30 watts each. We mitigated the heating issues by venting the IR lights on the top of the cabinet to room air using slits cut in the top section of the cabinet and venting heat from the PDU and network switch through ventilation holes cut in the bottom of the cabinet. The normal power dissipated in the area of the fish is 26 watts when the system is idle, and 46 watts when video recording which does not lead to a significant temperature rise. A heater is used at the beginning of the day to heat the embryo solution [14] to 28.5 degrees, and embryo solution temperature remains in the acceptable range throughout testing.

It is difficult to distinguish between shadows and fish in a low contrast environment. We strive to have the lighting as diffuse as possible to eliminate shadowed areas in the camera’s field of view. Three-millimeter thick semi-transparent
optical diffusing acrylic rests just below the infrared floodlights, and a sheet of three-millimeter thick IR filtering material rests just below the optical diffusing material, which further diffuses the infrared light and prevents stray visible light entering through the ventilation slots in the top section from reaching the area where the experiments are conducted. The joints between the access door to the swimming lanes and chambers is inset into the cabinet in a way that prevents visible light from entering the cabinet from around the edges of the access door. During operation, a three-millimeter sheet of translucent acrylic covers the swimming chambers, further diffusing the light and providing a light background against which the fish are video recorded. The area between the upper and lower sheets of acrylic is approximately 12 inches high, and is covered with a reflective matte finish to further diffuse the light inside the cabinet. This arrangement minimizes shadows within the cabinet and provides a background with somewhat uniform illumination.

**Tank and Swimming Lanes**
Each cabinet contains a tank (See Figure 2.2) which measures 40 inches wide, 18 inches deep, and 15 inches tall. Sixteen swimming lanes are mounted on top of the tank; they are 14 inches long, 2 inches wide, and 2 inches high. The purpose of the swimming lanes is to direct water flow to the fish while minimizing optical disturbance to the video recording. The bottom of the swimming lanes is constructed from 0.75-mm transparent acrylic sheet; the thickness was chosen to minimize optical distortions caused by the change in index of refraction between air, acrylic, and water (index of refraction of air is 1.0, acrylic is 1.6 and water is 1.3). Three-millimeter acrylic is used for the sides of the swimming chambers. The thickness of this material allows the very thin base to be bonded to the edges of the swimming chambers, giving the swimming lanes their structural integrity. Two one-inch-high dams are placed one inch from the beginning and one inch from the end of the swimming lanes. These dams trap water and hold a one-inch water depth in the swimming lanes while the pump is not running. The front edge of the swimming lanes is a bulkhead made of 6-mm acrylic. Holes for plumbing fixtures are drilled and tapped into this bulkhead for each swimming lane. Quarter-inch-diameter hose attaches to these fixtures, where water is routed from a manifold through valves to the swimming lanes. A second bulkhead exists at the rear of the swimming lanes. The bulkhead is shaped to direct water flow to a water capturing device. Flow rates through the swimming chamber are determined by measuring the volume of water exiting the swimming lane over a timed period. Figure 2.3 illustrates these components. The tank and components were designed using Computer Aided Design Software (CAD) and laser cut. The acrylic components were bonded using Acrylic solvent such as “Plexiglass Acrylic Glue”. Two varieties were used, the first is a viscous glue used to fill seams to make the joints water tight, while the second has a very low viscosity; it is absorbed into the seams via capillary action. Both types of glue contain plastic solvents which chemically weld the acrylic parts together [15].

**Swimming Chambers**
Swimming lanes accept removable cassette boxes called swimming chambers (Figure 2.4). The swimming chambers are designed to ensure larvae zebrafish cannot escape, to minimize optical distortion, to allow the highest amount of light through, to present the fewest number of seams or other areas where the fish can hide, and to allow free flow of water. Swimming chambers fit within the swimming lanes; they have an outside width of 1.98 inches, and a length of 5 inches. The bottom of the swimming chamber is made from 1.5-mm acrylic, while the sides are 3-mm. This material is stiff enough so that the chamber maintains its shape when handled, and allows a gluing surface for a fine mesh attached to the chamber. The swimming chambers have a rectangular hole in the side, extending from the bottom of the chamber to a height of 1.25 inches, where the hole is covered with a fine mesh (400 µm) made from nylon used for wedding veils. Nylon mesh is secured to the swimming chamber using hot-melt adhesive [16] where it is applied along the bottom edge of the top lip, along the outsides of the swimming chamber and the bottom edge. This arrangement minimizes the seams seen within the chamber.
Figure 2.4: Swimming chamber. Water flows through mesh at ends of the swimming chamber. The swimming chambers are inserted into the swimming lanes. Care is taken in the design of the swimming chamber to minimize optical distortion. The swimming chambers provide a fish-tight seal.

from the video camera, located seven inches below the bottom of the swimming lanes.

**Reservoir**

The reservoir is part of the tank and is located at the rear. Water flows from the rear bulkheads of the swimming lanes into the reservoir. We measure the water flow rate by using a specialized water capture tool that we created. The capture tool is held in the reservoir, and water is collected from individual swimming lanes over a period of time. Water flow rate in the swimming lane is determined by the amount of water collected in the water capture tool integrated over time. The capacity of the reservoir is five gallons, though it is normally filled to 3.5 gallons. Flow rate through the system is dependent on the water height in the reservoir, which must contain a constant amount of water. An error of one inch of water height produces
just over a one percent error in flow rate (See Section Flow Rates). Internally, the reservoir has braces that connect the front and back sides to relieve stress on the edges of the tank due to the water pressure integrated over the reservoir surface area. Near the bottom of the reservoir is an outlet for tubing which connects to the pump.

**Pump**

The pump is a Danner 500 GPH pond pump; it is used to cycle water from the reservoir to a manifold. The pump is capable of pumping 500 gallons per hour when the inlet and outlet are at the same height, and can lift water to a maximum of ten feet. The sixteen swimming lanes use one-third of the pump’s capacity. The overcapacity of the pump provides regulation of water flow. This topic is discussed below under Flow Rates.

**Manifold**

The manifold runs the length of the tank and it distributes water from the pump through half-inch PVC pipe to the swimming lanes. Holes are drilled and tapped into the manifold, accommodating plumbing fixtures that accept quarter-inch vinyl tubing. The tubing, connected to PVC valves, is routed into the plumbing fixtures on the front of the bulkhead on the swimming lanes. The water that flows within the system is called embryo solution, and contains of distilled water and various corrosive salts [14]. We replaced the brass valves with PVC because the PCV valves are immune to this type of corrosion.

**Camera Bar**

Each Raspberry Pi camera is mounted on a camera holder which is then attached to the camera bar that extends the length of the tank. Quarter-inch holes are located along the bar’s length, accommodating bolts used to fasten camera holders to the bar. The Raspberry Pi cameras are mounted on a circuit board with four two-millimeter diameter holes. The camera holders include a quarter-inch nut that allows the bolts in the camera bar to attach to the camera holders. The field
of view of the cameras is adjusted by tilting the cameras in the camera holders, and rotating the cameras on the camera bar. The optimum position of the cameras is seven inches below the swimming lanes. This position maximizes the resolution of the fish swimming in the swimming lanes.

Factory-new cameras have a focus that starts at one meter and extends to infinity. The cameras are modified; their focus has been changed by breaking the glue that holds the lens and unscrewing it by a quarter of a turn. This adaptation allows the camera to focus on the bottom of the swimming chamber. A depth of field study was conducted to determine various camera options. (See “Raspberry Pi Cameras” below).

**Front Cover**

The Raspberry Pis are connected to the cameras mounted on the camera bar through an eighteen-inch ribbon cable. The Raspberry Pis are mounted in cases, which are then mounted on the inside of the front cover of the tank using screws tapped into the acrylic. Ventilation and cable holes are located on the front cover. Power and Ethernet cabling from the 24-port network switch passes through these holes.

**Power Distribution Unit**

Each cabinet contains a power distribution unit (PDU). The PDU connects to the ethernet switch and contains computer-controlled power outlets, it provides the interface between hardware within the cabinet and software running on the control computer. Software running on the control computer sends messages to the PDU over the network telling it to power on and off the lights and the pump. The PDU also controls power to the Raspberry Pis allowing the Pis to be remotely reset if necessary. Multiple vendors supply PDUs, each with their own set of commands used to control the status of the power switches. The selection of which command set to use for this particular PDU is maintained in system tables. These tables are described in Chapter 3, Configuration.
Flow Rates

Over the course of system testing, different flow rates were tried to determine which would give the best results for fish rheotaxis behavior [10]. The original target flow rate was 1 mm/s. Experiments with fish showed that faster flow rates were required. Unfortunately, the flow rate is a function of valve-opening cross sectional area. Having too big a cross-sectional area of the valve makes fine-tuning of the valve adjustments difficult. A series of tests with flow rates set from 1 to 10 mm/s were conducted. These flow rates span an order of magnitude, and given the precision required, different valves were tested. The first valve (left, Figure 2.5) is a lawn sprinkler valve. The fastest flow rate obtained with this valve was 2 mm/s, and the valve was very hard to control. The second valve (left center, Figure 2.5) tested was a needle valve. While the flow rates for this valve were more controllable than the sprinkler valve, its capacity was still too low. The third valve (center right, Figure 2.5) is a ball valve. This valve has the diameter and control sufficient to give the required flow rate. We used this valve and achieved flow rates of 10 mm/s, and measured the rheotaxis behavior of fish at different flow rates. We found a peaking of rheotaxis behavior at 5 mm/s. Lower flow rates did not induce the rheotaxis response, while higher flow rates caused the fish to tire before the end of the experiment. Based on these experiments, we set the flow rates to 5 mm/s +/− 10 percent. We were able use this valve and calibrate all lanes to 5 mm/s. Unfortunately, the embryo solution is loaded with salts and other corrosive materials which react with brass in the valve. The fourth valve (right, Figure 2.5) is a ball valve made from PVC plastic which is inert to the embryo solution. This valve allows all flow rates to be adjusted to the chosen value and is the one currently used.

The overcapacity of the pump is used to stabilize water flow through the system. In operation, the reservoir is filled to the level at which the flow rates were calibrated. The question is what happens to the flow rate if the reservoir is filled to a different level than when the system flow rates were calibrated and the valves
were adjusted. We analyzed the sensitivity of flow through the swimming chambers versus a change in head height in the water reservoir to determine the stability of the system for flow rate. We found that a one-inch change of head height in the reservoir changed the water flow rate by just over 1 percent.

The pump parameters from the manufacturers data sheet are given in gallons per hour and head height in feet. The first step in the calculation is to determine the system’s gallon per hour requirement. Ninety-eight gallons per hour are required to provide each swimming lane with a flow rate of 5 mm/s. Second, the required head height seen at the valve provided by the pump is determined from the data sheet provided by the pump’s manufacturer. This turns out to be 7.84 feet. The third step is to determine the valve resistance, which is the pressure seen on the valve divided by the flow rate through the valve [17], which is numerically 76 micro-feet per millimeter water flow through the swimming lane. The fourth step is to define an incremental change in head height, caused by operators not filling
the reservoir to the same level that it was calibrated at. The water level of the reservoir is marked when the system is calibrated, and operators are expected to fill the reservoir to within one inch of that mark before using the system. The fifth step is to determine the flow rate by dividing the change in pressure (the head height) by the valve resistance, which for a 1-inch change in head height gives a change of 0.053 mm/s from the nominal flow rate of 5 mm/s, resulting in a 1.06 percent change in flow rate. One way to think of this is that the pump has an internal resistance, and the valves create additional resistance. Flow rate is given by the pump’s pressure (10 feet head height) divided by the sum of the valve and the pump’s internal resistance. We can calculate the pressure (7.84 feet head height) at the valve using the manufacturer’s data sheet. The water height in the reservoir is marked when the system is calibrated. Operators are instructed to check the water height, and maintain the water level to within one inch of the marked value. A one percent error is allowed based on water reservoir head height, and a nine percent error is allowed for valve-to-valve variations.

**Raspberry Pi Cameras**

In 2012 the low-cost ($35) Raspberry Pi computer was introduced. The Raspberry Pi computer supports a low-cost ($30) 5 megapixel camera, and is capable of recording high-definition (1080P) video. The camera, computer and case together cost about $70 dollars. This camera normally focuses images from one meter to infinity, but can be modified to change its focus. Cameras sold by the Raspberry Pi Foundation come in two varieties. The first option includes a filtering material between the camera’s lens and the focal plane which excludes infrared light, and this lens has a better color balance than the second camera option. The second camera, called the NoIRcamera, has the infrared absorbing material removed, and is suitable for recording using 940-nm infrared lighting.

Normally, the Raspberry Pi camera can clearly image objects from three feet to infinity. We require maximizing the view of the fish within the swimming lanes; the camera should be as close to the swimming lanes as possible. A distance of seven
inches was found to give a full field of view of the areas where the fish could swim, while maintaining sufficient spatial resolution of the fish within the video frame. This means that the cameras needed to be refocused to maximize the field of view.

Three solutions were tried: 1) use a magnifying glass in front of the camera, 2) use a camera from a different manufacturer, and 3) modify the Raspberry Pi camera by breaking the seal that holds the lens in place and unscrewing the lens, changing its focus. The advantage of using a magnifying glass was that the cameras would not be damaged and could potentially be reused. However, the space required for the lenses and the costs excluded this option. The second trial was performed using a different lens and camera solution from another manufacturer. A camera that had an adjustable-focus lens was purchased from Ebay. We selected a “5MP OV5647 Sensor Camera Board with CS-Mount Lens for Raspberry Pi”. We ordered the camera with the NO-IR filter option. The advantage to this system was that the lens could be easily refocused. However, the pictures it took at the required range had a “pincushion” effect. Given that the lenses should only have to be focused once, the better picture quality from the original Raspberry Pi lenses won over the adjustable lens. A small study was taken to compare the depth of field and distortion of the adjustable-lens camera compared to the Raspberry Pi camera. The larger the depth of field, the less critical the focusing requirements. Comparisons between the lenses were made with a rectangular target measuring 5 inches on one side. The cameras were focused on the target to maximize their resolution. Once focused, the cameras’ range to the target was adjusted and the depth of field of the cameras was approximated. Further, a comparison of the straightness of the lines in the images were compared. The depth of field from the adjustable lens was found to be approximately 1 inch, while the depth of field from the Raspberry Pi camera was found to be approximately 3 inches. Lines on the target near the edges appear straighter with the Raspberry Pi camera and do not show a “pincushioning” effect. Compare the picture from Figure 2.6, which shows a target with straight lines taken with the adjustable lens, to the picture taken with the Raspberry Pi camera lens Figure 2.7. One speculation about the better image
Figure 2.6: Test image using an alternative lens for the Raspberry Pi. This lens has significant “pinchushion” effects, and a limited depth of field. We rejected the use of this lens.

quality from the Raspberry Pi lens is that there is some digital processing of the image done within the camera module. Parameters for this processing are tuned to the original lens; the replacement lens does not have a similar advantage. The results of the study showed that the best solution was to slightly unscrew the lens from the Raspberry Pi camera by to bring the swimming lanes into clear focus. We created a custom tool, made from brass, that fits around the lens of the camera for refocusing the Raspberry Pi cameras.

The program running on the Raspberry Pi that records video from the Raspberry Pi cameras has adjustments that are made by command-line parameters. We found that under infrared conditions, the recorded picture would sometimes take on a reddish hue. This was corrected by setting the camera’s saturation variable to the minimum value of -100, disabling color. We adjusted other parameters as well: sharpness is set at the maximum value, while contrast and brightness are set at 70 percent of their maximum values.

The Raspberry Pi cameras record video at a rate of 17 Mbits per second. This implementation uses 16 cameras per trial giving a required video bandwidth of 272
megabits per second. We limit the number of concurrent video transfers to 32 (2 trials) which gives a peak required bandwidth of 544 Mbps. The limitation of two sets concurrent trials is conservative, the hardware allows three.

\textbf{Distortion Caused by Projection to a Plane}

Fish are constrained to swim within a water depth of one inch. Each swimming chamber has a single camera, and multiple fish at various depths within the swimming chamber are projected onto a single two dimensional plane when video recorded. Because the camera is located below the swimming lanes and looks through the bottom of the swimming chambers, fish that are near the surface appear further away from the center of the swimming lane than they would appear if they were closer to the bottom. Figure 2.8 schematically shows the relationship between the images of fish at the closer bottom surface and fish at the further top surface.

The lack of fish height information (z-axis information) and the different optical densities of materials leads to three types of errors. First, fish may be at the same
Figure 2.8: Illustration showing changes in apparent X,Y position as a function of fish height in water. Errors are greatest near the far edges of the swimming chambers.

distance along the length of the swimming lane, but appear at different distances within the lane based on their depth - fish near the bottom of the tank will appear further from the camera and larger, while fish at the top will appear closer and smaller. Second, the index of refraction of water causes the fish to appear at 77 percent of their actual distance through the water, and light within the swimming chamber will bend toward the camera. Third, if a fish is swimming directly into the camera, the fish’s orientation cannot be determined.

Figure 2.9 demonstrates the apparent location error due to fish at different heights. The blue dots in the figure are spaced a half-inch apart on the bottom of the swimming chamber. The red dots are the locations on the surface of the water correspond to the blue dots on the bottom. Points between the red and blue dots will be mapped to the blue dot positions on the bottom plane of the swimming chamber. The error in location from bottom to top of the chamber increases as the camera angle increases, and reaches a maximum of 6 mm near the borders of the swimming chambers.

Fortunately, the required measurements rely only on the orientation of the fish, and because the depth is restricted, the fish are unlikely to change depth. Figure 2.10 illustrates that both sides of the fish are distorted nearly equally.
Figure 2.9: Simulated distortion artifacts. Blue dots are on the bottom of the swimming chamber. Matching red dots are at the surface of the water. The errors increase to approximately six millimeters from the bottom surface to the top, and also increase with distance from the center of the image. The simulation accounts for changes in optical density.

The distortion effects were measured in a simulator. Figure 2.10 demonstrates what happens to the image of a small square measuring 4 mm per side asymmetrically located on the bottom of the tank when projected to the top surface of the water. The blue dots in the figure are spaced on a grid four mm apart apart on the bottom of the swimming chambers; the red dots are the same points projected to the surface. The box locations are offset by six millimeters, showing an error in absolute position, but the changes in the relative angles are not measurable. Measurements for the simulation showed a ten-percent growth in the box size as the box is moved from the top surface of the swimming lane to the bottom surface. However, the change in orientation due to shear effects is in the range of parts per million and is not observable.

The third error that occurs is when the fish is swimming directly into the
Figure 2.10: The blue points represent a square (4mm x 4mm) on the bottom of the swimming chamber, and the red points are the corresponding points on the surface of the water. The red points are displaced from the blue points by six milimeters due to positional errors, but the red points still form a square. The angular error of the red points is insignificant.

camera. This is illustrated in Figure 2.11. In this case, the orientation of the fish is symmetrical; the head and tail are along the same axis and cannot be distinguished. The fish may have a directional component along the horizontal axis, but that component cannot be determined. Swimming lane geometry generally precludes this condition; the fish are 4 to 10 mm long, and the depth of water is 1 inch. The fish cannot swim in the vertical direction for long before hitting the bottom or the top of the swimming chamber.

**Summary**

The hardware systems represented several engineering challenges. Among them were thermal loading, illumination, pump stability, camera parameters and choices of lenses, and distortion artifacts when the fish were projected to a plane. Analysis
Figure 2.11: The fish orientation cannot be determined if it is swimming directly into the camera. This is a very rare occurrence due to the limited water height.

models were created, allowing these parameters to be simulated and understood.
CHAPTER 3
SYSTEM TEST AND CONTROL SOFTWARE

Figure 3.1 illustrates the major components of the system software architecture. The system software provides functionality for the user interface, control of the hardware components, and maintenance of the overall system. Each trial involves a cabinet and its associated lights, pumps, tanks and zebrafish. Trials are a collection of experiments that run within a cabinet. Two separate independent pieces of information need to be merged to support a trial; they are the test parameters, and hardware configuration information. A user selects a “test” to start where “test” identifies test timing parameters, and a cabinet on which to run the test. The control software merges this information and starts a “trial” with the user specified parameters. Example of experiments are the drug or dosage levels given to the fish. Multiple trials may be conducted using different cabinets concurrently. The only restriction is that no more than 48 (see Chapter 2) simultaneous video transfers are ongoing through any network switch at a particular instant. System software interfaces to two different networks, one external and the other internal (the hardware was discussed in Chapter 2). Operators configure tests and review results using a web-based interface. The system multitasks, allowing concurrent trial runs and operator interaction. System software delivers video to analysis processes running on a variety of computers as discussed in Chapter 2. These processes transfer results back to the control computer for display to the operator. Each Raspberry Pi runs a daemon process [18], allowing the control computer to issue commands to the Raspberry Pi to control the camera. The daemon process running on the Raspberry Pi is the “camera daemon”. The daemon process running on the control computer is the “state machine daemon”. The state machine daemon running on the control computer manages the automation of cameras by sending commands to the camera daemons running on the Raspberry Pis. System software includes a
Figure 3.1: Users running web-client software log into an Apache server which creates dynamic HTML content. The server queries test state and configuration information from the state machine daemon that is responsible for issuing commands to the Raspberry Pi cameras and other hardware. Video data from the cameras is sent to the control computer for processing. Reports are made available to the user through the web interface.

The video processing subsystem, which is independent from the control software. The state machine daemon looks up the names of scripts from configuration files, and as video data becomes available, executes the scripts. These scripts manage any necessary movement of data to the video processing software, which may reside on the control computer, the high performance cluster (HPC), or on the private network. Figure 3.1 shows the system architecture.

Updating the Raspberry Pi computers requires automation. Maintainers use scripts to configure multiple Raspberry Pis, which eases maintenance and manage-
The software and control systems use a loose message-based coupling mechanism between software components, allowing flexibility and scalability in software design. For example, hardware computational requirements to process every frame of the video may differ from those required to process one frame every six seconds; however, these differences are confined to the video processing subsystem. The data collection and user interface systems need not change because of possible future additional computation requirements. The video processing subsystem allows multiple videos to be processed concurrently. The video processing subsystem can be configured to process videos on the control computer, on computers within the private network, or on a high performance computer (HPC) cluster. What’s important from a software point of view is not which computers run which pieces of code, but the interrelationship between the code pieces. The capability of the Raspberry Pi hardware is increasing; soon it may be possible for the video processing software to run on the Raspberry Pis themselves without a need for an external data processing computer.

This chapter describes the interrelationship between the various software components and their connection to the hardware:

- First: The state machine that manages the running of trials.
- Second: The interaction between the user interfaces, configuration files, and test state machine.
- Third: The data flow from the user interface through the web server to the state machine.
- Fourth: The tasks running on the Raspberry Pis, how they are started, and their responsibilities.
- Fifth: Communications between the controlling process and the data analysis tasks.
The software is written in high-level languages familiar to internet programmers, which include Perl [19], JavaScript [20], Bash [21], and C++ [22–24]. The high-level languages include a feature known as “garbage collection,” which minimizes the chance of memory leaks. We leverage against high-level utilities; we use Apache [25] to manage dynamically generated web pages, Network File System (NFS) [26,27], automount for the routing and storage of video traffic, and Secure Shell (SSH) [27] for remote administration and, in the event of processing video on the computer cluster, remote execution.

Interprocess communications take place through commands sent as text strings, or a command and an XML (Extensible Markup Language) [28] encoded message. One of the design goals is to use common, well-known languages and utilities to minimize the learning curve required for future maintainers.

State Machines

The core of the system is a set of state machines running within the state machine daemon that runs on the control computer (Figure 3.2). Communications to and from these state machines are through ASCII string messages sent over network transmission control protocol (TCP) socket interfaces [29] to the state machine daemon. A line is added to the control computers startup scripts which starts execution of the state machine daemon process [18]; this daemon opens a communications socket and waits for incoming connections. The state machine daemon manages persistent configuration tables used for testing, manages state machines to perform and monitor tests, and coordinates message traffic between the Raspberry Pis and data processing tasks. Users interact with web pages generated by Apache CGI scripts that create messages that interact with the state machine daemon. These messages are interpreted as commands to carry out on behalf of the user.

The state machine daemon maintains a list of data structures which maintain state for each cabinet (state machines). One of the elements in the state machine data structure is the “state” element. Each of these state machines maintains
a list of Raspberry Pis and their current condition (recording or idle) associated with this cabinet as well as the current state of the trial and timeout variables. As messages arrive at the state machine daemon for a trial, the daemon looks up the instance of the state machine and makes function calls which include a reference to the appropriate state machine. The functions take different actions depending on the current state of the state machine, and the incoming message. For example, executing a “timeout” function on a state machine with the current time causes the machine to compare the current time with a previous time stored in a variable maintained in the state machine structure, and change state if the time is exceeded. Generally, one of the states in a state machine is “idle”. State machines remain in the “idle” condition until they receive a “start” message, at which time they normally send messages and transition to a non-idle condition.

A state machine is “executing” if it is not in the idle state. State machine execution is started by a “start” function which usually performs some initialization, sets some hardware state, and changes the state machine’s state from “idle” to some other state. The state machine should be thought of as an object with the functions operating on it as methods; incoming messages invoke methods on the state machine which advance the machine’s state. The transition from one state to another generally causes some form of output from the state machine, such as changing the pump or light’s status, or sending messages to the video processing system.

Messages arriving from the Raspberry Pis over Transmission Control Protocol - Internet Protocol (TCPIP) [30] are directed to the correct state machine by looking up the source address from the incoming message, and using the source address as the key into the list of states. These messages update the state machines’ view of the Raspberry Pis involved in the trial. Messages arriving from the Apache web server that involve a state machine identify the relevant state machine in the message. All input to the state machine daemon is routed through the “select” system call. Messages retrieved from the “select” system call are parsed and the
appropriate functions within the state machine daemon. The first field identifies the functionality the command wishes to execute, and subsequent fields are arguments which may be XML encoded strings. The daemon looks up the request from the incoming message and dispatches the request with arguments to a handler which may query or update configuration tables, return status information from the state machine, start trials or take other actions.

The daemon’s internal state, along with the configuration tables, are updated as necessary based on input messages from multiple sources, including the user modifying the test content and Raspberry Pis camera daemons sending status messages. Table data is stored as XML encoded files and is persistent across reboots. Examples of these tables include "cameras.xml" and "tests.xml", which identify the current camera and test configurations. Each Raspberry Pi and PDU within the system is assigned a unique identifier on the private network. When the state machine daemon receives a "Start Test" message, it starts a state machine for the particular cabinet. As the state machine advances through its phases, commands are issued to the PDUs and Raspberry Pis associated with this cabinet.

The Raspberry Pis start the camera daemon when booted. It accepts commands for the cameras over the network such as "start recording for 5 minutes", and provides status to the control computer when the camera state changes. Typically, the control computer requests a status report from the Raspberry Pi, which reports an "idle" condition if it is not recording. The state machine daemon on the control computer asks the camera daemon to start recording for five minutes. The camera daemon replies that it is in the "recording" state. When the Raspberry Pi has completed recording the camera daemon enters the "idle" state and reports that condition to the state machine daemon process running on the control computer.

The actual test conducted consists of the following stages. Figure 3.3 illustrates the timing:

- Dark Adaption The fish are allowed to rest and acclimatize to this environment for a thirty-minute period (dark adaptation time), after which five
Figure 3.2: States that each trial (set of tests conducted within one cabinet) progresses through. The state machine controls the timing of the lights and pumps within the system.

minutes of video is recorded. This video is divided into three epochs, corresponding to the following three stages.

Epoch 1 The first epoch records video with the pump off for a period of one minute. We expect to see all fish swimming (moving) and approximately 1/6 of them oriented within 30 degrees of a given reference direction (assuming random fish orientation). During this time, two rheotaxis index (RI) values are computed, as defined in Chapter 4, in order to verify the fish are swimming with a random orientation.

Epoch 2 The pump is turned on. The pump stabilization time is defined as a transition period from the time the pump is turned on until the water flow in the swimming lane reaches steady state. The stabilization time is set for one minute. During this period, two RI values are computed. These values are expected to rise during
Figure 3.3: Test timeline. Fish are allowed to acclimate for 30 minutes, then video recorded for 2 minutes. The pump is turned on and one minute is allowed for the flow to stabilize. Three minutes of video are recorded where the fish are expected to be in rheotaxis.

Epoch 3

The fish are video recorded for an additional period of two minutes. Healthy fish align with the upstream direction, while fish that are damaged show more random orientation. The RI values should peak during this period. If the fish are damaged, the RI values may decline. A total of six RI values are computed over a period of three minutes. Healthy fish will have RI values of about 85 percent. Damaged fish will have lower values.

When the user selects a test to be started, the CGI scripts send a message to the state machine daemon. This message contains both the “test” information, and an identifier which describes on which cabinet the test is to be run on. The daemon looks up the timing information from the test information provided by the user, and cabinet’s configuration information from configuration tables (see Configuration section below). The daemon locates the state machine data within its internal tables and begins execution of the states that control the trial. Multiple state machines may run concurrently, with each state machine running a different trial. The state machine daemon routes message traffic from the user interface and Raspberry Pis to the correct executing machine; additionally, the executing
states receive timing messages from the state machine daemon. When running, the state machine looks up configuration and test timing information from various tables and uses that information to determine timing information for control of the Raspberry Pi cameras, PDU and video routing. Users define test parameters which include the “Dark Adaption Time”, “Power Off Time”, “Pump Stabilization Time” and “Pump On Time”. Different tests are defined for different purposes, such as calibration and simple operational tests. The trial starts when the user specifies a test and a cabinet, at which point the state machine daemon applies test configuration data to a particular set of hardware. The process of gathering configuration data is described below under “Configuration”. The state machine advances through the following states, implementing the protocol described above.

**Idle**

The state machine is not running. It is waiting to be started. When started, it sends a message to the PDU to apply power to the IR floodlights, and sets an internal timeout variable to be equal to the configuration test configuration parameter Dark Adaption Time plus the current time. Nominally, the dark adaption time is 30 minutes.

**DarkTime**

The state machine will advance state when the current time is greater than the timeout value stored in the internal timeout variable. Timeout messages are internally generated by the state machine daemon and sent to all state machines every five seconds. Once this timeout value has expired, compute the recording time from the configuration information gathered when the state machine was started. The recording time is the sum of the Power Off Time, Pump Stabilization Time and Power On Time. Next, the state machine sends a message to all the cameras associated with this trial telling them to start recording for the recording time and where to send their video. Set the timeout variable to the Power Off Time, which is typically one minute plus the current
time, and advance the state machine to the RecordPumpOff state. The purpose of the DarkTime state is to allow the fish time to acclimatize to the dark conditions of the test, and normally lasts for 30 minutes.

**RecordPumpOff** During this period, the cameras are recording video for the Power Off Time of typically one minute. The state machine receives current time messages from the state machine daemon. When the current time exceeds the timeout value set by the DarkTime state, the state machine sends a message to the PDU associated with this trial to turn on the pump power. The state machine sets the timeout variable to the current time plus the Pump On Time plus the Pump Stabilization Time plus the current time, and advances the state to the RecordPumpOn state. The purpose of this state is to record fish for a period of (nominally) one minute before turning on the pump.

**RecordPumpOn** The state machine running within the camera daemon on each Raspberry Pi will report an Idle condition when it has completed its recording process (including any flushing of buffers to the network). Processing of video information can start after this message is sent. The state machine keeps track of which cameras are active for this trial. The state machine notifies the video processing subsystem when a camera has completed processing, as well as when all cameras have completed recording video. This information is used by the video processing subsystem to enable processing. (See Video Processing Subsystem below.) Once all cameras have reported the Idle condition, and the notification messages have been sent, the state machine sends messages to the PDU telling it to turn off the lights and pump. The state machine then transitions to the Idle state. The purpose of this state
is to wait until all the cameras have completed recording and to notify the video processing subsystem that video has arrived.

At the end of the trial, the state machine transfers messages to the video processing subsystem (described below in Video Processing Subsystem), which analyze the fish video and report the RI values.

**Configuration**

The assignment of Raspberry Pis and PDUs within tanks, and tanks and processing resources within the system, allows flexibility. Video processing resources may be co-located with test equipment, or located at some remote facility; they are decoupled from test assets. The state machines described above know how to run a test and know the operations that need to be carried out. State machines are given configuration data describing which Raspberry Pis and PDUs are connected to which cabinets, where the Raspberry Pis are to store the data, and which computer resources and processing models are used to process the video data. Options for processing models include processing on the control computer, processing on an external computer cluster, or processing internally on computers located on the local network.

Currently, we have a system that supports four cameras, and another system that supports sixteen cameras. The systems support different PDU hardware. The configuration architecture allows the hardware devices and processing to be abstracted to the state machines running on the control computer. Specified within the configuration options is the name of a script that accepts commands from the state machine and generates the correct sequence of commands to the PDU to control the lights and pump. This provides an abstract interface to the PDU hardware from the state machine. An abstract interface allows the same state machine daemon code to operate with different PDU command sets from different PDU manufacturers in the system. The video processing is similarly abstracted, allowing the same state machine daemon code to generate reports regardless of whether the video processing software is located on the control computer, on the
Figure 3.4: When user selects Cabinet A “Set.xml” is consulted, tank A is found under “tank” column, and its ID field found to be is A0. Next, “cameras.xml” is consulted and the cameras with where the “Set” matches the “ID” field are used in this trial.

private network, or on the public network.

The user specifies both the test to run and the cabinet that contains the resources to run the trial. The state machine needs a list of camera identifiers and the PDU identifier associated with this cabinet; it gets this information using the following procedure.

1. Each row in the “sets.xml” table (Figure 3.4) contains a TANK field. Code selects rows from the “sets” table, where the tank name matches the user-requested cabinet, and where the “enable” field in the row is marked “y”. Note that there may be multiple matches; this may be useful in the future as it allows multiple computers to process data for one cabinet. The EXECPRM field and the SWITCHIP field are extracted from the selected rows within the table. EXECPRM names the program that binds commands sent by the state machine to a sequence of strings that needs to be sent to the PDU to control the lights and pump. Currently, we use two different programs, “dlLogggers.pl” and “ip9258.pl”. The test state machine
calls these programs with the IP address supplied in the SWITCHCHIP field with arguments to turn on or off the pumps and lights. These programs are examples of implementations of the abstract PDU interface described above. If multiple rows contain the same value for the TANK, then the first row is used.

2. A list of cameras is constructed using the ID field from the “sets” table and the SET field from the camera table. See Figure 3.4. If more than one row in the sets table matches the requested TANK parameter, the list of cameras taken is the union of all rows in the “cameras” table where the SET field of the camera matches the ID field from the “set” table. This allows flexibility in how the data is routed in the system. We do not expect that this flexibility will typically be used.

3. The state machine tells the cameras where to send the video information. The Raspberry Pis identify the control computer as “swimmy”; however, other computers on the private network are allowed. The commands sent by the state machine to the cameras include the fully qualified path name (FQPN) where part of the path name includes the machine name of a NFS mounted directory. From the Raspberry Pi’s point of view, it is storing a file; it just happens to be storing it on a non-local machine. This configuration allows Raspberry Pi cameras within the same cabinet to send data to different data processing computers.

4. The post-processing system receives updates at different points during the test. The state machine running on the control computer calls the script identified in the POSTPROCESS field with a parameter that identifies the epoch of the trial.

The state machine needs two pieces of information to run a trial. The first is the test information which is contained in the tests table (Figure 3.5) Fields within this table identify timing information for the tests. The second set of information is the
Figure 3.5: Timing information for tests. The user selects the test ID, the other fields are test parameters.

<table>
<thead>
<tr>
<th>Test ID</th>
<th>Dark Time</th>
<th>Pump off</th>
<th>Pump on</th>
<th>Stab. Time</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>10Sec</td>
<td>0</td>
<td>10</td>
<td>10</td>
<td>0</td>
<td>Sanity test. Is it working</td>
</tr>
<tr>
<td>NoDark</td>
<td>0</td>
<td>120</td>
<td>120</td>
<td>60</td>
<td>Already dark adapted</td>
</tr>
<tr>
<td>DrugTest</td>
<td>1800</td>
<td>60</td>
<td>120</td>
<td>60</td>
<td>Test the drug</td>
</tr>
</tbody>
</table>

hardware configuration information which is maintained in the tables “sets.xml” and “cameras.xml” (Figure 3.4). This configuration represents the physical assets and connectivity of the system. The user is allowed to enable and disable cameras and set the “Region Of Interest” (ROI) values maintained in the table, but otherwise does not have access to this table. Configuration information is updated using two methods. Operators use web pages to define tests and update parameters within “test.xml” and “cameras.xml”. The tables that contain the cabinet routing and processing information (“sets.xml”) are accessible using secure shell and logging into the control computer.

Secure shell is supported on the Raspberry Pi camera. From an outside network, maintainers log onto the control computer, and from there may log onto a Raspberry Pi.

User Interface

The user interacts with the system through a series of web pages. Test operators are normally co-located with the control computer and test data collection is not affected when the internet is down. The system uses an Apache web server [25] and Perl Common Gateway Interface (CGI) [31] as the enabling technologies to create dynamic web page content. Briefly, the steps used to retrieve web pages are as follows.

- Users use a web browser such as Firefox or Chrome as a client program which makes a connection over the public network to the Apache web server. The
The web browser sends a request to the server specifying a particular web page.

- The web server sends Hypertext Markup Language (HTML) to the client. The returned content may include requests to load other pages, buttons, forms, and programs written in a language (typically JavaScript) [20], which is understood and executed by the web browser.

- The client examines received content from the web server, and further downloads content from the network. For example, if the initial HTML included a command to download a picture, then a subsequent call is made to download the picture’s content. The client will recursively download content from the server to build the web page displayed to the user.

- JavaScript programs may be started by the client. In our case, the “Display Status” page, which is downloaded as part of the “configuration” page, tells the client browser to poll the web server running on the control computer for camera status updates every five seconds. The web Apache web server running on the control computer requests current test state information from the state machine daemon, encodes it in XML, and sends the information back to the user’s browser. The browser decodes the XML reply and updates the current status to the user. This technology is known as “Ajax” [32]. Other common terms include “Client-side scripting” and “Client-side dynamic web pages.”

The HTML is delivered as a text file containing both code and data. Client software doesn’t care how the text file was created; it could be a static page, or dynamically computed. The Apache web server receives requests from the browser, determines from the request if it can deliver static (non-computed) content to the user, or if it needs to run a CGI script to compute the content for the web page. CGI scripts may interface with other programs via a TCP socket interface to retrieve information and make requests, such as starting a test or updating configuration information. CGI programs are only active while the HTML for the content of the
web page is being built, and cannot maintain state. Users want a coherent view of the system. For example, they want the system to remember who they are without having to enter a username and password on every screen. Three mechanisms are used to maintain state: hidden fields, cookies, and configuration tables.

The server may request the client to accept a cookie. In our case, a session cookie is created when the user logs in and identifies the user as someone who can access the system. The cookie is protected using cryptography techniques – web page content is encrypted over the network channel between the server and the client. Once logged into the system, the cookie is transmitted by the client on every page requested. CGI scripts on the server check to see that the cookie is valid, and will refuse access if the cookie is invalid.

Configuration state is maintained in a configuration files. The user has options to update the camera and test information. When the user presses the update button, the information is transmitted to the Apache web server, invoking a CGI script, which then invokes a network transaction between the CGI script and the state machine daemon. Depending on the message, the daemon process may respond in a variety of ways, including simply acknowledging the completion of the message, updating a configuration file, starting a test, or responding with current system status.

A simplified example of the test status screen shows these interactions. Users select a tab on the status screen to select a tank, and the status screen then displays the test status for that tank. There is a box for each Raspberry Pi, which is colored gray when idle, and orange when active. This screen updates every five seconds, and reflects the current state of the system. The following interactions take place.

1. The browser requests the status update page for a particular tank.

2. The Apache web server invokes a script written in CGI to build the page. The HTML encoding for this page includes JavaScript source code that is interpreted by the browser along with geometry for the placement of status
indicators that the browser renders; these status indicators change color reflecting the system’s state. The browser invokes JavaScript code which polls the server every five seconds.

3. JavaScript code running within the user’s browser requests the current status for a particular cabinet from the state machine daemon process running on the control computer. The request is transmitted to the Apache web server, invoking a script that makes a request to the daemon process for the current status of the Raspberry Pis associated with this cabinet. The daemon process returns this information to the CGI script, which in turn returns the information to the browser.

The previous process is repeated until the user navigates away from the webpage.

Note that both the users web browser and the state machine daemon process running on the control computer maintain state information. The daemon process maintains the current test state, while the client knows which cabinet the user is querying. State information is not maintained by the Apache web server.

**Raspberry Pi and Camera Control**

The Raspberry Pis are located on the private Ethernet network. The video is transferred at a rate of 17 megabits per second. The network capacity is one gigabit per second, allowing a maximum of 58 Raspberry Pis to be video recording at any one time. Each cabinet contains 16 Raspberry Pis, so a maximum of three tanks may be operated at one time giving a cap of 48 simultaneous video connections. The current test protocol records the video for five minutes, and users are instructed to wait three minutes for recording video from successive cabinets. This limits the number of in-flight videos to 32, which results in a bandwidth of 544 megabits per second. Further analysis of this is in Chapter 5.

The Raspberry Pis run a camera daemon that is invoked by the startup scripts when the Pis boot. This camera daemon opens a socket and waits for commands.
The camera daemon running on the Raspberry Pi allows multiple connections. It replies with queries about the current camera status (busy or idle), and responds to requests to record video. The control computer makes requests to the Raspberry Pis to record video and save the recorded files to specific file names, where the destination for the video file is on an NFS mounted drive. This drive is typically on the control computer, although it can be anywhere on the private network. Changes in the camera state are sent to all programs that have connected to camera daemon; in particular, the state machine running on the control computer is notified when the camera’s state changes from active to idle. The state of the Raspberry Pi’s camera is cached within the state machine daemon. This cached state information is reported to the user when he or she is viewing the status page.

The firewall rules on the control computer enable outgoing internet connections originating from the Raspberry Pis. These connections enable the Pis to update their operating system software and retrieve current time of day information. The system contains scripts to maintain the Raspberry Pi cameras. Maintainers log onto the control computer and navigate to the directory containing scripts used to maintain the Raspberry Pis, and run the script “runSetup.sh” that copies all the files contained in the directory “cameraScripts” to the Raspberry Pi, as well as the script “setupSwimmy.sh”. Once the files have been copied to the target Raspberry Pi, the script “setupSwimmy.sh” is automatically invoked on the Raspberry Pi. This script moves the files to the appropriate locations, connects to the internet and updates system software, and performs general housekeeping.

**Video Processing Subsystem**

The video processing subsystem receives notifications from the state machines that new video is available for processing. The processing software can be configured to process videos as they are completed (used for processing on the control computer), or process them in batch (useful on the HPC). The state machine daemon creates one or more directories where the Raspberry Pis send their videos. The location of these directories is determined from information in the “Sets.xml”
table. Normally, all video created within a cabinet is sent to the same directory, but his is not a requirement. The video processing subsystem is responsible for starting the video processing software. The video processing software requires a configuration file “configData.xml”, a file that identifies the user processing options such as the type of detection algorithms to perform “reports.xml”, the name of a video file to process, and the directory name to place its results. The video processing software generates a file called “RtReports.xml” which contains the results of the processing, its contents are displayed to the user. The video processing software may leave other results in the directory for further analysis, such as the images used for analysis, or spreadsheet information.

The software which does the actual video processing is unaware of the larger environment in which it is running. It knows its inputs and outputs; however, it may be running on any number of different configurations. We have three different configuration models for video processing, and it is straightforward to create more. Our defined environments are 1) local processing where video processing is done on the control computer, 2) processing on the HPC, and 3) processing on multiple processors located on the private network. Note that processing on the HPC requires sufficient network bandwidth on the public facing network to move video between the control computer and the HPC.

At the beginning of a test, the state machine creates a directory (the data directory) where the videos recorded by the Raspberry Pis are sent. Two files are created within this directory. The file “configData.xml” identifies which cameras are used in the test and the image coordinates of the region of interest (ROI), as well as test timing information. The file ‘reports.xml’ identifies the processing algorithms and parameters needed by the video processing software. The Raspberry Pis write video to the data directory, and video processing for each video is started sometime after the video has been sent to the data directory. The video processing system creates a file called ‘RtReports.xml’; this file summarizes the output from all videos recorded in this trial. Results from this file are used to generate the
summary information displayed to the user on the reports web page.

Three different processing models have been created to run the video processing software; these models were outlined in Chapter 2. The state machine identifies the processing model using the POSTPROCESSING field in the "sets.xml" database. The different scripts identified in this field cause processing to be performed using one of the three processing models: processing locally, processing on the HPC, or processing on the local network. These scripts package the videos along with the relevant configuration information, and send the information to some data processing computer. The scripts then start the video processing software on the data processing computer, wait (if necessary) for the processing to be completed and send the results back to the control computer. The results are made available to the user through the graphical user interface (GUI).

**Summary**

This chapter discussed the overall software architecture for the automation and control. It outlined the operation of the test state machine, options for video processing, responsibilities of the control computer and Raspberry Pis, data flow from the web pages to the server processes, maintenance scripts, configuration tables, and models for processing reports.
CHAPTER 4

VIDEO PROCESSING SOFTWARE

After video data has been captured by the system, the video processing software performs tasks such as automatic fish detection, rheotaxis analysis, and motion analysis. The video processing software receives a video file and configuration information, processes the video, and provides quantitative results to an end user. The video processing involves several stages: preprocessing, intensity thresholding, connected components labeling, feature extraction, and optional motion analysis. We refer to the first two stages (preprocessing and thresholding) collectively as detection. We have implemented alternative methods for fish detection, and we discuss their relative advantages and disadvantages. We also discuss our protocol for manual fish detection and rheotaxis analysis by human operators, which is used to determine the accuracy of the automated system. Finally, we show how objects from a sequence of frames can be connected into tracks for motion analysis.

Detectors

One approach to distinguishing foreground objects (i.e., fish) from the background in video data is to preprocess each frame to estimate a background image, and then detect fish by selecting each pixel whose intensity differs substantially from the corresponding pixel of the background image. One method for background estimation is the adaptive Gaussian mixture model (GMM) [33,34], which dynamically adapts the interpretation of foreground and background pixels based on how long an object has been stationary. This method works well if all the fish are swimming. Unfortunately, fish can sometimes stick to a wall of the swimming chamber. If a fish is stuck to the swimming chamber, the GMM may mistakenly consider the stationary fish as part of the background, which can lead to the fish not being detected.
Another preprocessing method involves the use of the bottom hat transform (BHT) [35–38], which enhances features smaller than a user selected structuring element and darker than surrounding pixels. (In our case, the fish are darker than the background.) Objects that are smaller than the structuring element and darker than the surrounding pixels will become the foreground pixels with a higher intensity value than the background after transformation. BHT works on a frame by frame basis. It can separate foreground from background information in a single image, and it does not rely on image history. BHT is followed by intensity thresholding to complete the fish detection task. The primary disadvantage of BHT (for our application) is the amount of time required to form the background image if the structuring element is large. Also, the structuring element must be applied to each incoming frame, and the structuring element must be larger than the size of the objects to be detected.

The details of the BHT computation are as follows. The bottom hat transform of image $f$ with structuring element $b$ is defined as $T_b(f) = f \bullet b - f = (f \oplus b) \ominus b - f$, where $\bullet$ is the morphological grayscale closing operator, $\oplus$ is the dilation operator, and $\ominus$ is the erosion operator. The structuring element $b$ is a disk with a radius greater than the size of a fish. In BHT, the closing operation produces a background reference image where dark objects smaller than the structuring element have been replaced by lighter background pixels. Each pixel in the foreground image is then subtracted from the constructed background image, creating a difference image. In this image, the dark objects of the input image that are smaller than the structuring element have been replaced by an intensity that is the difference between the background image and the object, while elements of the image that are larger than the structuring element have pixel intensities near zero. The result of BHT is that the input image’s dark objects (that are smaller than the structuring element) are enhanced and the unimportant intensity variations are suppressed. Detection is performed by thresholding this enhanced image. BHT creates a new reference image for each video frame. One advantage of doing this is that the background image tracks changes in lighting intensity.
In order to overcome the shortcomings of the aforementioned detectors, I have developed a spatiotemporal normalized thresholding (STENT) method for normalizing the intensities in video data both spatially and temporally in order to make the background have near-constant intensity. This preprocessing method has a training phase and a normalization phase. STENT builds a reference image during the training phase that is subsequently used in the normalization phase. We assume that the background from incoming frames differs only in overall intensity from the reference image. We normalize the incoming frame so that the its overall intensity matches the reference image, and then subtract on a pixel-by-pixel basis the normalized frame from the reference image to produce an enhanced frame. Intensity thresholding is applied to the enhanced frame and completes the detection process. STENT does not need to recompute the reference image for each frame since we assume that the spatial intensity variation of the background in the training frames is representative of the background in subsequent frames. The normalization process in STENT compensates for changes in lighting conditions across the scene and over time.

STENT operates similarly to BHT. It computes a reference image and then subtracts the scaled incoming frame from the reference image. However, STENT computes the reference image once per video by averaging the grayscale intensities of multiple video frames, and then performing morphological operations to remove any residual foreground objects from this image. It computes the average grayscale intensity of all pixels within this reference image, and as new frames are decoded from the video stream, it scales each pixel grayscale level so that the average grayscale intensity of the new frame matches the grayscale intensity of the reference image. The scaled input frame is then subtracted from the reference frame, resulting in an enhanced image. STENT assumes that the variation in the background over time can be modeled by one scaling factor that represents the change in illumination intensity over time. The illumination also varies spatially across the background, but STENT assumes that this spatial variation is the same for each frame. In our system the illumination is somewhat brighter near the cen-
ter of the swimming lane due to the geometry of the lighting system. Also, in our system when the pump turns on, the water level changes slightly, decreasing the overall illumination. STENT’s spatiotemporal normalization of pixel intensity compensates for the variation in illumination. The reference image in STENT is computed from multiple frames, so this tends to average out the effect of temporal variation in intensity when computing the reference image.

In the training phase, STENT uses a training set of frames to compute a reference image (i.e., a background estimate). During the normalization phase, it uses the reference image to normalize the subsequent frames. To obtain an initial estimate of the background, we compute the temporal average of a training set of $N_0$ video frames (we use the first 200 frames of the video):

$$\mu_t(x,y) = \frac{1}{N_0} \sum_t I(x,y,t)$$  \hspace{1cm} (4.1)

However, this average may contain dark areas where fish were stationary in the training set. We remove the dark areas using the morphological closing operation with a circular structuring element size of about 41 pixels (see Chapter 5 for details) to obtain the reference image (i.e., a filtered background estimate):

$$\hat{\mu}_t(x,y) = \mu_t(x,y) \bullet d(x,y)$$  \hspace{1cm} (4.2)

Next, we compute the spatial average of the reference image to obtain the scalar, spatiotemporal average of the sampled video frames:

$$\mu_{xyt} = \frac{1}{N_x N_y} \sum_x \sum_y \hat{\mu}_t(x,y)$$  \hspace{1cm} (4.3)

Once we have computed the reference image $\hat{\mu}_t(x,y)$ and the spatiotemporal average $\mu_{xyt}$, we can proceed to normalize each video frame during the normalization phase. To normalize with respect to just the temporal dimension, we can multiply $I(x,y,t)$ by $\mu_{xyt}/\mu_{xy}(t)$, where $\mu_{xy}(t)$ is the spatial average of $I(x,y,t)$. To normalize just across the spatial dimension, we can multiply $I(x,y,t)$ by $\mu_{xyt}/\hat{\mu}_t(x,y)$. 
We combine both of these normalization steps into a single, spatiotemporal normalization for each frame:

\[
I_{\text{normalized}}(x, y, t) = I(x, y, t) \frac{\mu_{xyt}^2}{\mu_{xyt}(t) \hat{\mu}_t(x, y)}
\]  (4.4)

The normalized image is then subtracted from the reference image to obtain an enhanced image, which will be thresholded to complete the detection process.

Note that \( \mu_{xyt} \) and \( \hat{\mu}_t(x, y) \) are computed just once per video, but \( \mu_{xy}(t) \) is computed for each frame. With the assumption that the number of foreground object pixels is small, the mean background intensity of a single, normalized frame \( I_{\text{normalized}}(x, y, t) \) closely matches the spatiotemporal average, \( \mu_{xyt} \), of the frames used in the training set.

We have a choice on how to process images once they have been transformed by either BHT or STENT. One approach is simple thresholding. Another approach is to use histogram-based thresholding in the detector, such as Otsu’s thresholding [39]. We will apply a fixed biasing term to Otsu’s method to bias the detectors against false positives. In Chapter 5, we compare the performance of the detection schemes, focusing on two principal methods: 1) BHT using simple thresholding, and 2) STENT with simple thresholding. In addition, we show results comparing Otsu’s thresholding with simple thresholding.

The STENT and BHT detectors described above are used to generate fish detections from video frames. Fish positions are determined using the following steps. First, each frame from the video is decoded and transformed into a grayscale image. A frame counter is incremented for each video frame decoded. Second, an operator-selected detector (or Otsu threshold) determines which pixels in the image are foreground or background, producing a binary image (see bottom of Figure 4.3). Third, connected component labeling [40] is applied to the binary image containing the detection results for a given frame, and the contours of the foreground components are determined. We compute the raw moments \( m_{00}, m_{10}, m_{01} \), and the central moments \( \mu_{20}, \mu_{11}, \mu_{02}, \mu_{30}, \mu_{12} \) and \( \mu_{03} \) using Greens function
for each contour found, and save the moments along with the video frame number to a file as features for later processing. Raw moments are defined in equation 4.5. Central moments are defined in equation 4.7 [42]. The function $I(x,y)$ is the indicator function, and is equal to 1 if the fish covers the pixel at location $(x,y)$.

\[
m_{jk} = \sum_{x=-\infty}^{\infty} \sum_{y=-\infty}^{\infty} I(x,y) \quad (4.5)
\]

\[
\bar{x} = \frac{m_{10}}{m_{00}}, \quad \bar{y} = \frac{m_{01}}{m_{00}} \quad (4.6)
\]

\[
\mu_{jk} = \sum_{x=-\infty}^{\infty} \sum_{y=-\infty}^{\infty} (x - \bar{x})^i(y - \bar{y})^j I(x,y) \quad (4.7)
\]

The centroid is calculated for each fish in each frame using the stored raw moment information. The centroid for each fish is given by equation 4.6.

**Orientation Analysis**

The orientation calculation is a two-step process. First, the major axis along the fish is calculated. We use principal component analysis (PCA) [43] to compute the major axis by finding the eigenvectors and eigenvalues of the covariance matrix of the second central moments. The covariance matrix is defined as

\[
C = \begin{bmatrix}
\mu_{20} & \mu_{11} \\
\mu_{11} & \mu_{02}
\end{bmatrix} \quad (4.8)
\]

The fish’s orientation is parallel to the eigenvector given by the largest eigenvalue. There are two possible directions the fish can be oriented; this ambiguity is called the “head-tail ambiguity”. We use asymmetry information contained in the third central moment to resolve the head-tail ambiguity. One approach is to use PCA to rotate the fish to be parallel to the $x$ axis, and then use the skewness of the fish to determine which end is the tail. If the skewness is positive, then the tail aligns with the positive $x$ axis.

Figure 4.1 gives an example of a one-dimensional case where the fish has been
Figure 4.1: Head-tail ambiguity resolution. Assume fish is oriented along X axis, and all pixels have equal weight. For each value of $x$, the mass along the X axis is $M(x) = \sum_{y=-\infty}^{\infty} I(x, y)$, where $I(x, y) = 1$ if there is a detection at position $x, y$ (bottom of figure).

rotated to align with the $x$ axis. At any point along the $x$ axis, the mass distribution along the $x$ axis is the count of pixels along the $y$ axis at position $x$, which is $M(x) = \sum_{y=-\infty}^{\infty} I(x, y)$, where $I(x, y) = 1$ if a fish covers position $x, y$, else 0. The one-dimensional third central moment along the $x$ axis is defined as $\mu_3 = \sum_{x=-\infty}^{\infty} M(x)(x - \bar{x})^3$ where $\bar{x}$ is the $x$ value at the centroid. If $\mu_3 > 0$ then the tail is deemed to be in the direction of the positive $x$ axis, else the direction of the negative $x$ axis. The single dimensional skewness is related to the third moment in the following way. The skewness is a measure of asymmetry of a distribution about a mean, and is the third moment $\mu_3$ [42]. The one dimensional K’th central
moment is given by

\[ \mu_k = \sum_{j=1}^{N} (x_j - \bar{x})^k f(x_j) \] (4.9)

\[ \mu_3 = \sum_{j=1}^{N} (x_j - \bar{x})^3 f(x_j) \] (4.10)

\[ \bar{x} = \frac{1}{N} \sum_{i=1}^{N} x_i \] (4.11)

In equation 4.9 [44], the function \( f(x_j) \) is the weight at location \( x \). We take this as \( f(x_j) = M(x) \) (Figure 4.1) = \( \sum_{y=-\infty}^{\infty} I(x,y) \) which is computed for each position along the \( x \) axis.

\[ M(x) = \sum_{y=-\infty}^{\infty} I(x,y) \] (4.12)

\[ \mu_{30} = \sum_{x=-\infty}^{\infty} \sum_{y=-\infty}^{\infty} I(x,y)(x - \bar{x})^3(y - \bar{y})^0 \] (4.13)

\[ = \sum_{x=-\infty}^{\infty} \sum_{y=-\infty}^{\infty} I(x,y)(x - \bar{x})^3 \] (4.14)

\[ = \sum_{x=-\infty}^{\infty} M(x)(x - \bar{x})^3 = \mu_3 \] (4.16)

Instead of rotating the fish to perform the above skew analysis, we avoid that computation by using the 2-D third central moments to obtain a heuristic estimate of the skewness along the major axis of the unrotated fish. Our feature sets for each fish includes the third central moments, \( \mu_{30} \) and, \( \mu_{03} \). The sign of these two moments indicates which quadrant (related to the major and minor axes) the tail of the fish is located in, and the four-quadrant arctangent (atan2) [45] gives a general direction of the tail. To resolve the head-tail ambiguity, we choose the tail direction to be the end of the PCA major axis that most closely matches the angle found by \( \theta = \text{atan2}(\mu_{03}, \mu_{30}) \).
Rheotaxis Index (RI) = \frac{\text{Number of zebrafish within 30 degrees}}{\text{Total number of detected zebrafish}}

Figure 4.2: Rheotaxis Index (RI) computation. The water flows from the left in our setup. The total number of detected zebrafish that are swimming within 30 degrees of the oncoming flow direction is divided by the total number of detected zebrafish to determine RI.

The RI is computed from the fish orientations relative to the water flow direction. RI is given as the percentage of fish swimming within 30 degrees of the oncoming flow of water. The angles where the fish is swimming in rheotaxis are shown Figure 4.2. The value of 30 degrees was chosen to agree with work of other investigators [10,12].

Figure 4.3 illustrates the orientation estimation for a sample video frame. The bottom section shows the detections after thresholding. The white areas represent fish detections. The top section of the figure shows the fish marked with ellipses (with axes derived from the principal component analysis), and the orientation of each fish marked with a dark line on one end of the ellipse.

**Method for Assessing Orientation Accuracy**

The accuracy of the system was determined by comparing the computed fish positions and orientations against those identified by four human observers. The criterion for success was that the orientation error of objects detected by the au-
Figure 4.3: Detection process. The upper part of the frame shows the detected fish. The fish are orientated with their head near the mark shown on the ellipse. Bottom frame shows outline of fish detections after thresholding. White areas are pixel values that are less than the background value.

tomated system be no greater than the human inter-operator error. A tool was developed to record human observations where user clicks on the head of a fish to mark its position and drags a line from the fish’s head to mark the fish orientation. The user presses the “n” key on the keyboard to advance the video by six seconds. The user can rewind the video and replay the fish swimming to its next position by pressing the “a” key. A separate scoring file was generated for each tester. Of the 49 frames analyzed, two testers found 538 detections; one tester found 541 detections; and the fourth tester detected 530 fish. Each tester detected approximately 11 fish per frame. The measurements by the human observers were then processed to derive the ground-truth fish positions and orientations using the following procedure.

1. The position, orientation and frame number of each fish found by each tester was tabulated.

2. For each frame, the mean shift clustering algorithm [46] [47] was applied to
the tabulated fish positions, and the computed cluster centers were deemed to be the ground truth fish positions. We used a circular disk kernel with a radius of 11 pixels to form the clusters.

3. For each ground truth fish cluster center, several values were tabulated: the circular mean orientation (Eqn. 4.17) of the positions marked by the individual testers, the angular mean absolute deviation (Eqn 4.20), and the number of samples in the cluster. These quantities are discussed in more detail below.

Circular statistics [48,49] are used to compute average orientation angle. Let \( M \) be the number of ground-truth fish, and let \( N \) be the number of human observers. Let \( \hat{\theta}_{ij} \) be the orientation of the \( j \)th ground-truth fish as measured by the \( i \)th human observer. For the \( j \)th ground-truth fish (i.e., cluster center), the ground-truth orientation is computed as the circular mean of the orientations measured by the human observers:

\[
\hat{\theta}_j = \text{atan2}(\hat{S}_j, \hat{C}_j), -\pi < \hat{\theta}_j \leq \pi
\]  

(4.17)

where

\[
\hat{S}_j = \frac{1}{N} \sum_{i=1}^{N} \sin(\theta_{ij})
\]  

(4.18)

\[
\hat{C}_j = \frac{1}{N} \sum_{i=1}^{N} \cos(\theta_{ij})
\]  

(4.19)

To quantify how much the orientation measurements deviate from the ground truth, we use mean absolute deviation (MAD) –inline [50] as the figure of merit. The individual human orientation error is computed as the (MAD) between the human-measured orientation and the ground-truth orientation:

\[
\hat{\phi}_j = \frac{1}{N} \sum_{i=1}^{N} \left| \theta_{ij} - \hat{\theta}_j \right|
\]  

(4.20)

The average human orientation error is the average of the MAD values over all detections:

\[
\bar{\phi} = \frac{1}{M} \sum_{j=1}^{M} \hat{\phi}_j
\]  

(4.21)
We only compute MAD values for fish where we have both machine detections and ground-truth.

For each ground-truth fish, the machine orientation error is computed as the angular difference between the ground-truth orientation and the machine-computed orientation of the nearest machine-detected fish. Rules are applied to qualify the pairings of fish for each such computed orientation error. There are several instances where the humans significantly disagree among themselves in the orientation measurement. Human errors include not marking a fish (in this case, a cluster will have fewer than four observations) and marking the wrong fish orientation. Fish swim in and out of the field of view, and it is difficult to determine the orientation of a fish that is not fully observed. We apply the following rules to qualify the orientation errors. 1) If either the ground-truth position or the machine-computed position is within 11 pixels of the border, eliminate this sample (the fish must be fully in view) from the set of orientation errors. 2) Eliminate the sample if the fish was not detected by all four human observers. 3) If the human observers disagree in orientation by more than 50 degrees, eliminate the sample.

The average machine orientation error is the MAD between ground-truth orientation $\hat{\theta}_j$ and the machine-computed orientation $\alpha_j$ of the nearest machine-detected fish (for qualified pairings):

$$\alpha = \frac{1}{M} \sum_{j=1}^{M} |\alpha_j - \hat{\theta}_j|$$

(4.22)

**Human Orientation Error vs. Machine Orientation Error**

Circular statistics makes orientation error analysis difficult. However, it is possible to get a sense of the system performance by comparing the machine orientation error to the inter-operator error. Assuming that the human orientation errors are Gaussian distributed (a big assumption), we can approximate the human standard deviation as 1.25 times the machine-to-human MAD [51]. Then we can compare the human standard deviation to the standard deviation of the machine orientation errors.
Our observation-by-observation comparison subtracts two random variables; the first is the average of human-determined ground truth orientation, and the second is the machine orientation. Random variable subtraction is convolution in the probability density function (PDF) domain.

**Motion Analysis**

A current topic of study in the image processing community is the particular case of object tracking where each tracked object in the image is indistinguishable. In 2000, McCormick [52] proposed tracking multiple objects based on exclusion principles and particle filters. Khan [53] discusses the tracking of ants using particle filters. The ants’ behavior changes as the ants come within a certain distance of each other, and the filter changes its characteristics based on ant proximity. The tracking of multiple zebrafish larvae is, in general, an open problem. Martineau surveyed different methods and difficulties related to this topic [54].

We distinguish motion from tracking. Tracking implies that a label assigned to a particular fish is maintained throughout the run of the trial. The fish do not have individualized markings. It is possible to stain or otherwise mark the fish, but this option was rejected because the interactions between the chemistry of marking the fish and drug interactions are unknown and need to be determined on a case-by-case basis. An alternative to tracking is to place one fish in each swimming chamber; that way the identity of the fish in the swimming chamber is known.

Motion detection allows us to determine whether the fish are swimming and, for the duration of the track, the percentage of time that the fish spend swimming within 30 degrees of the water flow.

Lighting conditions vary within the swimming chambers. The illumination near the edges of the swimming chambers falls off around the center, giving lower contrast between the fish and the background, resulting in a lower signal-to-clutter ratio (SCR). There are areas in the swimming chamber where multiple fish can congregate, appearing as one fish. It is impossible to determine either fish’s identity once they diverge. We feel that improvements in detection and tracing algorithms,
lighting conditions, and perhaps swimming chamber geometry are necessary to extend unique labeling of fish identity over time, coming closer to “tracking”.

Our motion analysis approach does not guarantee unique labeling of each fish across the trial. However, it does provide useful information. Conventional tracking models involve a sequence of measurements and a model of the dynamics of the system. The model state is advanced in time, and the probability of each the measurement given the particular state of the model is calculated. The model is then updated based on the error in its current state and the measured values. Our approach to motion is simpler. We have three cases. First, the fish in subsequent video frames are positioned close enough to the current frame that this fish is unambiguously the same fish (this is set by a user-defined cutoff). Second, multiple fish may occur in the subsequent frames that could match the fish in this frame. Third, no fish in subsequent frames is likely to be a match for this fish.

Our matching algorithm is divided into two parts. The first part forms partial tracks based on a sequence of detections. A partial track maintains the frame number and position of the first and last detections added, and the list of detections associated with this partial track. Partial tracks are created on an “open” list. If, after a certain user-settable amount of time, the partial track has not been appended to, the track is moved to the “closed” list. Detections are not added to partial tracks on the “closed” list. The partial track formation algorithm consists of the following steps:

1. On initialization, or if there are no partial tracks, we form partial tracks from the current detection list.

2. Create an $M \times N$ array of detections by partial tracks. Each element in the matrix has four parts; they are the distance between the partial track and the fish detection, an index to the partial track, an index to the detection, and a flag that is initially set to “unused.”

3. Flatten the matrix created in Step 2 and sort the results by distance. Walk
Figure 4.4: Formation of tracks using multiple detections. Fish detections from multiple frames are grouped into tracks based on the proximity between detections over time into track fragments, and the grouping of track fragments into tracks. Tracks are groupings of detections for our best guess of the same fish over time.

through this list performing the following:

(a) If the flag is “unused,” do the following:

i. If the distance is less than the threshold value, associate this detection with the partial track. Change the flag to “used,” and mark all other elements that contain this detection or this track as “used.”

ii. If the distance is greater than the threshold value and the flag is marked as “unused,” create a new partial track with this detection. Mark all further elements that contain this detection as “used.”

4. If the partial track has not been appended for some user settable number of frames (default is 10), move the partial track to a “closed” list, which means that this track can no longer be used.

5. Repeat the above steps until all frames have been examined.

Partial tracks are associations of the closest detections over time. The process model is simple; the closest detection that appears within some distance and time of the last detection of the partial track is appended to the partial track. Detections that cannot be matched to partial tracks create new partial tracks.
Similar logic forms full tracks. We want to merge the closest end time and position of each track with the beginning time and position of a partial track. There are some restrictions. The start time of the partial track must be greater than the current track end time. However, that time is bounded. The difference in position is also bounded. The fish have different swimming behavior when the water is flowing verses when it is stationary; they tend to congregate near the outlet area of the swimming chamber. The probability of a partial track being part of a track fall off quickly over time. The system is causal; the start of a partial track must occur after the end of the current track for the partial track to be merged with the current track. See Figure 4.4 for an example.

We can determine useful metrics, such as the percentage of time a fish is orientated upstream, and the percentage of fish that are swimming. Other uses for track information are to apply median filtering algorithms to reduce the probability of false orientations due to head-to-tail ambiguity, or partially observed fish.

The tracking problem where the items observed have identical features is very difficult, and as of 2013 has not been solved [54]. This system makes progress towards that goal, and our partial solution provides benefits. It is unclear how much effort will be required to uniquely identify a fish over the course of an experiment. It may be possible to stain or otherwise mark the fish so that they are distinguishable. This topic will be discussed in Chapter 6.

**Summary**

This chapter outlined the detection processing time requirements, and how STENT minimizes processing overhead. Chapter 5 will show that the accuracy of STENT and BHT detection are comparable. BHT detection is adequate for our current needs when processing one frame out of 180; however, motion processing will require faster algorithms such as STENT or more hardware. Full tracking requires more than algorithmic change; currently, the fish can overlap in areas of the swimming chamber, and the ambiguities cannot be determined. Lighting may correct these issues, or in the worst case, some form of redesign of the swimming
chambers may be required, being careful not to introduce new distortion artifacts into the chamber.
CHAPTER 5

ANALYSIS AND PERFORMANCE

This chapter discusses performance measurements for the system’s accuracy and throughput. We tested different processing configurations, and from these measurements, we determined the number and type of hardware and computing resources required to meet a target goal of 1000 drug combinations per year. Testing of processing throughput was conducted using the STENT and BHT detection methods. We also evaluated Otsu thresholding versus fixed thresholding, and we compared processing each frame (beneficial for motion analysis) to processing one frame every six seconds (sufficient for RI calculations). Tests were conducted using an Intel NUC I5 computer running at 1.3 GHz with 16 GB of memory, and a 48-core dual-threaded Xeon 2.1 GHz processor with 256 GB of memory as the video processing computers. We use the following definitions for the detectors within this chapter. The STENT method using a fixed threshold is referred to as the STENT detector, the STENT method with Otsu thresholding is the STENT-O detector, and the BHT method with fixed thresholding is the BHT detector.

Consider a goal of processing 1000 drugs per year, where we test 10 dosage levels per drug, for a total of 10 000 experiments per year. Assuming 250 working days per year, this requires 400 experiments per day where each experiment is one drug at one dosage level and uses one swimming lane. Our system groups 16 experiments into one trial. We need to run twenty-five trials per day (400/16 = 25). We assume a trial rate of one trial per hour per cabinet, (Each trial requires 35 minutes of test time plus time to change fish) and that we have at least four cabinets available so that the tests can be accomplished in an eight hour day. (25 hours for trials / 8 hours per day = 3.1 cabinets). Six hundred megabytes (0.6GB) of data is generated per lane, 61GB are generated per drug dosage level, and 244GB are generated per day (400 experiments per day times 0.61GB per experiment). Each
video is five minutes long and is recorded at 30 frames per second, giving 9000 total frames per video. The current RI calculation extracts one frame for processing every six seconds for a total of 50 frames to process per video. If we want to do motion analysis at 30 frames per second, we need to process 9000 video frames. We examine the processing times for STENT detection and BHT detection within two different computer environments, first, processing on the control computer and second, processing on a 48 core / 256 GB of memory computer. We do not present a separate throughput analysis for STENT-O detection processing because the processing requirements are similar to STENT.

**Processing on Control Computer**

We timed the video processing using the BHT and STENT detectors for both the 50 and 9000 frame cases for one trial (16 swimming lanes) on the control computer using the Linux “time” command. The control computer is an Intel NUC i5 running at 1.3 GHz with 16 GB of memory. This computer is a dual-core device with two threads running concurrently per core, for a total of four concurrent threads. The video processing code (FFmpeg) [55] is multithreaded. The times taken to process videos on the control computer are shown in Table 5.1.

<table>
<thead>
<tr>
<th></th>
<th>Wall Clock (Sec)</th>
<th>User Process Time Per Lane(Sec)</th>
<th>System Time Per Lane(Sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>STENT, 50 Frames</td>
<td>1323 (22 Min)</td>
<td>327</td>
<td>3</td>
</tr>
<tr>
<td>BHT, 50 Frames</td>
<td>1427 (23 Min)</td>
<td>409</td>
<td>11</td>
</tr>
<tr>
<td>STENT 9000 Frames</td>
<td>3301 (55 Min)</td>
<td>937</td>
<td>3</td>
</tr>
<tr>
<td>BHT 9000 Frames</td>
<td>43393 (12 Hr)</td>
<td>12641 (3.5 Hr)</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 5.1: Example video processing times on an Intel NUC i5 computer. The current system uses the NUC as the control computer. See text for details.
libraries that start threads which prefetch, read and decode video data from disk. Our code that performs detections operates on a separate thread from the library created threads and takes a different amount of processing time depending on if we are running STENT or BHT detection.

Table 5.1 shows that both STENT and BHT require approximately the same amount of computer time for one frame processed per six seconds (the 50-frame cases). The system time per lane which represents the amount of time the operating system is performing operations on behalf of the process is small compared to overall processing time. In our case, most of the system time is spent performing disk read operations. For the 50-frame cases, four threads are operating concurrently. The reading and decoding of video frames are time multiplexed with our detection processing.

It is possible to process 25 trials within 24 hours on the control computer for both 50 frame detection schemes (STENT and BHT). The processing of all frames using STENT for motion on the control computer is tight, requiring 23 hours of processing time. We need roughly 12 times the processing time to process BHT 9000 frame detections for motion (12 hours of processing per trial times 25 trials is 300 run time hours per day, hours divided by 24 hours per day gives us the amount the level of increase).

We have three alternatives to processing BHT 9000 motion data from all the video generated by the system in a day. 1) use a faster control computer, 2) use an alternate computer attached to the network to process information, or 3) use more computationally efficient algorithms such as STENT. Scripts have been written and tested which transfer video information to the universitys high-performance cluster (HPC), instruct the cluster to process the data, and return the video to the control computer (see Chapter 3). Tests were performed using this cluster for BHT detection. The results of these tests indicated that on a core-by-core basis, timings are comparable with the control computer – approximately 5 minutes of time are required per BHT video with one frame processed every 180 frames. The advantage
Table 5.2: Example processing times on a 48 core computer. This processing was done on an otherwise idle server.

<table>
<thead>
<tr>
<th></th>
<th>Wall Clock (Sec)</th>
<th>User Process Time Per Lane(sec)</th>
<th>System Time Per Lane(Sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>STENT, 50 Frames</td>
<td>72 (1 Min)</td>
<td>325</td>
<td>5</td>
</tr>
<tr>
<td>BHT, 50 Frames</td>
<td>73 (1 Min)</td>
<td>318</td>
<td>5</td>
</tr>
<tr>
<td>STENT 9000 Frames</td>
<td>259 (4 Min)</td>
<td>500</td>
<td>16</td>
</tr>
<tr>
<td>BHT 9000 Frames</td>
<td>861 (14 min)</td>
<td>1087 (18 min)</td>
<td>10</td>
</tr>
</tbody>
</table>

of using the HPC is that many processors operated in parallel, potentially one processor for each video.

**Processing on Multiple Core Machine**

A newer machine with 48 cores, 256 GB of memory and 2.1 GHz processors was made available. The number of available threads on this machine greatly outnumbered that required for concurrent processing of the videos in a trial. The timings are given in Table 5.2.

Several items stand out in Table 5.2. First, the system time using the 48 core computer is comparable to the system time using the NUC control computer, indicating that kernel operations conducted on behalf of the process took a similar amount of time. Second, the user process time per lane 50 frame STENT detection cases is the same. The User Process time divided by the wall clock time gives an indication of the number of processor tasks operating in parallel on the problem. The User Process time is the sum of time for threads working on generating detections for this video. For the 50 frame cases, an average of 4.5 processors operated in parallel decoding video data (325 seconds/72 seconds). An average of 1.9 threads per video processed the STENT 9000 case (500/259), and one thread dominated the timing in the BHT case. The average number of threads used in the BHT 9000 case is 1.3 (1087/861) threads operating per video detected.
The effective throughput for trials per hour for BHT detection can be calculated by:

- Determining the effective number of threads required per video by dividing the user process time per lane by the wall clock time (1087/861) = 1.26.
- Determining the number of trials per hour = 60 minutes/14 minutes = 4.28 trials per hour.
- Determining the number of threads required per trial = 16 * 1.26 = 20.16 threads per trial.
- Dividing total threads available by the threads required per trial = 96/20.16 = 4.76. This number gives the number of trials that can be simultaneous computed.
- Multiply the number of simultaneous trials by the number of trials per hour = 4.76 * 4.28 = 20 trials per hour supported.

Roughly, this is saying that each video analyzed with BHT detection requires 1.3 threads, and that 80 swimming lanes are supported using all cores, and four sets of 80 swimming lanes can be processed in an hour. A processing rate of 320 lanes per hour is 20 trials per hour. We anticipate that at the capacity of 1000 drugs per year the processing load will be 25 trials per day.

Recently computers have shown up on Ebay with sixteen cores for under $1000. This class of machine is sufficient to process detections using the BHT detector.

**Data Transfer Timing**

There is a penalty for moving data from the control computer to the data processing computer. The network bandwidth needs to be sufficient so that the movement of video data to the data processing computer does not create a system bottleneck. We wish this data transfer to take less than the approximate time
<table>
<thead>
<tr>
<th>Source</th>
<th>Destination</th>
<th>Rate MB/s</th>
<th>Transfer time (hours)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grumpy</td>
<td>HPC</td>
<td>54 MBPS</td>
<td>1.26</td>
</tr>
<tr>
<td>Grumpy</td>
<td>Khalessi</td>
<td>61 MBPS</td>
<td>1.12</td>
</tr>
<tr>
<td>Doc</td>
<td>HPC</td>
<td>49 MBPS</td>
<td>1.39</td>
</tr>
<tr>
<td>Doc</td>
<td>Khalessi</td>
<td>80 MBPS</td>
<td>0.85</td>
</tr>
</tbody>
</table>

Table 5.3: Data movement tests from different control computers to computer clusters and servers. The results show that the network throughput is sufficient to sustain traffic between control computer and the data processing computers required to process the data. The reports sent by the video processing computer back to the control computer are small (less than 1 percent) of the size of the video sent to the video processing computer.

We have measured throughput from different computers on campus to estimate throughput from the control computer to the high-performance cluster and the 48 core computer described above (Khalessi). The data sources for these tests were: 1) A Core 2 Duo XPS 410 (Doc), and 2) An Intel I3 series notebook (Grumpy). Table 5.3 summarizes the time taken for data transfer for 256 GB of data collected per day from the 24 trials to transfer between the various machines. These tests indicate that sufficient bandwidth is available to move the 256 GB of data collected per day.

**Accuracy Analysis**

This section discusses the accuracy of STENT preprocessing with fixed thresholding, the BHT with fixed thresholding and STENT with Otsu thresholding detectors. We simplify these terms to STENT, BHT, and STENT-O detectors. The detectors were operated under two conditions, one condition optimized the accuracy of fish location, the other optimized the fish orientation.

Four observers were asked to score fish detections at six-second intervals. We analyze accuracy using two criteria. We examine angular and position accuracy of automated detections compared to human observers. We call each fish location a
“cluster”, where all observers have agreed on the location to within 11 pixels. Our test data consists of approximately 500 observations of fish made over 49 frames by four observers. Our analysis begins by determining how well humans agree on the orientation of the clusters. We do this in two stages. First we will create a histogram of the differences between each human observation of a cluster orientation and the average value of the cluster orientation where the average value is defined in equation `<ch4:circMean>`. Second, we will determine the MAD between the human observers and the average orientation of the cluster. Third, we will create histograms using the three detectors. Each of the three detectors was operated under two conditions, one was to optimize position accuracy compared human observers, the second was to optimize orientation accuracy. Fourth, we will examine the F-score of the three detectors when optimized for accuracy in position and orientation compared to human observers. Fifth, we will summarize tabulate the results. Sixth, we will show a plot of RI by Epoch of the three different detectors, along with the results from manual analysis.

Figure 5.1 shows the distribution of human-to-human inter-operator error. The mean angle of the a cluster of fish from human observers is calculated from 500 clusters and total 2000 human observations. A list of differences was created by calculating the circular mean angle of cluster of fish observations, and for each cluster, subtracting the human observed angle from the cluster mean. Figure 5.1 is a histogram of the resulting list. It shows four overlapping distribution of differences (one for each human observer) when looking at the same fish. The MAD differences for each observer for this was computed using equation 4.21 and found to be 4.77, 4.42, 4.23 and 4.23 degrees. The average human to human variation using the MAD score is 4.41 degrees.

Figure 5.2 shows the angular deviation between the clusters and the machine-generated detections using the three detectors optimize for both position and orientation accuracy. One plot shows the best F-score (described below), the second plot shows the best angular match. Their are two tunable parameters. The structuring
element size is the size of a circular structuring element using in the morphological closing operation described in Chapter 4. The “parameter” value for STENT and BHT is the fixed threshold level used in the detector to determine foreground from background pixels. In the case of STENT, this threshold is the grayscale intensity of the input image when normalized to the mean background value. For BHT, this is the threshold applied to the reference image (calculated by morphological processing) minus the input image. For STENT-O, this is a bias term that shifts the Otsu determined threshold in a way that increases margin against false positives. This has the side the effect of decreasing the number of pixels in a fish observation.

We quantify the position accuracy using F-score. For each of the detectors we

Figure 5.1: Histogram of Human to human differences for four human observers. MAD differences are 4.77, 4.42, 4.23, and 4.23 degrees.
Figure 5.2: Histogram of angular differences between the averages of human observations, which are taken as truth, and the corresponding machine generated detections. Curves shown for best F score and smallest angular deviation for STENT, BHT and STENT-O thresholding. Machine to Human errors range from 3.91 to 4.55 degrees depending on selected detector and parameters. Numbers in caption are arguments for the detection algorithm and are discussed in the text.

have varied the parameter and recorded the precision and recall values and have determined the F-score. A true positive is where the system identifies something as true; in our case, the machine matches the human observers. A false positive is when the condition is false, but the machine gave a true answer. A false negative is where the condition is true but the machine gave the answer as false. Precision, recall, and F-score are calculated from true positives (TP), false positives (FP), and false negatives (FN) using the following:

Precision $p$, Recall $r$, and F-score $F$ as:
Figure 5.3: Precision and Recall plot showing location errors as the threshold variable is varied. All detectors approach an F score in precision or recall of 1.0 depending which “truth” detections were included. Numbers in caption are arguments for the detection algorithm and are discussed in the text.

\[ p = \frac{TP}{TP + FP} \]  \hspace{1cm} (5.1)  

\[ r = \frac{TP}{TP + FN} \]  \hspace{1cm} (5.2)  

\[ F = \frac{2pr}{p + r} \]  \hspace{1cm} (5.3)  

The precision and recall show in Figure 5.3 approach 1.0. Two curves are plotted for each detector; one curve represents the best match to F-score, the other the best match in orientation. Structuring element sizes are held constant and given as the parameter for the curve. The STENT and BHT detectors varied the threshold used to determine the foreground versus background pixels. The parameter value for STENT-O was an offset applied to the value returned by the Otsu threshold function. We found that varying this offset increased noise immunity and resulted
Table 5.4: Best detection MAD scores

<table>
<thead>
<tr>
<th>Detector</th>
<th>Parameter</th>
<th>SE Size</th>
<th>F-Score</th>
<th>MAD Angle</th>
</tr>
</thead>
<tbody>
<tr>
<td>STENT</td>
<td>200</td>
<td>41</td>
<td>0.9922</td>
<td>3.91</td>
</tr>
<tr>
<td>STENT-O</td>
<td>-7</td>
<td>45</td>
<td>0.9922</td>
<td>3.99</td>
</tr>
<tr>
<td>BHT</td>
<td>12</td>
<td>19</td>
<td>0.9886</td>
<td>4.13</td>
</tr>
</tbody>
</table>

Table 5.5: Best detection F-score

<table>
<thead>
<tr>
<th>Detector</th>
<th>Parameter</th>
<th>SE Size</th>
<th>F-score</th>
<th>MAD Angle</th>
</tr>
</thead>
<tbody>
<tr>
<td>BHT</td>
<td>14</td>
<td>23</td>
<td>0.9961</td>
<td>4.11</td>
</tr>
<tr>
<td>STENT-O</td>
<td>-11</td>
<td>27</td>
<td>0.9961</td>
<td>4.55</td>
</tr>
<tr>
<td>STENT</td>
<td>198</td>
<td>39</td>
<td>0.9960</td>
<td>4.04</td>
</tr>
</tbody>
</table>

in fewer false positives.

Table 5.4 shows the results when the detector’s parameters are chosen for best performance that minimizes angular deviations. For these cases, the detectors exceed the performance bounds set by human observers. We conclude that all three detection strategies meet the orientation requirements.

Additionally, we determined the best parameters for locating the position of the fish based on F-score. We see virtually no difference in F-scores between the two sets of parameters indicating that we can optimize for angle without suffering in detections.

Finally, we compare the RI indexes from the four human observers to the RI scores determined by the machines observations. Figure 5.4 shows the four human observers (triangles in aqua color) and the machine observations (circles of different colors). Machine-generated RI scores are within the ranges of the human-generated scores, indicating agreement between machine and human observations.

The Otsu threshold values calculated at runtime are nearly constant over the running of the trial. We expect this. The STENT preprocessing step maintains nearly the same background characteristics over the course of the trial. The thresh-
old value can be set using Otsu’s method on the training set, and held constant during the test. A slight modification can be made to the STENT algorithm forcing the reference frame to a particular intensity; that modification should allow a fixed threshold for STENT.

It should not be surprising that the system and the human observers agree. Humans and machine both scored the same videos. Any optical distortion or artifacts that occurred in the video affected both the humans and the machine detections. Nonetheless, the experiments showed the system accuracy matches the RI results generated by human observers.

We have demonstrated that the two forms of the STENT algorithm and the BHT algorithm achieve sufficient levels of accuracy for both orientation and posi-
tion. We observe that STENT processing on the control computer takes 55 minutes to process all frames in a five-minute video for a trial that consists of 16 swimming lanes and that BHT takes 12 hours to process the same information. The current control computer is capable of processing 25 trials in a 24 hour period using STENT. However, BHT processing requires a computer with more robust computational abilities to process every frame. The distinction may be important in the future if motion detection and processing is implemented.

**Summary**

This chapter outlined the throughput necessary to support the testing of 1000 drugs per year and demonstrated three sets of computation and algorithmic methods to support that level of throughput. Throughput can be achieved using a low-cost computer and the STENT algorithm. Alternatively, a computer with more capacity, or a computer cluster, can be used to process data. Network throughput was measured; it is sufficient to support the data traffic necessary to sustain the testing rates. The experiments run on the system show that the automated system achieves accuracy results comparable to human observers.
CHAPTER 6
CONCLUSIONS AND FUTURE WORK

The primary purpose of this system as built is to screen drugs by examining the swimming behavior of zebrafish, and measuring the percentage of time that they spend swimming upstream. The fish are constrained to use their sense of touch which is a property of the health of the hair cells along their lateral line to detect the water current. The hair cells along the fish’s lateral line serves as a biological model for hair cells in the human inner ear. Drugs which help recovery to damage hair cells on the zebrafish are further screened using more advanced and expensive animals such as mice.

Tracking

Individual fish tracking may add useful information. This section identifies some of the issues involved with tracking. We will define fish tracking as assigning a label to each fish in the video and retaining the association between the label and the physical fish for the duration of the experiment. This is a different problem than described in chapter 4 where we were concerned with motion and motion estimation. The tracking problem is similar to placing multiple ball bearings into a box, and keeping the correct association of each ball bearing with its corresponding label while the box is being shaken.

There are a few approaches to this problem:

One approach is to put only one fish into each swimming chamber. In this case, the identity of the fish in the swimming chamber is known, and the fish can be tracked. This is the simplest solution and was discussed in chapter 4

Each fish can be uniquely marked. It may be possible to use dyes to put unique markings on each fish, where the dye pattern or color identifies the fish. Optical fluorescent dies have been used to uniquely mark eals with different colors, and
track them when swimming [56, 57], however in our system, the zebrafish must remain in the dark, they are very small and that makes the task of painting unique markings on them difficult. Further the video resolution used to image the fish needs to be sufficient to make unambiguous detections of the fish. The fish present different poses when swimming, and the view of the markings would change over time. In the current system, each fish image is typically less than 100 pixels in size. Organic dyes are available that when illuminated with UV will fluoresce in the near infrared spectrum [58, 59], however zebrafish are sensitive to ultraviolet light (350 NM) [60,61]. The current experimental protocol relies on the fish navigating in dark conditions.

Multiple cameras on orthogonal axes may reduce the problem. Currently our swimming lanes are mounted on a platform with cameras mounted seven inches below the swimming lanes. The apparatus allows a foot of clearance below the swimming lanes for the cameras, cabling and other hardware. The apparatus is sixteen lanes wide. If we allow the same clearance for the cameras taking a side view picture, each swimming lane occupies a space of one foot by one foot, and the overall width of the system increases to sixteen feet. One alternative is to arrange the swimming chambers so one edge of the chamber is in the “downward” direction, forming a “V” shaped trough. Cameras would then be mounted on either side of the swimming chamber. This scheme allows a better view of the fish (the view from the side is larger than the view from the top) and using synchronized video frames from both cameras, allows the fish’s location to be uniquely determined in all three spatial dimensions.

Conclusions

This thesis introduced a scalable system that contains various hardware and software components. Analysis and testing were performed at different levels. This analysis demonstrates that the system performs as well as human observers in a small fraction of time required for determining RI values by hand. In fact, this system expands the testing capacity by over 60 times that of Niihori’s manual analysis.
The network architecture allows a scalable design. Computational resources may be added as necessary to the system to enable more capability without interfering with the function of the current system. The system delivers end-to-end results; operators start a test, and after some time, test analysis results are available on the system’s web site.

Temperature measurements made while the system is in operation (Chapter 2 Cabinets) demonstrate that the system remains within limits while working in the operating environment. Optical considerations minimized distortion artifacts. Simulation demonstrates (Chapter 2 Distortion Caused by Projection to a Plane) that the projection of fish swimming in three dimensions to two dimensions has negligible impact on the relative geometry of the fish. In particular, fish orientation and swimming direction are not adversely affected. Analysis demonstrates acceptable deviation of water flow rates with change of water head height in the reservoir; a one-inch change of water head height results in a one-percent change of water flow rate. Camera lens tests were conducted to determine which lenses to use in the system.

Configuration option settings information are maintained in XML tables. Maintainers modify these tables when the hardware configuration changes, such as when new cabinets are added. Hardware-specific commands are abstracted away from the core system operation; for example, a different PDU manufacturer is supported by adding a script to provide the relevant commands and modify a database entry to point to the new script to use. Provisions exist to use multiple internal computers, if necessary, to support increased network throughput in the event that more than 24 tests need to be in test at the same time.

Control software was written using common tools and languages; programmers are available with the required skills to maintain this system.

**Future Work**

Raspberry Pi hardware continues to improve. The current generation Raspberry Pi 3 processors have ten times the processing capability as the first genera-
tion Raspberry Pi. Currently, the system contains a mix of Raspberry Pi version 1 and 2, the Raspberry Pi is a single core processor, while the Raspberry Pi 2 is a quad core chip. Analysis software could be re-designed to take advantage of the newer generation Raspberry Pis. A set of library functions (picamera [62] ) is available that allows direct camera manipulation from Python scripts and the import of video frames to the ARM processors as video recording takes place. With clever manipulation of the GPU on the Raspberry Pi, it may be possible to perform some or all the detections on the Raspberry Pis themselves. This greatly reduces the data processing workload on the other computers in the system. The number of computations the system needs to perform increases with the number of videos that the system records, which scales with the number of Raspberry Pis within the system. If the Pis have sufficient processing capability to perform the video analysis, then the systems analysis capability becomes independent of external computational resources and network throughput.

The lighting geometry will need to be improved if effective tracking is to be implemented. We cannot currently resolve multiple fish in the corners of the swimming chambers. Currently, there are three alternatives to improve the chances of successfully tracking fish. The first alternative is to use some form of spot lighting to better illuminate the fish in the corners of the swimming chambers. Multiple fish can “stack up” in the corners, and the current swimming chamber and camera configuration cannot resolve this challenge. A second alternative is to possibly change the geometry of the swimming chambers to minimize the shadowing effects of the walls and remove locations where the fish merge in the corners of the swimming chambers. A third solution is to develop and use better motion tracking software to compensate for the wall artifacts. One idea is to shape the swimming chamber so that it matches the viewing frustum of the camera. However, it will be difficult to maintain the required mechanical tolerances. The current (tested) motion software can be extended to incorporate a realistic model of fish swimming behavior in order to enhance the ability to track fish even when they swim in shadowed areas or near walls. There are known restrictions on the fish behavior: they cannot
swim through the sides of the tank, they congregate in the downstream area of the swimming chambers, and sometimes they stick themselves to the surfaces of the swimming lanes. These constraints can be used to build a state model that influences the distribution model of, say, a particle filter [53] to better estimate the future location of the fish. It is likely that all three approaches will increase the ability for the system to track fish.

Currently, the system projects the fish to a two-dimensional surface. Multiple cameras per swimming chamber would allow depth information to be collected. One option is to use a lensing arrangement that presents two different views of the swimming chamber to the same camera, and a second option is to use a Raspberry Pi that supports multiple cameras, such as the Raspberry Pi compute module [63]. There may be other low-cost boards that will synchronously support the video acquisition of multiple cameras. One problem to overcome is keeping the frames from both cameras synchronized. Through novel solutions to such problems, more precise tracking of zebrafish can be achieved, which may open up new avenues to use behavioral analysis to test the efficacy of new drug therapies.
REFERENCES


