Development of an Autonomous Robotic Air Hockey Player

by

Bee Vang

Submitted to the
Department of Mechanical Engineering
in Partial Fulfillment of the Requirements for the Degree of

Bachelor of Science in Mechanical Engineering

at the

Massachusetts Institute of Technology

June 2013

Signature of Author: _____
Department of Mechanical Engineering
May 10, 2013

Certified by: _____
Karl D. Iagnemma
Principal Investigator of The Robotic Mobility Group
Thesis Supervisor

Accepted by: _____
Anette Hosoi
Professor of Mechanical Engineering
Undergraduate Officer

(this page intentionally left blank)

Development of an Autonomous Robotic Air Hockey Player

by

Bee Vang

Submitted to the Department of Mechanical Engineering
on May 10, 2013 in Partial Fulfillment of the
Requirements for the Degree of

Bachelor of Science in Mechanical Engineering

ABSTRACT

Air hockey is a widely played sport in the United States and is expanding to other countries. It is usually played on a frictionless table between two or more people. The dependency of another individual makes it impossible for a person to practice or play alone.

An autonomous two-link manipulator was designed and developed to allow a person to individually play air hockey. The two-link manipulator consisted of a mechanical, electrical, and software system. The mechanical system was designed to actuate the two-link manipulator through geared motors and a belt transmission while being self-contained. The electrical system was designed to provide feedback of the physical system and motor control capabilities. The software system was designed to integrate the information from the mechanical and electrical systems to create the controls and behavior of the mechanical arm.

The autonomous two-link manipulator proved to be a robust design. It had a 68.5 percent hit rate, suggesting that the design will be capable of playing air hockey.

Thesis Supervisor: Karl D. Iagnemma
Title: Principal Investigator of The Robotic Mobility Group

(this page intentionally left blank)

Acknowledgements

I would like to thank Professor Karl Iagnemma for giving me the opportunity to explore my interest in robotics. With his guidance and support, I was able to lead the development of my project from conception to a working prototype.

(this page intentionally left blank)

Contents

List of Figures

(this page intentionally left blank)

Chapter 1

Introduction

Air hockey is a widely played sport in the United States and is expanding to other countries. It first became popular in the 1970s where it could be found in arcade rooms [1], but has since found a common place in personal entertainment rooms and competitive tournaments. Air hockey is played on a frictionless table (usually a thin sheet of air) with side railings to prevent the puck and the mallet from exiting the area of play. There are many variation to the rule of play, but the game is simple. There are two players on opposing ends of the table who try to strike a puck with a mallet into the opponent's goal. The game of air hockey has not experienced much change since its birth. One of the difficulties of playing air hockey is the dependency of another player. A robotic arm was developed to address this issue. This thesis discusses and evaluates the development of an autonomous robotic arm to allow an individual to independently play air hockey.

The arm's design and development was heavily influenced by a Do-It-Yourself mentality. The overall cost of the arm and electrical components, excluding the computer, was $469.53. The end goal was to design an arm that could be easily reproduced by a person with minimal mechanical engineering background and workshop experience. It also had to be affordable, autonomous, self-contained, and played with realistic force and movements. A two-link manipulator with a webcam visual system was designed to address the problem with these constraints in mind. The two-link manipulator consisted of three individual systems: mechanical, electrical, and software. The mechanical system was designed to actuate the two-link manipulator through geared motors and a belt transmission while being self-contained. The electrical system was designed to provide feedback of the physical system and motor control capabilities. The software system was designed to integrate the information from the mechanical and electrical systems to create the controls and behavior of the mechanical arm.

Chapter 2

Mechanical System Design

The mechanical system was the starting point in the design process of the robotic arm. The placement and selection of the actuators and links determined the necessary components in the electrical and software systems. The mechanical system was designed to house all the electrical and mechanical components in one transferable unit. This allowed the arm to be moved easily from one table to another. In the design of the mechanical system, many constraints had to be defined in order to create a robotic arm capable of playing air hockey with the speed and force of a person.

2.1 Design Constraints

The robotic arm was designed with consideration of cost, end-effector speed and torque, adaptability to different table sizes, ability to reach the area of play, and self-containment. Although there was no set cost, the arm was designed to be low cost and easy to reproduce for people who already owned an air hockey table. This design constraint limited the types of motors and electronics that could be used in this project. In order to create an arm that was capable of playing on various table sizes, the arm needed a versatile mount. This led to a design that used clamps to hold the arm onto the table, shown in Figure 2.1. The clamps hooked onto the table with a C-structure and was secured to the air hockey table with an adjustable screw.

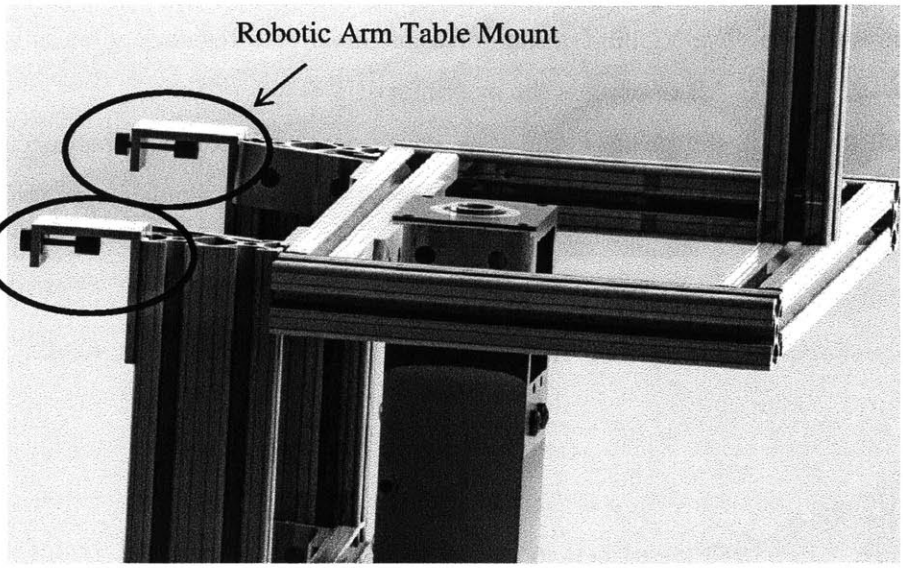

Figure 2.1: The C-structure of the Robotic Arm Table Mount allowed the adjustable screws, shown in red, to secure the mount to the table.

The most deterministic constraint was the end-effector speed and torque. These constraints determined the arm's movement speed and force, the motors, the motor controllers and the design of the mechanical structure. A Vicon Infrared Camera system was used to gather the necessary data to estimate the average human hand speed and force being applied to a puck on impact. The implementation of these design constraints kept the arm's movements within that of an average person.

2.1.1 Vicon Infrared Camera System

A Vicon Infrared (IR) Camera System was used to gather the position data of the puck and mallet to determine the respective average speed and acceleration during play. This information was used to determine the required end-effector speed and torque to simulate a person playing air hockey. The physical implementation of these design constraints was in the choice of the joint motors.

The Vicon IR system consisted of six individual IR cameras mounted to observe a fixed area. The cameras used IR reflectors to determine the location of objects in three-dimensional space with millimeter precision. The position data was then uploaded to a laptop running the

Vicon Tracker software. The Vicon Tracker software forwarded this data wirelessly to another computer that used MATLAB to process the incoming data.

MATLAB processed the position and time data from the Vicon cameras to calculate the velocity and acceleration of the puck and mallet. The velocity was calculated using the forward difference, backward difference and centered difference methods, expressed by equation 2.1, 2.2, and 2.3 respectively [2]. In the equations, x is the position data, $\dot{x}$ is the calculated velocity, h is the time step between each data point, and i is the location of the current data point.

$$\dot{x} = \frac{1}{h} * (x_{i+1} - x_i) \tag{2.1}$$

$$\dot{x} = \frac{1}{h} * (x_i - x_{i-1}) \tag{2.2}$$

$$\dot{x} = \frac{1}{2h} * (x_{i+1} - x_{i-1}) \tag{2.3}$$

The forward and backward difference are first order approximation, while the center difference is second order. This means that the center difference is typically a more accurate approximation. However, the center difference could not be used at the beginning and end of each data set, because it would require additional data. In these situations, the forward and backward difference was used.

A second order center difference calculation was used to determine the acceleration of the puck and mallet. The second order center difference equation is given by [2]:

$$\dot{x} = \frac{1}{h^2} * (x_{i-1} - 2x_i + x_{i+1}) \tag{2.4}$$

Instead of starting and stopping at the first and last data point, this calculation started at the second and stopped at the penultimate data point to avoid the same issue in the velocity calculations.

2.1.2 Motor Torque and Speed Requirements

The calculated average speed and acceleration from MATLAB was used to select appropriate motors. Figure 2.2, 2.3, 2.4, and 2.5 shows the resulting histogram of the average speed and acceleration of the puck and mallet. This data was gathered from a person playing air hockey on a goal-less table in which the puck could not leave the area of play. The puck

experienced an acceleration of less than 60 m/s^2 for approximately 80 percent of the time and a velocity of less than 3 m/s for 90 percent of the time. The mallet experienced similar results for the acceleration, but its velocity was below 2 m/s for 90 percent of the time. From the data, it was determined that the maximum end-effector speed needed to be 3 m/s and the acceleration needed to be 60 m/s^2.

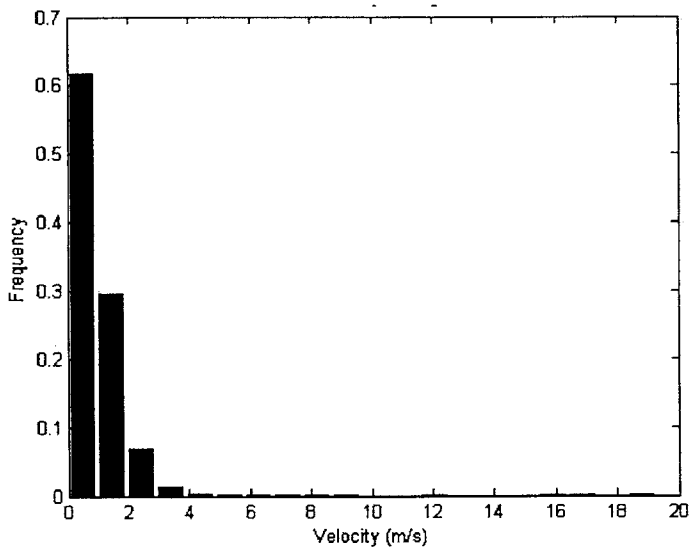

Figure 2.2: A histogram of the average puck velocity collected from a Vicon IR Camera system.

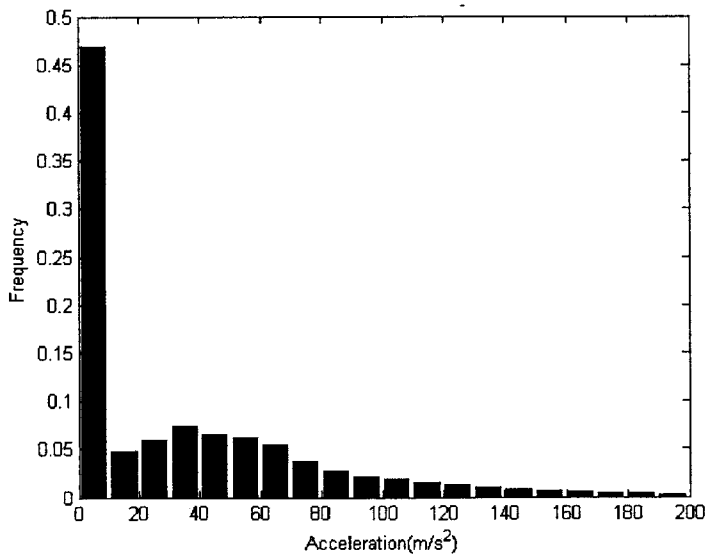

Figure 2.3: A histogram of the average puck acceleration collected from a Vicon IR Camera system.

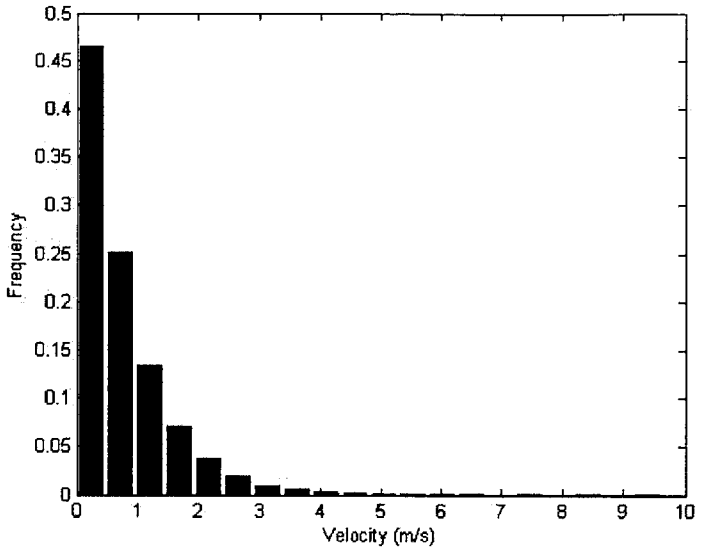

Figure 2.4: A histogram of the average mallet velocity collected from a Vicon IR Camera system.

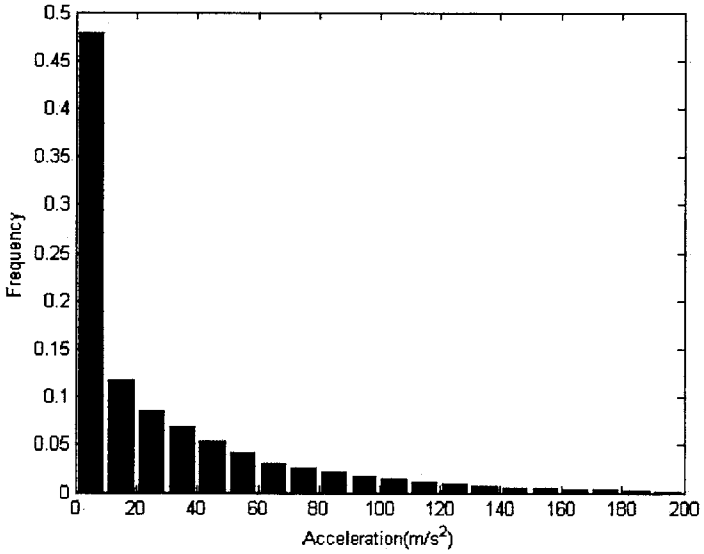

Figure 2.5: A histogram of the average mallet acceleration collected from a Vicon IR Camera system.

2.2 Two-Link Manipulator

A two-link rotational joint manipulator was chosen as the ideal candidate, because it fulfilled all the design requirements. It was selected over a two-axis prismatic design and a slide-and-piston design. The two-link manipulator can transverse over the area, A, as a function of the length of link one, l_1, and link two, l_2.

$$A = \pi * ((l_1 + l_2)^2 - (l_1 - l_2)^2) \tag{2.5}$$

It required less infrastructure, utilized rotational motors, and has well understood dynamics and controls. It also resembled a human arm, making it easier for people to grasp its purpose.

The dynamics of a two-link manipulator is expressed by equation 2.6. This equation does not include gravity, because the arm is mounted on a horizontal plane perpendicular to gravity. The torques on the motor, τ, depends on the inertia matrix, H, the coriolis and centripetal torque matrix, C, the viscous dissipation matrix, D, the angular acceleration, $\ddot{q}$, and the angular velocity, $\dot{q}$.

$$H\ddot{q} + C\dot{q} + D\dot{q} = \tau \tag{2.6}$$

2.3 Motor and Gearhead Selection

To actuate the two-link manipulator, direct current (DC) motors and proper gearheads were chosen according to the design requirements for speed and torque. DC motors were chosen, because they are generally cheaper than both servo and stepper motors with the same motor specifications. The appropriate motor for the shoulder joint needed to be able to rotate the fully extended arm at 3.0 radian/s with an angular acceleration of 60.6 radian/s^2. The motor requirements for the distal joint only needed to be able to rotate the distal arm at 6.8 radian/s with an angular acceleration of 136.9 radian/s^2. If both of these conditions are met, the maximum desired end-effector speed and angular acceleration will also be met.

The torque-speed curve of a motor needed to be analyzed to determine if that motor met the required specifications. The torque-speed curve shows the maximum power, the stall torque and the no-load speed. The maximum power is when the motor is operating at half the no-load speed

and half the stall torque. The no-load speed is the maximum rotational speed of the motor and the stall torque is the maximum output torque of the motor. If the maximum speed and torque requirement of the joint motors were below the torque-speed curve of a specific motor, that motor will be able to actuate the arm. It is better to have the design requirements fall closer to the no-load speed, because the motor will be more efficient and will operate for longer periods of time without damage [3].

A RS-550 motor with a back-shaft and a P60 Gearbox with a gear ratio of 81:1 was chosen as the shoulder joint actuator. The torque speed curve of this motor and gear head combination is shown in Figure 2.6. The power requirement for this joint motor was below the torque-speed curve of the motor and gear head combination. This indicated that the motor was sufficient to actuate the arm. If the motor was to operate at the maximum required torque, it will have a very low efficiency, because the majority of the input power will be converted to heat. However, the motor and gearhead combination was adequate in driving the arm, because the motor only needed to operate for short amount of time between each strike.

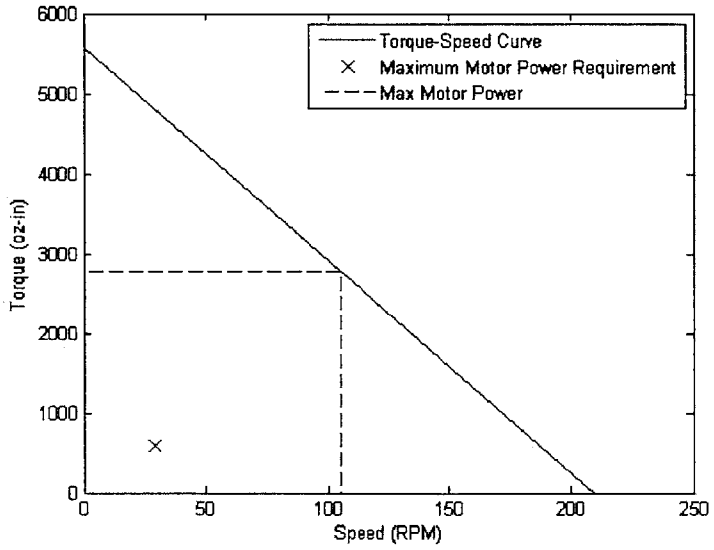

Figure 2.6: The torque-speed curve of the shoulder motor at 12V. The required maximum power output is well below the motor's torque-speed curve. This indicated that the motor and gear head combination was sufficient to drive the arm at the required end-effector speed and torque.

A geared motor from Pololu was chosen to drive the distal arm. It has a stall torque of 200 oz-in and a no-load speed of 150 rpm. The speed and torque requirement for the distal arm motor

was 65.4 rpm and 85.7 oz-in. The requirements fell close to the maximum power of the chosen motor. This meant that the motor could not operate at the required end-effector speed and torque for an extended period of time. The torque-speed curve of the distal motor is shown in Figure 2.7.

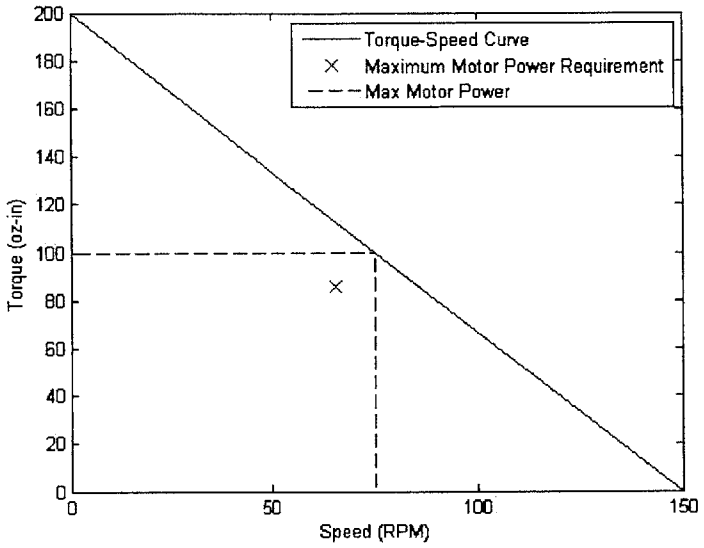

Figure 2.7: The torque-speed curve of the distal motor at 12V. The power requirement for this joint motor was close to the maximum power output of the chosen motor. This indicated that the motor would not be able to operate at the maximum end-effector speed and torque for a prolong period of time.

2.4 Kinematics and Controls

The kinematics of the arm was used to control the arm's movement to a desired location. This project did not consider the system's dynamics in its control scheme to keep the controller simple. The controller was designed to move the end-effector to a desired position through the control of the joint angles.

2.4.1 System Kinematics and Analysis

The mechanical system consisted of two links with two rotational joints. The system moved in a two-dimensional plane on the air hockey table. Figure 2.8 shows the arm's configuration along with the joint angles, q_1 and q_2, and the length of the links, l_1 and l_2. The joint angle q_1 was measured with respect to the x-axis and the joint angle q_2 was measured with respect to q_1.

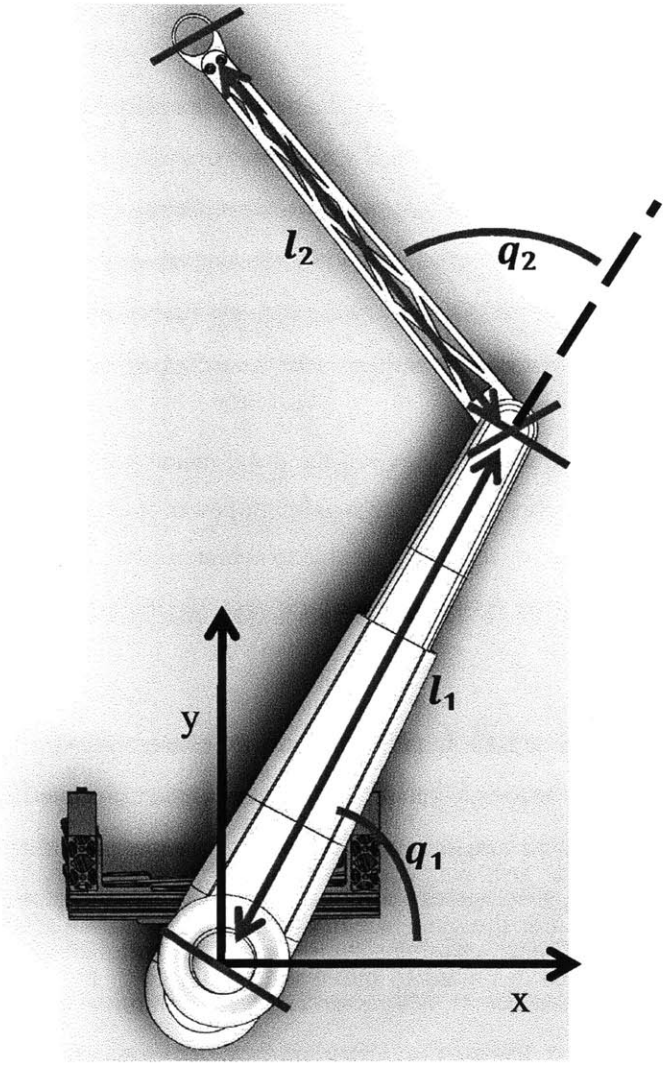

Figure 2.8: System diagram of the two-link manipulator with the origin at the shoulder joint. Angle q_1 was measured with respect to the x-axis and q_2 was measured with respect to q_1.

The forward kinematic equation for the end-effector is given in equation 2.7 where the shoulder joint is the origin for the position plane. A Proportional-Derivative position controller was chosen over a more complex controller that also considered the dynamics to reduce the implementation and testing time

$$\begin{aligned} x &= l_1 \cos q_1 + l_2 \cos(q_1 + q_2) \\ y &= l_1 \sin q_1 + l_2 \sin(q_1 + q_2) \end{aligned} \tag{2.7}$$

Inverse kinematics was used to determine the joint angles for any desired position. The control objective of this project was to move the arm to the desired joint angles. Equation 2.8 gives the inverse kinematic equation for a desired position (x_d, y_d). These equation shows that there are two possible solution for any given position: elbow up or elbow down.

$$\begin{aligned} q_1 &= \tan^{-1}\left(\frac{y_d}{x_d}\right) - \cos^{-1}\left(\frac{{x_d}^2 + {y_d}^2 + {l_1}^2 - {l_2}^2}{2l_1\sqrt{{x_d}^2 + {y_d}^2}}\right) \\ q_2 &= \pi - \cos^{-1}\left(\frac{{l_1}^2 + {l_2}^2 - {x_d}^2 - {y_d}^2}{2l_1 l_2}\right) \end{aligned} \tag{2.8}$$

In order to resolve this issue the arm was limited to a right-handed orientation. This was accomplished by allowing the computer equation solver to use the first solution for q_1 and q_2 when x_d was positive. When x_d was negative, q_1 was adjusted by equation 2.9. This proved to be robust enough to control and determine the position of the arm's end-effector with a right-handed orientation.

$$q_1 = \pi - \tan^{-1}\left(\frac{y_d}{x_d}\right) - \cos^{-1}\left(\frac{{x_d}^2 + {y_d}^2 + {l_1}^2 - {l_2}^2}{2l_1\sqrt{{x_d}^2 + {y_d}^2}}\right) \tag{2.9}$$

2.4.2 Proportional-Derivative Controller

A Proportional-Derivative (PD) controller was designed to move the end-effector from its current position to a desired position. The PD controller factored in the distance from the current position, (x, y), to the desired position, (x_d, y_d), and the velocity, $\dot{x}$, of each joint. The controller is given in equation 2.10. It used a constant gain factor, K_p and K_d, to scale the motor output. This controller design simulated a spring and dashpot behavior between the current location and the desired location. The controller was implemented through the software system.

$$output\ signal = -K_p\left(\sqrt{(x-x_d)^2+(y-y_d)^2}\right) - K_d\dot{x} \tag{2.10}$$

The gains were determined experimentally. The arm was tested with many gain factors until the performance and motion of the arm was deemed stable and satisfactory. It was not possible to use a root locus to determine the best gains, because there was no direct conversion from pulse-width-modulation (PWM) to motor power. The PWM signals will be discussed in chapter three. The PD controller's performance is shown in Figure 2.9. The simulations suggested that the arm will have a settling time of less than one second. The actual setting time of the system was somewhere between one and two seconds depending on the distance between the current and desired position.

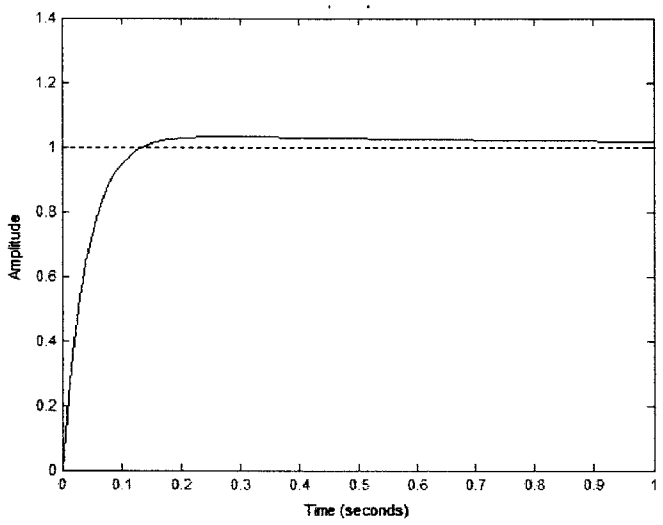

Figure 2.9: The settling time of the system with a step input. The simulations suggested that the arm will move to the desired location within one second.

2.5 Complete Mechanical System

The mechanical system served as the structure of the robotic arm. A two-link manipulator was designed to strike a moving puck with comparable speed and force to that of a person. The shoulder motor was connected directly to the first link. The distal motor drove the distal arm through a belt and pulley system. The torque created by the distal motor's mass was eliminated by placing the mass of the distal motor on the axis of rotation of the whole arm. This design

choice reduced the required motor torque of the shoulder joint motor and lessen the overall cost. The complete mechanical system is shown in Figure 2.10.

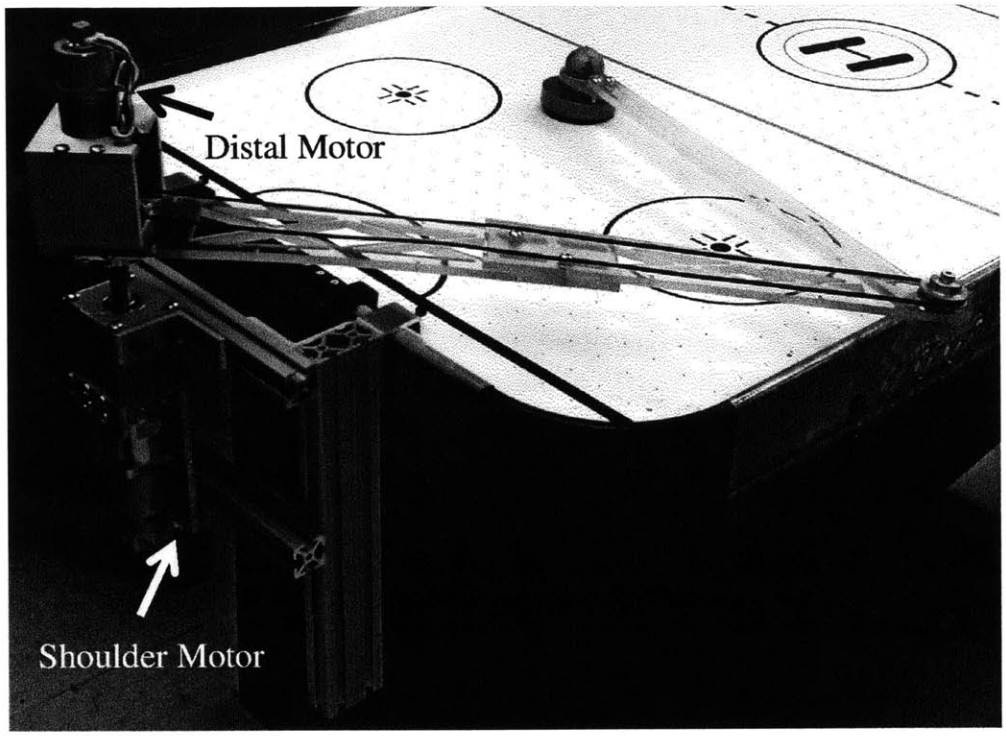

Figure 2.10: The complete mechanical system; a two-link manipulator that was driven by two rotational motors. The shoulder joint was directly connected to the first link while the distal motor drove the distal link through a belt and pulley. This design tried to place all the mass of the system on the axis of rotation at the shoulder joint.

A bearing support structure was also designed to protect the motors from backlash and bending moments. Each motor shaft was connected to a drive shaft by a shaft coupler and then supported at two points by ball bearings. The bearing structure is shown in Figure 2.11.

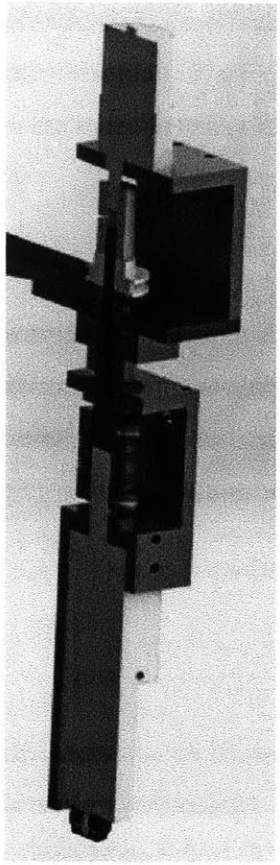

Figure 2.11: The bearing support structure for the joint motors. The motors are shown in teal, the shaft couplers are shown in green, the bearings are shown in red, and the drive shafts are shown in blue and black. The bearing support structure prevented the motor shaft from experiencing backlash and bending moments.

Chapter 3

Electrical System Design

The electrical system integrated the controls and sensing aspects of the arm. It took in data via encoders and a webcam and output the motor commands to an arduino and motor controller. The electrical system allowed the arm to gather the data from the physical world and used it to determine the movement of the arm.

3.1 Motor Controllers

The direction of the current and the voltage determines the speed and direction of the motor. Special circuit boards can be used to manipulate these properties to control the motors. In this case two circuit boards were used for each motor: a microcontroller and a motor controller. The motor controller connected a power supply to the motor and provided the circuit necessary to manipulate the voltage and current. The microcontroller was used to transmit control signals from a computer to the motor controller.

3.1.1 Motor Controller Selection

The motor controller is a circuit board designed to control the speed and direction of motors. Important factors to keep in mind are: operational current, maximum current, voltage, and control signal. It is important to select a motor controller that can operate above the motor specifications so that the motor controller does not get damaged.

The SyRen 50A Regenerative Motor Controller, shown in Figure 3.1, was chosen to operate the shoulder motor. This motor controller can handle a continuous 50Amp current, a peak current of 100Amps, a 12 Volt input, and a PWM control signal. The shoulder motor operates at 12

Volts with a stall current of 85 Amps. From Figure 2.6, it can be inferred that the maximum operational current of the shoulder motor will not exceed 50Amps. Therefore, the SyRen 50A motor control was a suitable motor controller for the shoulder motor.

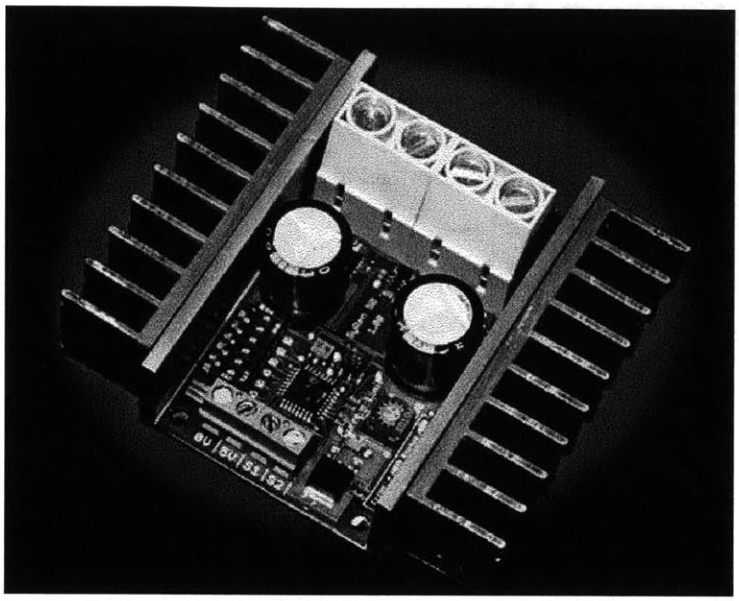

Figure 3.1: The SyRen 50A Regenerative Motor Controller was selected to control the shoulder joint.[1]

The SyRen 10A Regenerative Single Channel Motor Controller, shown in Figure 3.2, was chosen to drive the distal joint. The distal motor operates at 12 Volts with a stall current of 5 Amps. The motor controller can handle a continuous 10 Amp current, a peak current of 15 Amps, a 12 Volt input, and a PWM control signal. The maximum current draw from the distal motor will not exceed the peak current. From Figure 2.7, the maximum operational torque is near the maximum motor power, therefore the motor will not experience a current draw close to the stall current. As a result, the SyRen 10A was a suitable choice for the distal motor.

[1] Image courtesy of www.robotmarktplace.com

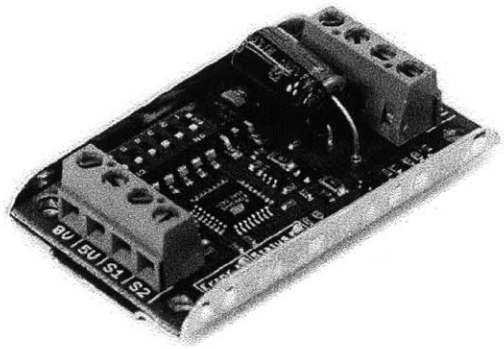

Figure 3.2: The SyRen 10A Regenerative Single Channel Motor Controller was selected to control the distal motor.[2]

A microcontroller was used to control the motor controller with PWM signals. The microcontroller received its information from a computer and output that command as a PWM signal to the motor controller.

3.1.2 Arduino Microcontroller

A microcontroller is a compact integrated computer. The microcontroller board has everything it needs in order to run calculation and execute computer codes. It has a processor, memory, and digital/analog inputs and outputs. The Arduino Uno microcontroller was chosen for this project, because it was inexpensive and has PWM output pins, a strong support community, and C language based integrated development environment (IDE).

The microcontroller acted as a relying agent for the motor controller. It received data from a computer, process it, and sent the motor controller a PWM signal. The Arduino used the code in Appendix C to actuate the motors.

3.2 Encoders

Quadrature encoders were used to measure the joint angles. The encoders had a spinning disc with a sensor that recorded "ticks" or markers on the spinning disc. Quadrature encoders

[2] Image courtesy of www.lynxmotion.com

output two signals that can be processed to determine the number of ticks and motor direction. Figure 3.3 shows an example of the encoder output. The order of the output signals indicates direction and the frequency of the signals indicates speed [4].

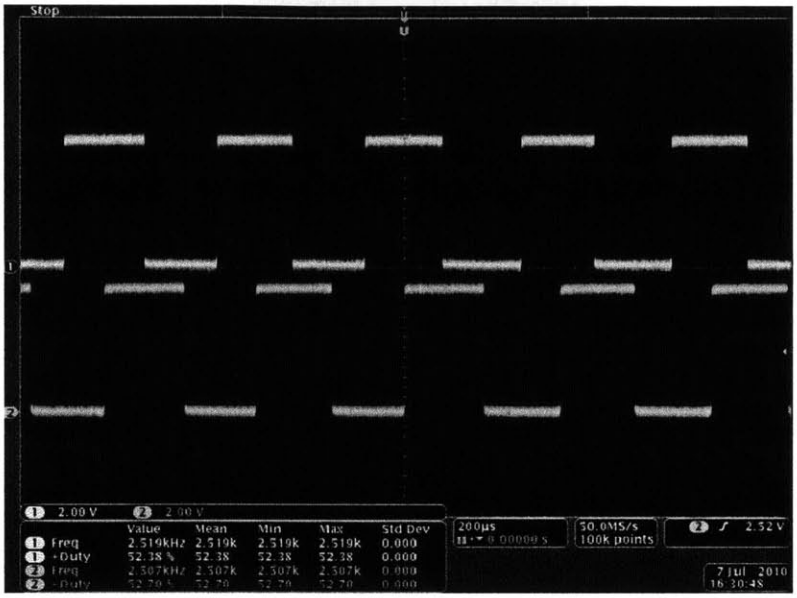

Figure 3.3: A sample output of a quadrature encoder. The order of the two signals indicates direction and the frequency of the signals indicates speed [4].[3]

The joint angles were used to calculate the position of the end-effector and determined the respective joint angular velocities. These measurements were important in determining the control signals for the PD controller. There were some calibration issues with the encoder that affect the accuracy of the end-effector. The most notable issues were the rotation of the motor shafts when the end-effector was physically constrained from moving and the backlash of the motor shafts. Both of these issues will be discussed later.

3.2.1 Encoder Selection

An E4P OEM Miniature Optical encoder with 100 counts per revolution was chosen for the shoulder motor. With the gear ratio, this encoder measured 8100 counts per revolution of the motor shaft. The distal arm had a built-in encoder with 64 counts per revolution. With the gearhead, this encoder measured 4331 counts per revolution of the motor shaft. The counts per

[3] Image courtesy of www.pololu.com

revolution was used to calculate the angle change over time for each joint. The joint angles were calculated using equation 3.1. This equation states that the current joint angle is equal to the current joint angle plus the amount of ticks over a time period, *ticks*, multiplied by a tick to angle conversion factor. The *ticks* also took the direction of rotation into consideration.

$$q_1 = q_1 + ticks * \left(\frac{2\pi}{8100}\right)$$
$$q_2 = q_2 + ticks * \left(\frac{2\pi}{4331}\right) \quad (3.1)$$

3.2.2 Phidgets Highspeed Encoder Board

The PhidgetEncoder HighSpeed 4-Input board was selected to interpret the output signals of the encoders. The encoder board utilizes a hardware and software interface to allow a computer to read important information from the encoders via usb. The board takes in the encoder signals and process it into relative position change, incremental position, and time data. The encoder information is called upon by the software system to determine the location of the end-effector and the arm's joint angular velocities.

3.3 Visual System

For the two-link manipulator to be completely autonomous, it needed to be able to detect the location of the puck in relation to the end-effector. This was accomplished through the use of a webcam. The webcam was mounted above the manipulator with a field of view of the table surface. A software system was used to process these images to determine the puck's speed and position. The puck's speed and position allowed the arm to predict where the puck will be and how to best reach it. A webcam was chosen because it was affordable and accessible.

3.4 Electrical System Layout

The overall goal of the electrical system was to provide input and output streams for the software system so that it could control the mechanical arm using real physical parameters. Figure 3.4 shows a complete diagram of the electrical system.

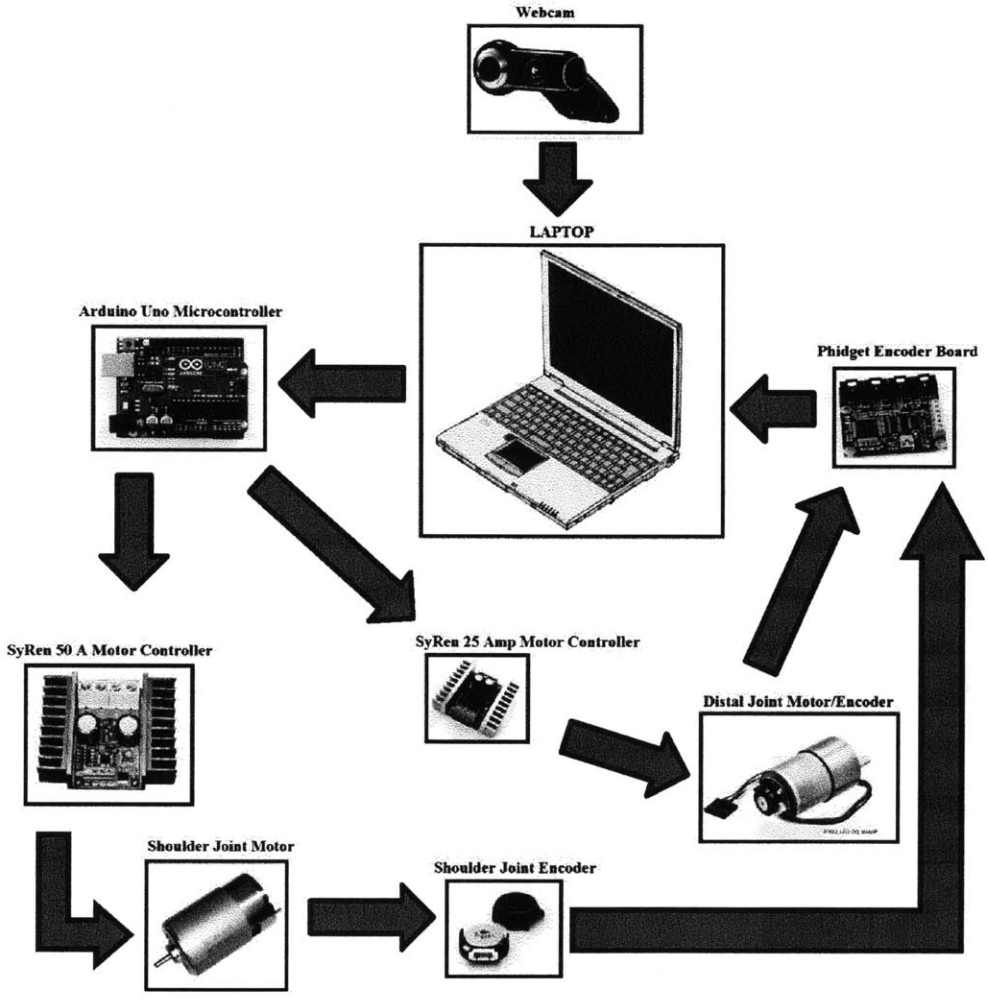

Figure 3.4: A complete electrical system diagram. The blue arrow shows the flow of data. The computer took in input from the Phidgets encoder board and the webcam and then output motor commands to the Arduino microcontroller.

The subsystems interacted together to allow the arm to move, track the puck, and determine the location of the end-effector. The central part of the electrical system was the computer. The encoders were connected to the Phidgets encoder board which was connected to the laptop. The

motors were connected to the motor controllers, then to the arduino microcontroller and then to the laptop. The arrangement of the subsystems created a closed-loop feedback of the whole system.

Chapter 4

Software System Design

The software system acted as the brain of the robotic arm. It gathered data from the electrical components and decided how to best actuate the mechanical system. The software system was composed of many separate libraries that functioned together to allow the computer to process the input data and output the control command in real time. The integration of the software system allowed the arm to be completely autonomous.

4.1 Boost C++ Libraries and Multi-Threading

The most important subsystem of the software system was the ability to multi-task or multi-thread. This allowed the computer to process many data input streams, calculations, and output streams in real time. Without this library, the computer would only be able to execute one process at a time which would greatly increase the computation time and result in a slower arm reaction.

Each of the subsystem in the software system existed on a separate thread, and in some cases, created their own sub-threads. The special thing about the boost library is that it automatically utilizes the multiple cores in a computer's processor. Once the subsystems use all the process capability of one core, boost automatically delegate task to the other cores to allow for continuous data processing and to reduce the delay time between data input and output.

4.2 OpenCV and Visual System

The visual processing subsystem was built with the OpenCV libraries. OpenCV is an open source, multi-platform software that implements many of the standard image processing techniques. It is used widely in computer vision applications.

The main OpenCV functions used in this project were: color space transformation, color detection, image threshold, and contour detection. The color transformation converts an image from the standard Red-Green-Blue (RGB) color space to the Hue-Saturation-Value (HSV) color space. The HSV color space was beneficial because it arranges the color in a way that makes it easier to specify a range of similar colors by only varying one value. The difference between the RGB and HSV color space is shown in Figure 4.1. The hue determines the color, for example red, blue, green, yellow. The saturation level defines the intensity of the color. The value represents the brightness of the color. Both the saturation and value are affected by adding black or white to the hue.

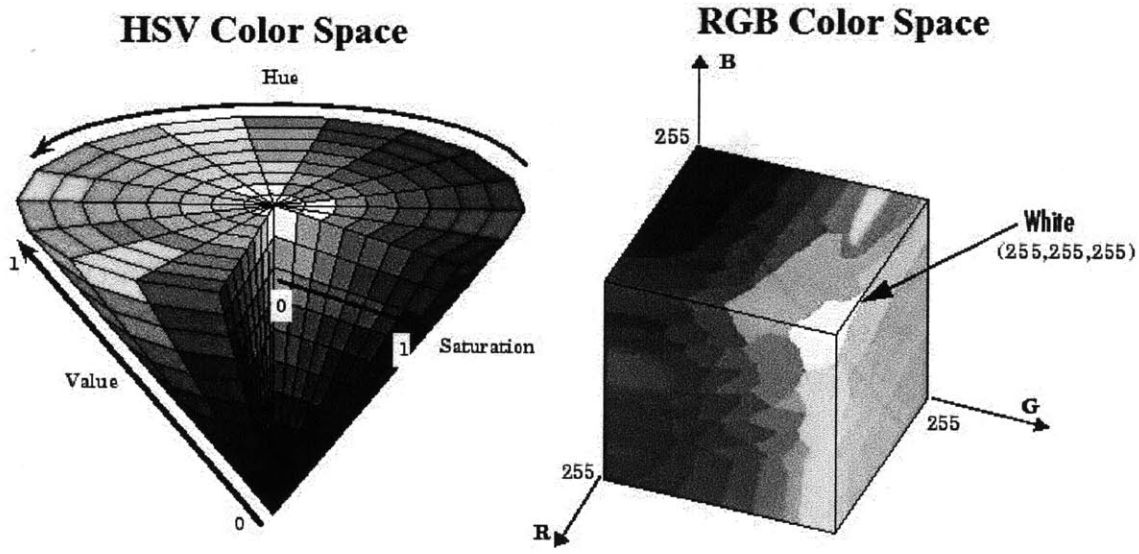

Figure 4.1: The HSV color space uses Hue-Saturation-Value to determine a color, while the RGB color space uses Red-Green-Blue. The HSV color space arrange the colors in a way that makes it easier to select a gradient of one color.[4]

[4] Image used courtesy of www.mathworks.com

The color detection algorithm was used to process the image to look for the red puck in the HSV color space. The image threshold process was used in conjunction with the color detection algorithm to create a black and white image. The white represented areas on the image where the color of interest was found. The black represented everything else. By creating this black and white image, the computer can easily determine where a color is on the image.

The contour detection process took the threshold image and found the boundary lines of the white spaces. Then a contour area of each contour was calculated and compared so that only the largest white space was taken into consideration by the software system. This proved to be robust enough to locate the red puck on the air hockey table through all trials. A sample output of the vision system is shown in Figure 4.2.

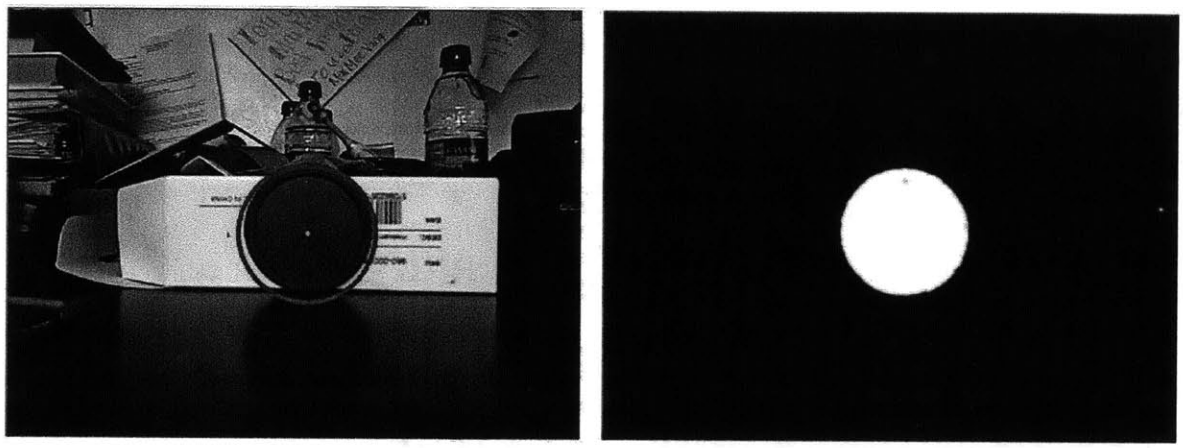

Figure 4.2: The image on the left was captured by a webcam. The software system used the OpenCV algorithms to locate the red puck and to create a threshold image, shown on right. It then drew a circle on the captured image to make it easier for the user to see which object the software system was tracking.

4.3 Phidgets Libraries and Encoders

The Phidgets libraries were used to retrieve the encoder data. The Phidgets encoder board requires this certain library to communicate with the computer. The Phidgets board took the encoder signals and created increment position, relative position, and time data. The Phidgets libraries also created multi-threads in the software system to listen for changes in the encoders

and to notify the software system. This allowed for the real time detection of the end-effector location.

4.4 Serial Communication and Arduino

The arduino requires a serial communication to exchange data with the computer. The serial library from www.teuniz.net was selected because it was built for C++ and the call functions were easy to comprehend. The serial communication link was created on a separate thread that allowed the arduino and computer to communicate without interruptions. The serial communication line can only transmit one byte at a time. This meant that only one character could be sent at a time. For example, if the computer wanted to send "1500" to the arduino, it would have to break "1500" into four parts: "1", "5", "0", "0". On the arduino end, it would receive these four bytes, but it would not know that the data should be "1500".

Because of this, a code was implemented on the arduino to sum the data together. In order to distinguish between the two motors, a motor code was assigned: "a" for the shoulder motor and "b" for the distal motor. Also, an end signal, "\n" indicating an empty space in characters, was added to tell the arduino that the computer was done sending data for a specific motor. With this system, if the computer wanted to send "1500" to the shoulder motor, it would send "a", "1", "5", "0", "0", "\n". The arduino would read the "a" and assign the resulting value to the shoulder motor. The complete code and byte calculation is shown in Appendix C.

4.5 Graphic User Interface and Gtkmm

A Graphic User Interface (GUI) is a visual interface that allows a user to interact with the software system. The same user interface could have been created with only text inputs, but a GUI was determined to be more effective and easier to use. The GUI allowed for the control of the arm in manual and autonomous modes and for the input of physical constraints. In the manual mode, the user could control the motors through the use of the keyboard or text inputs

into the GUI. The autonomous mode disabled the user's control and placed it in the software. This mode allowed the arm to autonomously play air hockey.

C++ does not come with its own GUI library. GTKmm was selected as an open source GUI library, because it is cross platform, uses the C++ language, has a GUI builder, and has many built-in functionality that make the GUI run effectively. GTKmm was used to create the GUI, along with the interactive buttons, text inputs, and labels. This created an informative, interactive display for the user in the operation of the arm. The air hockey GUI is shown in Figure 4.3.

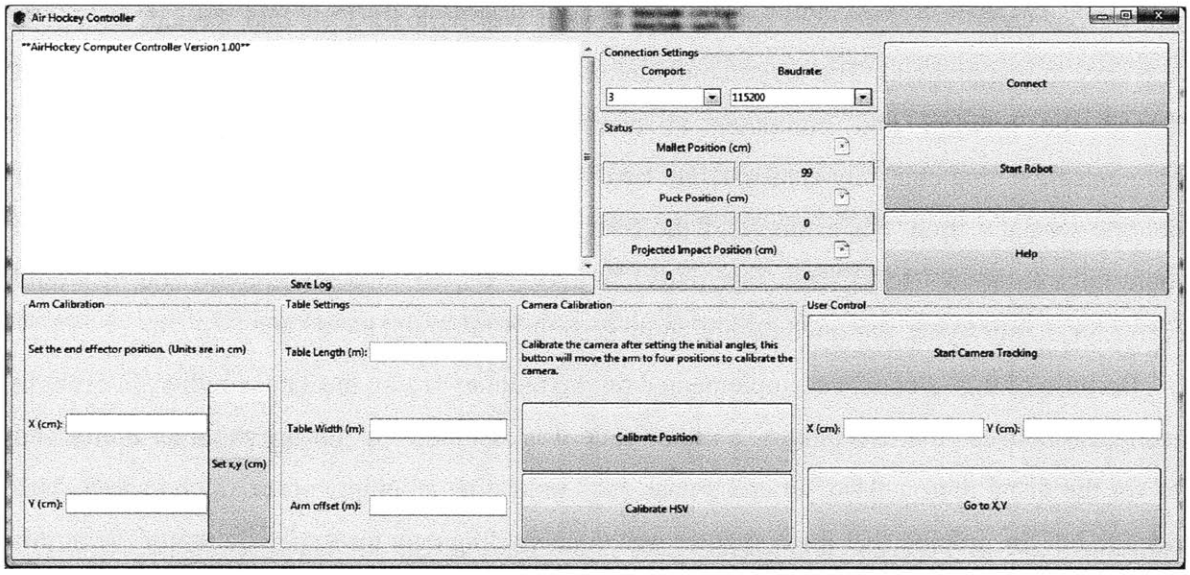

Figure 4.3: A Graphic User Interface built from the GTKmm libraries. The GUI consisted of buttons, text inputs, and display labels. The GUI allowed for the control of the robotic arm.

4.6 Arm Behavior and Controls

The arm's behavior and control thread took in the camera data and the end-effector location to determine a PD control output. The arm was programmed to strike the puck if it was moving toward the arm and in an area specified by the software system. If there is no puck or the puck is moving away from the arm, the arm would go to a home position and wait. The area was designated to prevent the arm from hitting the edge or getting stuck in its own goal. When the end-effector hit the edge or was caught in its own goal, the encoders reported change in joint

angles when, in actuality, the end-effector was not moving. The end-effector was physically constrained by the wall or goal from moving outside the play area, but the motor shaft kept spinning and caused the shaft couplers or timing belt to slip. This caused the software system to lose track of the real position of the end-effector and prevented the arm from hitting the puck.

4.7 Software System Layout

The layout of the software system is shown in Figure 4.4. The main thread was the GUI window. From the main window, other threads were created to handle the various task of the software system.

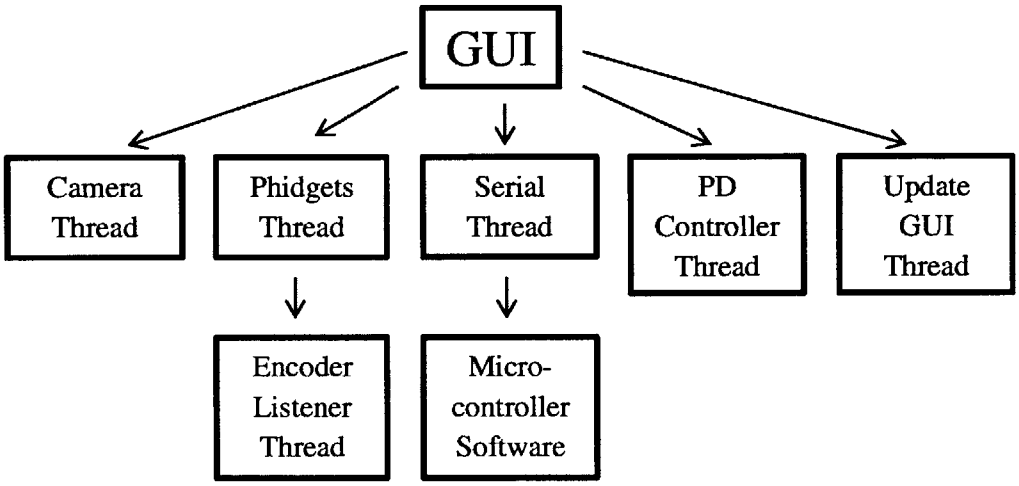

Figure 4.4: The layout of the software system. The main thread was the GUI window. Subsequent threads were created to break up the various task in the software system. The most important threads are shown in the figure.

The camera thread initialized the webcam to capture a live stream at 30 frames per second of the air hockey table and puck. It used the OpenCV libraries to process the images and to determine the puck's location. The position data was passed to a global variable that was shared among all the threads. The Phidgets thread initialized the PhidgetEndcoder HighSpeed board and created additional sub-threads to listen for changes in the encoders and to notify the GUI. The serial thread opened a communication

link via usb between the computer and microcontroller. It was used to pass control signals from the GUI and PD controller thread to the motor controllers. The PD controller thread processed the data from the input sensors and determined where the end-effector should go to hit the puck. The Update GUI thread updated the labels on the GUI to reflect the real time information.

Chapter 5

Experimental Evaluation

The robotic arm was tested on a Harvard Acclaim 4.6 ft. Air-Powered hockey table. The table was on the smaller spectrum of air hockey tables, but it was within the design space. The performance test of the arm began with the mechanical assembly. The two joint motors were mounted along the rotational axis of the shoulder joint. Then the electrical components were connected to the mechanical structure to provide the sensory input and output for the software system. The webcam was mounted above the shoulder joint with an extension on the arm table mount. An overview of the assembled robotic arm is shown in Figure 5.1.

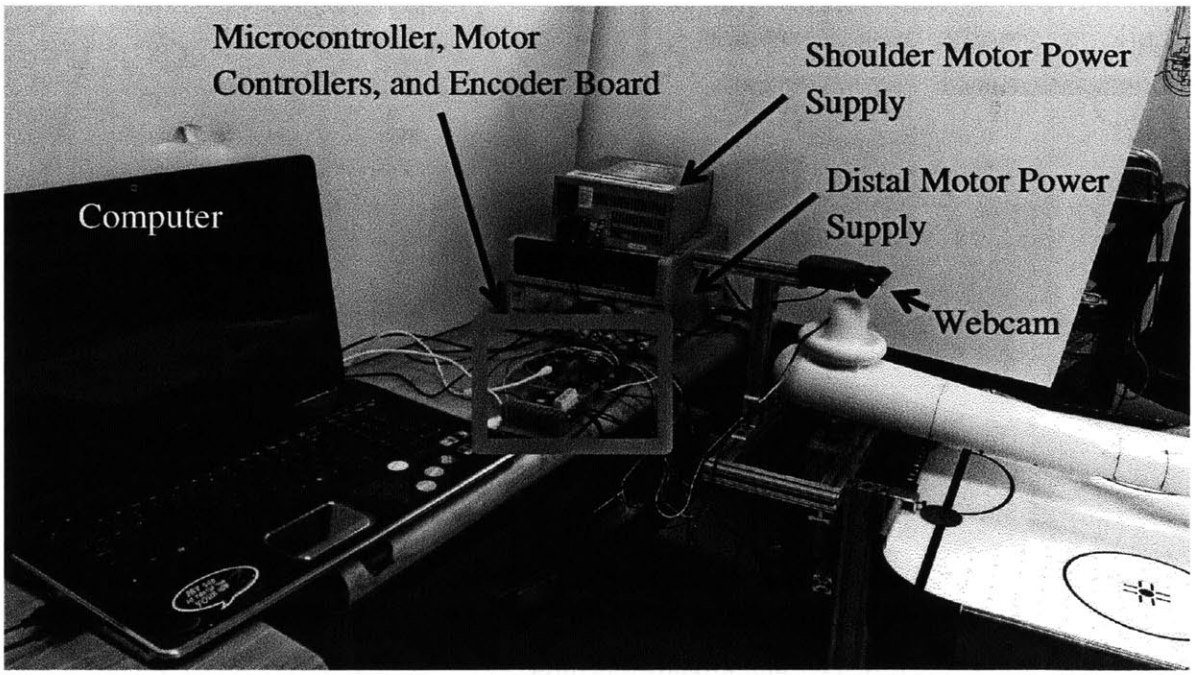

Figure 5.1: The complete setup of the two-link manipulator system. The two-link manipulator was mounted on the Harvard Acclaim air hockey table. The electrical sensors and actuators allowed the arm to autonomously play air hockey. A computer interpreted the sensory data and used it actuate the mechanical arm.

The shoulder motor was powered by a 12 Volt, 100 Amp PowerMax DC power supply. The distal motor was powered by a 12 Volt, 3 Amp DC power supply. These power supplies were chosen to accommodate the stall current of each motor. After many trials, it was observed that the maximum current on both motors never exceeded 3 Amps. This confirmed that the power supplies were adequate in providing power to each motors.

Before any testing could be done, the webcam needed to be calibrated to the same coordinate plane as the end-effector. Four end-effector positions were used to calibrate the webcam. The four points made a rectangle in the end-effector plane, but a converging quadrilateral on the camera image. The configuration points are shown in Figure 5.2.

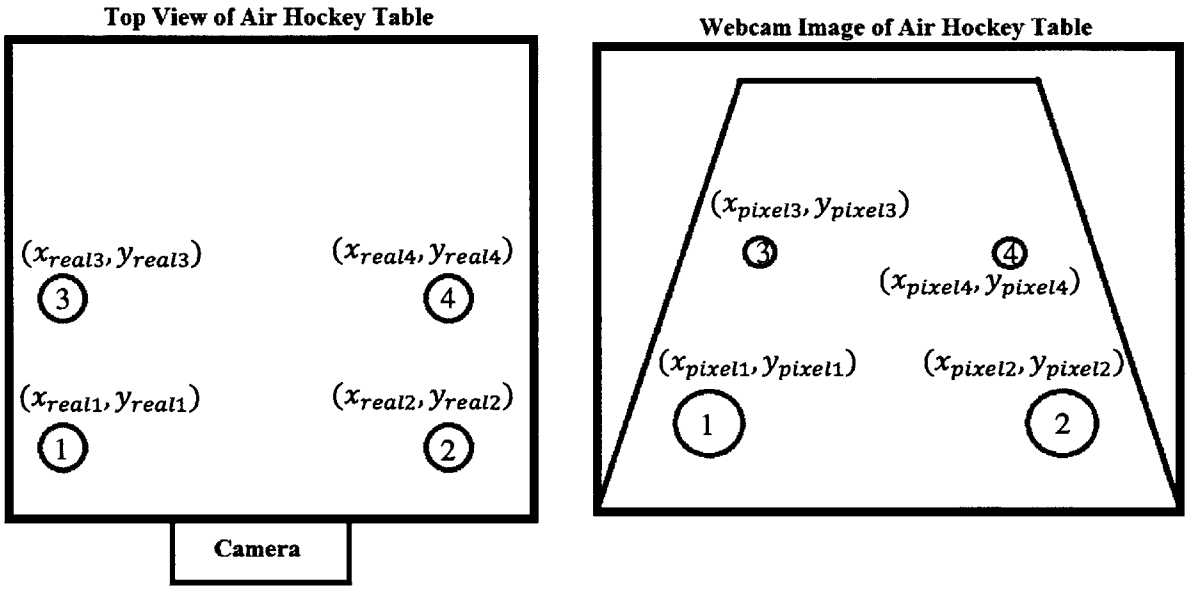

Figure 5.2: The end-effector plane is shown on the left. Four calibration points were used to calibrate the camera's perspective to the end-effector coordinate system. The camera's perspective, on right, shows the perpendicular lines of the real system converging to a point near the center.

To compensate for the converging image, a set of linear equations were used to relate the pixel position, (x_{pixel}, y_{pixel}), to real end-effector position, (x_{real}, y_{real}). Equation 5.1 relates the y positions.

$$y_{real} = m_y * y_{pixel} + b_y \tag{5.1}$$

where

$$m_y = \frac{y_{real1} - y_{real2}}{y_{pixel1} - y_{pixel2}}, \qquad b_y = y_{real1} - m_y * y_{pixel1}$$

The x position required a bit more work because the x-pixel length is dependent on the y values. Equation 5.2 defines the x_{real} position transformation as a function of the current values y_{real}, y_{pixel}, and x_{pixel} and calibration points $x_{real1}, x_{real2}, x_{real}$:

$$x_{real} = m_x * x_{pixel} + b_x \tag{5.2}$$

where

$$m_x = \frac{x_{real1} - x_{real2}}{c_1 - c_2}, \qquad b_x = x_{real1} - m_x * c_1, \qquad c_1 = m_{c1} * y_{pixel} + b_{c1},$$

$$m_{c1} = \frac{x_{pixel1} - x_{pixel3}}{y_{pixel1} - y_{pixel3}}, \qquad b_{c1} = x_{pixel1} - m_{c1} * y_{pixel1}$$

and c_2 is defined similarly to c_1 using points 2 and 4.

The purpose of this project was to design and develop a robot that allowed a person to independently play air hockey. Therefore, the most compelling validation of the robotic arm's performance is its ability to autonomously hit the puck.

5.1 Performance Validation

The arm was given 73 attempts to autonomously hit a moving puck. To hit a moving puck, the computer had to calculate a desired striking point. First, it used a center difference method to find the velocity of the puck. Then, it estimated the puck's position in ten milliseconds using the current position and velocity vector. Ten milliseconds was experimentally chosen as the time step, because it gave the arm enough time to reach the puck from anywhere on the table. Out of the 73 attempts, the arm was able to strike 50 pucks resulting in a 68.5 percent hit rate. This was an excellent result for the first prototype. Due to the complexity of the system and the simplicity of the controls, the hit rate was expected to be lower than 10 percent.

The behavior of the arm was observed to follow the programmed behavior and controls. The arm did not attempt to strike at pucks that were moving away from it or too close to the walls. It returned to its home position after every attempt and did not move when there was no puck present.

Although the results were promising, there were many issues in the operation and controls of the robotic arm. The largest issue was the misalignment of the end-effector position

and the calibrated camera position. Sometimes, the motor overshot the puck's position and hit one of the side walls. In this case, the motors kept rotating but the end-effector did not move. As a result, the computer lost track of the real end-effector position and could not hit the puck accurately.

Another issue was the backlash in the joint motors. The distal arm had a five to ten degree of rotation before the encoder registered any movement. The shoulder motor had even greater backlash. The backlash also resulted in miscalculations of the end-effector position.

Chapter 6

Summary and Conclusion

Air hockey is a sport that is usually played between two or more individuals. There is an inherent issue with the dependency of another person. Sometime, it is more favorable to play air hockey by oneself. An autonomous robotic arm was designed and developed to allow a person to independently play air hockey. The robotic arm consisted of three main systems: mechanical, electrical, and software. The mechanical system was designed as a two-link manipulator with Proportional-Derivate position control. The electrical system contained encoders, a microcontroller, motor controllers and a webcam. The encoders were used to determine the joint angles. The microcontroller and motor controllers were used to actuate the joint motors. The webcam was used to track the position of the puck. The software system used the sensory data to control and actuate the mechanical system.

The robotic arm's performance was well above the expected value. The arm had a hit rate of 68.5 percent and behaved as expected. This indicated that the arm was not hitting the puck by chance. The result suggests that the arm will be able to autonomously play air hockey, but it could be improved by making a few adjusts as outlined in the future work.

6.1 Future Work

Many improvements can be made to increase the performance and robustness of the robotic arm. The major issue that needs to be addressed is the calibration between the real and calculated end-effector position. A more sensitive encoder, more ticks per revolution, might be able to resolve the miscalibration issues. It is also possible that the bearing support structure and shaft couplers are not rigid enough to transfer the rotation of the drive shaft to the motor shaft and encoder.

Instead of a hardware fix, a software solution might prove to be the better option. The webcam can be programmed to track the puck and end-effector in the pixel coordinate system. A control loop can be found to move the end-effector to any desired pixel coordinate. In this solution, the location of the puck and end-effector will always be known and in the same coordinate system. This should prevent the miscalibration issues of a calculated end-effector position that only depended on the encoders. The encoder will still be useful as a velocity sensor and secondary estimate of the end-effector position.

Other secondary improvements can be made to the camera, controller design, and behavior and strategy. Through experimental observation, the current camera at 30 frames per second was not able to track fast moving pucks. A faster camera capable of capturing more frames per second can give the software system a better estimate of the real position and velocity of the puck. A more complex controller design that takes into account the dynamics and kinematics can also improve the arm's performance. And lastly, the arm's behavior and strategy can be improved to make the arm play at a more realistic and competitive level.

Bibliography

[1] Dazadi, "Air Hockey History," 18 Feb 2009, www.dazadi.com (24 April 2013).

[2] George Konidaris *et al.*, Math, Numerics, & Programming (for Mechanical Engineers), 2011.

[3] S. Colton. Notes on DC Motors. 2011.

[4] Pololu Robotics & Electronics, "67:1 Metal Gearmotor 37Dx54L mm with 64 CPR Encoder," 2013, www.pololu.com (26 April 2013).

(this page intentionally left blank)

Appendix A

Matlab Code

```
1.  %%this script is used to process velocity and acceleration of puck/striker%%
2.
3.  %clean workspace
4.  close all
5.  clear all
6.  clc
7.
8.  %load data (the coords are in mm)
9.  %column 1 = marker number
10. %column 2 = x coord
11. %column 3 = y coord
12. %column 4 = z coord
13. %column 5 = blank
14. load('puck3.mat');
15.
16. figVar = figure; %create figure
17. MarkerPosVar = MarkerPosVar/1000;
18. %V = 1/(2h) * ( F(xi + 1) - F(xi-1) )
19. Velocity = zeros(iCounter-1,2); %x and y velocity
20.
21. for i = 1:(iCounter-1)
22.     if i == 1%do foward difference at the start of data
23.         Velocity(1,1) = 1/timeLap(1) * ( MarkerPosVar(2,2) -
     MarkerPosVar(1,2) ); %x velocity
24.         Velocity(1,2) = 1/timeLap(1) * ( MarkerPosVar(2,3) -
     MarkerPosVar(1,3) ); % y velocity
25.     elseif i == (iCounter - 1) %do backward difference
26.         Velocity(i,1) = 1/timeLap(i) * ( MarkerPosVar(i,2) - MarkerPosVar(i-
    1,2) ); %x velocity
27.         Velocity(i,2) = 1/timeLap(i) * ( MarkerPosVar(i,3) - MarkerPosVar(i-
    1,3) ); %y velocity
28.     else %do centered difference
29.         Velocity(i,1) = 1/(2*timeLap(i)) * ( MarkerPosVar(i+1,2) - MarkerPosVar(i-
    1,2) ); %x velocity
30.         Velocity(i,2) = 1/(2*timeLap(i)) * ( MarkerPosVar(i+1,3) - MarkerPosVar(i-
    1,3) ); %y velocity
31.     end
32. end
33.
34. VelMag = sqrt(Velocity(:,1).^2 + Velocity(:,2).^2);
35. tempA = find( VelMag > 20);
36.
37. VelMag(tempA) = [];
38.
39. plot([1:size(VelMag,1)],VelMag);
40. xlabel('Time (s)')
41. ylabel('Speed (m/s)')
```

```
title('Striker Velocity')

fig2 = figure; %acceleration figure
accel = zeros(iCounter-3,2);

for i = 2:(iCounter-2)
    accel(i-1,1) = (1/timeLap(i)^2) * ( MarkerPosVar(i-1,2) - 2*MarkerPosVar(i,2) + MarkerPosVar(i+1,2) ); %x accel
    accel(i-1,2) = (1/timeLap(i)^2) * ( MarkerPosVar(i-1,3) - 2*MarkerPosVar(i,3) + MarkerPosVar(i+1,3) ); %y accel
end

AccelMag = sqrt( accel(:,1).^2 + accel(:,2).^2 );
tempB = find( AccelMag > 200);
AccelMag(tempB) = [];

plot([1:size(AccelMag,1)],AccelMag);
xlabel('Time (s)')
ylabel('Acceleration (m/s^2)')
title('Striker Accel')

%{
fig3 = figure;
mass_puck = 12.4/1000;
force = mass_puck * accelMag;
plot(time(2:iCounter-2),force);
xlabel('Time (s)')
ylabel('Force (N)')
title('Striker Force')
%}



%%histogram calculations%%

clear all
close all
clc

load('puck1Calc.mat');
load('puck2Calc.mat');
load('puck3Calc.mat');
load('strikerCalc.mat');

allPuckVel = [Puck1VelMag;Puck2VelMag;Puck3VelMag];
allPuckAcc = [Puck1AccelMag;Puck2AccelMag;Puck3AccelMag];

puckVelFig = figure;
[puckN,puckXOut] = hist(allPuckVel,20);
bar(puckXOut,puckN/size(allPuckVel,1));
xlabel('Velocity (m/s)');
ylabel('Frequency')
title('Puck Velocity Histogram');

puckAccelFig = figure;
[puckN2,puckXOut2] = hist(allPuckAcc,20);
bar(puckXOut2,puckN2/size(allPuckAcc,1));
xlabel('Acceleration(m/s^2)');
ylabel('Frequency')
    title('Puck Acceleration Histogram');
```

```
101.
102.        strikerVelFig = figure;
103.        [strN,strXOut] = hist(StrikerVelMag,20);
104.        bar(strXOut,strN/size(StrikerVelMag,1));
105.        xlabel('Velocity (m/s)');
106.        ylabel('Frequency')
107.        title('Striker Velocity Histogram');
108.
109.        strikerMagFig = figure;
110.        [strN2,strXOut2] = hist(StrikerAccelMag,20);
111.        bar(strXOut2,strN2/size(StrikerAccelMag,1));
112.        xlabel('Acceleration(m/s^2)');
113.        ylabel('Frequency')
114.        title('Striker Acceleration Histogram');
```

(this page intentionally left blank)

Appendix B

C++ Code

```
/*
 ============================================================================
 Name        : airHockey.cpp
 Author      : Bee
 Version     : 1.00
 Copyright   : Your copyright notice
 Description : Air Hockey Software with GTKmm
 ============================================================================
 */

#include <gtkmm/main.h>
#include <gtk/gtk.h>
#include <windows.h>
#include "mainWindow.h"

using namespace std;

int main (int argc, char *argv[])
{
    Glib::thread_init();
    Gtk::Main kit(argc, argv);
    mainWindow mainWindow;

    std::cout << "main: start" << std::endl;
    //This is the main thread
    //Gtk::Main::run(mainWindow);
    kit.run(mainWindow);
    std::cout << "main: done" << std::endl;
    return 0;
}
```

```
/*
 * global.h
 *
 *  Created on: Jan 29, 2013
 *      Author: bvang13
 */

#ifndef GLOBAL_H_
#define GLOBAL_H_
#define BOOST_THREAD_USE_LIB
#include <boost/thread.hpp>
#include <boost/date_time.hpp>
```

```
13. #include <boost/math/special_functions/round.hpp>
14. #include <string>
15. #include <math.h>
16. #include <boost/timer.hpp>
17.
18. using namespace std;
19.
20. extern const double Len1, Len2, THETA_TOL1, THETA_TOL2, Kp1, Kd1, Kp2, Kd2, SPEED_TOL,
    HOME_POS[2];
21. extern const int V1_MAX, V1_MIN, V2_MAX, V2_MIN, V_NEUTRAL;
22. extern const unsigned char END_SIGNAL;
23. extern int cport_nr, bdrate;
24. extern double currentTheta1, currentTheta2, desiredTheta1, desiredTheta2,
25.                 wcurrent1, wcurrent2, error1, error2;
26. extern int indicator;
27. extern double time1[3], time2[3], thetaStore1[3], thetaStore2[3], tableLength, tableWid
    th, armOffset;
28. extern char vOut1[], vOut2[];
29. extern bool runMotorController, runCalcPuckTrajectory, killThread, cameraTracking;
30.
31. extern string malletX, malletY, puckX, puckY, projectedX, projectedY;
32.
33. extern double calibratePoint1[4], calibratePoint2[4], calibratePoint3[4], calibratePoin
    t4[4];
34. //using y = mx+b equation to find scale of px to real in the perspective of the camera

35. extern double calibrationXSlopePX[2], calibrationXConstantPX[2], calibrationYSlope, cal
    ibrationYConstant,
36.                 calibrationXSlope, calibrationXConstant;
37.
38. extern int encoderPosition[2]; //store encoder position until encoder is loaded
39.
40. extern double puckTime[3], puckPositionX[3], puckPositionY[3], puckVelocity[2];
41. #endif /* GLOBAL_H_ */
42.
43. /*
44.  * global.cpp
45.  *
46.  *  Created on: Jan 29, 2013
47.  *      Author: bvang13
48.  */
49.
50. #include "global.h"
51.
52. //constants
53. const double Len1 = 0.55834526;//base arm length in meters
54. const double Len2 = 0.43815;//forearm length in meters
55. const double THETA_TOL1 = 0.1;//radian
56. const double THETA_TOL2 = 0.1;//radian
57. const int V1_MAX = 1520;//base max
58. const int V1_MIN = 1480;//base min
59. const int V2_MAX = 1700;//fore max
60. const int V2_MIN = 1300;//fore min
61. const int V_NEUTRAL = 1500;
62. const unsigned char END_SIGNAL  = 10;//signal to end transmission of data
63. const double Kp1 = 50.0;//base arm
64. const double Kd1 = -50.0;//base earm (-20.0)
65. const double Kp2 = 250;//base arm
66. const double Kd2 = -150.0;//forearm (-100)
67. const double SPEED_TOL = 0.01;//m/s
68. const double HOME_POS[2] = {0,0.3};//home position in meters (x,y)
```

```
69.
70. //global serial
71. int cport_nr,        /* /dev/ttyS0 (COM1 on windows) */
72. bdrate;       /* 115200 baud */
73.
74. //global variables
75. double currentTheta1 = M_PI/2.0; //store the counter variable of the encoder from zero

76. double currentTheta2 = 0.0;
77. double desiredTheta1 = M_PI/2.0;
78. double desiredTheta2 = 0.0;
79. int indicator = 0;//0 = phidget not connected, 1 = phidget connected
80. double time1[3] = {0,0,1};//holds previous elapsed time
81. double time2[3] = {0,0,1};
82. double thetaStore1[3] = {0,0,0};//holds previous positions
83. double thetaStore2[3] = {0,0,0};
84. double wcurrent1 = 0.0;
85. double wcurrent2 = 0.0;
86. double error1 = 0.0;
87. double error2 = 0.0;
88. char vOut1[] = "1500";//pwm ctrl signal for base motor
89. char vOut2[] = "1500";//pwm ctrl signal for forearm motor
90.
91. bool runMotorController = false;
92. bool killThread = false;
93. bool cameraTracking = false;
94.
95. string malletX = boost::lexical_cast<string>(int((Len1*cos(currentTheta1)+Len2*cos(curr
    entTheta1+currentTheta2))*100.0));
96. string malletY = boost::lexical_cast<string>(int((Len1*sin(currentTheta1)+Len2*sin(curr
    entTheta1+currentTheta2))*100.0));
97. string puckX = boost::lexical_cast<string>(0);
98. string puckY = boost::lexical_cast<string>(0);
99. string projectedX = boost::lexical_cast<string>(0);
100.        string projectedY = boost::lexical_cast<string>(0);
101.
102.        double tableLength = 1.36;//meters
103.        double tableWidth = 0.6;//meters
104.        double armOffset = 0.12;//meters
105.
106.        //calibration points, each contains [pixel X, pixel Y, real X, real Y]
107.        double calibratePoint1[4] = {0,0,0,0}; //bottom left point
108.        double calibratePoint2[4] = {0,0,0,0}; //bottom right point
109.        double calibratePoint3[4] = {0,0,0,0}; //top left point
110.        double calibratePoint4[4] = {0,0,0,0}; //top right point
111.        double calibrationXSlopePX[2] = {0.0,0.0}; //scaling factor, left/right side, fo
    r pixel line
112.        double calibrationXConstantPX[2] = {0.0,0.0}; //constant, left/right side, for p
    ixel line
113.        double calibrationXSlope = 0.0; //scaling factor, m, xreal/xpx
114.        double calibrationXConstant = 0.0; //constant, b, xreal/xpx
115.        double calibrationYSlope = 0.0; //scaling factor, m, yreal/ypx
116.        double calibrationYConstant = 0.0; //constant, b, yreal/ypx
117.
118.        int encoderPosition[2] = {0,0};
119.
120.        //tracking puck
121.        double puckTime[3] = {0,0,0};
122.        double puckPositionX[3] = {0,0,0};
123.        double puckPositionY[3] = {0,0,0};
124.        double puckVelocity[2] = {0,0};//x,y unit vector
```

```
125.
126.        bool runCalcPuckTrajectory = false;//do not run calc puck trajectory thread
```

```
1.  #ifndef MAINWINDOW_H
2.  #define MAINWINDOW_H
3.  #include <gtkmm.h>
4.  #include "CameraSensor.h"
5.
6.  using namespace std;
7.
8.  class mainWindow : public Gtk::Window
9.  {
10. public:
11.   mainWindow();
12.   virtual ~mainWindow();
13.
14. private:
15.   // Signal handlers:
16.   void on_button_clicked(Glib::ustring data);
17.   void connect_button(void);
18.   void save_log(void);
19.   void help(void);
20.   void helpHide(void);
21.   bool onKeyPress( GdkEventKey *event );
22.   void start_robot(void);
23.   void startCameraThread(void);
24.   void calibrateHSV(void);
25.   void loadHSV(void);
26.   void setXYPos(void);
27.   void setTheta2(void); //unused
28.   void goToXY(void);
29.   void updatePositions(void);
30.   void updatePosDisplay(void);
31.   void calibrateCamera(void);
32.   void calibrateAlgorithm(double xPX, double yPX);
33.   void calculatePuckTrajectory(void);
34.
35.   // Child widgets:
36.   Gtk::Table m_Table1, m_Table2, m_Table3, m_Table4, m_tableSetting, m_tableStatus;
37.   Gtk::ToggleButton m_toggleConnect, m_toggleMode;
38.   Gtk::Button m_button2, m_button15, m_textSave, m_buttonHelpOk, m_buttonSetTheta1, m_b
    uttonSetTheta2;
39.   Gtk::ScrolledWindow m_scrolledWindow;
40.   Gtk::TextView m_textView1;
41.   Glib::RefPtr<Gtk::TextBuffer> m_textBuffer1;
42.   Gtk::VBox m_VBox;
43.   Gdk::Color m_Color;
44.   Gtk::Frame m_frameStatus,m_frameConcSetting,m_frameButton, m_frameTest, m_frameMallet
    X, m_frameMalletY,
45.                   m_framePuckX, m_framePuckY, m_frameProjX, m_frameProjY, m_frameArmSet
    ting, m_frameCameraSetting, m_frameControl,
46.                   m_frameTableSetting;
47.   Gtk::Label m_labelComport, m_labelBaudrate, m_labelArmPos, m_labelPuckPos, m_labelPro
    jectedPos,
48.                 m_labelArmSetting, m_labelCameraSetting, m_labelGoX, m_labelGoY, m_lab
    elTableLength,
```

```
49.                 m_labelTableWidth, m_labelArmOffset, m_labelMalletX, m_labelMalletY, m
    _labelPuckX,
50.                 m_labelPuckY, m_labelProjX, m_labelProjY, m_labelSetX, m_labelSetY;
51.   Gtk::ComboBoxText m_comboComport, m_comboBaudrate;
52.   //Gtk::InputDialog m_inputDialogTheta1, m_inputDialogTheta2;
53.   Gtk::Entry m_inputDialogTheta1, m_inputDialogTheta2, m_inputDialogX, m_inputDialogY,
    m_inputLength,
54.                  m_inputWidth, m_inputOffset;
55.   Gtk::Dialog dialog;
56.   Gtk::Label label1, label2,label3,label4;
57.   Gtk::Button m_buttonCameraCalibrate, m_buttonGoToPos, m_buttonCalibrateHSV, m_buttonS
    tartCamera;
58.   Gtk::Table  m_tableTemp, m_tableAdvSetting, m_tableArmSetting, m_tableCameraSetting,
    m_tableControl,
59.                  m_tableTableSetting;
60.   Gtk::Image m_imageMallet, m_imagePuck, m_imageGuess, m_imageControl;
61.
62.   //display position thread
63.   Glib::Thread* m_threadPosDisplay;
64.   Glib::Dispatcher m_threadPosDisplay_Dispatcher;
65.
66.   //class Variables
67.   bool errorMsg;
68.   CameraSensor cameraSensor;
69.   CPhidgetEncoderHandle encoderReader;
70. };
71.
72. #endif // MAINWINDOW_H
73.
74. /*
75.  ============================================================================
76.  Name        : mainWindow.cpp
77.  Author      : Bee
78.  Version     :
79.  Copyright   : Your copyright notice
80.  Description : Air Hockey Software with GTKmm
81.  ============================================================================
82.  */
83. //includes
84. #include "mainWindow.h"
85. #include "function.h"
86. using namespace std;
87.
88.
89.
90. mainWindow::mainWindow()
91. : errorMsg(true),
92.   m_Table1(2,2,true),//main table
93.   m_Table2(1,2,true),//split top right corner into halves
94.   m_Table3(3,1,true),//button tables
95.   m_Table4(4,2,false),//status indicators table
96.   m_tableAdvSetting(4,1,false),//advance settings
97.   m_tableSetting(2,2,false),//table for m_frameConcSetting
98.   m_tableStatus(6,4,true),//table for status indicators
99.   m_tableArmSetting(3,3,false),
100.         m_tableTableSetting(3,2,false),
101.         m_tableCameraSetting(3,1,false),
102.         m_tableControl(3,4,false),
103.         m_button2("Status"),
104.         m_buttonCameraCalibrate("Calibrate Position"),
105.         m_buttonCalibrateHSV("Calibrate HSV"),
```

```
106.          m_buttonStartCamera("Start Camera Tracking"),
107.          m_buttonGoToPos("Go to X,Y"),
108.          m_labelMalletX("X coord"),
109.          m_labelMalletY("Y coord"),
110.          m_labelPuckX("X coord"),
111.          m_labelPuckY("Y coord"),
112.          m_labelProjX("X coord"),
113.          m_labelProjY("Y coord"),
114.          m_labelGoX("X (cm): "),
115.          m_labelGoY(" Y (cm): "),
116.          m_labelSetX("X (cm): "),
117.          m_labelSetY("Y (cm): "),
118.          m_labelTableLength("Table Length (m): "),
119.          m_labelTableWidth("Table Width (m): "),
120.          m_labelArmOffset("Arm offset (m): "),
121.          m_toggleConnect("Connect"),
122.          m_toggleMode("Start Robot"),
123.          m_button15("Help"),
124.          m_buttonHelpOk("Ok"),
125.          m_buttonSetTheta1("Set x,y (cm)"),
126.          m_buttonSetTheta2("Set Theta2 = 0"),
127.          m_textSave("Save Log"),
128.          m_frameStatus("Status"),
129.          m_frameConcSetting("Connection Settings"),
130.          m_frameArmSetting("Arm Calibration"),
131.          m_frameTableSetting("Table Settings"),
132.          m_frameCameraSetting("Camera Calibration"),
133.          m_frameControl("User Control"),
134.          m_labelComport("Comport:"),
135.          m_labelBaudrate("Baudrate: "),
136.          m_labelArmPos("Mallet Position (cm)"),
137.          m_labelPuckPos("Puck Position (cm)"),
138.          m_labelProjectedPos("Projected Impact Position (cm)"),
139.          m_tableTemp(2,2,false),
140.          m_imageMallet("./test.jpg"),
141.          m_imagePuck("./test.jpg"),
142.          m_imageGuess("./test.jpg"),
143.          m_imageControl("./img/ControlButtons.jpg")
144.
145.      {
146.          // This just sets the title of our new window.
147.          set_title("Air Hockey Controller");
148.          //set_position(GTK_WIN_POS_CENTER);//work on this later
149.
150.          // sets the border width of the window.
151.          set_border_width(10);
152.
153.          //Action Log
154.          m_textView1.set_editable(false);
155.          m_textView1.set_cursor_visible(false);
156.          m_textBuffer1 = Gtk::TextBuffer::create();
157.          m_textBuffer1-
   >set_text("**AirHockey Computer Controller Version 1.00** \n");
158.          m_textView1.set_buffer(m_textBuffer1);
159.
160.          //define Vbox for Action Log
161.          m_VBox.pack_start(m_scrolledWindow);
162.          m_VBox.pack_start(m_textSave, Gtk::PACK_SHRINK);
163.
164.          //settings section
165.          m_comboComport.append_text("1");
```

```
166.            m_comboComport.append_text("2");
167.            m_comboComport.append_text("3");
168.            m_comboComport.append_text("4");
169.            m_comboComport.append_text("5");
170.            m_comboComport.append_text("6");
171.            m_comboComport.append_text("7");
172.            m_comboComport.append_text("8");
173.            m_comboComport.append_text("9");
174.            m_comboComport.append_text("10");
175.            m_comboComport.append_text("11");
176.            m_comboComport.append_text("12");
177.            m_comboComport.set_active_text("3");
178.            m_comboBaudrate.append_text("300");
179.            m_comboBaudrate.append_text("1200");
180.            m_comboBaudrate.append_text("2400");
181.            m_comboBaudrate.append_text("4800");
182.            m_comboBaudrate.append_text("9600");
183.            m_comboBaudrate.append_text("14400");
184.            m_comboBaudrate.append_text("19200");
185.            m_comboBaudrate.append_text("28800");
186.            m_comboBaudrate.append_text("38400");
187.            m_comboBaudrate.append_text("57600");
188.            m_comboBaudrate.append_text("115200");
189.            m_comboBaudrate.set_active_text("115200");
190.            m_tableSetting.attach(m_labelComport,0,1,0,1);
191.            m_tableSetting.attach(m_labelBaudrate,1,2,0,1);
192.            m_tableSetting.attach(m_comboComport,0,1,1,2,Gtk::FILL,Gtk::FILL,5,5);
193.            m_tableSetting.attach(m_comboBaudrate,1,2,1,2,Gtk::FILL,Gtk::FILL,5,5);
194.            m_frameConcSetting.add(m_tableSetting);
195.            m_frameConcSetting.set_border_width(5);
196.
197.            //add lables to frame
198.            m_frameMalletX.add(m_labelMalletX);
199.            m_frameMalletY.add(m_labelMalletY);
200.            m_framePuckX.add(m_labelPuckX);
201.            m_framePuckY.add(m_labelPuckY);
202.            m_frameProjX.add(m_labelProjX);
203.            m_frameProjY.add(m_labelProjY);
204.            m_frameMalletX.set_shadow_type(Gtk::SHADOW_ETCHED_IN);
205.            m_frameMalletX.set_border_width(2);
206.            m_frameMalletY.set_shadow_type(Gtk::SHADOW_ETCHED_IN);
207.            m_frameMalletY.set_border_width(2);
208.            m_framePuckX.set_shadow_type(Gtk::SHADOW_ETCHED_IN);
209.            m_framePuckX.set_border_width(2);
210.            m_framePuckY.set_shadow_type(Gtk::SHADOW_ETCHED_IN);
211.            m_framePuckY.set_border_width(2);
212.            m_frameProjX.set_shadow_type(Gtk::SHADOW_ETCHED_IN);
213.            m_frameProjX.set_border_width(2);
214.            m_frameProjY.set_shadow_type(Gtk::SHADOW_ETCHED_IN);
215.            m_frameProjY.set_border_width(2);
216.
217.            //define table 4 widgets
218.            m_Table4.attach(m_frameConcSetting,0,2,0,1);//settings section
219.            m_tableStatus.attach(m_labelArmPos,0,3,0,1);
220.            m_tableStatus.attach(m_imageMallet,3,4,0,1);
221.            m_tableStatus.attach(m_frameMalletX,0,2,1,2);
222.            m_tableStatus.attach(m_frameMalletY,2,4,1,2);
223.            m_tableStatus.attach(m_labelPuckPos,0,3,2,3);
224.            m_tableStatus.attach(m_imagePuck,3,4,2,3);
225.            m_tableStatus.attach(m_framePuckX,0,2,3,4);
226.            m_tableStatus.attach(m_framePuckY,2,4,3,4);
```

```
227.            m_tableStatus.attach(m_labelProjectedPos,0,3,4,5);
228.            m_tableStatus.attach(m_imageGuess,3,4,4,5);
229.            m_tableStatus.attach(m_frameProjX,0,2,5,6);
230.            m_tableStatus.attach(m_frameProjY,2,4,5,6);
231.            m_frameStatus.add(m_tableStatus);
232.            m_frameStatus.set_border_width(5);
233.            m_Table4.attach(m_frameStatus,0,1,1,4);
234.            //m_frameTest.set_shadow_type(Gtk::SHADOW_IN);
235.
236.            //define table 3 widgets
237.            m_Table3.attach(m_toggleConnect,0,1,0,1);
238.            m_Table3.attach(m_toggleMode,0,1,1,2);
239.            m_Table3.attach(m_button15,0,1,2,3);
240.
241.            //define table 2 widgets
242.            m_Table2.attach(m_Table4,0,1,0,1);
243.            m_Table2.attach(m_Table3,1,2,0,1);
244.
245.            //add text box to scroll window
246.            m_scrolledWindow.add(m_textView1);
247.            m_scrolledWindow.set_policy(Gtk::POLICY_AUTOMATIC, Gtk::POLICY_ALWAYS);
248.
249.            // put the box into the main window.
250.            add(m_Table1);
251.            m_Table1.attach(m_VBox,0,1,0,1);
252.            m_Table1.attach(m_tableAdvSetting,0,2,1,2);
253.            //m_Table1.attach(m_frameDisplay,0,2,1,2);
254.            //m_frameDisplay.set_border_width(3);
255.            m_Table1.attach(m_Table2,1,2,0,1);
256.
257.            //table settings
258.            m_tableTableSetting.set_border_width(3);
259.            m_tableTableSetting.attach(m_labelTableLength,0,1,0,1);
260.            m_tableTableSetting.attach(m_labelTableWidth,0,1,1,2);
261.            m_tableTableSetting.attach(m_labelArmOffset,0,1,2,3);
262.            m_tableTableSetting.attach(m_inputLength,1,2,0,1);
263.            m_tableTableSetting.attach(m_inputWidth,1,2,1,2);
264.            m_tableTableSetting.attach(m_inputOffset,1,2,2,3);
265.            m_inputLength.set_text(boost::lexical_cast<string>(tableLength));
266.            m_inputWidth.set_text(boost::lexical_cast<string>(tableWidth));
267.            m_inputOffset.set_text(boost::lexical_cast<string>(armOffset));
268.
269.            //add control settings to bottom of controller table
270.            m_tableAdvSetting.attach(m_frameArmSetting,0,1,0,1);
271.            m_tableAdvSetting.attach(m_frameTableSetting,1,2,0,1);
272.            m_tableAdvSetting.attach(m_frameCameraSetting,2,3,0,1);
273.            m_tableAdvSetting.attach(m_frameControl,3,4,0,1);
274.            m_frameArmSetting.set_border_width(3);
275.            m_frameTableSetting.set_border_width(3);
276.            m_frameCameraSetting.set_border_width(3);
277.            m_frameControl.set_border_width(3);
278.            m_frameArmSetting.add(m_tableArmSetting);
279.            m_frameTableSetting.add(m_tableTableSetting);
280.            m_frameCameraSetting.add(m_tableCameraSetting);
281.            m_frameControl.add(m_tableControl);
282.            m_labelArmSetting.set_text("Set the end effector position. (Units are in cm)
    ");
283.            m_labelArmSetting.set_line_wrap(true);
284.            m_labelArmSetting.set_alignment(Gtk::ALIGN_LEFT);
285.            m_labelCameraSetting.set_text("Calibrate the camera after setting the initia
    l angles, this button will move the arm to four positions to calibrate the camera.");
```

```
286.            m_labelCameraSetting.set_line_wrap(true);
287.            m_labelCameraSetting.set_alignment(Gtk::ALIGN_LEFT);
288.            m_tableArmSetting.attach(m_labelArmSetting,0,3,0,1);
289.            m_tableArmSetting.set_border_width(3);
290.            m_tableArmSetting.attach(m_buttonSetTheta1,2,3,1,3);
291.            //m_tableArmSetting.attach(m_buttonSetTheta2,1,2,2,3);
292.            m_tableArmSetting.attach(m_labelSetX,0,1,1,2);
293.            m_tableArmSetting.attach(m_labelSetY,0,1,2,3);
294.            m_tableArmSetting.attach(m_inputDialogTheta1,1,2,1,2);
295.            m_tableArmSetting.attach(m_inputDialogTheta2,1,2,2,3);
296.            m_tableCameraSetting.attach(m_labelCameraSetting,0,2,0,1);
297.            m_tableCameraSetting.attach(m_buttonCameraCalibrate,0,1,1,2);
298.            m_tableCameraSetting.attach(m_buttonCalibrateHSV,0,1,2,3);
299.            m_tableCameraSetting.set_border_width(3);
300.            m_tableControl.attach(m_buttonStartCamera,0,4,0,1);
301.            m_tableControl.attach(m_labelGoX,0,1,1,2);
302.            m_tableControl.attach(m_inputDialogX,1,2,1,2);
303.            m_tableControl.attach(m_labelGoY,2,3,1,2);
304.            m_tableControl.attach(m_inputDialogY,3,4,1,2);
305.            m_tableControl.attach(m_buttonGoToPos,0,4,2,3);
306.            m_tableControl.set_border_width(3);
307.            /*m_inputDialogTheta1.set_text(boost::lexical_cast<string>(17));
308.            m_inputDialogTheta2.set_text(boost::lexical_cast<string>(63));*/
309.
310.            // Now when the button is clicked, we call the "on_button_clicked" function

311.            // with a pointer to "button 1" as it's argument
312.            //m_button1.signal_clicked().connect(sigc::bind<Glib::ustring>(
313.            //sigc::mem_fun(*this, &HelloWorld::on_button_clicked), "button 1"));
314.
315.            m_textSave.signal_clicked().connect(sigc::mem_fun(*this, &mainWindow::save_l
    og));
316.            m_button15.signal_clicked().connect(sigc::mem_fun(*this, &mainWindow::help))
    ;
317.            m_toggleConnect.signal_clicked().connect(sigc::mem_fun(*this, &mainWindow::c
    onnect_button));
318.            m_toggleMode.signal_clicked().connect(sigc::mem_fun(*this, &mainWindow::star
    t_robot));
319.            m_button2.signal_clicked().connect(sigc::bind<-1, Glib::ustring>(
320.                    sigc::mem_fun(*this, &mainWindow::on_button_clicked), "button 2"));

321.            this->add_events(Gdk::KEY_PRESS_MASK);
322.            this-
    >signal_key_press_event().connect( sigc::mem_fun( *this, &mainWindow::onKeyPress ) );
323.            m_buttonCameraCalibrate.signal_clicked().connect(sigc::mem_fun(*this, &mainW
    indow::calibrateCamera));
324.            m_buttonCalibrateHSV.signal_clicked().connect(sigc::mem_fun(*this, &mainWind
    ow::calibrateHSV));
325.            m_buttonStartCamera.signal_clicked().connect(sigc::mem_fun(*this, &mainWindo
    w::startCameraThread));
326.            m_buttonGoToPos.signal_clicked().connect(sigc::mem_fun(*this, &mainWindow::g
    oToXY));
327.            m_buttonSetTheta1.signal_clicked().connect(sigc::mem_fun(*this, &mainWindow:
    :setXYPos));
328.            m_buttonSetTheta2.signal_clicked().connect(sigc::mem_fun(*this, &mainWindow:
    :setTheta2));
329.            m_buttonHelpOk.signal_clicked().connect(sigc::mem_fun(*this, &mainWindow::he
    lpHide));
330.            m_threadPosDisplay_Dispatcher.connect(sigc::mem_fun(*this,&mainWindow::updat
    ePosDisplay));
331.
```

```
332.            show_all_children();
333.
334.            label1.set_text("Manual Mode:");
335.            label1.modify_font(Pango::FontDescription("Sans Bold Not-Rotated 24"));
336.            label2.set_text("text for manual mode");
337.            label3.set_text("Automatic Mode:");
338.            label3.modify_font(Pango::FontDescription("Sans Bold Not-Rotated 24"));
339.            label4.set_text(" 1) Press 'Connect' \n 2) Press 'Start Robot' \n 3) Press '
    Start Camera Tracking'");
340.            dialog.add_action_widget(m_buttonHelpOk,0);
341.            dialog.set_title("Help");
342.            //dialog.set_default_size(300,100);
343.            dialog.set_resizable(false);
344.            m_tableTemp.attach(label1,0,1,0,1);
345.            m_tableTemp.attach(m_imageControl,0,1,1,2,Gtk::FILL,Gtk::FILL,10,10);
346.            m_tableTemp.attach(label3,1,2,0,1);
347.            m_tableTemp.attach(label4,1,2,1,2);
348.
349.            Gtk::Box *box = dialog.get_vbox();
350.            box->pack_start(m_tableTemp);
351.            dialog.show_all_children();
352.            dialog.set_position(Gtk::WIN_POS_CENTER);
353.            // Show the widgets.
354.            // They will not really be shown until this Window is shown.
355.
356.            //set_resizable(false);
357.            set_default_size(800,600);
358.            set_position(Gtk::WIN_POS_CENTER);
359.
360.            m_threadPosDisplay = Glib::Thread::create(sigc::mem_fun(*this, &mainWindow::
    updatePositions),true);
361.        }
362.
363.        mainWindow::~mainWindow()
364.        {
365.            //save position files
366.            ofstream myfile ("positionConfig.ini");
367.            if (myfile.is_open())
368.            {
369.                //save calibration points
370.                myfile << calibratePoint1[0] <<"\n";
371.                myfile << calibratePoint1[1] <<"\n";
372.                myfile << calibratePoint1[2] <<"\n";
373.                myfile << calibratePoint1[3] <<"\n";
374.                myfile << calibratePoint2[0] <<"\n";
375.                myfile << calibratePoint2[1] <<"\n";
376.                myfile << calibratePoint2[2] <<"\n";
377.                myfile << calibratePoint2[3] <<"\n";
378.                myfile << calibratePoint3[0] <<"\n";
379.                myfile << calibratePoint3[1] <<"\n";
380.                myfile << calibratePoint3[2] <<"\n";
381.                myfile << calibratePoint3[3] <<"\n";
382.                myfile << calibratePoint4[0] <<"\n";
383.                myfile << calibratePoint4[1] <<"\n";
384.                myfile << calibratePoint4[2] <<"\n";
385.                myfile << calibratePoint4[3] <<"\n";
386.
387.                //save slopes
388.                myfile << calibrationYSlope <<"\n";
389.                myfile << calibrationYConstant <<"\n";
390.                myfile << calibrationXSlopePX[0] <<"\n";
```

```
391.                 myfile << calibrationXSlopePX[1] <<"\n";
392.                 myfile << calibrationXConstantPX[0] <<"\n";
393.                 myfile << calibrationXConstantPX[1] <<"\n";
394.
395.                 //save current theta/encoder position
396.                 myfile << currentTheta1 <<"\n";
397.                 myfile << currentTheta2 <<"\n";
398.
399.                 myfile.close();
400.                 cout << "Position Calibration file updated." << endl;
401.             }
402.             else
403.             {
404.                 cout << "Cannot write to position Calibration file." << endl;
405.             }
406.         }
```

```
1.  /*
2.   * function.h
3.   *
4.   *  Created on: Jan 29, 2013
5.   *      Author: bvang13
6.   */
7.
8.  #ifndef FUNCTION_H_
9.  #define FUNCTION_H_
10.
11. #include "mainWindow.h"
12. #include <iostream>
13. #include <fstream>
14. #include <string>
15. #include <algorithm>
16. #include <stdlib.h>
17. #include <stdio.h>
18. #include <Windows.h>
19. #include "rs232.h"
20. #include <time.h>
21. using namespace std;
22.
23.
24. #endif /* FUNCTION_H_ */
25.
26. /*
27.  * function.cpp
28.  *
29.  *  Created on: Jan 28, 2013
30.  *      Author: bvang13
31.  */
32.
33. #include "function.h"
34.
35. // functions and events
36. void mainWindow::connect_button(void)
37. {
38.     if (m_toggleConnect.get_active())//clicking connect
39.     {
40.         cport_nr = atoi(m_comboComport.get_active_text().c_str()) - 1;
```

```
41.         bdrate = atoi(m_comboBaudrate.get_active_text().c_str());
42.         if (indicator == 0)//phidget is not connected
43.         {
44.             encoderReader = encoder_simple();
45.             if (encoderReader != 0)//try to connect to phidget
46.             {
47.                 indicator = 1;//phidget is connected
48.                 m_textBuffer1->insert_at_cursor("Phidget Connected\n");
49.                 m_textView1.set_buffer(m_textBuffer1);
50.                 cout << "loaded settings" << endl;
51.                 if(OpenComport(cport_nr, bdrate))//try to connect to arduino
52.                 {
53.                     m_textBuffer1-
    >insert_at_cursor("ERROR: Cannot connect to Arduino\n");
54.                     m_textView1.set_buffer(m_textBuffer1);
55.                     errorMsg = true; //indicates that there is error connecting
56.                     m_toggleConnect.set_active(false);
57.                     runMotorController = false;//close PD control threads
58.                 }
59.                 else
60.                 {//case where everything is connected
61.                     m_textBuffer1->insert_at_cursor("Connected to Arduino\n");
62.                     m_textView1.set_buffer(m_textBuffer1);
63.                     m_toggleConnect.set_label("Disconnect");
64.                     errorMsg = false; // no error connecting to comport
65.                     //m_comboComport.set_editable(false);
66.                     //m_comboBaudrate.set_editable(false);
67.                 }
68.             }
69.             else//cannot connect to phidget
70.             {
71.                 m_textBuffer1-
    >insert_at_cursor("ERROR: Cannot connect to Phidget\n");
72.                 m_textView1.set_buffer(m_textBuffer1);
73.                 errorMsg = true;
74.                 m_toggleConnect.set_active(false);
75.                 runMotorController = false;//close PD control threads
76.             }
77.         }
78.         else//phidget already connected
79.         {
80.             m_textBuffer1->insert_at_cursor("Phidget Connected\n");
81.             m_textView1.set_buffer(m_textBuffer1);
82.             if(OpenComport(cport_nr, bdrate))
83.             {
84.                 m_textBuffer1-
    >insert_at_cursor("ERROR: Cannot connect to Arduino\n");
85.                 m_textView1.set_buffer(m_textBuffer1);
86.                 errorMsg = true; //indicates that there is error connecting
87.                 m_toggleConnect.set_active(false);
88.                 runMotorController = false;//close PD control threads
89.             }
90.             else
91.             {//case where everything is connected
92.                 m_textBuffer1->insert_at_cursor("Connected to Arduino\n");
93.                 m_textView1.set_buffer(m_textBuffer1);
94.                 m_toggleConnect.set_label("Disconnect");
95.                 errorMsg = false; // no error connecting to comport
96.                 boost::thread(displayInfo);
97.                 runMotorController = true;
98.                 //m_comboComport.set_editable(false);
```

```
99.                 //m_comboBaudrate.set_editable(false);
100.                    }
101.                }
102.            }
103.            else//click disconnect
104.            {
105.                if (!errorMsg)
106.                {
107.                    CloseComport(cport_nr);
108.                    m_textBuffer1->insert_at_cursor("Disconnected from Arduino\n");
109.                    m_textView1.set_buffer(m_textBuffer1);
110.                    m_toggleConnect.set_label("Connect");
111.                    //m_comboBaudrate.set_editable(true);
112.                    //m_comboComport.set_editable(true);
113.                    errorMsg = true;
114.                    runMotorController = false;//close PD control threads
115.                }
116.            }
117.        }
118.
119.        bool mainWindow::onKeyPress( GdkEventKey *event )
120.        {
121.            if (!errorMsg)// only check keys if we are connected to the robot
122.            {
123.                int output = 1500;
124.                //base motor control
125.                if ( (event->keyval == GDK_KEY_s) || (event-
    >keyval == 65464) )//numpad 8
126.                {
127.                    cout << "up" << endl;
128.                    output = 1505;
129.                    itoa(output,vOut1,10);
130.                    SendByte(cport_nr,'a');
131.                    SendBuf(cport_nr,(unsigned char*)vOut1,4);
132.                    SendByte(cport_nr,END_SIGNAL);
133.                    Sleep(100);
134.                    output = 1498;//offset so motor doesnt move
135.                    itoa(output,vOut1,10);
136.                    SendByte(cport_nr,'a');
137.                    SendBuf(cport_nr,(unsigned char*)vOut1,4);
138.                    SendByte(cport_nr,END_SIGNAL);
139.                }
140.                //base motor control
141.                else if ( (event->keyval == GDK_KEY_w) || (event-
    >keyval == 65461) )//numpad 5
142.                {
143.                    cout << "down" << endl;
144.                    output = 1495;
145.                    itoa(output,vOut1,10);
146.                    SendByte(cport_nr,'a');
147.                    SendBuf(cport_nr,(unsigned char*)vOut1,4);
148.                    SendByte(cport_nr,END_SIGNAL);
149.                    Sleep(100);
150.                    output = 1498;//offset so motor doesnt move
151.                    itoa(output,vOut1,10);
152.                    SendByte(cport_nr,'a');
153.                    SendBuf(cport_nr,(unsigned char*)vOut1,4);
154.                    SendByte(cport_nr,END_SIGNAL);
155.                }
156.                //distal motor control
```

```
157.                else if ( (event->keyval == GDK_KEY_a) || (event-
    >keyval == 65460) )//numpad 4
158.                {
159.                    cout << "left" << endl;
160.                    output = 1620;
161.                    itoa(output,vOut2,10);
162.                    SendByte(cport_nr,'b');
163.                    SendBuf(cport_nr,(unsigned char*)vOut2,4);
164.                    SendByte(cport_nr,END_SIGNAL);
165.                    Sleep(10);
166.                    output = 1500;
167.                    itoa(output,vOut2,10);
168.                    SendByte(cport_nr,'b');
169.                    SendBuf(cport_nr,(unsigned char*)vOut2,4);
170.                    SendByte(cport_nr,END_SIGNAL);
171.
172.                }
173.                //distal motor control
174.                else if ( (event->keyval == GDK_KEY_d) || (event-
    >keyval == 65462) )//numpad 6
175.                {
176.                    cout <<  "right" << endl;
177.                    output = 1370;
178.                    itoa(output,vOut2,10);
179.                    SendByte(cport_nr,'b');
180.                    SendBuf(cport_nr,(unsigned char*)vOut2,4);
181.                    SendByte(cport_nr,END_SIGNAL);
182.                    Sleep(10);
183.                    output = 1500;
184.                    itoa(output,vOut2,10);
185.                    SendByte(cport_nr,'b');
186.                    SendBuf(cport_nr,(unsigned char*)vOut2,4);
187.                    SendByte(cport_nr,END_SIGNAL);
188.                }
189.                updateMalletPositionDisplay();
190.            }
191.
192.            //returning true, cancels the propagation of the event
193.            return true;
194.        }
195.
196.
197.        void mainWindow::on_button_clicked(Glib::ustring data)
198.        {
199.            cout << "Hello World - " << data << " was pressed" << endl;
200.        }
201.
202.        void mainWindow::save_log(void) //done for the most part
203.        {
204.            ofstream myfile;
205.            time_t rawTime;
206.            mkdir("./logs/");
207.            struct tm * timeinfo;
208.            time(&rawTime);
209.            timeinfo = localtime( &rawTime);
210.            string filename = "./logs/ActionLog_";
211.            filename += asctime(timeinfo);
212.            replace(filename.begin(), filename.end(), ' ','_');
213.            replace(filename.begin(), filename.end(), ':','_');
214.            replace(filename.begin(), filename.end(), '\n','_');
215.            replace(filename.begin(), filename.end(), '\0','_');
```

```
216.
217.            filename += ".txt";
218.            myfile.open (filename.c_str());
219.            if (myfile.is_open() )
220.            {
221.                myfile << m_textView1.get_buffer()->get_text();
222.                myfile.close();
223.                Gtk::MessageDialog dialog(*this, "Action log saved");
224.                Glib::ustring tempString = "Filename: ";
225.                tempString += filename;
226.                dialog.set_secondary_text(tempString);
227.                dialog.set_title("File Saved");
228.                dialog.run();
229.            }
230.            else
231.            {
232.                Gtk::MessageDialog dialog(*this, "Error");
233.                dialog.set_title("Error");
234.                dialog.set_secondary_text("The file could not be saved.");
235.            }
236.        }
237.
238.        void mainWindow::help(void) //not done, only template
239.        {
240.            dialog.show();
241.            dialog.set_position(Gtk::WIN_POS_CENTER);
242.        }
243.
244.        void mainWindow::helpHide(void)
245.        {
246.            dialog.hide();
247.        }
248.
249.        void mainWindow::start_robot()
250.        {
251.            if (m_toggleMode.get_active() && indicator != 0)
252.            {
253.                m_toggleMode.set_label("Stop");
254.                //startCameraThread
255.                runMotorController = true;
256.                boost::thread threadPD(displayInfo);
257.                m_textBuffer1->insert_at_cursor("Start PD control\n");
258.                m_textView1.set_buffer(m_textBuffer1);
259.
260.                //start calc puck trajectory
261.                runCalcPuckTrajectory = true;
262.                Glib::Thread::create(sigc::mem_fun(*this, &mainWindow::calculatePuckTraj
    ectory),true);
263.                m_textBuffer1-
    >insert_at_cursor("Start Calculate Puck Trajectory thread\n");
264.                m_textView1.set_buffer(m_textBuffer1);
265.            }
266.            else
267.            {
268.                m_toggleMode.set_label("Start");
269.                runMotorController = false;
270.                runCalcPuckTrajectory = false;
271.                m_textBuffer1-
    >insert_at_cursor("Stopped PD control and CalcPuck Trajectory thread\n");
272.                m_textView1.set_buffer(m_textBuffer1);
273.            }
```

```
274.        }
275.
276.        void mainWindow::startCameraThread(void)
277.        {
278.            //cout << "thread camera started" << endl;
279.            m_textBuffer1->insert_at_cursor("Camera Tracking Started\n");
280.            m_textView1.set_buffer(m_textBuffer1);
281.
282.            //check to see if camera is already tracking
283.            if (!cameraTracking)
284.            {
285.                cameraTracking = true;
286.                boost::thread thread3(cameraSensor);
287.            }
288.        }
289.
290.        void mainWindow::updatePositions(void)
291.        {
292.            while (true)
293.            {
294.                m_threadPosDisplay_Dispatcher();
295.                boost::this_thread::sleep(boost::posix_time::millisec(50));
296.            }
297.        }
298.
299.        void mainWindow::updatePosDisplay(void)
300.        {
301.            m_labelMalletX.set_text(Glib::ustring(malletX));
302.            m_labelMalletY.set_text(Glib::ustring(malletY));
303.            m_labelPuckX.set_text(Glib::ustring(puckX));
304.            m_labelPuckY.set_text(Glib::ustring(puckY));
305.            m_labelProjX.set_text(Glib::ustring(projectedX));
306.            m_labelProjY.set_text(Glib::ustring(projectedY));
307.        }
308.
309.        void mainWindow::calibrateHSV(void)
310.        {
311.
312.            m_textBuffer1->insert_at_cursor("Calibrating HSV values\n");
313.            m_textView1.set_buffer(m_textBuffer1);
314.            cameraSensor.calibrateHSV();
315.        }
316.
317.        void mainWindow::calibrateCamera(void)
318.        {
319.            m_textBuffer1->insert_at_cursor("Calibrating Camera units\n");
320.            m_textView1.set_buffer(m_textBuffer1);
321.            //Glib::Thread::create(sigc::mem_fun(*this, &mainWindow::updateProjY),true);
    */
322.            Glib::Thread::create(sigc::mem_fun(cameraSensor, &CameraSensor::calibrateCam
    era),true);
323.            //cameraSensor.calibrateCamera();
324.        }
325.
326.        void mainWindow::setXYPos(void)
327.        {
328.            double x = atof(m_inputDialogTheta1.get_text().c_str())/100.0;
329.            double y = atof(m_inputDialogTheta2.get_text().c_str())/100.0;
330.            double alpha = 0.0;
331.            if (x < 0)
332.            {
```

```
333.                alpha = M_PI - atan(abs(y/x));
334.            }
335.            else
336.            {
337.                alpha = atan(y/x);
338.            }
339.
340.            currentTheta2 = M_PI - acos( (Len1*Len1 + Len2*Len2 - x*x -
    y*y)/(2*Len1*Len2));
341.            currentTheta1 = alpha - acos( (x*x + y*y + Len1*Len1 -
    Len2*Len2)/(2*Len1*sqrt(x*x + y*y)));
342.        }
343.
344.        void mainWindow::setTheta2(void)
345.        {
346.            //currentTheta2 = atof(m_inputDialogTheta2.get_text().c_str());
347.            currentTheta2 = 0.0;
348.            cout << "theta2 set to " << currentTheta2 << endl;
349.            CPhidgetEncoder_setPosition(encoderReader, 1,0);
350.            updateMalletPositionDisplay();
351.        }
352.
353.        void mainWindow::goToXY(void)
354.        {
355.            cout << "going to xy" << endl;
356.            double xe = atof(m_inputDialogX.get_text().c_str())/100.0;
357.            double ye = atof(m_inputDialogY.get_text().c_str())/100.0;
358.            setXY(xe,ye);
359.            cout << "t1 = " << desiredTheta1 << "t2 = " << desiredTheta2 << endl;
360.        }
361.
362.        void mainWindow::calculatePuckTrajectory(void)
363.        {
364.            double tempProjX = HOME_POS[0], tempProjY = HOME_POS[1];
365.            tableLength = atof(m_inputLength.get_text().c_str()),
366.                    tableWidth = atof(m_inputWidth.get_text().c_str()),
367.                    armOffset = atof(m_inputOffset.get_text().c_str());
368.            double timeStep = 0.15;
369.            //arm will only hit in between these two lines
370.            double yUpperHitLine = HOME_POS[1]+0.5;
371.            double yLowerHitLine = HOME_POS[1];
372.            cout << "Calc Puck Traj Thread started" << endl;
373.            //double xTime = 0.0, yTime = 0.0;
374.            double malletXPos = boost::lexical_cast<double>(malletX)/100.0;
375.            double malletYPos = boost::lexical_cast<double>(malletY)/100.0;
376.            double distanceBetweenPuckAndMallet = 0.0;
377.            while (runCalcPuckTrajectory)
378.            {
379.
380.                if (puckVelocity[1] < -0.1) //puck is moving towards arm
381.                {
382.                    //predict where puck will be in timeStep seconds
383.                    //smart time step
384.                    malletXPos = boost::lexical_cast<double>(malletX)/100.0;
385.                    malletYPos = boost::lexical_cast<double>(malletY)/100.0;
386.                    distanceBetweenPuckAndMallet = sqrt( (puckPositionX[2]-
    malletXPos)*(puckPositionX[2]-malletXPos) +
387.                                (puckPositionY[2]-malletYPos)*(puckPositionY[2]-
    malletYPos))/abs(wcurrent2*(Len1+Len2));
388.                    //cout << distanceBetweenPuckAndMallet << endl;
389.                    cout << malletXPos << endl;
```

```
390.                    if (distanceBetweenPuckAndMallet > .3)
391.                    {
392.                        timeStep = 0.15;
393.                    }
394.                    else
395.                    {
396.                        timeStep = 0.10;
397.                    }
398.                    tempProjX = puckVelocity[0]*timeStep + puckPositionX[2];
399.                    tempProjY = puckVelocity[1]*timeStep + puckPositionY[2];
400.                    //malletYPos = boost::lexical_cast<double>(malletY)/100.0;;
401.                    if (abs(tempProjX) > (tableWidth/2-
    .12) || tempProjY < yLowerHitLine || tempProjY > yUpperHitLine)
402.                    {
403.                        //go to home position
404.                        cout << "puck moving towards me, i'm going to home " << endl;
405.                        projectedX = boost::lexical_cast<string>(int(HOME_POS[0]*100));

406.                        projectedY = boost::lexical_cast<string>(int(HOME_POS[1]*100));

407.                        setXY(HOME_POS[0], HOME_POS[1]);
408.                    }
409.                    else
410.                    {
411.                        //go to projected location
412.                        cout << "go to proj loc" << endl;
413.                        projectedX = boost::lexical_cast<string>(int(tempProjX*100));
414.                        projectedY = boost::lexical_cast<string>(int(tempProjY*100));
415.                        Glib::ustring tempString = "Moving to: (" + projectedX + "," + p
    rojectedY + ")\n";
416.                        m_textBuffer1->insert_at_cursor(tempString);
417.                        m_textView1.set_buffer(m_textBuffer1);
418.                        setXY(tempProjX,tempProjY);
419.                    }
420.                }
421.                else if ( (puckPositionY[2] < yUpperHitLine) && ( abs(puckVelocity[1]) <
     0.05) && (puckPositionY[2] > yLowerHitLine) )
422.                {
423.                    cout << "puck not moving" << endl;
424.                    if (abs(puckVelocity[0]) < 0.05) //no movement in x or y
425.                    {
426.                        projectedX = boost::lexical_cast<string>(int(puckPositionX[2]*10
    0));
427.                        projectedY = boost::lexical_cast<string>(int(puckPositionY[2]*10
    0));
428.                        if (abs(puckPositionX[2]) > (tableWidth/2-.15))
429.                        {
430.                            //go to home position
431.                            cout << "home 2" << endl;
432.                            projectedX = boost::lexical_cast<string>(int(HOME_POS[0]*100
    ));
433.                            projectedY = boost::lexical_cast<string>(int(HOME_POS[1]*100
    ));
434.                            setXY(HOME_POS[0], HOME_POS[1]);
435.                        }
436.                        else
437.                        {
438.                            //go to projected location
439.                            cout << "go to puck" << endl;
440.                            projectedX = boost::lexical_cast<string>(int(puckPositionX[2
    ]*100));
```

```
441.                             projectedY = boost::lexical_cast<string>(int(puckPositionY[2
    ]*100));
442.                             setXY(puckPositionX[2],puckPositionY[2]);//go to puck's loca
    tion, assume no movement
443.                         }
444.                     }
445.                     else //puck is moving horizontally
446.                     {
447.                         tempProjX = puckVelocity[0]*timeStep + puckPositionX[2];
448.                         projectedX = boost::lexical_cast<string>(int(tempProjX*100));
449.                         projectedY = boost::lexical_cast<string>(int(puckPositionY[2]*10
    0));
450.                         setXY(tempProjX, puckPositionY[2]);
451.                     }
452.                 }
453.                 else //puck is moving away or not on arm side
454.                 {
455.                     //go to home position
456.                     cout << "im home" << endl;
457.                     projectedX = boost::lexical_cast<string>(int(HOME_POS[0]*100));
458.                     projectedY = boost::lexical_cast<string>(int(HOME_POS[1]*100));
459.                     setXY(HOME_POS[0],HOME_POS[1]);//go to home position
460.                 }
461.                 //this sort of works
462.                 /*//cout << puckVelocity[1] << endl;
463.                 if (puckVelocity[1] < -
    0.1) //if puck is moving towards arm, y velocity is negative
464.                 {
465.                     //final goal: hit puck in hit-box
466.                     cout<<"calcing traj" << endl;
467.                     //find out where puck will hit wall, if any
468.                     //x time to wall
469.                     if (abs(puckVelocity[0]) < SPEED_TOL) //puck is not moving in x
470.                     {
471.                         xTime = 100000;//means we will never reach it
472.                     }
473.                     else //puck is moving in x
474.                     {
475.                         //xTime is when we'll hit either side of the table
476.                         xTime = abs((tableWidth/2.0 -
     abs(puckPositionX[2]))/puckVelocity[0]);
477.                     }
478.
479.                     //yTime to upper hit line
480.                     yTime = (yUpperHitLine - puckPositionY[2])/puckVelocity[1];
481.                     if (yTime < 0.0)
482.                     {
483.                         //puck is pass the upper hit line
484.                         //check for lower hit line
485.                         yTime = (yLowerHitLine - puckPositionY[2])/puckVelocity[1];
486.                         if (yTime < 0.0)
487.                         {
488.                             yTime = 1000000;//means we will never reach it
489.                         }
490.                     }
491.                     if (xTime < yTime) //hit sides first, ignore
492.                     {
493.                         //go to home position
494.                         cout << "go home" << endl;
495.                         projectedX = boost::lexical_cast<string>(int(HOME_POS[0]*100));
496.                         projectedY = boost::lexical_cast<string>(int(HOME_POS[1]*100));
```

```
497.                         setXY(HOME_POS[0], HOME_POS[1]);
498.                     }
499.                     else if (yTime < 2.0) //if puck will be between lines and yTime < 2.
    0 seconds
500.                     {
501.                         //hit puck where it'll be in 2sec
502.                         tempProjX = puckVelocity[0]*0.1 + puckPositionX[2];
503.                         tempProjY = puckVelocity[1]*0.1 + puckPositionY[2];
504.                         if (abs(tempProjX) > (tableWidth/2-
    .15) || tempProjY < yLowerHitLine || tempProjY > yUpperHitLine)
505.                         {
506.                             //go to home position
507.                             cout << "home 2" << endl;
508.                             projectedX = boost::lexical_cast<string>(int(HOME_POS[0]*100
    ));
509.                             projectedY = boost::lexical_cast<string>(int(HOME_POS[1]*100
    ));
510.                             setXY(HOME_POS[0], HOME_POS[1]);
511.                         }
512.                         else
513.                         {
514.                             //go to projected location
515.                             cout << "go to proj loc" << endl;
516.                             projectedX = boost::lexical_cast<string>(int(tempProjX*100))
    ;
517.                             projectedY = boost::lexical_cast<string>(int(tempProjY*100))
    ;
518.                             Glib::ustring tempString = "Moving to: (" + projectedX + ","
     + projectedY + ")\n";
519.                             m_textBuffer1->insert_at_cursor(tempString);
520.                             m_textView1.set_buffer(m_textBuffer1);
521.                             setXY(tempProjX,tempProjY);
522.                         }
523.                     }
524.                     else
525.                     {
526.                         //go to home position
527.                         cout << "home 3" << endl;
528.                         projectedX = boost::lexical_cast<string>(int(HOME_POS[0]*100));
529.                         projectedY = boost::lexical_cast<string>(int(HOME_POS[1]*100));
530.                         setXY(HOME_POS[0], HOME_POS[1]);
531.                     }
532.                 }
533.                 //if puck is on arm side and not moving towards arm
534.                 else if ( (puckPositionY[2] < (armOffset + (tableLength/2.0))) && ( abs(
    puckVelocity[1]) < 0.05) && (puckPositionY[2] > yLowerHitLine) )
535.                 {
536.                     cout << "puck not moving" << endl;
537.                     if (abs(puckVelocity[0]) < 0.05) //no movement in x or y
538.                     {
539.                         projectedX = boost::lexical_cast<string>(int(puckPositionX[2]*10
    0));
540.                         projectedY = boost::lexical_cast<string>(int(puckPositionY[2]*10
    0));
541.                         if (abs(puckPositionX[2]) > (tableWidth-.05))
542.                         {
543.                             //go to home position
544.                             cout << "home 2" << endl;
545.                             projectedX = boost::lexical_cast<string>(int(HOME_POS[0]*100
    ));
```

```
546.                            projectedY = boost::lexical_cast<string>(int(HOME_POS[1]*100
    ));
547.                            setXY(HOME_POS[0], HOME_POS[1]);
548.                        }
549.                        else
550.                        {
551.                            //go to projected location
552.                            cout << "go to puck" << endl;
553.                            projectedX = boost::lexical_cast<string>(int(puckPositionX[2
    ]*100));
554.                            projectedY = boost::lexical_cast<string>(int(puckPositionY[2
    ]*100));
555.                            setXY(puckPositionX[2],puckPositionY[2]);//go to puck's loca
    tion, assume no movement
556.                        }
557.                    }
558.                    else //puck is moving horizontally
559.                    {
560.                        tempProjX = puckVelocity[0]*2.0 + puckPositionX[2];
561.                        projectedX = boost::lexical_cast<string>(int(tempProjX*100));
562.                        projectedY = boost::lexical_cast<string>(int(puckPositionY[2]*10
    0));
563.                        setXY(tempProjX, puckPositionY[2]);
564.                    }
565.                }
566.                else //puck is moving away or not on arm side
567.                {
568.                    //go to home position
569.                    cout << "im home" << endl;
570.                    projectedX = boost::lexical_cast<string>(int(HOME_POS[0]*100));
571.                    projectedY = boost::lexical_cast<string>(int(HOME_POS[1]*100));
572.                    setXY(HOME_POS[0],HOME_POS[1]);//go to home position
573.                }*/
574.                boost::this_thread::sleep(boost::posix_time::millisec(50));
575.            }
576.            cout << "calc puck trajectory thread end" << endl;
577.        }
```

```
1.  /*
2.   * CameraSensor.h
3.   *
4.   *  Created on: Jan 28, 2013
5.   *      Author: bvang13
6.   */
7.
8.  #ifndef CAMERASENSOR_H_
9.  #define CAMERASENSOR_H_
10.
11. #include <iostream>
12. #include <fstream>
13. #include "opencv2/opencv.hpp"
14. #include <opencv2/highgui/highgui.hpp>
15. #include <opencv2/core/core.hpp>
16. #include <opencv2/imgproc/imgproc.hpp>
17. #include <stdio.h>
18. #include <stdlib.h>
19. #include <string.h>
```

```
20. #include <assert.h>
21. #include <math.h>
22. #include <float.h>
23. #include <limits.h>
24. #include <time.h>
25. #include <ctype.h>
26. #include <windows.h>
27. #include "motorController.h"
28.
29. using namespace cv;
30. using namespace std;
31.
32. class CameraSensor {
33. public:
34.     CameraSensor();
35.     virtual ~CameraSensor();
36.     void processImage(Mat frame);
37.     void operator()();
38.     void shutDown();
39.     void calibrateHSV();
40.     void updatePuckPosition();//need to finish, will track and update the position of t
    he puck
41.     void calibrateCamera();//calibrate px to meters
42.
43. private:
44.     void loadData(void);
45.     Mat image;
46.     bool killThread;
47.     int hsv_min[3], hsv_max[3];
48.
49. };
50.
51. #endif /* CAMERASENSOR_H_ */
52.
53. /*
54.  * CameraSensor.cpp
55.  *
56.  *  Created on: Jan 28, 2013
57.  *      Author: bvang13
58.  */
59.
60. #include "CameraSensor.h"
61.
62. CameraSensor::CameraSensor() {
63.     hsv_min[0] = 0;
64.     hsv_min[1] = 0;
65.     hsv_min[2] = 0;
66.     hsv_max[0] = 0;
67.     hsv_max[1] = 0;
68.     hsv_max[2] = 0;
69.     loadData();
70. }
71.
72. CameraSensor::~CameraSensor() {
73. }
74.
75. void CameraSensor::loadData(void)
76. {
77.     string line;
78.     int hsv[6];
79.     ifstream myfile ("hsvConfig.ini");
```

```
80.     if (myfile.is_open())
81.     {
82.         for (int i = 0; i < 6; i++)
83.         {
84.             getline (myfile, line);
85.             const char *num = line.c_str();
86.             hsv[i] = atoi( num );
87.         }
88.         hsv_min[0] = hsv[0];
89.         hsv_min[1] = hsv[1];
90.         hsv_min[2] = hsv[2];
91.         hsv_max[0] = hsv[3];
92.         hsv_max[1] = hsv[4];
93.         hsv_max[2] = hsv[5];
94.         myfile.close();
95.         cout << "hsvConfig.ini was loaded successfully." << endl;
96.     }
97.     else
98.     {
99.         cout << "Cannot open hsvConfig.ini or it does not exist." << endl;
100.        }
101.
102.        //load all info of positionConfig.ini file
103.        ifstream myPos ("positionConfig.ini");
104.        if (myPos.is_open())
105.        {
106.            for (int i = 0; i < 24; i++)
107.            {
108.                getline (myPos, line);
109.                const char *num = line.c_str();
110.                if (i < 4)
111.                {
112.                    calibratePoint1[i] = atof(num);
113.                }
114.                else if (i < 8)
115.                {
116.                    calibratePoint2[i-4] = atof(num);
117.                }
118.                else if (i < 12)
119.                {
120.                    calibratePoint3[i-8] = atof(num);
121.                }
122.                else if (i < 16)
123.                {
124.                    calibratePoint4[i-12] = atof(num);
125.                }
126.                else if (i == 16)
127.                {
128.                    calibrationYSlope = atof(num);
129.                }
130.                else if (i == 17)
131.                {
132.                    calibrationYConstant = atof(num);
133.                }
134.                else if (i < 20)
135.                {
136.                    calibrationXSlopePX[i-18] = atof(num);
137.                }
138.                else if (i < 22)
139.                {
140.                    calibrationXConstantPX[i-20] = atof(num);
```

```
141.                    }
142.                    else if (i == 22)
143.                    {
144.                        currentTheta1 = atof(num);
145.                        desiredTheta1 = currentTheta1;
146.                    }
147.                    else if (i == 23)
148.                    {
149.                        currentTheta2 = atof(num);
150.                        desiredTheta2 = currentTheta2;
151.                    }
152.                }
153.                myPos.close();
154.                cout << "positionConfig.ini was loaded successfully." << endl;
155.            }
156.            else
157.            {
158.                cout << "Cannot open positionConfig.ini or it does not exist." << endl;

159.            }
160.        }
161.
162.
163.        void CameraSensor::processImage(Mat frame)
164.        {
165.            vector<vector<Point> > contours;
166.            Mat bimage = image >= 70;
167.            findContours(bimage, contours,CV_RETR_LIST,CV_CHAIN_APPROX_NONE);
168.            Mat cimage = Mat::zeros(bimage.size(), CV_8UC3);
169.
170.            double maxContour = -1.0;
171.            double maxArea = -1.0;
172.            for (size_t i = 0; i < contours.size(); i ++)
173.            {
174.                size_t count = contours[i].size();
175.                if ( count < 6 )
176.                    continue;
177.
178.                if ( contourArea(contours[i]) < maxArea)
179.                    continue;
180.
181.                maxArea = contourArea(contours[i]);
182.                maxContour = i;
183.            }
184.
185.            //only print the largest circle's position
186.            if ((maxContour != -1.0) &&(maxArea != -1.0)) {
187.                Mat pointsf;
188.                Mat(contours[maxContour]).convertTo(pointsf, CV_32F);
189.                RotatedRect box = fitEllipse(pointsf);
190.                ellipse(frame,box.center,box.size*0.5f, box.angle, 0, 360, Scalar(0,256,
    0),1,CV_AA);
191.                cout <<box.center<<endl;
192.            }
193.        }
194.
195.        void CameraSensor::calibrateHSV()
196.        {
197.            //Image variables
198.            CvCapture* capture = cvCaptureFromCAM( 1 );
199.            int h1=hsv_min[0];int s1=hsv_min[1];int v1=hsv_min[2];
```

```
200.        int h2=hsv_max[0];int s2=hsv_max[1];int v2=hsv_max[2];
201.        CvSize size = cvSize(640,480);
202.        cvNamedWindow("cnt",CV_WINDOW_AUTOSIZE);
203.        //track bars
204.        cvCreateTrackbar("H1","cnt",&h1,510,0);
205.        cvCreateTrackbar("S1","cnt",&s1,510,0);
206.        cvCreateTrackbar("V1","cnt",&v1,510,0);
207.        cvCreateTrackbar("H2","cnt",&h2,510,0);
208.        cvCreateTrackbar("S2","cnt",&s2,510,0);
209.        cvCreateTrackbar("V2","cnt",&v2,510,0);
210.        IplImage* hsvimg=cvCreateImage(size,IPL_DEPTH_8U,3);
211.        IplImage* thresh=cvCreateImage(size,IPL_DEPTH_8U,1);
212.        IplImage* frame = cvQueryFrame( capture );
213.        if( !capture )
214.        {
215.            fprintf( stderr, "ERROR: capture is NULL \n" );
216.            getchar();
217.            return;
218.            //return -1;
219.        }
220.        while(true)
221.        {
222.            frame= cvQueryFrame(capture);
223.            if( !frame )
224.            {
225.                fprintf( stderr, "ERROR: frame is null...\n" );
226.                getchar();
227.                break;
228.            }
229.            //Windows
230.            cvNamedWindow("Original Image",CV_WINDOW_AUTOSIZE);
231.            cvNamedWindow("Thresholded Image",CV_WINDOW_AUTOSIZE);
232.            //Changing into HSV plane
233.            cvCvtColor(frame,hsvimg,CV_BGR2HSV);
234.            //Thresholding the image
235.            cvInRangeS(hsvimg,cvScalar(h1,s1,v1),cvScalar(h2,s2,v2),thresh);
236.            //Showing the images
237.            cvShowImage("Original Image",frame);
238.            cvShowImage("Thresholded Image",thresh);
239.            //Waiting for user to press any key
240.            if( (cvWaitKey(10) & 255) == 27 ) break;
241.        }
242.        hsv_min[0] = h1;
243.        hsv_min[1] = s1;
244.        hsv_min[2] = v1;
245.        hsv_max[0] = h2;
246.        hsv_max[1] = s2;
247.        hsv_max[2] = v2;
248.        ofstream myfile ("hsvConfig.ini");
249.        if (myfile.is_open())
250.        {
251.            myfile << h1 <<"\n";
252.            myfile << s1 <<"\n";
253.            myfile << v1 <<"\n";
254.            myfile << h2 <<"\n";
255.            myfile << s2 <<"\n";
256.            myfile << v2 <<"\n";
257.            myfile.close();
258.            cout << "Configuration file updated." << endl;
259.        }
260.        else
```

```
261.            {
262.                cout << "Cannot write to file." << endl;
263.            }
264.            cvReleaseCapture( &capture);
265.            cvDestroyAllWindows();
266.        }
267.
268.        void CameraSensor::calibrateCamera()
269.        {
270.            // Default capture size - 640x480
271.            CvSize size = cvSize(640,480);
272.            // Open capture device. 0 is /dev/video0, 1 is /dev/video1, etc.
273.            CvCapture* capture = cvCaptureFromCAM( 1 );
274.            if( !capture )
275.            {
276.                fprintf( stderr, "ERROR: capture is NULL \n" );
277.                getchar();
278.                return;
279.            }
280.            // Create a window in which the captured images will be presented
281.            cvNamedWindow( "Camera", CV_WINDOW_AUTOSIZE );
282.            // Detect puck color
283.            CvScalar hsv_min_scalar = cvScalar(hsv_min[0], hsv_min[1], hsv_min[2]);
284.            CvScalar hsv_max_scalar = cvScalar(hsv_max[0], hsv_max[1], hsv_max[2]);
285.            IplImage*  hsv_frame    = cvCreateImage(size, IPL_DEPTH_8U, 3);
286.            IplImage*  thresholded   = cvCreateImage(size, IPL_DEPTH_8U, 1);
287.            IplImage* frame = cvQueryFrame(capture);
288.            while( true )
289.            {
290.                // Get one frame
291.                frame = cvQueryFrame( capture );
292.                if( !frame )
293.                {
294.                    fprintf( stderr, "ERROR: frame is null...\n" );
295.                    getchar();
296.                    break;
297.                }
298.                // Covert color space to HSV as it is much easier to filter colors in th
    e HSV color-space.
299.                cvCvtColor(frame, hsv_frame, CV_BGR2HSV);
300.                // Filter out colors which are out of range.
301.                cvInRangeS(hsv_frame, hsv_min_scalar, hsv_max_scalar, thresholded);
302.                // Memory for hough circles
303.                CvMemStorage* storage = cvCreateMemStorage(0);
304.                // hough detector works better with some smoothing of the image
305.                cvSmooth( thresholded, thresholded, CV_GAUSSIAN, 9, 9 );
306.                image = Mat(thresholded);
307.                //processImage(Mat(frame));
308.                vector<vector<Point> > contours;
309.                Mat bimage = image >= 70;
310.                findContours(bimage, contours,CV_RETR_LIST,CV_CHAIN_APPROX_NONE);
311.                Mat cimage = Mat::zeros(bimage.size(), CV_8UC3);
312.                double maxContour = -1.0;
313.                double maxArea = -1.0;
314.                for (size_t i = 0; i < contours.size(); i ++)
315.                {
316.                    size_t count = contours[i].size();
317.                    if ( count < 6 )
318.                        continue;
319.
320.                    if ( contourArea(contours[i]) < maxArea)
```

```
321.                         continue;
322.
323.                     maxArea = contourArea(contours[i]);
324.                     maxContour = i;
325.                 }
326.                 Mat tempFrame = Mat(frame);
327.                 //only print the largest circle's position
328.                 RotatedRect box;
329.                 if ((maxContour != -1.0) &&(maxArea != -1.0)) {
330.                     Mat pointsf;
331.                     Mat(contours[maxContour]).convertTo(pointsf, CV_32F);
332.                     box = fitEllipse(pointsf);
333.                     ellipse(tempFrame,box.center,box.size*0.5f, box.angle, 0, 360, Scala
    r(0,256,0),1,CV_AA);
334.                     //cout <<box.center<<endl;
335.                 }
336.                 cvShowImage( "Camera", frame ); // Original stream with detected ball ov
    erlay
337.                 //cvShowImage( "HSV&", hsv_frame); // Original stream in the HSV color s
    pace
338.                 cvShowImage( "After Color Filtering", thresholded ); // The stream after
     color filtering
339.                 cvReleaseMemStorage(&storage);
340.                 // Do not release the frame!
341.                 int keyPressed = cvWaitKey(10)&255;
342.                 //cout<<keyPressed<<endl;
343.                 if( keyPressed == 27 )
344.                 {
345.                     double yAvg[4];
346.                     yAvg[0] = (calibratePoint1[1] + calibratePoint2[1])/2.0; //avg Y px
    for bottom
347.                     yAvg[1] = (calibratePoint1[3] + calibratePoint2[3])/2.0; //avg Y rea
    l for bottm
348.                     yAvg[2] = (calibratePoint3[1] + calibratePoint4[1])/2.0; //avg Y px
    for top
349.                     yAvg[3] = (calibratePoint3[3] + calibratePoint4[3])/2.0; //avg Y rea
    l for top
350.
351.                     //finding line equation for Y conversion, this won't change
352.                     calibrationYSlope = (yAvg[1]-yAvg[3])/(yAvg[0]-
    yAvg[2]); // slope real/px
353.                     calibrationYConstant = yAvg[1] -
     calibrationYSlope*yAvg[0]; //constant for Y equation
354.
355.                     //finding x px lines for left and right
356.                     //assume x are of same length in x and y direction
357.                     //x is dependent, y is given
358.                     calibrationXSlopePX[0] = (calibratePoint1[0]-
    calibratePoint3[0])/(calibratePoint1[1]-calibratePoint3[1]);//using left points
359.                     calibrationXSlopePX[1] = (calibratePoint2[0]-
    calibratePoint4[0])/(calibratePoint2[1]-calibratePoint4[1]);//using right points
360.                     calibrationXConstantPX[0] = calibratePoint1[0] -
     calibrationXSlopePX[0]*calibratePoint1[1];
361.                     calibrationXConstantPX[1] = calibratePoint2[0] -
     calibrationXSlopePX[1]*calibratePoint2[1];
362.                     //found xpx = f(ypx), this will allow us to see how the real length
    scales in the camera view
363.
364.                     cout <<"Calibration done!" << endl;
365.                     break;
366.                 }
```

```
367.                else if ( keyPressed == 106) //j is pressed, bottom left
368.                {
369.                    calibratePoint1[0] = box.center.x; //px X
370.                    calibratePoint1[1] = box.center.y; //px Y
371.                    calibratePoint1[2] = boost::lexical_cast<double>(malletX)/100.0; //r
    eal x (m)
372.                    calibratePoint1[3] = boost::lexical_cast<double>(malletY)/100.0; //r
    eal y (m)
373.                    cout<<"calibration point 1 stored" << endl;
374.
375.                }
376.                else if ( keyPressed == 107) //k is pressed, bottom right
377.                {
378.                    calibratePoint2[0] = box.center.x; //px X
379.                    calibratePoint2[1] = box.center.y; //px Y
380.                    calibratePoint2[2] = boost::lexical_cast<double>(malletX)/100.0; //r
    eal x (m)
381.                    calibratePoint2[3] = boost::lexical_cast<double>(malletY)/100.0; //r
    eal y (m)
382.                    cout<<"calibration point 2 stored" << endl;
383.                }
384.                /*      else if ( keyPressed == 108) //l is pressed
385.                {
386.
387.                }*/
388.                else if ( keyPressed == 117) //u is pressed, top left
389.                {
390.                    calibratePoint3[0] = box.center.x; //px X
391.                    calibratePoint3[1] = box.center.y; //px Y
392.                    calibratePoint3[2] = boost::lexical_cast<double>(malletX)/100.0; //r
    eal x (m)
393.                    calibratePoint3[3] = boost::lexical_cast<double>(malletY)/100.0; //r
    eal y (m)
394.                    cout<<"calibration point 3 stored" << endl;
395.                }
396.                else if ( keyPressed == 105) //i is pressed, top right
397.                {
398.                    calibratePoint4[0] = box.center.x; //px X
399.                    calibratePoint4[1] = box.center.y; //px Y
400.                    calibratePoint4[2] = boost::lexical_cast<double>(malletX)/100.0; //r
    eal x (m)
401.                    calibratePoint4[3] = boost::lexical_cast<double>(malletY)/100.0; //r
    eal y (m)
402.                    cout<<"calibration point 4 stored" << endl;
403.                }
404.                /*else if ( keyPressed == 111) //o is pressed
405.                {
406.
407.                }*/
408.
409.            }
410.            cvDestroyAllWindows();
411.            cvReleaseCapture( &capture );
412.        }
413.
414.        void CameraSensor::shutDown()
415.        {
416.            killThread = true;
417.        }
418.
419.        //detect puck and convert its pixel position into meters
```

```
420.        void CameraSensor::operator()()
421.        {
422.            // Default capture size - 640x480
423.            CvSize size = cvSize(640,480);
424.            // Open capture device. 0 is /dev/video0, 1 is /dev/video1, etc.
425.            CvCapture* capture = cvCaptureFromCAM( 1 );
426.            if( !capture )
427.            {
428.                fprintf( stderr, "ERROR: capture is NULL \n" );
429.                getchar();
430.                return;
431.            }
432.            // Create a window in which the captured images will be presented
433.            cvNamedWindow( "Camera", CV_WINDOW_AUTOSIZE );
434.            // Detect puck
435.            CvScalar hsv_min_scalar = cvScalar(hsv_min[0], hsv_min[1], hsv_min[2]);
436.            CvScalar hsv_max_scalar = cvScalar(hsv_max[0], hsv_max[1], hsv_max[2]);
437.            IplImage*  hsv_frame    = cvCreateImage(size, IPL_DEPTH_8U, 3);
438.            IplImage*  thresholded   = cvCreateImage(size, IPL_DEPTH_8U, 1);
439.            IplImage* frame = cvQueryFrame( capture );
440.            //reusable variables
441.            double yPxCurrent = 0.0, xPxCurrent = 0.0, xPx[2] = {0.0,0.0};
442.            boost::timer puckTimer; //start timer to keep track of time for tracking puc
    k
443.            while( true )
444.            {
445.                // Get one frame
446.                frame = cvQueryFrame( capture );
447.                if( !frame )
448.                {
449.                    fprintf( stderr, "ERROR: frame is null...\n" );
450.                    getchar();
451.                    break;
452.                }
453.                // Covert color space to HSV as it is much easier to filter colors in th
    e HSV color-space.
454.                cvCvtColor(frame, hsv_frame, CV_BGR2HSV);
455.                // Filter out colors which are out of range.
456.                cvInRangeS(hsv_frame, hsv_min_scalar, hsv_max_scalar, thresholded);
457.                // Memory for hough circles
458.                CvMemStorage* storage = cvCreateMemStorage(0);
459.                // hough detector works better with some smoothing of the image
460.                cvSmooth( thresholded, thresholded, CV_GAUSSIAN, 9, 9 );
461.                image = Mat(thresholded);
462.                vector<vector<Point> > contours;
463.                Mat bimage = image >= 70;
464.                findContours(bimage, contours,CV_RETR_LIST,CV_CHAIN_APPROX_NONE);
465.                Mat cimage = Mat::zeros(bimage.size(), CV_8UC3);
466.                double maxContour = -1.0;
467.                double maxArea = -1.0;
468.                for (size_t i = 0; i < contours.size(); i ++)
469.                {
470.                    size_t count = contours[i].size();
471.                    if ( count < 6 )
472.                        continue;
473.
474.                    if ( contourArea(contours[i]) < maxArea)
475.                        continue;
476.
477.                    maxArea = contourArea(contours[i]);
478.                    maxContour = i;
```

```
479.                }
480.                Mat tempFrame = Mat(frame);
481.                //only print the largest circle's position
482.                RotatedRect box;
483.                //reset location
484.                xPxCurrent = -1.0;
485.                yPxCurrent = -1.0;
486.                if ((maxContour != -1.0) &&(maxArea != -1.0)) {
487.                    Mat pointsf;
488.                    Mat(contours[maxContour]).convertTo(pointsf, CV_32F);
489.                    box = fitEllipse(pointsf);
490.                    ellipse(tempFrame,box.center,box.size*0.5f, box.angle, 0, 360, Scala
    r(0,256,0),1,CV_AA);
491.                    xPxCurrent = box.center.x;
492.                    yPxCurrent = box.center.y;
493.                }
494.                //store x positions
495.                puckPositionX[0] = puckPositionX[1];
496.                puckPositionX[1] = puckPositionX[2];
497.                //store y positions
498.                puckPositionY[0] = puckPositionY[1];
499.                puckPositionY[1] = puckPositionY[2];
500.                //store time elapsed since we started tracking
501.                puckTime[0] = puckTime[1];
502.                puckTime[1] = puckTime[2];
503.                puckTime[2] = puckTimer.elapsed();//seconds
504.                if (xPxCurrent != -1.0 && yPxCurrent != -1.0)
505.                {
506.                    //calc puck position base on camera
507.                    //first find x length base on pixels
508.                    xPx[0] = calibrationXSlopePX[0]*yPxCurrent + calibrationXConstantPX[
    0]; //left side x px
509.                    xPx[1] = calibrationXSlopePX[1]*yPxCurrent + calibrationXConstantPX[
    1]; //right side x px
510.                    //now find xreal = f(xpx) using calibration length and left/right x
    px, result = m and b
511.                    calibrationXSlope = (calibratePoint1[2]-calibratePoint2[2])/(xPx[0]-
    xPx[1]);
512.                    calibrationXConstant = calibratePoint1[2] -
     calibrationXSlope*xPx[0];
513.                    //find what current xPx and yPx of camera are in real units
514.
515.                    puckPositionX[2] = calibrationXSlope*xPxCurrent + calibrationXConsta
    nt;//m
516.                    puckPositionY[2] = calibrationYSlope*yPxCurrent + calibrationYConsta
    nt;//m
517.
518.                    //velocity vector
519.                    puckVelocity[0] = (puckPositionX[2] -
     puckPositionX[0])/(puckTime[2]-puckTime[1]);//x dir
520.                    puckVelocity[1] = (puckPositionY[2] -
     puckPositionY[0])/(puckTime[2]-puckTime[1]);//y dir
521.
522.                    if (puckVelocity[0] > 10.0)
523.                    {
524.                        puckVelocity[0] = 10.0;
525.                    }
526.                    else if (puckVelocity[0] < -10.0)
527.                    {
528.                        puckVelocity[0] = -10.0;
529.                    }
```

```
530.
531.                     if (puckVelocity[1] > 10.0)
532.                     {
533.                         puckVelocity[1] = 10.0;
534.                     }
535.                     else if (puckVelocity[1] < -10.0)
536.                     {
537.                         puckVelocity[1] = -10.0;
538.                     }
539.
540.                 }
541.                 else //no puck found, will direct arm to home location
542.                 {
543.                     //set puck position to home
544.                     puckPositionX[2] = HOME_POS[0];
545.                     puckPositionY[2] = HOME_POS[1];
546.
547.                     //set puck velocity to zero
548.                     puckVelocity[0] = 0.0;
549.                     puckVelocity[1] = 0.0;
550.                 }
551.                 puckX = boost::lexical_cast<string>(int(puckPositionX[2]*100)); // (cm)

552.                 puckY = boost::lexical_cast<string>(int(puckPositionY[2]*100)); // (cm)

553.                 cvShowImage( "Camera", frame ); // Original stream with detected ball ov
    erlay
554.                 //cvShowImage( "HSV&", hsv_frame); // Original stream in the HSV color s
    pace
555.                 cvShowImage( "After Color Filtering", thresholded ); // The stream after
     color filtering
556.                 cvReleaseMemStorage(&storage);
557.                 // Do not release the frame!
558.                 if ( (cvWaitKey(10)&255) == 27)
559.                 {
560.                     cameraTracking = false;
561.                     break;
562.                 }
563.                 //if (killThread)   break;
564.
565.             }
566.             cvDestroyAllWindows();
567.             cvReleaseCapture( &capture );
568.         }
```

```
1.  /*
2.   * motorController.h
3.   *
4.   *  Created on: Jan 28, 2013
5.   *      Author: bvang13
6.   */
7.
8.  #ifndef MOTORCONTROLLER_H_
9.  #define MOTORCONTROLLER_H_
10.
11. #define BOOST_THREAD_USE_LIB
12. #include <boost/thread.hpp>
```

```
13. #include <boost/date_time.hpp>
14. #include <iostream>
15. #include <string>
16. #include <stdio.h>
17. #include <phidget21.h>
18. #include <windows.h>
19. #include <math.h>
20. #include "rs232.h"
21. #include "global.h"
22. using namespace std;
23.
24. int CCONV AttachHandler(CPhidgetHandle ENC, void *userptr);
25. int CCONV DetachHandler(CPhidgetHandle ENC, void *userptr);
26. int CCONV ErrorHandler(CPhidgetHandle ENC, void *userptr, int ErrorCode, const char *De
    scription);
27. int CCONV InputChangeHandler(CPhidgetEncoderHandle ENC, void *usrptr, int Index, int St
    ate);
28. int CCONV PositionChangeHandler(CPhidgetEncoderHandle ENC, void *usrptr, int Index, int
     Time, int RelativePosition);
29. int display_properties(CPhidgetEncoderHandle phid);
30. CPhidgetEncoderHandle encoder_simple();
31. void PDCtrl1();
32. void PDCtrl2();
33. void displayInfo();
34. void setXY(double x, double y);
35. void updateMalletPositionDisplay(void);
36.
37.
38. #endif /* MOTORCONTROLLER_H_ */
39.
40. /*
41.  * motorController.cpp
42.  *
43.  *  Created on: Jan 28, 2013
44.  *      Author: bvang13
45.  */
46.
47. #include "motorController.h"
48.
49. int CCONV AttachHandler(CPhidgetHandle ENC, void *userptr)
50. {
51.     int serialNo;
52.     CPhidget_DeviceID deviceID;
53.     int i, inputcount;
54.
55.     CPhidget_getSerialNumber(ENC, &serialNo);
56.
57.     //Retrieve the device ID and number of encoders so that we can set the enables if n
    eeded
58.     CPhidget_getDeviceID(ENC, &deviceID);
59.     CPhidgetEncoder_getEncoderCount((CPhidgetEncoderHandle)ENC, &inputcount);
60.     printf("Encoder %10d attached! \n", serialNo);
61.
62.     //the 1047 requires enabling of the encoder inputs, so enable them if this is a 104
    7
63.     if (deviceID == PHIDID_ENCODER_HS_4ENCODER_4INPUT)
64.     {
65.         printf("Encoder requires Enable. Enabling inputs....\n");
66.         for (i = 0 ; i < inputcount ; i++)
67.             CPhidgetEncoder_setEnabled((CPhidgetEncoderHandle)ENC, i, 1);
68.     }
```

```
69.     return 0;
70. }
71.
72.
73. int CCONV DetachHandler(CPhidgetHandle ENC, void *userptr)
74. {
75.     int serialNo;
76.     CPhidget_getSerialNumber(ENC, &serialNo);
77.     cout << "Device: " << serialNo << " is detached." << endl;
78.     indicator = 0;
79.     return 0;
80. }
81.
82. int CCONV ErrorHandler(CPhidgetHandle ENC, void *userptr, int ErrorCode, const char *De
    scription)
83. {
84.     printf("Error handled. %d - %s \n", ErrorCode, Description);
85.
86.     return 0;
87. }
88.
89. int CCONV InputChangeHandler(CPhidgetEncoderHandle ENC, void *usrptr, int Index, int St
    ate)
90. {
91.     printf("Input #%i - State: %i \n", Index, State);
92.
93.     return 0;
94. }
95.
96.
97. int CCONV PositionChangeHandler(CPhidgetEncoderHandle ENC, void *usrptr, int Index, int
     Time, int RelativePosition)
98. {
99.     int Position;
100.          CPhidgetEncoder_getPosition(ENC, Index, &Position);
101.          //printf("Encoder #%i - Position: %5d -- Relative Change %2d --
     Elapsed Time: %5d \n", Index, Position, RelativePosition, Time);
102.          if (Index == 0)
103.          {//this is the base arm encoder
104.              //currentTheta1 = M_PI/2 + (double)Position*(2.0*M_PI/31000.0);//pi/2 +
    count*conversion factor of CPR to rad
105.              currentTheta1 += (double)RelativePosition*(2.0*M_PI/31000.0);
106.              time1[0] = time1[1];
107.              time1[1] = time1[2];
108.              time1[2] = (double) Time/1000.0;
109.              thetaStore1[0] = thetaStore1[1];
110.              thetaStore1[1] = thetaStore1[2];
111.              thetaStore1[2] = currentTheta1;
112.              encoderPosition[0] = Position;
113.          }
114.          else if (Index == 1)
115.          {//this is the distal encoder
116.              currentTheta2 -= (double)RelativePosition*(2.0*M_PI/4330.88);
117.              time2[0] = time2[1];
118.              time2[1] = time2[2];
119.              time2[2] = (double) Time/1000.0;
120.              thetaStore2[0] = thetaStore2[1];
121.              thetaStore2[1] = thetaStore2[2];
122.              thetaStore2[2] = currentTheta2;
123.              encoderPosition[1] = Position;
124.          }
```

```
125.            return 0;
126.        }
127.
128.        //Display the properties of the attached phidget to the screen.  We will be disp
    laying the name, serial number and version of the attached device.
129.        //Will also display the number of inputs and encoders on this device
130.        int display_properties(CPhidgetEncoderHandle phid)
131.        {
132.            int serialNo, version, num_inputs, num_encoders;
133.            const char* ptr;
134.
135.            CPhidget_getDeviceType((CPhidgetHandle)phid, &ptr);
136.            CPhidget_getSerialNumber((CPhidgetHandle)phid, &serialNo);
137.            CPhidget_getDeviceVersion((CPhidgetHandle)phid, &version);
138.
139.            CPhidgetEncoder_getInputCount(phid, &num_inputs);
140.            CPhidgetEncoder_getEncoderCount(phid, &num_encoders);
141.
142.            printf("%s\n", ptr);
143.            printf("Serial Number: %10d\nVersion: %8d\n", serialNo, version);
144.            printf("Num Encoders: %d\nNum Inputs: %d\n", num_encoders, num_inputs);
145.
146.            return 0;
147.        }
148.
149.        CPhidgetEncoderHandle encoder_simple()
150.        {
151.            int result;
152.            const char *err;
153.
154.            //Declare an encoder handle
155.            CPhidgetEncoderHandle encoder = 0;
156.            //create the encoder object
157.            CPhidgetEncoder_create(&encoder);
158.
159.            //Set the handlers to be run when the device is plugged in or opened from so
    ftware, unplugged or closed from software, or generates an error.
160.            CPhidget_set_OnAttach_Handler((CPhidgetHandle)encoder, AttachHandler, NULL);
161.            CPhidget_set_OnDetach_Handler((CPhidgetHandle)encoder, DetachHandler, NULL);
162.            CPhidget_set_OnError_Handler((CPhidgetHandle)encoder, ErrorHandler, NULL);
163.
164.            //Registers a callback that will run if an input changes.
165.            //Requires the handle for the Phidget, the function that will be called, and
     an arbitrary pointer that will be supplied to the callback function (may be NULL).
166.            CPhidgetEncoder_set_OnInputChange_Handler(encoder, InputChangeHandler, NULL)
    ;
167.
168.            //Registers a callback that will run if the encoder changes.
169.            //Requires the handle for the Encoder, the function that will be called, and
     an arbitrary pointer that will be supplied to the callback function (may be NULL).
170.            CPhidgetEncoder_set_OnPositionChange_Handler (encoder, PositionChangeHandler
    , NULL);
171.
172.            CPhidget_open((CPhidgetHandle)encoder, -1);
173.
174.            //get the program to wait for an encoder device to be attached
175.            cout << "Waiting for encoder to be attached...." << endl;
176.            if((result = CPhidget_waitForAttachment((CPhidgetHandle)encoder, 10000)))
177.            {
```

```
178.                 CPhidget_getErrorDescription(result, &err);
179.                 cout << "Problem waiting for attachment: " << err << endl;
180.             }
181.
182.             //Display the properties of the attached encoder device
183.             display_properties(encoder);
184.
185.             //read encoder event data
186.             cout <<"Reading....." << endl;
187.             return encoder;
188.         }
189.
190.         void PDCtrl1()
191.         {//pd ctrl for base arm
192.             wcurrent1 = 1/(time1[2]-time1[0])*(thetaStore1[2]-thetaStore1[0]);
193.             error1 = desiredTheta1 - currentTheta1;
194.             int output = 1498;//offset
195.             output = 1500 - (Kp1*error1 + Kd1*wcurrent1);//generate pwm signal to send
196.             //check to make sure velocities are below the limits
197.             if (output >= V1_MAX) {
198.                 output = V1_MAX;
199.             }
200.             else if (output <= V1_MIN ) {
201.                 output = V1_MIN;
202.             }
203.             itoa(output,vOut1,10);//assign vOut2 to be the output value (from int to cha
    r[])
204.
205.             //cout << "output1 = " << output << endl;
206.             SendByte(cport_nr,'a');//forearm motor indicator
207.             SendBuf(cport_nr,(unsigned char*)vOut1,4);//send motor value
208.             //cout << (unsigned char*)vOut1 <<endl;
209.             SendByte(cport_nr,END_SIGNAL);//send end of transmission signal
210.         }
211.
212.
213.         void PDCtrl2()
214.         {//pd ctrl for forearm
215.             wcurrent2 = -(thetaStore2[2]-thetaStore2[0])/(time2[2]-time2[0]);
216.             error2 = desiredTheta2 - currentTheta2;
217.             int output = 1500;
218.             output = 1500 + (Kp2*error2 + Kd2*wcurrent2);//generate pwm signal to send
219.             //projectedY = boost::lexical_cast<string>(output);
220.             //check to make sure velocities are below the limits
221.             if (output >= V2_MAX) {
222.                 output = V2_MAX;
223.             }
224.             else if (output <= V2_MIN ) {
225.                 output = V2_MIN;
226.             }
227.             itoa(output,vOut2,10);//assign vOut2 to be the output value (from int to cha
    r[])
228.             //cout << "output2 = " << output << endl;
229.             SendByte(cport_nr,'b');//forearm motor indicator
230.             SendBuf(cport_nr,(unsigned char*)vOut2,4);//send motor value
231.             SendByte(cport_nr,END_SIGNAL);//send end of transmission signal
232.         }
233.
234.         void setXY(double x, double y) //takes in meters
235.         {
236.             double alpha = 0.0;
```

```
237.            if (x < 0)
238.            {
239.                alpha = M_PI - atan(abs(y/x));
240.            }
241.            else
242.            {
243.                alpha = atan(y/x);
244.            }
245.
246.            desiredTheta2 = M_PI - acos( (Len1*Len1 + Len2*Len2 - x*x -
    y*y)/(2*Len1*Len2));
247.            desiredTheta1 = alpha - acos( (x*x + y*y + Len1*Len1 -
    Len2*Len2)/(2*Len1*sqrt(x*x + y*y)));
248.        }
249.
250.        void updateMalletPositionDisplay(void)
251.        {
252.            malletX = boost::lexical_cast<string>(int((Len1*cos(currentTheta1)+Len2*cos(
    currentTheta1+currentTheta2))*100.0));
253.            malletY = boost::lexical_cast<string>(int((Len1*sin(currentTheta1)+Len2*sin(
    currentTheta1+currentTheta2))*100.0));
254.        }
255.
256.        void displayInfo()
257.        {
258.            cout << "dislpayInfo() created" << endl;
259.            while (true)
260.            {
261.                PDCtrl1();
262.                PDCtrl2();
263.                updateMalletPositionDisplay();
264.                //cout<<"Theta1: "<<currentTheta1<<" Theta2: " <<currentTheta2<<endl;
265.                if (!runMotorController)
266.                {
267.                    break;//end this thread
268.                    cout << "displayInfo() tread killed" << endl;
269.                }
270.                boost::this_thread::sleep(boost::posix_time::millisec(60));
271.            }
272.            cout << "displayInfo() finished" << endl;
273.        }
```

Appendix C

Arduino Code

```
1.  /*Arduino will recieve 'a' for base motor or 'b' for forearm motor
2.  then the pwm signal following with an end signal '\n'
3.  Arudino reads the signal, check for max/min speeds
4.  then send control signal to microcontrollers
5.  PD implementation on C++
6.  */
7.
8.  //include files
9.  #include <Servo.h>
10.
11. //define objects
12. Servo baseMotor;
13. Servo foreArmMotor;
14.
15. //constant variables
16. #define baseMotorPin 3
17. #define foreArmMotorPin 5
18. #define neutral 1500
19. //#define foreMaxCap 1700
20. //#define foreMinCap 1300
21. //#define baseMaxCap 1550
22. //#define baseMinCap 1450
23.
24. //global variables
25. int incomingByte;
26. int baseSpeed;
27. int foreSpeed;
28.
29. void setup() {
30.   Serial.begin(115200);
31.   while (!Serial) {
32.     ;
33.   }
34.
35.   baseMotor.attach(baseMotorPin);
36.   foreArmMotor.attach(foreArmMotorPin);
37.
38.   baseMotor.writeMicroseconds(neutral);
39.   foreArmMotor.writeMicroseconds(neutral);
40.   baseSpeed = neutral;
41.   foreSpeed = neutral;
42. }
43.
44. void loop() {
45.   if ( Serial.available() > 0 ) {
46.     incomingByte = Serial.read(); //should be a number
47.   }
```

```
48.   if (incomingByte == 'b') {
49.     foreSpeed = 0;
50.     while(1) {
51.       incomingByte = Serial.read();
52.       if (incomingByte == '\n') break;
53.       if (incomingByte == -1) continue;
54.       foreSpeed *= 10;
55.       foreSpeed = ((incomingByte - 48) +foreSpeed);
56.     }
57.   }
58.   //speed check
59. //   if (foreSpeed >= foreMaxCap) {
60. //     foreSpeed = foreMaxCap;
61. //   }
62. //   else if (foreSpeed <= foreMinCap) {
63. //     foreSpeed = foreMinCap;
64. //   }
65.
66.   if (incomingByte == 'a') {
67.     baseSpeed = 0;
68.     while(1) {
69.       incomingByte = Serial.read();
70.       if (incomingByte == '\n') break;
71.       if (incomingByte == -1) continue;
72.       baseSpeed *= 10;
73.       baseSpeed = ((incomingByte - 48) +baseSpeed);
74.     }
75.   }
76.
77. //   if (baseSpeed >= baseMaxCap) {
78. //     baseSpeed = baseMaxCap;
79. //   }
80. //   else if (baseSpeed <= baseMinCap) {
81. //     baseSpeed = baseMinCap;
82. //   }
83.
84.   foreArmMotor.writeMicroseconds(foreSpeed);
85.   baseMotor.writeMicroseconds(baseSpeed);
86.   delay(17);//original 17
87.   incomingByte = 0;
88. }
```