AL5A Robotic Arm Project: Web-Based Control with Spatial Awareness and Intuitive Manipulation

Final Draft
April 26, 2011

By Robert S. LaForge
Instructor Janusz Zalewski

Senior Software Engineering Project
Florida Gulf Coast University
Fort Myers, Florida
Spring 2011
Table of Contents

1. Introduction ........................................................................................................................................... 3
  1.1 Project Overview ............................................................................................................................ 3
  1.2 Hardware Configuration .................................................................................................................. 4
  1.3 Software .......................................................................................................................................... 6
      1.3.1 Client/Server Application ....................................................................................................... 6
      1.3.2 Simplified Remote Control and Spatial Awareness Application ........................................... 7
      1.3.3 Phidget Framework ................................................................................................................. 7

2. Definition of the Problem ....................................................................................................................... 8
  2.1 Overview ........................................................................................................................................ 8
  2.2 General Problem Description .......................................................................................................... 10
  2.3 Constraints and Dependencies ....................................................................................................... 10
  2.4 Definitions ..................................................................................................................................... 11
  2.5 User Characteristics ....................................................................................................................... 11
  2.6 Specific Requirements .................................................................................................................... 12

3. Design .................................................................................................................................................. 21
  3.1 Overview ...................................................................................................................................... 21
  3.2 Class Diagrams .............................................................................................................................. 21
  3.3 Spatial Mapping .............................................................................................................................. 25
  3.4 Graphical User Interface Map and Destination Plotting ................................................................. 25
  3.5 User Control .................................................................................................................................. 26
  3.6 Arm Base Rotation Algorithm ...................................................................................................... 27
  3.7 X and Y Position to Servo Value Algorithm ................................................................................... 28
  3.8 Scroll Wheel to Servo Adjustment Algorithm ................................................................................ 32

4. Implementation ..................................................................................................................................... 34
  4.1 Determining X, Y, and Radial Positions from Left Mouse Click .................................................... 34
  4.2 Determining End Point Arm Base Servo Value ............................................................................ 35
  4.3 Determining End Point Shoulder, Frame, and Hand Servo Values ............................................. 36
  4.4 Calculating Servo Adjustments and Scroll Wheel Rotations ....................................................... 38
  4.5 Testing .......................................................................................................................................... 39

5. Experimental Results ............................................................................................................................. 42
  5.1 Accuracy in Calculated versus Observed End Positions ................................................................. 42
  5.2 TCP Transmission Performance ................................................................................................... 46

6. Conclusion .......................................................................................................................................... 50

References .................................................................................................................................................. 53

Appendix A: Code ..................................................................................................................................... 54

Appendix B: User Manual ....................................................................................................................... 98
1. Introduction

1.1 Project Overview

Robotic devices are used in all aspects of our world. Whether in automation for factories and businesses, personal prosthetics, or used for off-world exploration in space and on other planets, we rely on robotics every day to reduce the costs of products and services and to make our lives easier. The ability to control, modify and update these robotic devices remotely is crucial. This allows robots to go in places where humans may not. It also provides mobility to people with disabilities who must control them by other means. To develop truly useful robotic equipment, the devices must be designed in a manner that considers a functional, precise, intuitive, and simple interface.

The purpose of this project is to design and develop such an interface to provide control over a robotic arm from a remote location in a simulated environment applicable to real-world scenarios. The interface will include a video feed broadcasted from the remote location and shall allow users to accurately position the arm to points within the robot’s reach in order to manipulate objects or perform simple tasks.

This project expands on Carlos Daboin’s 2009 study.[7] In his study, Daboin developed a client/server application based on C# whereby video streams and servo motor rotation commands could be transmitted using socket connections. These transmissions provided communication allowing remote surveillance and control over the robotic arm.

In order to expand on Daboin’s work and develop this new project, his work was adapted. First, the robot’s spatial reach was mapped in order to determine the minimum and maximum range of motion for any given height. Second, an algorithm was written to interpret multiple servo rotations to given spatial coordinates. Third, these coordinates were scaled for the user interface in order to provide a visual map for plotting destinations. Lastly, the user interface over the robot was simplified so that multiple servo rotations may occur in one set of mouse commands. The requirements are described in greater detail in the following sections.
1.2 Hardware Configuration

This section provides details on the hardware used in this project as well as a layout of how they are configured.

1) eBox 2300SX thin client, as a target.[1]

2) Windows PC (Windows 7), as a host computer.

3) Logitech Quick Cam Pro 5000. [2]

4) Servo Controller for 8 servo motors. [3]

5) Power Supply. [3]

6) AL5A Arm Frame for servos. [4]
7) Servos Needed. [4]

- 3 X Servos HS-422
- 1 X Servo HS-475HB
- 1 X Servo HS 755HB

8) Cables and connectors. [4]

- 3 servo extender cable 6"
- 3 servo extender cable 12"
- 3 servo extender cable 24"

9) Keyboard and a mouse (single USB cable connection is recommended).

10) Network infrastructure (LAN).

The eBox 2300 is considered a thin client, which is sometimes called a lean client. This is a “low-cost, centrally-managed computer devoid of CD-ROM players, diskette drives, and expansion slots” [1]. The eBox 2300 is the device used in this application and is accessible to clients/users in a LAN or WAN.

The AL5A Robotic Arm made by Lynxmotion is a small robot kit that utilizes servos which allow base rotation, single plane shoulder rotation, elbow rotation, wrist motion, and a functional gripper. The AL5A is equipped with a SSC-32 servo controller, allowing the robotic arm to iterate commands fast and accurately [5]. The AL5A is connected to the eBox via an USB cable.

The QuickCam Pro 5000 features a 640x480 VGA sensor. It uses RightLight 2 Technology, an imaging system from Logitech that delivers sharp video even in dim lighting [6]. The Logitech camera is connected to the eBox via an USB cable.

The hardware components are connected in a system shown in figure 1.6.
1.3 Software

1) Microsoft® Visual Studio® 2005 Professional Edition

2) Microsoft® Visual Studio® 2010 Express Edition

3) Windows® Embedded CE 6.0 Platform Builder

4) Microsoft® XNA Game Studio 3.1/4.0

4) Phidget® Framework for eBox

6) Client/Server Application developed by Carlos Daboin

7) Simplified Remote Control and Spatial Awareness Application

1.3.1 Client/Server Application

The application was developed by Carlos Daboin and then incorporated into the Simplified Control and Spatial Awareness Application described in section 1.3.2. Written in C#, its function is to transmit video and rotation commands using sockets. The Server application
streams live video captured from an USB camera and delivers commands to the servos, both of which are connected to the eBox 2300. The server application runs under the Windows CE 6.0 operating system. The Client application receives the streaming video from eBox2300 via the Internet and displays it onto the computer running this application. The client application sends rotation commands to rotate the various servo motors attached to the AL5A Robotic Arm. The program uses a TCP/IP connection [7].

1.3.2 Simplified Remote Control and Spatial Awareness Application

This application is the project that was developed in this experiment. It is written in C# and utilizes both Windows Forms and Microsoft XNA Game Studio 4.0. It also incorporates the TCP/IP socket code written in C# as developed by Carlos Daboin. The program sends servo rotation commands to the eBox and robot and receives video transmissions back. Through a circular map, the user may choose the radial position and distance out from the base of the robot in inches. When the left mouse button is clicked and held down, a height map is displayed. The user may then select a height, also in inches. On release, the program calculates the necessary servo adjustments to reach the destination point and prepares them for transmission. Through use of the scroll wheel, the program allows the user to move the robot to the destination at a desired speed depending on the scroll wheel rotation speed. The robot gripper may be adjusted by holding down the right mouse button. The detailed specifications will be outlined in the remaining sections.

1.3.3 Phidget Framework

The Phidget Framework installs the necessary drivers for the operating Phidget Devices in addition to the Phidget Manager and Web service.
2. Definition of the Problem

2.1 Overview

The project objective is to develop an application that provides remote control of an AL5A robotic arm connected to a low-powered, small CPU embedded system, called an eBox. The program communicates via a client/server application involving TCP protocol. The eBox acts as a server and receives servo transmissions from the client to control the robotic arm while transmitting video back.

In Daboin’s study, the user interface displays the video and provides a means for arm control through the use of sliders or text boxes to send individual rotation commands. This interface is depicted immediately below and includes five controls for each of the robotic arm’s servos: base, shoulder, elbow, wrist, and gripper.
The new program utilizes the same hardware configuration and TCP protocol but simplifies the user control. In it, the application provides a spatial map that allows the user to plot specific destination coordinates in one set of mouse commands rather than five separate adjustments. With each left mouse click, the necessary rotation adjustments to reach the plotted destination coordinate are calculated from the spatial location. The user then scrolls the mouse wheel in order to reach the destination as fast or as slow as desired. This provides finesse and precision. Depending on the direction of the scroll, the user reaches the desired location or returns to the current starting location, or to any coordinate along this vector. To adjust the gripper, the user simply clicks and holds down the right mouse button. The program also provides a means for sending five individual rotation commands. In this way, a user may make minor adjustments to any given servo or servos should they not be satisfied with their plotted one.
2.2 General Problem Description

The fundamental problem with the Daboin’s AL5A Robotic Arm control is the user interface. In order to perform any detailed operations, the user must either compose and send individual servo commands, or populate each individual sensor with its new location and then send all the commands at once. Although it provides adequate control, the method is slow and leaves the user without precise knowledge of the arm’s location except through a slow TCP video transmission that drops frames due to the low processing power of the eBox.

To define the problems, the system:

1. Lacks spatial awareness of the robotic arm’s reach limitations.
2. Lacks a graphical user interface that maps these limitations.
3. Lacks a means for plotting destination coordinates according to these limitations.
4. Lacks a simplified control that will calculate and send complex multi-servo commands to reposition the arm.
5. Lacks a method that allows the user to control the speed at which the arm shall reach its destination or return to its current starting point.
6. Lacks a means for starting and stopping between each end point.

2.3 Constraints and Dependencies

The system communicates via sockets, and the transmission of data uses TCP/IP protocol, so it will capable to run on any Ethernet network. The system transmission will be limited by networks (LAN/WAN) transmission speed and reliability. The system is assumed to run over Ethernet network, and the remote device, thin client, and client computer, PC, are always connecting to a network. The remote device will have a remote fix address, IP, which can be reachable from remote client application. Finally, the thin
client, and the personal computer shall have Ethernet ports to connect to a local network that uses TCP/IP protocol [7].

2.4 Definitions

2.4.1 Arm Base (AB): This is the lowest servo on the robotic arm. It swivels the base of the robot up to 195°.

2.4.2 Arm Shoulder (AS): This is the next servo above the arm base on the robotic arm and rotates the shoulder forwards or backwards. This servo can rotate up to 120°.

2.4.3 Arm Frame (AF): This is the next servo above the arm shoulder. It is the elbow of the robotic arm and can rotate forwards or backwards. This servo can rotate up to 135°.

2.4.4 Arm Hand (AH): This is the next joint above the arm frame and acts as the wrist of the robotic arm. It can swivel forwards or backwards, however it is unable to swivel side to side. This servo can rotate up to 190°.

2.4.5 Arm Gripper (AG): This is the top most joint of the robot arm and controls how wide or far apart the opposite sides of the gripper are from each other. This servo can rotate up to 110°.

2.5 User Characteristics

2.5.1 The user controls the five Phidget servo motors attached to the AL5A robotic arm in order to conduct arm rotation movements.

2.5.2 The user determines when to connect and disconnect the client/server connection.

2.5.3 The user interacts with the robotic arm through the client application’s GUI.

2.5.4 The user monitors the robotic arm movements through a video stream connected to the target component.
2.6 Specific Requirements

2.6.1 The client application shall have a button labeled, ‘Connect’, to connect to the server application.
   a. The application shall connect to the server, web camera, and robotic arm when selected.
   b. The button shall have white text when disconnected.
   c. The button shall have blue text when connected.
   d. The application shall connect to the designated IP address and port number as designated in the code of the application.
   e. When connected, the program shall enable the ‘Disconnect’, ‘Reset Position’, and ‘Last Position’ buttons.

2.6.2 The client application shall have a button labeled, ‘Disconnect’, to disconnect from the server application.
   a. The application shall disconnect from the eBox, robotic arm and web camera when selected.
   b. The button shall have blue text when disconnected.
   c. The button shall have white text when connected.
   d. When disconnected, the program shall disable the ‘Disconnect’, ‘Reset Position’, and ‘Last Position’ buttons.
   e. When disconnected, the program shall clear all text boxes on the GUI interface.

Figure 2.3 Connect and Disconnect buttons.
2.6.3 The client application shall provide a picture box to view the remote robot in real time.

a. When disconnected from the eBox, the picture box shall be simple a black box display.

b. When connected, the picture box shall display the remotely connected robotic arm via the web camera connected to the eBox.

![Image](image.png)

Figure 2.4 Picture Box.

2.6.4 The client application shall provide a radial map:

a. The radial map shall be displayed while the left mouse button is not pressed.

b. The radial map shall allow users to select arm base rotation in degrees to which the robot may rotate.

c. The radial map shall allow users to select the distance in inches out from the center of the robotic arm base to which the robotic arm may reach.

d. The radial map shall include eleven rings that grow in size from the center of the map out.
e. Each ring of the radial map shall represent one inch from the previous ring.

f. The center 2.25 inches of the radial map shall be a different color designating the base diameter of the robotic arm.

g. The radial map shall have two red lines starting from the center of the map to the outer edge designating whether the reach is front or behind the robot.

h. The radial map shall include numbers designating the degrees from the robot’s start position.

![Figure 2.5 Radial Map.](image)

2.6.5 The client application shall provide a height map for selecting the height from the ground to which the robotic arm shall move.

a. The height map shall be displayed while the left mouse button is pressed.

b. The height map shall allow users to select a height to which the arm may move.
c. The height map shall include a twenty-two by twenty-two grid of squares, where each square designates one square inch.

d. The height map shall include a plotted graph showing the limits of its reach for a given height.

e. The height map shall have a vertical line running up the center in green displaying height values in inches.

f. At the base of the vertical green line, the height map shall have numbers from 1 to 11 starting from the center out to the left.

g. At the base of the vertical green line, the height map shall have numbers from -1 to -11 starting from the center out to the right.

h. The numbered system from 1-11 and -1 to -11 shall correlate to the plotted graph displaying the reach limitations.

2.6.6 The application shall provide a text box labeled, ‘X’:

a. The text box shall display the selected distance out from the base to which the robotic arm may move.
b. The number displayed shall display the minimum x value along with the text, ‘min’, for a given height if the user selects an x value less than allowed for a given height.

c. The number displayed shall display the maximum x value along with the text, ‘max’, for a given height if the user selects an x value greater than allowed for a given height.

2.6.7 The application shall provide a text box labeled, ‘Y’, displaying the selected height from the ground to which the robotic arm may move.

2.6.8 The application shall provide a text box labeled, ‘R’, displaying the selected rotation, in degrees, to which the base of the robot shall rotate from its center position at zero degrees.

2.6.9 The application shall have a label titled, ‘End Position’, to denote the purpose of the position text boxes, ‘X’, ‘Y’, and ‘R’.

2.6.10 The application shall not allow user input in the ‘X’, ‘Y’, and ‘R’ text boxes.

2.6.11 The application shall provide a text box displaying the arm base servo rotation according the end point selected by the user labeled, ‘AB’.

2.6.12 The application shall provide a text box displaying the arm shoulder servo rotation according the end point selected by the user labeled, ‘AS’.

2.6.13 The application shall provide a text box displaying the arm frame servo rotation according the end point selected by the user labeled, ‘AF’.

Figure 2.7 X, Y, Z textboxes and ‘End Position’ label
2.6.14 The application shall provide a text box displaying the arm hand servo rotation according the end point selected by the user labeled, ‘AH’.

2.6.15 The application shall provide a text box displaying the arm gripper servo rotation according the end point selected by the user labeled, ‘AG’.

2.6.16 The application shall allow the user to manually enter servo positions in each servo position text box.

2.6.17 The application shall have a button, labeled ‘Send Manual Servo Commands’:
   a. The button shall have blue text when disabled.
   b. The button shall have white text when enabled.
   c. The button shall be disabled when disconnected from the eBox.
   d. The application shall send servo commands to reposition the arm according to the populated data in the ‘AB’, ‘AS’, ‘AF’, ‘AH’, and ‘AG’ text boxes without delay when selected.
   e. The application shall send the stored value of a sensor as a command if a text box is left blank.

2.6.18 The application shall have a label titled, ‘Current Servos’, to denote the purpose of the servo text boxes, ‘AB’, ‘AS’, ‘AF’, ‘AH’, and ‘AG’.

![Current Servos]

Figure 2.8 ‘AB’, ‘AS’, ‘AF’, ‘AH’, ‘AG’ and ‘Current Servos’ label

2.6.19 The application shall have a button, labeled “Reset Position”: 

2.6.20 The application shall have a button, labeled ‘Last Position’:

a. The button shall send servo commands to reposition the arm to its current starting position without delay when selected.

b. The application shall remove the current end coordinates from its memory to allow the user to select a new end position when selected.

c. The application shall populate the ‘AB’, ‘AS’, ‘AF’, ‘AH’, and ‘AG’ text boxes with the current starting servo values when selected.

d. The button shall be disabled when disconnected from the eBox.

e. The button shall be enabled when connected to the eBox.

Figure 2.9 ‘Reset Position’ and ‘Last Position’ buttons
2.6.21 The application shall provide control of the robotic arm via interaction between the mouse and the GUI:

a. The program shall interpret a left mouse press on the radial map as an end coordinate vector designating the end point radial rotation in degrees of the robot arm base and the distance end point out in inches from the base that the robotic arm may reach.

b. The program shall interpret a left mouse release as a designated end height coordinate to which the robot may travel to.

c. The program shall interpret forward motion of the scroll wheel as the speed by which the robotic arm may travel from the starting point to the newly designated end point.
d. The program shall interpret backward motion of the scroll wheel as the speed by which the robotic arm may travel from its current position to the latest starting point.

e. The program shall send servo commands through the eBox to the robotic arm for every scroll increment.

f. The program shall send servo transmissions to adjust the robotic arm gripper while the right mouse button is pressed.

g. The program shall interpret each separate right button mouse press as a signal to rotate the gripper in the opposite direction of the last right button press.

![Figure 2.10 Mouse Control](image)
3. Design Solution

3.1 Overview

This section systematically defines the design solutions necessary to complete the project objectives. In terms of software development, the server side is maintained according to the previous robotic arm project developed by Daboin. This portion of the project was written using Visual Studio 2005 and C#. The published server programs reside on the eBox 2300 that runs a Windows Embedded CE operating system. One server program listens for clients and provides video transmissions using a TCP protocol. The other listens for servo commands and passes these transmissions to the AL5A robotic arm using the Phidget Framework.

The client side software is heavily redeveloped to include a graphical user interface that describes the limitations of the robotic arm’s reach and provides a map for spatial plotting of arm destination coordinates. It also offers a simplified control that automatically calculates and sends complex multi-servo commands, varies the speed at which the arm may travel back and forth between currently selected end positions as decided by the user in any given instance, and allows the user to stop the arm at any given point between. It is built in Visual Studio 2010, using C# and XNA Frameworks. It uses TCP protocol for receiving video transmissions and sending servo commands.

3.2 Class Diagrams

The ObjectModel class shown to the left works directly with the Game1 class shown below. Its purpose is to represent object being used on both the radial and height maps in the user interface. This class defines specific characteristics about the object, such as its position on the interface, its center point, what image represents it, its size, and whether it should be drawn on the GUI or not.
The SharedResources class is used by both the Game1 class and the mainForm class and allows these two classes to share information back and forth. This class also performs multiple calculations that transform pixel coordinate positions from the plotting map, into X, Y, and Z positions, and finally into servo values. The class also contains multiple arrays filled with the observed measurements of the robot’s range of motion. These arrays are referenced when trying to calculate servo adjustments for coordinates between array index values.

More details on these calculations are provided in the Design and Implementation sections of this report.
The `mainForm` class contains and controls all the text boxes, buttons, and picture boxes on the graphical user interface. It also establishes the TCP connection between the client and server, writes servo commands to the server based on user selections, displays the video feed from the server, and interprets mouse commands. It references the `SharedResources` class.
The Game1 class controls the picture box in the lower center of the mainForm class. It draws to the user interface either the radial map or the height map, depending on whether the user is making a mouse press. It references both the SharedResources class and the ObjectModel classes.
3.3 Spatial Mapping

Prior to doing any software development, the limitations of the robot reach is mapped. This involves measuring each servo position according to specific spatial locations. For the arm base (AB), this requires measuring the maximum rotation in either direction and defining a starting position that is exactly half way between the two (Radial-axis). The maximum rotation of the base from the start position in either direction designates where the robot must reach in front of or behind itself. The arm shoulder (AS), arm frame (AF), and arm hand (AH) are considered as a set. Together, each servo position is recorded for minimum, mid, and maximum reach (X-axis) according to a specific height (Y-axis). These measurements include both the reach in front of and behind itself. Height measurements begin from zero or the lowest height possible above zero according to the servo limitations. These measurements are used to calculate the servo adjustments for any given coordinate. As an aside, the arm gripper (AG) is not measured for spatial mapping and is discussed later.

3.4 Graphical User Interface Map and Destination Plotting

Now that the range of motion has been recorded for the AL5A robotic arm, it is mapped to an interface interpretable by a user. This interface was described in requirements 2.6.4 and 2.6.5 and depicted in figures 2.5 and 2.6. For either the radial or height map, the entire GUI size is 400 by 400 pixels and is linked to a Windows form picture box utilizing XNA Frameworks. In its initial state, the interface displays the radial map. When a point is selected on the radial map, the program retrieves the pixel coordinate within the picture box and converts it from pixels to either degrees or inches using the slope and a scale. These calculations are considered from the very center of the map out and the slope is calculated from this point to the selected point. Upon radial map selection, the program automatically displays the height map, thus allowing the user to select a Y-value. The limitations of the robot reach are superimposed on the grid and show the range of motion in front of and behind it. Upon selection release, the program takes the
release point and calculates the height by transposing the pixel location into inches. This calculation is determined from the center and bottom of the height map.

### 3.5 User Control

The program has a simplified user control to manage complex servo operations. To plot destinations on the graphical map described in 3.3, the program uses mouse commands. The application interprets left mouse presses as a coordinate selection on the radial map. The program defines a left mouse release as a coordinate selection on the height map. A selection on the radial map is only allowed while the mouse pointer is over the radial map. On release, if the mouse pointer is no longer in the map area and instead off to the left or right, the program will still select the closest pixel point from that release coordinate to calculate the height. If the release is above the map, the program will set the height to maximum height. If the release is below the height map, the program will automatically designate zero as the Y-value.

The program favors the height value. Should a user select an X-value on mouse press that is outside the range of motion for the Y-value designated on mouse release, the program will determine a new X-value equal to the maximum X-value for the chosen height.

The mouse scroll wheel is used to reposition the arm between the current start and end positions. For each change in delta of the scroll wheel, the program writes an output stream containing new servo values for the arm shoulder (AS), arm frame (AF), and the arm hand (AH) through the TCP connection to the server and on to the robotic arm. Each transmission repositions the arm to its new location. A positive delta repositions the arm towards the currently selected end coordinate. A negative delta repositions the arm closer to its current starting point. The faster the scroll wheel is moved, the faster the transmissions are posted through the TCP connection. This provides the user with a method for varying the speed at which the arm moves and allows the user to stop the arm at any desired position between the two end points.

The program offers some flexibility to the user in regards to choosing begin and end positions. Should the user decide to stop at a position between the start and end points currently
designated, and then choose a new destination coordinate on the plotter, the program marks its current position as the new start position and the selected position as its new end position.

The arm gripper is controlled using the right mouse button. The program sets off a timer and a Boolean when the right mouse button is selected. While it is being held, the servo controlling the gripper rotates at a set speed. Upon release, the timer resets and the Boolean toggles. The Boolean controls the servo rotation direction. The next time it is pressed, the servo rotates in the opposite direction.

The ‘reset’ button allows the user to reset the robot’s position to the designated start position. When pressed, the program sends a command transmission to the eBox and arm with the servo values for the resting position. The arm then repositions itself straight away. In this position, the robot rests at its maximum height, with a zero X-value and zero degrees. When the ‘last position’ button is selected, the program sends a stream of servo values to the eBox and arm and causes the arm to move immediately to the current start position. It also removes the currently selected end position, thus allowing the user to create a new end coordinate from the start position as well as a new path to traverse.

At any given time, the AB, AS, AF, AH, and AG textboxes display the end position servo values. The only case where this is not true is at program start up, connecting to the server, and disconnecting from the server. At startup, the text boxes are all blank. When the ‘connect’ button is selected, the program automatically repositions the arm to its startup position and populates the text boxes with the startup servo values. At server disconnection, the textboxes are blank. These text boxes are editable and allow users to populate specified servo values. When the ‘send servo commands’ button is selected, the servo values are transmitted through the TCP connection, and the arm automatically repositions. Should a field be left blank, the program populates the blank field with the value stored globally for the particular servo.

### 3.6 Arm Base Rotation Algorithm (ABRA)

An algorithm has been created to determine the number of degrees necessary to rotate the arm base (AB) so that it faces the desired end-location directly. If the end-
point destination is beyond the servos rotational range, the algorithm must determine the position opposite the end-point position so that the arm may reach directly behind itself. Since total Arm Base (AB) rotation possible is 195°, the resting position for the arm would be at 97.5° normally. Since the parameter values used in the prior design for the arm base servo begin at a minimum of 20 and a maximum of 215 and each servo parameter value has a 1° difference, the center resting position parameter value for the AB would be 117°. This is now designated as 0° in the new design. The AB will be able to adjust to a maximum rotation up to ±97.5°. Should the end-point destination be behind the robot, the program will subtract 180° from the calculated rotation value and position the robot arm base opposite of where it will reach. This occurs in step 4 below. Afterwards, a Boolean flag is set true or false, depending on whether the AB calculation leaves the robotic arm facing opposite its end-point destination. This flags the program to use a specific set of measured values for calculating the arm shoulder (AS), arm frame (AF), and arm hand (AH) servo end point values.

ABRA Algorithm

1. Retrieve radial map coordinate in pixels from user selection.

2. Determine slope of coordinate from the center pixel coordinate:

   \[ m = \frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - x_1} \]

3. Determine angle \( \theta \):

   \[ m = \tan \theta \]
   \[ \theta = \arctan m \]

4. If angle is greater than ±97.5°, subtract 180° from angle.

   a. Set Boolean flag to false, meaning that the robot will reach backwards and to use appropriate arrays of measured values.

5. Else set Boolean flag to true, meaning robot will reach forward and to use appropriate arrays of measured values.
3.7 X and Y Position to Servo Value Algorithm (PTSV)

An algorithm has been created that transforms X and Y axis coordinates into servo rotation values for the Arm Shoulder (AS), Arm Frame (AF), and Arm Hand (AH).

PTSV Algorithm

1. Retrieve pixel coordinates selected for x and y values.

2. Determine length of X from center point of radial map:

\[ c = \frac{\sqrt{A^2 + B^2}}{\text{scale}} \]

3. Determine height of Y from center ground point of height map:

\[ y = \frac{Y}{\text{scale}} \]

4. Knowing height, find the index in the height array where its height value is just greater than the chosen Y value.

5. Check the Boolean flag to determine whether the robot is reaching in front of or behind itself and select the group of measured X-value arrays that pertain to it (minimum reach values, mid reach values, and maximum reach values).

6. Determine the range of motion, displacement, and scale of X for a given height, Y.
   
   a. Compare the chosen X value to the array containing the mid-point reach of the robot using the height index.
      
      i. If X is less than the midpoint at index, the program will use the minimum and midpoint measured X-value arrays.
      
      ii. Else, the program will use the midpoint and maximum X-value arrays.

   b. Calculate the minimum and maximum x values for a given height:
      
      i. If the index equals the length of the height array, then set the minimum and maximum reach values equal to each other and set the total range of reach (X-axis) to zero.
ii. Else, determine the minimum X value for the chosen height using the measured Y values and minimum measured X values, each at index and index-1:

1. Divide the difference of lower bound X from upper bound X and divide it by the lower bound Y subtracted from the upper bound Y to get a ration of X’s to Y.

\[
\text{tempScale} = \frac{\text{min}_x[\text{index}] - \text{min}_x[\text{index} - 1]}{\text{height}[\text{index}] - \text{height}[\text{index} - 1]}
\]

2. Calculate lower limit of X for chosen X value:

\[
\text{minX} = (\text{Y} - \text{height}[\text{index} - 1]) \ast \text{tempScale} \\
\quad + \text{min}_x[\text{index} - 1]
\]

iii. Determine the maximum X value for the chosen height using the measured Y values and maximum measured X values, each at index and index-1:

1. Divide the difference of lower bound X from upper bound X and divide it by the lower bound Y subtracted from the upper bound Y to get a ration of X’s to Y.

\[
\text{tempScale} = \frac{\text{max}_x[\text{index}] - \text{max}_x[\text{index} - 1]}{\text{height}[\text{index}] - \text{height}[\text{index} - 1]}
\]

2. Calculate upper limit of X for chosen X value:

\[
\text{maxX} = ((\text{Y} - \text{height}[\text{index} - 1]) \ast \text{tempScale}) \\
\quad + \text{max}_x[\text{index} - 1]
\]

c. Calculate total range of X:

\[
\text{XRange} = \text{maxX} - \text{minX}
\]

d. Calculate displacement:
e. Calculate displacement scale to be used for determining new servo values:

\[
\text{displacementScale} = \frac{\text{displacement}}{X \text{Range}}
\]

7. For each servo, calculate the new servo value:

a. If \(Y\) equals the maximum height possible for the robotic arm, then set all servo values to their starting, rest position values.

b. Otherwise, determine minimum servo value for a specific height:

i. Find ratio of the servo adjustment to height by using index to retrieve the upper and lower bound measured parameters for both height and measured servo values from the height array and the minimum servo value array.

\[
\text{tempScale} = \frac{\text{min}_\text{servo}[\text{index}] - \text{min}_\text{servo}[\text{index} - 1]}{\text{height}[\text{index}] - \text{height}[\text{index} - 1]}
\]

ii. Find the actual minimum servo value for a given height:

\[
\text{min}_\text{ServoValue} = ((Y - \text{height}[\text{index} - 1]) \times \text{tempScale}) + \text{min}_\text{servo}[\text{index} - 1]
\]

c. Determine maximum servo value for a specific height:

i. Find ratio of the servo adjustment to height by using index to retrieve the upper and lower bound measured parameters for both height and measured servo values from the height array and the maximum servo value array.

\[
\text{tempScale} = \frac{\text{max}_\text{servo}[\text{index}] - \text{max}_\text{servo}[\text{index} - 1]}{\text{height}[\text{index}] - \text{height}[\text{index} - 1]}
\]

ii. Find the actual maximum servo value for a given height:
3.8 Scroll Wheel to Servo Adjustment Algorithm (SWSA)

The program utilizes the scroll wheel in order to incrementally reposition the AS, AF, and AH servos from its start point to its end point, or to some destination along the path between. An algorithm is used to determine the amount each servo will reposition per increment. Since servo values are integers, but division may establish absolute values greater than zero and less than one, the program sets these values to increments of one servo position. The program then sets a Boolean determining whether the end position is a greater or lesser servo value. In this way, the program knows to stop sending transmissions to the eBox and robotic arm when the end value is reached. The program uses a hardcoded scroll wheel factor to establish a reasonable increment range per servo adjustment.

SWSA Algorithm

1. Calculate servo adjustments for AS, AF, and AH:

   \[
   \text{servo Adjustment} = \frac{(\text{end Servo Value} - \text{start Servo Value})}{\text{scroll Wheel Factor}}
   \]

2. If the absolute servo adjustment value is a fraction between one and zero:
   
   a. If the actual servo adjustment is less than zero
      
      i. If true, set the servo adjustment value to negative one
      
      ii. Else, set the servo adjustment value to positive one.
b. Check to see whether the start position is less than or equal to the end position:
   
   i. If false, set the Boolean to false.
   
   ii. Else, set the Boolean to true.

3. If scrolling begins

   a. Add the adjustment value to the current servo value.

   b. Check to see whether the new servo value has reached the final position.

   i. If true, current servo value equals the final servo value.
4. Implementation

This section discusses code in the program relevant to calculating positions on the user map interface and how they are converted to servo rotation values. All position and servo calculations are conducted in the SharedResources.cs class.

This class is used to pass information between the main windows form and the graphical user interface plotting map as well as to calculate X, Y, and radial coordinates and corresponding servo values. It also has multiple arrays made of measured values for relating height and reach to servo values. It sets servo values to previous positions and is used to collect information for text boxes. The first piece of code below shows the implementation of the Arm Base Algorithm described in section 3.6. The code after presents the implementation for the X and Y position to Servo Algorithm as described in section 3.7.

4.1 Determining X, Y, and radial Positions from Left Mouse Click

The setX function receives a value in pixels, finds the slope length and then divides by the scale from pixels to inches.

```csharp
// sets the X value based on angle A and B for a triangle A^2 + B^2 = C^2
public void setX(double X)
{
    radianX = X;
    if (X == 0)
        this.X = X;
    else
    {
        double A = radianX - centerX;
        double B = radianZ - centerZ;
        this.X = (Math.Sqrt(A * A + B * B)) / scale;
    }

    if (this.X > 11.5)
        this.X = 11.5; // this may be changed to set a max for a particular width
}
```

The setY function receives a value in pixels and then divides by the scale from pixels to inches.
public void setY(double Y)
{
    if (Y == 0)
        this.Y = Y;
    else
        this.Y = ((Y - centerY) * -1) / scale;
    if (this.Y > 14.25) //
        this.Y = 14.25;
    else if (this.Y < 0)
        this.Y = 0;
}

The setDegrees function retrieves two parameters and then calculates the angle.

    // sets the angle for determining Arm Base Rotation
    public void setDegrees()
    {
        if (Z == 0 && X == 0)
            angle = 0;
        else
        {
            radians = Math.Atan2((radianZ - centerZ), (radianX - centerX));
            angle = ((radians * (180 / Math.PI)) * -1) - 90;
            if (angle < 0)
            {
                angle += 360;
            }
        }

        // determining whether the robot is reaching in front of itself
        if (angle >= 0 && angle <= 97 || angle >= 263 && angle <= 360)
        {
            isForward = true;
        }
        else
        {
            isForward = false;
        }
    }

4.2 Determining End Point Arm Base Servo Value

This is an excerpt from the calculateServoPositions function from the
SharedResources.cs file and shows the implementation for calculating a servo position value.
Together with section 4.2.1, this comprises the arm base algorithm described in section 3.6.

    // ********** ARM BASE CALCULATIONS **********
    double tempAngle = angle;
    if (tempAngle > 97 && tempAngle < 263)
tempAngle = 360 + (tempAngle - 180);
// if point is in forward reach
if (tempAngle >= 0 && tempAngle <= 97)
{
    AB = (int)(startAB - tempAngle);
}
// else it will rotate to a point behind itself
else
{
    AB = (int)(startAB - (tempAngle - 360));
}

4.3 Determining End Point Arm Shoulder, Arm Frame, and Arm Hand Servo Values

This is also an excerpt from the calculateServoPositions function from the SharedResources.cs class. It shows the implementation for calculating the arm shoulder servo values. Calculations for the arm frame and arm hand are identical. At the end of the method, it calls the adjustServos function which determines the amount to adjust per increment of the scroll wheel (described in 4.2.4).

// calculates the servo values for a given x,y,z position
public void calculateServoPositions()
{
    double min_y;
    double max_y;
    double min_servo;
    double max_servo;
    double tempScale;

    max_y = height[index];
    min_y = height[index - 1];
    int[] arrayMinServoAS;
    int[] arrayMaxServoAS;

    // determines the set of arrays holding measured values to be used in the calculation process
    if (isForward)
    {
        if (X < midForward[index])
        {
            arrayMinServoAS = ASminForward;
            arrayMaxServoAS = ASmidForward;
        }
        else
        {
            arrayMinServoAS = ASmidForward;
        }
    }
arrayMaxServoAS = ASmaxForward;
}
}
Else // the robot is reaching behind itself
{
  if (X < midBackward[index])
  {
    arrayMinServoAS = ASminBackward;
    arrayMaxServoAS = ASmidBackward;
  }
  else
  {
    arrayMinServoAS = ASmidBackward;
    arrayMaxServoAS = ASmaxBackward;
  }
}

if (Y < 14.25)
{
  // ************ ARM SHOULDER CALCULATIONS ************
  // getting max arm shoulder value
  max_servo = arrayMaxServoAS[index];
  min_servo = arrayMaxServoAS[index - 1];

  // determines a scale for servo adjustment to height
  tempScale = (max_servo - min_servo) / (max_y - min_y);

  if (tempScale == 0)
    maxServo = max_servo;
  else
    maxServo = ((Y - min_y) * tempScale) + min_servo;

  // getting min arm shoulder value
  max_servo = arrayMinServoAS[index];
  min_servo = arrayMinServoAS[index - 1];

  tempScale = (max_servo - min_servo) / (max_y - min_y);

  if (tempScale == 0)
    minServo = min_servo;
  else
    minServo = min_servo + ((Y - min_y) * tempScale);

  AS = (int)(((maxServo - minServo) * displacementScale) + minServo);
  if (AS < 20)
    AS = 20;
  else if (AS > 145)
    AS = 145;
}
else
{
  // set the values to the start value because this is at the max height with
  // no X-minimum or maximum

  AS = startAS;
}
4.4 Calculating Servo Adjustments and Scroll Wheel Rotations

The following code lines are excerpts from the calculateServoAdjustments and adjustServos functions from the SharedResources.cs class. They show the implementation for calculating the arm shoulder servo adjustment per scroll wheel increment and the actual incrementing process that occurs while the user is scrolling. Calculations for the arm base, arm frame, and arm hand are an identical process. This correlates to section 3.8.

```csharp
// sets booleans for how the servos must adjust from the starting point in order to get the end point
calculateServoAdjustments();

public void calculateServoAdjustments()
{
    ASadjust = ((double)AS - (double)prevAS) / scrollWheelFactor;
    if (Math.Abs(ASadjust) < 1)
    {
        if (ASadjust < 0)
        {
            ASadjust = -1;
        }
        else
        {
            ASadjust = 1;
        }
    }
    if (prevAS <= AS)
    {
        // then must increment to reach end point
        ASLess = false;
    }
    else
    {
        // must decrement to reach end point
        ASLess = true;
    }
}

calculateServoAdjustments();

public void adjustServos(int direction)
{
    if (direction == 0)
    {
        curAS += (int)ASadjust;
        if (ASLess)
        {
            if (curAS <= AS)
            {
```
```c
    curAS = AS;
}
else
{
    if (curAS > AS)
    {
        curAS = AS;
    }
}
else
{
    curAS -= (int)ASadjust;
    if (!ASLess)
    {
        if (curAS <= prevAS)
        {
            curAS = prevAS;
        }
        else
        {
            if (curAS > prevAS)
            {
                curAS = prevAS;
            }
        }
    }
}
```

### 4.5 Testing

1. Measure accuracy of calculated versus observed robot end positions.
   a. Test values that are exactly those of the measured values stored in the height, minimum X, middle X, and maximum X arrays at specific indexes.
   b. Test values that are near a measured minimum, middle, and maximum X-value and that are near a measured Y-value.
   c. Test values that are at an exact measured Y-Value but only near the minimum, middle, and maximum measured X values.
   d. Test values that are at an exact measured minimum, middle, and maximum X value but only near a measured Y value.
   e. Test values that are at an exact measured Y-value and exactly half way between minimum and middle measured X-values.
   f. Test values that are an exact measured Y-value and exactly half way between middle and maximum measured X-values.
g. Test values that are at an exact measured minimum, middle, and maximum X value and exactly halfway between two measured Y-values.

h. Test values that are an exact minimum, middle, and maximum X-value but that have a Y-value outside of the robot range of motion.

i. Test values that are at exact measured Y-values but that are outside of the minimum and maximum X-values.

2. Measure and compare TCP transmission performance with and without a connection to the video server.

a. Test transmission performance without a making a connection to the video server.

i. In close proximity between client and server, measure time between client connection request and server connection acknowledgement.

ii. At a distance between client and server, measure time between client connection request and server connection acknowledgement.

iii. In close proximity between client and server, measure time between client sending servo adjustment commands to server receipt of commands.

iv. At a distance between client and server, measure time between client sending servo adjustment commands to server receipt of commands.

v. In close proximity between client and server, measure time to disconnect client from server.

vi. At a distance between client and server, measure time to disconnect client from server.

b. Test transmission performance with a connection to the video server.

i. In close proximity between client and server, measure time between client connection request and server connection acknowledgement.

ii. At a distance between client and server, measure time between client connection request and server connection acknowledgement.

iii. In close proximity between client and server, measure time between client sending servo adjustment commands to server receipt of commands.

iv. At a distance between client and server, measure time between client
sending servo adjustment commands to server receipt of commands.

v. In close proximity between client and server, measure time to disconnect client from server.

vi. At a distance between client and server, measure time to disconnect client from server.
5. Experimental Results

This section describes the results of the testing as designated in section 4.2.5.

5.1 Accuracy in Calculated versus Observed End Positions

The first set of tests revolves around the accuracy of the actual robot position versus program calculation of end position. A total of 54 tests were conducted. Each test case below describes input values injected into the program and the output values that were derived. In each corresponding table, the left two sets of columns are the test values inserted. The right two sets of values are the actual end position measurements of the robot. Each test case sets values for X and Y in inches, sets a Boolean as to whether the robot is reaching in front of or behind itself, and then calls the two methods in the SharedResources.cs file, determineMinMaxRangeValues() and calculateServoPositions(). An example of an implemented test case is show immediately below.

```csharp
/* h. Test values that are an exact minimum, middle, and maximum X-value but that * have a Y-values outside of the robot range of motion. */

public void test1h1()
{
    testX(11.37); // setting the X value
    testY(15.0); // setting the Y value above the possible Y value
    testForward(true);
    determineMinMaxRangeValues();
    calculateServoPositions();
}
```

In general, the tests clearly show that the output measurements are reasonable for purposes of this project and for future projects. Output values for tests 1A through 1G are all within one quarter of an inch of the injected values. The problems arise in tests 1H and 1I. In these two tests, values outside of the reach of the robot were injected.

In test 1H, Y values were inserted that were outside the possible range of motion. When the value was above the possible height, the program caught the error and automatically inserts the maximum height in its place. However, when a negative number was injected, the program crashed, causing an error. Due to the nature of the program, though, this was a problem in the
test case and not in the actual program functionality. When the program receives a value less than zero, it automatically inserts a zero as a replacement value. This is caught and altered in a method outside of those actually used in the test case.

In test 1I, X values were inserted into the program that were outside the possible range. This caused significant errors in the final resting position of the robot. As an example, inserting an X value of 15 and a Y value of one returned a destination of 8.62 for X and a Y value of 7.5. Other inserted values returned similar problems. Reviewing and processing other segments of code not used in the testing produced similar outcomes. Therefore, this is a bug in the program that will have to be dealt with in the next build.

**Test Case 1A:** Test values that are exactly those of the measured values stored in the height, minimum X, middle X, and maximum X arrays at specific indexes.

<table>
<thead>
<tr>
<th>Preset values</th>
<th>Observed Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>11.37</td>
</tr>
<tr>
<td>1</td>
<td>-8.62</td>
</tr>
<tr>
<td>4</td>
<td>11.37</td>
</tr>
<tr>
<td>4</td>
<td>-10.6</td>
</tr>
<tr>
<td>8</td>
<td>10.12</td>
</tr>
<tr>
<td>8</td>
<td>-10.12</td>
</tr>
</tbody>
</table>

**Test Case 1B:** Test values that are near a measured minimum, middle, and maximum X-value and that are near a measured Y-value.

<table>
<thead>
<tr>
<th>Preset values</th>
<th>Observed Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>X</td>
</tr>
<tr>
<td>1.25</td>
<td>11.12</td>
</tr>
<tr>
<td>1.25</td>
<td>-8.37</td>
</tr>
<tr>
<td>4.25</td>
<td>11.12</td>
</tr>
<tr>
<td>4.25</td>
<td>-10.35</td>
</tr>
<tr>
<td>8.25</td>
<td>9.87</td>
</tr>
<tr>
<td>8.25</td>
<td>-9.87</td>
</tr>
</tbody>
</table>
**Test Case 1C**: Test values that are at an exact measured Y-Value but only near the minimum, middle, and maximum measured X values.

<table>
<thead>
<tr>
<th>Preset values</th>
<th>Observed Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>11.12</td>
</tr>
<tr>
<td>1</td>
<td>-8.37</td>
</tr>
<tr>
<td>4</td>
<td>11.12</td>
</tr>
<tr>
<td>4</td>
<td>-10.35</td>
</tr>
<tr>
<td>8</td>
<td>9.87</td>
</tr>
<tr>
<td>8</td>
<td>-9.87</td>
</tr>
</tbody>
</table>

**Test Case 1D**: Test values that are at an exact measured minimum, middle, and maximum X value but only near a measured Y value.

<table>
<thead>
<tr>
<th>Preset values</th>
<th>Observed Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>X</td>
</tr>
<tr>
<td>1.25</td>
<td>11.37</td>
</tr>
<tr>
<td>1.25</td>
<td>-8.62</td>
</tr>
<tr>
<td>4.25</td>
<td>11.37</td>
</tr>
<tr>
<td>4.25</td>
<td>-10.6</td>
</tr>
<tr>
<td>8.25</td>
<td>10.12</td>
</tr>
<tr>
<td>8.25</td>
<td>-10.12</td>
</tr>
</tbody>
</table>

**Test Case 1E**: Test values that are at an exact measured Y-value and exactly half way between minimum and middle measured X-values.

<table>
<thead>
<tr>
<th>Preset values</th>
<th>Observed Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>4.94</td>
</tr>
<tr>
<td>1</td>
<td>-6.66</td>
</tr>
<tr>
<td>4</td>
<td>5.19</td>
</tr>
<tr>
<td>4</td>
<td>-7.06</td>
</tr>
<tr>
<td>8</td>
<td>3.64</td>
</tr>
<tr>
<td>8</td>
<td>-4.5</td>
</tr>
</tbody>
</table>
**Test Case 1F**: Test values that are an exact measured Y-value and exactly half way between middle and maximum measured X-values.

<table>
<thead>
<tr>
<th>Preset values</th>
<th>Observed Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>9.25</td>
</tr>
<tr>
<td>1</td>
<td>-7.97</td>
</tr>
<tr>
<td>4</td>
<td>9.31</td>
</tr>
<tr>
<td>4</td>
<td>-9.42</td>
</tr>
<tr>
<td>8</td>
<td>7.96</td>
</tr>
<tr>
<td>8</td>
<td>-8.25</td>
</tr>
</tbody>
</table>

**Test Case 1G**: Test values that are at an exact measured minimum, middle, and maximum X value and exactly half way between two measured Y-values.

<table>
<thead>
<tr>
<th>Preset values</th>
<th>Observed Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>X</td>
</tr>
<tr>
<td>1.75</td>
<td>11.37</td>
</tr>
<tr>
<td>1.75</td>
<td>-8.62</td>
</tr>
<tr>
<td>5</td>
<td>11.37</td>
</tr>
<tr>
<td>5</td>
<td>-10.6</td>
</tr>
<tr>
<td>8.25</td>
<td>10.12</td>
</tr>
<tr>
<td>8.25</td>
<td>-10.12</td>
</tr>
</tbody>
</table>

**Test Case 1H**: Test values that are an exact minimum, middle, and maximum X-value but that have a Y-value outside of the robot range of motion.

<table>
<thead>
<tr>
<th>Preset values</th>
<th>Observed Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>X</td>
</tr>
<tr>
<td>15</td>
<td>11.37</td>
</tr>
<tr>
<td>15</td>
<td>-8.62</td>
</tr>
<tr>
<td>15</td>
<td>11.37</td>
</tr>
<tr>
<td>-2</td>
<td>10.6</td>
</tr>
<tr>
<td>-2</td>
<td>10.12</td>
</tr>
<tr>
<td>-2</td>
<td>-10.12</td>
</tr>
</tbody>
</table>
**Test Case II:** Test values that are at exact measured Y-values but that are outside of the minimum and maximum X-values.

<table>
<thead>
<tr>
<th>Preset values</th>
<th>Observed Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>15</td>
</tr>
<tr>
<td>1</td>
<td>-15</td>
</tr>
<tr>
<td>4</td>
<td>15</td>
</tr>
<tr>
<td>4</td>
<td>-15</td>
</tr>
<tr>
<td>8</td>
<td>15</td>
</tr>
<tr>
<td>8</td>
<td>-15</td>
</tr>
</tbody>
</table>

### 5.2 TCP Transmission Performance

The purpose of this test set is to measure transmission performance of TCP packet exchange with and without video connection from the server. It is also to show that algorithmic calculations conducted on the client have no impact on this performance. In order to conduct this experiment, Wireshark was used. “Wireshark is a free and open-source packet analyzer. It is used for network troubleshooting, analysis, software and communications protocol development, and education.”[8] In this study, two set of measurements were taken. The first test was conducted campus using a wireless connection and within 15 feet of an access point and in the same room as the robotic arm server. The second test was taken off campus, approximately 30 miles away from the Florida Gulf Coast University campus also using a wireless connection.

Each set of tests included measuring the time required to establish a TCP connection and time to disconnect. They also included the time measurements for sending a servo command to the server from the client application. When video was being broadcasted, measurements were taken regarding the time a connection to the video server was established to when the actual video was received by the client.

Looking at the results for on and off campus connections without video, connecting to the server took longer at a distance. Connection was established on campus in 0.005015 seconds.
Connection off campus took 0.065679 seconds. Disconnect times were also shorter when on campus. On campus, it took 0.000112 seconds and off campus took 0.182433 seconds.

The two figures below are screen shots from Wireshark. Figure 5.1 shows when the TCP request was transmitted from the client. Figure 5.2 shows when the TCP connection was established on the server.

---

**Figure 5.1 TCP Request to Connect at line 67 on campus**
Next, a series of tests, each with 20 measurements, was conducted to test the time to send and receive a packet containing servo commands for both on and off campus, with and without a connection to the video server. Off campus and without a video server connection resulted in an average transmission time of 0.065768 seconds. On campus and without a video connection showed the average to be 0.002473 seconds. Off campus and with a video server connection, the average time to transmit a TCP packet with commands took an average of 0.068583 seconds. On campus, the same set of measurements resulted in an average of 0.010494 seconds. In all cases, the process of exchanging information between client and server, with or without video, was shorter on campus. Also, the transmission speed slowed down when video was also being transmitted. A histogram of the measurements is displayed in figures 5.3 through 5.6.
Figure 5.3 Histogram: Servo Command TCP Transmission Rate off Campus with No Video Connection in seconds

Figure 5.4 Histogram: Servo Command TCP Transmission Rate on Campus with No Video Connection in seconds
Another set of tests, each with 20 measurements, was conducted to measure the time span between forming a TCP connection and displaying the video feed on the client side. These were also tested both on and off campus. This also showed that the time span was less when conducted on campus. Off campus measurements proved an average of 2.68637 seconds while
on campus measurements averaged 1.797924 seconds. A histogram of the results is displayed in figures 5.7 and 5.8.

![Figure 5.7 Histogram: Time in seconds to Display Video Feed from TCP Connection Time off Campus](image1)

![Figure 5.8 Histogram: Time in seconds to Display Video Feed from TCP Connection Time on Campus](image2)

In terms of comparing results with and without a connection to the video server, the transmission rate for passing data to the server controlling robot motion was faster without the
video server running. The average difference in time taken to send new servo adjustments off campus was 0.002815 seconds. On campus results produced an average difference of 0.008021 seconds. Comparing on and off campus results, the average difference between transmitting data to the server controlling robot motion without a video connection was 0.063295 seconds. With the video server connection, the average difference was 0.058089 seconds.

Results favored the campus connection for time from initial TCP connection to the video server, and time to display the video frames to the client. From initial TCP connection to display time, the on campus transmission was faster by 0.888446 seconds.

In the end, the greater the distance, the longer it took to send and receive TCP packets. Also, connection to the video server directly impacted the speed at which servo command transmissions were passed, thus decreasing the performance. These results mirror the natural tendencies of network traffic and show that the algorithmic calculations performed on the client have no impact on robot command performance.
6. Conclusion

To reiterate, this document provides a detailed account of the robotic arm project. The project utilizes a client/server TCP protocol to broadcast video from the eBox server to the client. It also uses the protocol to transmit servo commands to the eBox and on to the robotic arm from the client. The server runs two programs simultaneously, one for video transmission and the other for receiving and passing the servo commands. This allows remote control of the robot. Both server-side and client side code are written in C#. The client-side also uses XNA Frameworks for handling the graphical map interface that collects points to be transformed into servo values.

This document also describes the starting point of the project that begins with an adequate user interface offering control to the user through adjustments to sliders, text boxes, and buttons where each servo may be manipulated separately. Once a servo adjustment was made, the program offered no recollection of previous servo position. The starting project also offered no means of spatial mapping, destination plotting, or user control to perform complex multi-servo maneuvers efficiently and relied entirely on a poor video transmission that often dropped frames and left the user wondering where the robot ended up.

The outcome of the project provides a means of control that relies much less on the video feed and relies on spatial plotting to set the destination points. The user can select radial, X, and Y values in a single mouse click and then move to the destination or to some point along the path between points at a variable speed. The program also allows the user to return to a previous position, either by reversing the direction of the scroll wheel movement or by clicking the ‘last position’ button.

By studying the results of the testing in section 5.1, it is clear that the accuracy increases as the user selects points closer to an actual measured value as stored in the arrays in the SharedResources.cs file. This is directly related the rotational nature of the servos themselves. As the chosen destination gets further from a measured value, the accuracy of the actual final position of the robot will decrease and rotate slightly above or below the plotted position. In the original design, servo measurements were only taken for the minimum and maximum X values.
of measured height values. This caused end positions of the robot to be off by as much as an inch from the plotted coordinate. By measuring an additional set of midpoint servo values to the midpoints in the range of X and the height of Y, the accuracy increased to within a quarter of an inch in the worst case scenarios except under the situations as described in section 5.1. This was a significant improvement and determined to be adequate for purposes of both this project and future projects.

The results of the performance testing in section 5.2 showed a direct correlation between distance and transmission speed. Servo command transmissions and video feed transmissions were faster the closer the client was to the server. Also, transmission rates were quicker without a connection to the video server. This is the typical nature of network traffic and implicitly shows that algorithmic calculations conducted in the client program have little to no impact on command performance.

In terms of future projects, the outcome of this one opens many doorways. First and foremost, it provides the possibility of creating playable motion files. By adding in a means of recording multiple destination plots to a file, a play button could be implemented to reiterate the motions. This would involve using the delta values of the scroll wheel to denote the speed of the action between points. It would also require a means of retrieving the files. To enhance this process, future students might consider buying pressure sensors to add to the gripper so that the user could be informed when the robot firmly grasps an object or when the robot successfully sets the object down and releases it. The student might also wish to create a timer, where a certain motion will be played for a period of time and then begin the next motion. It could also be programmed to perform certain tasks according to the pressures the robot senses.

The map interface was written using XNA frameworks. This platform allows the opportunity to create games or simulations for the user where virtual objects could be superimposed on the map and the user tries to interact with them, thus working as a robotic arm manipulation trainer.

Other improvements on the current project could be considered as well. Significant improvements could be made to the video transmission. One might consider writing a UDP protocol in order to reduce the frame loss ratio and gain a better video feed. This could also be
improved by adopting a different machine other than the eBox that has more processing power and RAM. A better algorithm for moving the arm incrementally with the scroll wheel should be looked at. Currently, the arm makes jerky motions with each adjustment. Very quickly, one could take the values that the program produces for each servo separately and find the average of all the servo adjustments per increment combined. When all servos increment at the same adjustment, the robot repositions smoothly. Instead of this, one could also establish a set speed for all servos per scroll increment. The intention of this project was to try and have all servos start and stop rotating at the same time, rather than one finish, then another, and so on. However, the outcome was the jerking motions clearly noted while scrolling. One last modification would be to work on the connection and disconnection from the server. Due to the low processing power of the eBox, the program will often disconnect from the server. If one continues with the eBox, this will be a natural and regular occurrence. The problem is that on some disconnections, but not all, when both the connection to the servo controller and the video controller on the eBox are lost, the program will crash and close. This should be looked at and resolved.
References


Appendix A:  Program Code

Game1.cs

using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Storage;

namespace RobotControl
{
    /// <summary>
    /// This is the main type for your game
    /// </summary>
    public class Game1 : Microsoft.Xna.Framework.Game
    {
        GraphicsDeviceManager graphics;
        SpriteBatch spriteBatch;

        SpriteFont coordinateFont;
        SpriteFont cursorLocation;
        private IntPtr drawSurface;
        SharedResources xyz;
        MouseState mouseStateCurrent;
        // Cursor cursor;

        Rectangle viewport;
        Vector2 x_letter = new Vector2(339, 328);
        Vector2 zy_letter = new Vector2(50, 40);
        Vector2 mousePos = new Vector2(0,0);
        ObjectModel degreeScreen;
        ObjectModel heightScreen;

        bool alreadyPressed;

        Vector2 ballPosOutput = new Vector2(25, 5);

        Vector2 minXPos = new Vector2(25, 5);
        Vector2 maxXPos = new Vector2(25, 30);
        Vector2 heightPos = new Vector2(25, 55);
        Vector2 rangePos = new Vector2(25, 80);
        Vector2 scalePos = new Vector2(25, 105);
        Vector2 ASPos = new Vector2(25, 130);
        Vector2 AFPos = new Vector2(25, 155);
        Vector2 AHPos = new Vector2(25, 180);
        Vector2 ABPos = new Vector2(25, 205);
Vector2 cursorPosOutput = new Vector2(200, 5);
Vector2 centerPos = new Vector2(200, 192); // center of degree graph
int yBaseCenter = 371; // center base position of x/y graph
Vector2 zeroPos = new Vector2(0, 0);
Random random;

public Game1(SharedResources xyz, IntPtr drawSurface)
{
    this.xyz = xyz;
    graphics = new GraphicsDeviceManager(this);
    Content.RootDirectory = "Content";
    this.drawSurface = drawSurface;
    graphics.PreparingDeviceSettings +=
        new EventHandler<PreparingDeviceSettingsEventArgs>
            (graphics_PreparingDeviceSettings);
    System.Windows.Forms.Control.FromHandle((this.Window.Handle)).VisibleChanged += new EventHandler(Game1_VisibleChanged);

    // sets center positions for plotting x, y, z positions from cursor
    xyz.setCenterX((int)centerPos.X);
    xyz.setCenterY(yBaseCenter);
    xyz.setCenterZ((int)centerPos.Y);
    viewport = new Rectangle(0,0,400,400);

    alreadyPressed = false;
}

/// <summary>
/// Event capturing the construction of a draw surface and makes sure
/// this gets redirected to a predesignated drawsurface marked by
/// pointer drawSurface
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
void graphics_PreparingDeviceSettings(object sender, PreparingDeviceSettingsEventArgs e)
{
    e.GraphicsDeviceInformation.PresentationParameters.DeviceWindowHandle = drawSurface;
    e.GraphicsDeviceInformation.PresentationParameters.BackBufferHeight = 400; //width of picturebox
    e.GraphicsDeviceInformation.PresentationParameters.BackBufferWidth = 400; //height of picturebox
}

/// <summary>
/// Occurs when the original gamewindows’ visibility changes and makes
/// sure it stays invisible
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void Game1_VisibleChanged(object sender, EventArgs e)
{
    if (System.Windows.Forms.Control.FromHandle((this.Window.Handle)).Visible == true)
        System.Windows.Forms.Control.FromHandle((this.Window.Handle))
protected override void Initialize()
{
    // TODO: Add your initialization logic here
    InitGraphicsMode(400, 720, false);
    this.IsMouseVisible = true;
    random = new Random();

    base.Initialize();
}

protected override void LoadContent()
{
    // Create a new SpriteBatch, which can be used to draw textures.
    spriteBatch = new SpriteBatch(GraphicsDevice);
    degreeScreen = new ObjectModel(Content.Load<Texture2D>("degreeGraph_new"));
    heightScreen = new ObjectModel(Content.Load<Texture2D>("x_y_Graph_new"));
    coordinateFont = Content.Load<SpriteFont>("x_Yfont");
    cursorLocation = Content.Load<SpriteFont>("Arial");
}

protected override void UnloadContent()
{
    // TODO: Unload any non ContentManager content here
}

protected override void Update(GameTime gameTime)
{
    // Allows the game to exit
    
        this.Exit();

    mouseStateCurrent = Mouse.GetState();
}
mousePos.X = mouseStateCurrent.X;
mousePos.Y = mouseStateCurrent.Y;

xyz.setDegrees();

// if left mouse button is pressed
if (!alreadyPressed && mouseStateCurrent.LeftButton == ButtonState.Pressed)
{
    alreadyPressed = true;
}

// if left mouse button is released
if (alreadyPressed && mouseStateCurrent.LeftButton == ButtonState.Released)
{
    alreadyPressed = false;
}

// if a right button event
if (mouseStateCurrent.RightButton == ButtonState.Pressed)
{
    xyz.setRightClickEvent(true);
}

base.Update(gameTime);
}

/// <summary>
/// This is called when the game should draw itself.
/// </summary>
protected override void Draw(GameTime gameTime)
{
    GraphicsDevice.Clear(Color.CornflowerBlue);

    // TODO: Add your drawing code here
    spriteBatch.Begin();

    // draws the XZ graph
    spriteBatch.Draw(degreeScreen.sprite, degreeScreen.positionXZ, Color.White);

    // draws the XY graph
    if (xyz.getKeyDown() && !xyz.getRightClickEvent())
    {
        spriteBatch.Draw(heightScreen.sprite, degreeScreen.positionXZ, Color.White);
    }
    spriteBatch.End();
    base.Draw(gameTime);
}

private bool InitGraphicsMode(int iWidth, int iHeight, bool bFullScreen)
{
    // If we aren't using a full screen mode, the height and width of
    // the window can be set to anything equal to or smaller than the
// actual screen size.
if (bFullScreen == false)
{
    {
        graphics.PreferredBackBufferWidth = iWidth;
        graphics.PreferredBackBufferHeight = iHeight;
        graphics.IsFullScreen = bFullScreen;
        graphics.ApplyChanges();
        return true;
    }
}
else
{
    // If we are using full screen mode, we should check to make sure that the display adapter can handle the video mode we are trying to set. To do this, we will iterate through the display modes supported by the adapter and check them against the mode we want to set.
    foreach (DisplayMode dm in GraphicsAdapter.DefaultAdapter.SupportedDisplayModes)
    {
        // Check the width and height of each mode against the passed values
        if ((dm.Width == iWidth) && (dm.Height == iHeight))
        {
            // The mode is supported, so set the buffer formats, apply changes and return
            graphics.PreferredBackBufferWidth = iWidth;
            graphics.PreferredBackBufferHeight = iHeight;
            graphics.IsFullScreen = bFullScreen;
            graphics.ApplyChanges();
            return true;
        }
    }
    return false;
}
return false;

mainForm.cs

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Net.Sockets;
using System.IO;
using System.Threading;

namespace RobotControl
{
    public partial class mainForm : Form
    {
        // client components to get video/commands communication
        TcpClient client;
        TcpClient client_video;
        NetworkStream ns;
        BinaryReader br;
        NetworkStream ns_v;
        BinaryReader br_v;
        SharedResources xyz;

        // network switch
        Boolean cmd;
        Boolean isConnected;
        Boolean firstTime;
        bool gripDirection;

        // thread to handle communication to server over port 17888
        ThreadStart ts;
        Thread th;

        delegate void PicHandler(Bitmap b);
        // method pointer when invoke requiered
        delegate void SetButtonEnableCallBack(Boolean input);
        // method pointer when invoke requiered
        delegate void SetButtonDisableCallBack(Boolean input);
        // method pointer when invoke requiered
        delegate void SetLabelText(string text);

        public mainForm(SharedResources xyz)
        {
            isConnected = false;
            this.xyz = xyz;
            firstTime = true;
            gripDirection = false;
            InitializeComponent();
        }

        public IntPtr getDrawSurface()
        {
            return pctSurface.Handle;
        }

        private void formMain_FormClosed(object sender, FormClosedEventArgs e)
        {
            Application.Exit();
        }
    }
}
// handles communication to port 17888, to send commands, // and to update arms' servos positions
private void Connect(IAsyncResult ia)
{
    TcpClient tc = ia.AsyncState as TcpClient;

    while (!tc.Connected)
    {
        output("Not Connected");
        continue;
    }

    output("Connected");
    ns = tc.GetStream();
    br = new BinaryReader(ns);

    WriteStream("1_" + xyz.getAB());
    WriteStream("2_" + xyz.getAS());
    WriteStream("3_" + xyz.getAF());
    WriteStream("4_" + xyz.getAH());
    WriteStream("5_" + xyz.getAG());

    while (cmd)
    {
        try
        {
            if (client.Available > 0)
            {
                // incoming string data
                string position = br.ReadString();
            }

            Thread.Sleep(0);
        }
        catch
        {
            MessageBox.Show("Connection Lost to Robot Server, press the connect button to connect again!");
            cmd = false;
        output("Disconnected");
        br.Close();
        ns.Close();
        client.Close();
        br_v.Close();
        ns_v.Close();
        client_video.Close();
        pictureBox3.BackColor = Color.Black;
        pictureBox3.Image = null;
        th.Abort();
        enables(true);
        disable(false);
        isConnected = false;
        ABTextBox.Text = "";
        ASTextBox.Text = "";
        AFTextBox.Text = "";
        AHTextBox.Text = "";
        AGTextBox.Text = "";
    }
xEndTextBox.Text = "";
yEndTextBox.Text = "";
zEndTextBox.Text = "";
resetXYZ();

// enable/disable form buttons
private void enable(bool change)
{
    if (change)
    {
        btnCon.Enabled = false;
        btnDCon.Enabled = true;
    }
    else
    {
        btnCon.Enabled = true;
        btnDCon.Enabled = false;
    }
}

// output commands/data
private void WriteStream(string pos)
{
    if (isConnected)
    {
        if (client.Connected)
        {
            NetworkStream ns_cmd = client.GetStream();
            BinaryWriter bw_cmd = new BinaryWriter(ns_cmd);
            bw_cmd.Write(pos);
        }
    }
}

// method pointer when invoke requiered
delegate void SetTextCallback(string text);

// handles communication to port 18888, to get real time video
private void Video(IAsyncResult ia_v)
{
    TcpClient tc_v = ia_v.AsyncState as TcpClient;

    // output message when port 18888 is not open
    if (!tc_v.Connected)
    {
        MessageBox.Show("Wait until port is open, or enter a valid IP");
        return;
    }

    while (!tc_v.Connected)
    {
        output("Not Connected");
        continue;
    }
}
output("Connected");
ns_v = tc_v.GetStream();
br_v = new BinaryReader(ns_v);

try {
    while (cmd) {
        output("Inside");
        if (client_video.Available > 0) {
            output("Inside2");
            int len = br_v.ReadInt32();
            if (len > 100) {
                output("Inside3");
                byte[] bt = br_v.ReadBytes(len);
                // transform incoming data into a bitmap
                MemoryStream ms = new MemoryStream(bt);
                Bitmap b = new Bitmap(ms);
                // place/refresh bitmap over a picture box
                this.Invoke(new PicHandler(UpdateImage), new object[]{b});
                ms.Close();
            }
        }
        Thread.Sleep(10);
    }
    output("OutsideAgain!");
} // when buffer overflows or connection lost
catch (Exception) {
    MessageBox.Show("Connection Lost to Video Server, press the connect button to connect again!");
    cmd = false;
    //output("Disconnected");
br.Close();
ns.Close();
client.Close();
br_v.Close();
ns_v.Close();
client_video.Close();
pictureBox3.BackColor = Color.Black;
pictureBox3.Image = null;
//th.Abort();
enables(true);
disable(false);
isConnected = false;

ABTextBox.Text = "";
ASTextBox.Text = "";
AFTTextBox.Text = "";
AHTextBox.Text = "";
AGTextBox.Text = "";
xEndTextBox.Text = "";
yEndTextBox.Text = "";  
zEndTextBox.Text = "";  
resetXYZ();  
}

// output connection status  
void output(string input)  
{
   try  
   {
      if (this.lblConn.InvokeRequired)
      {
         // It's on a different thread, so use Invoke.  
         SetLabelText text = new SetLabelText(output);  
         this.Invoke(text, new object[]{input});  
      }
      else
      {
         // It's on the same thread, no need for Invoke  
         this.lblConn.Text = input;
      }
   }
   catch {}  
}

void Listen()  
{
   // tcp object creation  
   client_video = new TcpClient();  
   // data buffer  
   client_video.ReceiveBufferSize = 2147483647;  
   // start communication  
   if (!client_video.Connected)
   {
      AsyncCallback ac_v = new AsyncCallback(Video);  
      client_video.BeginConnect("69.88.163.31", 18888, ac_v, client_video);
   }
}

void StartThread()  
{
   ts = new ThreadStart(Listen);  
   // start Listen for incoming client  
   th = new Thread(ts);  
   th.Start();  
}

private void btnCon_Click(object sender, EventArgs e)  
{
   xyz.resetServos();  
   resetXYZ();
   cmd = true;  
   StartThread();  
   lblConn.Text = "Waiting for Server Connection....";  
   // tcp object creation  
   client = new TcpClient();
// data buffer
client.ReceiveBufferSize = 20000000;

// start communication
if (!client.Connected)
{
    AsyncCallback ac = new AsyncCallback(Connect);
    client.BeginConnect("69.88.163.31", 17888, ac, client);
}

isConnected = true;
btnDCon.Enabled = true;
resetPostionButton.Enabled = true;
lastPositionButton.Enabled = true;
sendCMDButton.Enabled = true;
btnCon.Enabled = false;
firstTime = true;

ABTextBox.Text = xyz.getAB().ToString();
ASTextBox.Text = xyz.getAS().ToString();
AFTextBox.Text = xyz.getAF().ToString();
AHTextBox.Text = xyz.getAH().ToString();
AGTextBox.Text = xyz.getAG().ToString();

xEndTextBox.Text = "0";
yEndTextBox.Text = "14.25";
zEndTextBox.Text = "0";

// close servo connection
private void btnDCon_Click(object sender, EventArgs e)
{
    cmd = false;

    // disconnects tcp clients
    if (client_video.Connected || client.Connected)
    {
        MessageBox.Show("Connection Lost to Robot Server, press the connect button to connect again!");
        output("Disconnected");
        br.Close();
        ns.Close();
        client.Close();

        br_v.Close();
        ns_v.Close();
        client_video.Close();
pictureBox3.BackColor = Color.Black;
pictureBox3.Image = null;
        th.Abort();
        //enable(false);
        isConnected = false;
        btnDCon.Enabled = false;
        btnCon.Enabled = true;
        resetPostionButton.Enabled = false;
        lastPositionButton.Enabled = false;
        sendCMDButton.Enabled = false;
        ABTextBox.Text = "";
        ASTextBox.Text = "";
        AFTextBox.Text = "";
    }
}
AHTextBox.Text = "";
AGTextBox.Text = "";
xEndTextBox.Text = "";
yEndTextBox.Text = "";
zEndTextBox.Text = "";
resetXYZ();

//disable connect button
void enables(Boolean input)
{
    if (this.btnCon.InvokeRequired)
    {
        // It's on a different thread, so use Invoke.
        SetButtonEnableCallBack d = new SetButtonEnableCallBack(enables);
        this.Invoke
        (d, new object[] { input });
    }
    else
    {
        // It's on the same thread, no need for Invoke
        this.btnCon.Enabled = input;
    }
}

//enable disconnect button
void disable(Boolean input)
{
    if (this.btnDCon.InvokeRequired)
    {
        // It's on a different thread, so use Invoke.
        SetButtonDisableCallBack d = new SetButtonDisableCallBack(disable);
        this.Invoke
        (d, new object[] { input });
    }
    else
    {
        // It's on the same thread, no need for Invoke
        this.btnDCon.Enabled = input;
    }
}

// pictures capture by the server
private void UpdateImage(Bitmap b)
{
    this.pictureBox3.Image = b;
}

private void pctSurface_MouseDown(object sender, MouseEventArgs e)
{
    if (xyz.getRightClickEvent())
    {
        pctSurfaceTimer.Enabled = true;
        if (gripDirection)
        {
            gripDirection = false;
            // your code here
        }
    }
}
if (gripDirection = true)
else

else
// xyz.setZ(e.Y);
// xyz.setX(e.X);
// xyz.setKeyDown(true);

private void pctSurface_MouseUp(object sender, MouseEventArgs e)
{
    pctSurfaceTimer.Enabled = false;
    if (xyz.getRightClickEvent())
    {
        xyz.setRightClickEvent(false);
    }
    else
    {
        xyz.setY(e.Y);
        if (!xyz.isFor() && xyz.getY() < 0.75)
            xyz.replaceY(0.75);
        xyz.determineMinMaxRangeValues(); // calculates minimum/maximum/range values of robotic arm
        xyz.calculateServoPositions(); // determines servo values for a given point
        xyz.setKeyDown(false);
        // determining whether the final positions are greater than or less than what is actually possible for printing to textbox purposes
        double x;
        string msg = "";
        // is x less than minX
        if (xyz.getX() < xyz.getMinX())
        {
            x = xyz.getMinX();
            msg = " min";
        }
        // is x greater than maxX
        else if (xyz.getX() > xyz.getMaxX())
        {
            x = xyz.getMaxX();
            msg = " max";
        }
        // set to regular x value
        else
        {
        }
    }
}
x = xyz.getX();
}
exEndTextBox.Text = String.Format("{0:0.00}", x) + msg;
yEndTextBox.Text = String.Format("{0:0.00}", xyz.getY());
zEndTextBox.Text = String.Format("{0:0.00}", xyz.getDegrees());

ABTextBox.Text = xyz.getAB().ToString();
ASTextBox.Text = xyz.getAS().ToString();
AFTextBox.Text = xyz.getAF().ToString();
AHTextBox.Text = xyz.getAH().ToString();
AGTextBox.Text = xyz.getAG().ToString();

private void resetPostionButton_Click(object sender, EventArgs e)
{
    xyz.resetServos();
    resetXYZ();
    WriteStream("1_" + xyz.getAB());
    WriteStream("2_" + xyz.getAS());
    WriteStream("3_" + xyz.getAF());
    WriteStream("4_" + xyz.getAH());
    WriteStream("5_" + xyz.getAG());
    xEndTextBox.Text = "0";
    yEndTextBox.Text = "14.25";
    zEndTextBox.Text = "0";
    ABTextBox.Text = xyz.getAB().ToString();
    ASTextBox.Text = xyz.getAS().ToString();
    AFTextBox.Text = xyz.getAF().ToString();
    AHTextBox.Text = xyz.getAH().ToString();
    AGTextBox.Text = xyz.getAG().ToString();
}

protected override void OnMouseWheel(MouseEventArgs mea)
{
    Console.WriteLine("mea: " + mea.Delta);
    if (mea.Delta > 0)
    {
        if (xyz.getAB() != xyz.getcurAB() || xyz.getAS() != xyz.getcurAS() ||
            xyz.getAF() != xyz.getcurAF() || xyz.getAH() != xyz.getcurAH())
        {
            xyz.adjustServos(0);
            WriteStream("1_" + xyz.getcurAB().ToString());
            WriteStream("2_" + xyz.getcurAS().ToString());
            WriteStream("3_" + xyz.getcurAF().ToString());
            WriteStream("4_" + xyz.getcurAH().ToString());
        }
    }
    else
    {
        if (xyz.getPrevAB() != xyz.getcurAB() || xyz.getPrevAS() !=
            xyz.getcurAS() ||
            xyz.getPrevAF() != xyz.getcurAF() || xyz.getPrevAH() != xyz.getcurAH())
        {
            xyz.adjustServos(1);
        }
    }
}
private void resetXYZ()
{
    xyz.setX(0);
    xyz.setY(14.25);
    xyz.resetDegrees();
}

// this button resets the robotic arm to its current start position and allows the user to establish a new end location
private void lastPositionButton_Click(object sender, EventArgs e)
{
    xyz.resetToLastPosition();
    WriteStream("1_" + xyz.getAB().ToString());
    WriteStream("2_" + xyz.getAS().ToString());
    WriteStream("3_" + xyz.getAF().ToString());
    WriteStream("4_" + xyz.getAH().ToString());
    WriteStream("5_" + xyz.getAG().ToString());
    // resetting text boxes
    xEndTextBox.Text = "";
    yEndTextBox.Text = "";
    zEndTextBox.Text = "";
    ABTextBox.Text = xyz.getAB().ToString();
    ASTextBox.Text = xyz.getAS().ToString();
    AFTextBox.Text = xyz.getAF().ToString();
    AHTextBox.Text = xyz.getAH().ToString();
    AGTextBox.Text = xyz.getAG().ToString();
}

private void pctSurfaceTimer_Tick(object sender, EventArgs e)
{
    xyz.adjustAG(gripDirection);
    WriteStream("5_" + xyz.getAG().ToString());
}

private void sendCMDButton_Click(object sender, EventArgs e)
{
    if (ABTextBox.Text == "")
    {
        ABTextBox.Text = xyz.getPrevAB().ToString();
        WriteStream("1_" + xyz.getPrevAB().ToString());
    }
    else
    {
        WriteStream("1_" + ABTextBox.Text);
    }
}
if (ASTextBox.Text == "")
{
    ASTextBox.Text = xyz.getPrevAS().ToString();
    WriteStream("2_" + xyz.getPrevAS().ToString());
}
else
{
    WriteStream("2_" + ASTextBox.Text);
}
if (AFTextBox.Text == "")
{
    AFTextBox.Text = xyz.getPrevAF().ToString();
    WriteStream("3_" + xyz.getPrevAF().ToString());
}
else
{
    WriteStream("3_" + AFTextBox.Text);
}
if (AHTextBox.Text == "")
{
    AHTextBox.Text = xyz.getPrevAH().ToString();
    WriteStream("4_" + xyz.getPrevAH().ToString());
}
else
{
    WriteStream("4_" + AHTextBox.Text);
}
if (AGTextBox.Text == "")
{
    AGTextBox.Text = xyz.getPrevAG().ToString();
    WriteStream("5_" + xyz.getPrevAG().ToString());
}
else
{
    WriteStream("5_" + AGTextBox.Text);
}

xyz.setServos(Convert.ToInt32(ABTextBox.Text),
              Convert.ToInt32(ASTextBox.Text),
              Convert.ToInt32(AFTextBox.Text),
              Convert.ToInt32(AHTextBox.Text),
              Convert.ToInt32(AGTextBox.Text));
// since the formula for servos to X, Y, Z is not implemented
// this is intentionally left blank
xEndTextBox.Text = "";
yEndTextBox.Text = "";
zEndTextBox.Text = "";
namespace RobotControl
{
    partial class mainForm
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be disposed; otherwise, false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.components = new System.ComponentModel.Container();
            this.pctSurface = new System.Windows.Forms.PictureBox();
            this.pictureBox3 = new System.Windows.Forms.PictureBox();
            this.btnDCon = new System.Windows.Forms.Button();
            this.btnCon = new System.Windows.Forms.Button();
            this.lblConn = new System.Windows.Forms.Label();
            this.label1 = new System.Windows.Forms.Label();
            this.lastPositionButton = new System.Windows.Forms.Button();
            this.endPositionLabel = new System.Windows.Forms.Label();
            this.zELabel = new System.Windows.Forms.Label();
            this.yELabel = new System.Windows.Forms.Label();
            this.xELabel = new System.Windows.Forms.Label();
            this.zEndTextBox = new System.Windows.Forms.TextBox();
            this.yEndTextBox = new System.Windows.Forms.TextBox();
            this.xEndTextBox = new System.Windows.Forms.TextBox();
            this.AFTextBox = new System.Windows.Forms.TextBox();
            this.AHTextBox = new System.Windows.Forms.TextBox();
            this.pctSurfaceTimer = new System.Windows.Forms.Timer(this.components);
            this.zSLabel = new System.Windows.Forms.Label();
            this.ySLabel = new System.Windows.Forms.Label();
            this.xSLabel = new System.Windows.Forms.Label();
            this.AGTextBox = new System.Windows.Forms.TextBox();
            this.ABTextBox = new System.Windows.Forms.TextBox();
        }
    }
}
this.ASTextBox = new System.Windows.Forms.TextBox();
this.startPositionLabel = new System.Windows.Forms.Label();
this.AHLabel = new System.Windows.Forms.Label();
this.AGLabel = new System.Windows.Forms.Label();
this.sendCMDButton = new System.Windows.Forms.Button();
(System.ComponentModel.Model.ISupportInitialize)(this.pctSurface).BeginInit();
(System.ComponentModel.Model.ISupportInitialize)(this.pictureBox3).BeginInit();
this.SuspendLayout();
// pctSurface
this.pctSurface.Location = new System.Drawing.Point(215, 264);
this.pctSurface.Name = "pctSurface";
this.pctSurface.Size = new System.Drawing.Size(400, 400);
this.pctSurface.TabIndex = 0;
this.pctSurface.TabStop = false;
this.pctSurface.MouseDown += new System.Windows.Forms.MouseEventHandler(this.pctSurface_MouseDown);
this.pctSurface.MouseUp += new System.Windows.Forms.MouseEventHandler(this.pctSurface_MouseUp);
System.Windows.Forms.MouseEventHandler(this.pctSurface_MouseDown);
this.pctSurface.MouseUp += new System.Windows.Forms.MouseEventHandler(this.pctSurface_MouseUp);
// pictureBox3
this.pictureBox3.Location = new System.Drawing.Point(259, 10);
this.pictureBox3.Name = "pictureBox3";
this.pictureBox3.Size = new System.Drawing.Size(313, 235);
this.pictureBox3.TabIndex = 49;
this.pictureBox3.TabStop = false;
// btnDCon
(int)(((byte)(128)));
this.btnDCon.Enabled = false;
this.btnDCon.Font = new System.Drawing.Font("Arial Black", 9F,
this.btnDCon.ForeColor = System.Drawing.Color.White;
this.btnDCon.Location = new System.Drawing.Point(62, 127);
this.btnDCon.Name = "btnDCon";
this.btnDCon.Size = new System.Drawing.Size(130, 38);
this.btnDCon.TabIndex = 26;
this.btnDCon.Text = "Disconnect";
this.btnDCon.UseVisualStyleBackColor = false;
this.btnDCon.Click += new System.EventHandler(this.btnDCon_Click);
// btnCon
(int)(((byte)(255)));
this.btnCon.Enabled = false;
this.btnCon.Font = new System.Drawing.Font("Arial Black", 9F,
this.btnCon.ForeColor = System.Drawing.Color.White;
this.btnCon.Location = new System.Drawing.Point(62, 74);
this.btnCon.Name = "btnCon";
this.btnCon.Size = new System.Drawing.Size(130, 38);
this.btnCon.TabIndex = 24;
this.btnCon.Text = "Connect";
this.btnCon.UseVisualStyleBackColor = false;
this.btnCon.Click += new System.EventHandler(this.btnCon_Click);

// // lblConn
//
this.lblConn.AutoSize = true;
this.lblConn.Location = new System.Drawing.Point(25, 10);
this.lblConn.Name = "lblConn";
this.lblConn.Size = new System.Drawing.Size(0, 17);
this.lblConn.TabIndex = 34;

// // label1
//
this.label1.AutoSize = true;
this.label1.Location = new System.Drawing.Point(153, 10);
this.label1.Name = "label1";
this.label1.Size = new System.Drawing.Size(0, 13);
this.label1.TabIndex = 50;

// // resetPostionButton
//
this.resetPostionButton.Enabled = false;
this.resetPostionButton.Location = new System.Drawing.Point(641, 74);
this.resetPostionButton.Name = "resetPostionButton";
this.resetPostionButton.Size = new System.Drawing.Size(130, 38);
this.resetPostionButton.TabIndex = 51;
this.resetPostionButton.Text = "Reset Position";
this.resetPostionButton.UseVisualStyleBackColor = false;
this.resetPostionButton.Click += new System.EventHandler(this.resetPostionButton_Click);

// // lastPositionButton
//
this.lastPositionButton.Enabled = false;
this.lastPositionButton.Location = new System.Drawing.Point(641, 129);
this.lastPositionButton.Name = "lastPositionButton";
this.lastPositionButton.Size = new System.Drawing.Size(130, 38);
this.lastPositionButton.TabIndex = 52;
this.lastPositionButton.Text = "Last Position";
this.lastPositionButton.UseVisualStyleBackColor = false;
this.lastPositionButton.Click += new System.EventHandler(this.lastPositionButton_Click);

// endPositionLabel
//
this.endPositionLabel.AutoSize = true;
this.endPositionLabel.Location = new System.Drawing.Point(624, 365);
this.endPositionLabel.Name = "endPositionLabel";
this.endPositionLabel.Size = new System.Drawing.Size(147, 29);
this.endPositionLabel.TabIndex = 54;
this.endPositionLabel.Text = "End Position";
this.endPositionLabel.TextAlign = System.Drawing.ContentAlignment.MiddleCenter;

System.Drawing.ContentAlignment.MiddleCenter;

// zELabel
//
this.zELabel.AutoSize = true;
this.zELabel.ForeColor = System.Drawing.Color.Lime;
this.zELabel.Location = new System.Drawing.Point(646, 495);
this.zELabel.Name = "zELabel";
this.zELabel.Size = new System.Drawing.Size(16, 13);
this.zELabel.TabIndex = 79;
this.zELabel.Text = "R";
this.zELabel.TextAlign = System.Drawing.ContentAlignment.MiddleCenter;

// yELabel
//
this.yELabel.AutoSize = true;
this.yELabel.ForeColor = System.Drawing.Color.Lime;
this.yELabel.Location = new System.Drawing.Point(646, 457);
this.yELabel.Name = "yELabel";
this.yELabel.Size = new System.Drawing.Size(15, 13);
this.yELabel.TabIndex = 78;
this.yELabel.Text = "Y";
this.yELabel.TextAlign = System.Drawing.ContentAlignment.MiddleCenter;

// xELabel
//
this.xELabel.AutoSize = true;
this.xELabel.ForeColor = System.Drawing.Color.Lime;
this.xELabel.Location = new System.Drawing.Point(646, 418);
this.xELabel.Name = "xELabel";
this.xELabel.Size = new System.Drawing.Size(15, 13);
this.xELabel.TabIndex = 77;
this.xELabel.Text = "X";
this.xELabel.TextAlign = System.Drawing.ContentAlignment.MiddleCenter;

// zEndTextBox
this.zEndTextBox.Enabled = false;
this.zEndTextBox.Location = new System.Drawing.Point(671, 492);
this.zEndTextBox.Name = "zEndTextBox";
this.zEndTextBox.Size = new System.Drawing.Size(100, 20);
this.zEndTextBox.TabIndex = 76;

// yEndTextBox
this.yEndTextBox.Enabled = false;
this.yEndTextBox.Location = new System.Drawing.Point(671, 454);
this.yEndTextBox.Name = "yEndTextBox";
this.yEndTextBox.Size = new System.Drawing.Size(100, 20);
this.yEndTextBox.TabIndex = 75;

// xEndTextBox
this.xEndTextBox.Enabled = false;
this.xEndTextBox.Location = new System.Drawing.Point(671, 415);
this.xEndTextBox.Name = "xEndTextBox";
this.xEndTextBox.Size = new System.Drawing.Size(100, 20);
this.xEndTextBox.TabIndex = 74;

// AFTextBox
this.AFTextBox.Location = new System.Drawing.Point(76, 425);
this.AFTextBox.Name = "AFTextBox";
this.AFTextBox.Size = new System.Drawing.Size(100, 20);
this.AFTextBox.TabIndex = 81;

// AHTextBox
this.AHTextBox.Location = new System.Drawing.Point(76, 464);
this.AHTextBox.Name = "AHTextBox";
this.AHTextBox.Size = new System.Drawing.Size(100, 20);
this.AHTextBox.TabIndex = 82;

// pctSurfaceTimer
this.pctSurfaceTimer.Tick += new System.EventHandler(this.pctSurfaceTimer_Tick);

// zSLabel
this.zSLabel.AutoSize = true;
this.zSLabel.Location = new System.Drawing.Point(52, 428);
this.zSLabel.Name = "zSLabel";
this.zSLabel.Size = new System.Drawing.Size(22, 13);
this.zSLabel.Text = "AF";
this.zSLabel.TextAlign = System.Drawing.ContentAlignment.MiddleCenter;

//
//
ySLabel
//
this.ySLabel.AutoSize = true;
this.ySLabel.Location = new System.Drawing.Point(52, 390);
this.ySLabel.Name = "ySLabel";
this.ySLabel.Size = new System.Drawing.Size(23, 13);
this.ySLabel.Text = "AS";
this.ySLabel.TextAlign = System.Drawing.ContentAlignment.MiddleCenter;

//
//
xSLabel
//
this.xSLabel.AutoSize = true;
this.xSLabel.Location = new System.Drawing.Point(52, 351);
this.xSLabel.Name = "xSLabel";
this.xSLabel.Size = new System.Drawing.Size(23, 13);
this.xSLabel.Text = "AB";
this.xSLabel.TextAlign = System.Drawing.ContentAlignment.MiddleCenter;

//
//
AGTextBox
//
this.AGTextBox.Location = new System.Drawing.Point(76, 502);
this.AGTextBox.Name = "AGTextBox";
this.AGTextBox.Size = new System.Drawing.Size(100, 20);
this.AGTextBox.TabIndex = 58;

//
//
ABTextBox
//
this.ABTextBox.Location = new System.Drawing.Point(76, 348);
this.ABTextBox.Name = "ABTextBox";
this.ABTextBox.Size = new System.Drawing.Size(100, 20);
this.ABTextBox.TabIndex = 56;

//
//
ASTextBox
//

//
//

this.ASTextBox.Location = new System.Drawing.Point(76, 387);
this.ASTextBox.Name = "ASTextBox";
this.ASTextBox.Size = new System.Drawing.Size(100, 20);
this.ASTextBox.TabIndex = 57;

// startPositionLabel
//
this.startPositionLabel.AutoSize = true;
this.startPositionLabel.Font = new System.Drawing.Font("Miramonte", 18F,
((byte)(0)));
this.startPositionLabel.Location = new System.Drawing.Point(38, 298);
this.startPositionLabel.Name = "startPositionLabel";
this.startPositionLabel.Size = new System.Drawing.Size(171, 29);
this.startPositionLabel.TabIndex = 53;
this.startPositionLabel.Text = "Current Servos";
this.startPositionLabel.TextAlign = System.Drawing.ContentAlignment.MiddleCenter;

// AHLabel
//
this.AHLabel.AutoSize = true;
this.AHLabel.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F,
((byte)(0)));
this.AHLabel.Location = new System.Drawing.Point(52, 467);
this.AHLabel.Name = "AHLabel";
this.AHLabel.Size = new System.Drawing.Size(24, 13);
this.AHLabel.TabIndex = 87;
this.AHLabel.Text = "AH";
this.AHLabel.TextAlign = System.Drawing.ContentAlignment.MiddleCenter;

// AGLabel
//
this.AGLabel.AutoSize = true;
this.AGLabel.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F,
((byte)(0)));
this.AGLabel.Location = new System.Drawing.Point(52, 505);
this.AGLabel.Name = "AGLabel";
this.AGLabel.Size = new System.Drawing.Size(24, 13);
this.AGLabel.TabIndex = 88;
this.AGLabel.Text = "AG";
this.AGLabel.TextAlign = System.Drawing.ContentAlignment.MiddleCenter;

// sendCMDButton
//
((byte)(128)),
((byte)(255)));
this.sendCMDButton.Enabled = false;
this.sendCMDButton.Font = new System.Drawing.Font("Arial Black", 9F,
((byte)(0)));
this.sendCMDButton.ForeColor = System.Drawing.Color.White;
this.sendCMDButton.Location = new System.Drawing.Point(62, 544);
this.sendCMDButton.Name = "sendCMDButton";
this.sendCMDButton.Size = new System.Drawing.Size(130, 61);
this.sendCMDButton.TabIndex = 89;
this.sendCMDButton.Text = "Send Manual Servo Commands";
this.sendCMDButton.UseVisualStyleBackColor = false;
this.sendCMDButton.Click += new System.EventHandler(this.sendCMDButton_Click);

System.EventHandler(this.sendCMDButton_Click);

this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
((int)(((byte)(59))))), ((int)(((byte)(47)))));
this.ClientSize = new System.Drawing.Size(838, 681);
this.Controls.Add(this.sendCMDButton);
this.Controls.Add(this.AGLabel);
this.Controls.Add(this.AHLabel);
this.Controls.Add(this.AHTextBox);
this.Controls.Add(this.AFTextBox);
this.Controls.Add(this.zELabel);
this.Controls.Add(this.yELabel);
this.Controls.Add(this.yELabel);
this.Controls.Add(this.xELabel);
this.Controls.Add(this.zEndTextBox);
this.Controls.Add(this.yEndTextBox);
this.Controls.Add(this.xEndTextBox);
this.Controls.Add(this.zSLabel);
this.Controls.Add(this.ySLabel);
this.Controls.Add(this.xSLabel);
this.Controls.Add(this.AGTextBox);
this.Controls.Add(this.ASTextBox);
this.Controls.Add(this.ABTextBox);
this.Controls.Add(this.endPositionLabel);
this.Controls.Add(this.startPositionLabel);
this.Controls.Add(this.lastPositionButton);
this.Controls.Add(this.resetPositonButton);
this.Controls.Add(this.label1);
this.Controls.Add(this.lblConn);
this.Controls.Add(this.btnDCon);
this.Controls.Add(this.btnCon);
this.Controls.Add(this.pictureBox3);
this.Controls.Add(this.pctSurface);
this.MaximumSize = new System.Drawing.Size(854, 719);
this.MinimumSize = new System.Drawing.Size(854, 719);
this.Name = "mainForm";
this.Text = "Robotic Arm Control";
this.FormClosed += new System.Windows.Forms.FormClosedEventHandler(this.formMain_FormClosed);
((System.ComponentModel.ISupportInitialize)(this.pctSurface)).EndInit();
((System.ComponentModel.ISupportInitialize)(this.pictureBox3)).EndInit();
this.ResumeLayout(false);
this.PerformLayout();
}

#endregion
private System.Windows.Forms.PictureBox pictureBox3;
private System.Windows.Forms.Button btnDCon;
private System.Windows.Forms.Label lblConn;
private System.Windows.Forms.Label label1;
private System.Windows.Forms.Label endPositionLabel;
private System.Windows.Forms.Label zELabel;
private System.Windows.Forms.Label yELabel;
private System.Windows.Forms.Label xELabel;
private System.Windows.Forms.TextBox zEndTextBox;
private System.Windows.Forms.TextBox yEndTextBox;
private System.Windows.Forms.TextBox xEndTextBox;
private System.Windows.Forms.Label zSLabel;
private System.Windows.Forms.Label ySLabel;
private System.Windows.Forms.Label xSLabel;
private System.Windows.Forms.TextBox AGTextBox;
private System.Windows.Forms.TextBox ABTextBox;
private System.Windows.Forms.TextBox ASTextBox;
private System.Windows.Forms.Label startPositionLabel;
private System.Windows.Forms.Label AHLabel;
private System.Windows.Forms.Label AGLabel;

objectModel.cs

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Microsoft.Xna.Framework;

namespace RobotControl
{
    class ObjectModel
    {
        public Texture2D sprite;
        public Vector2 positionXZ;
        public Vector2 positionXY;
        public Vector2 center;
        public Vector2 velocity;
        public bool alive;
        public bool draw;
        public int height;
        public int width;
        public Rectangle rectXZ;
        public Rectangle rectXY;

        public ObjectModel(Texture2D loadedTexture)
{  
    positionXZ = Vector2.Zero;
    positionXY = Vector2.Zero;
    sprite = loadedTexture;
    center = new Vector2(sprite.Width / 2, sprite.Height / 2);
    velocity = Vector2.Zero;
    alive = false;
    draw = false;
    height = loadedTexture.Height;
    width = loadedTexture.Width;
    rectXZ = new Rectangle((int)positionXZ.X, (int)positionXZ.Y, width, height);
    rectXY = new Rectangle((int)positionXY.X, (int)positionXY.Y, width, height);
}

Program.cs

using System;
using System.Windows.Forms;
namespace RobotControl
{
#if WINDOWS || XBOX
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        static void Main(string[] args)
        {
            SharedResources xyz = new SharedResources();
            mainForm mainForm = new mainForm(xyz);
            mainForm.Show();
            /* Game1 game = new Game1(mainForm.getDrawSurface());
            game.Run();*/

            using (Game1 game = new Game1(xyz, mainForm.getDrawSurface()))
            {
                Form f = (Form)Form.FromHandle(game.Window.Handle);
                f.Text = "";
                f.Location = new System.Drawing.Point(0, 0);
                f.ShowIcon = false;
                f.MinimizeBox = false;
                f.MaximizeBox = false;
                f.ControlBox = false;
                game.Run();
            }
        }
    }
#endif
}

SharedResources.cs

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace RobotControl
{
    public class SharedResources
    {
        private double X; // distance from center
        private double Y; // height
        private double Z; // z is actually for degrees

        private double radianZ; // for calculating radians from pixels
        private Double radianX;
        private int centerX; // center of the graphs
        private int centerY;
        private int centerZ;

        private double scale = 16.4545;
        private double radians;
        private double angle;
        private bool rightClickEvent; // used to prevent recalculating x, y, z
        private bool keyDown;
        private bool isForward;

        private bool ASLess;
        private bool AFLess;
        private bool AHLess;
        private bool ABLess;

        private int index;

        private double XRange; // total range for a given height
        private double scaleX; // maximum displacement for a given height
        private double minScale; // minimum displacement for a given height

        private double displacementScale;

        private double minServo;
        private double maxServo;

        // initial position when program starts and when reset button is selected
        private int startAB = 90;
        private int startAS = 100;
        private int startAF = 90;
        private int startAH = 108;
        private int startAG = 49;

        // robot servos potential end position
        private int AB; // arm base
        private int AS; // arm shoulder
        private int AF; // arm frame
        private int AH; // arm hand
        private int AG; // arm gripper
private int prevAB; // prev arm base position
private int prevAS; // prev arm shoulder position
private int prevAF; // prev arm frame position
private int prevAH; // prev arm hand position
private int prevAG;

private int curAB; // current arm base position
private int curAS; // current arm shoulder position
private int curAF; // current arm frame position
private int curAH; // current arm hand position
private int curAG;

private int scrollWheelFactor = 20;

private double ABadjust = 0;
private double ASadjust = 0;
private double AFadjust = 0;
private double AHadjust = 0;

private double[] height = {0, 0.75, 1, 2.5, 3, 4, 6, 8, 8.5, 10, 12, 13, 13.5, 13.75, 14.25};

private double[] maxForward = {11.37, 11.37, 11.5, 11.5, 11.5, 11.37, 11.06, 10.12, 9.8, 8.75, 6.87, 5.16, 3.87, 2.87, 0};
private double[] midForward = {6.94, 7.0, 7.13, 7.4, 7.44, 7.25, 6.56, 5.8, 5.4, 4.38, 3.48, 2.58, 1.94, 1.44, 0};
private double[] minForward = {2.5, 2.62, 2.75, 3.31, 3.37, 3.12, 2.05, 1.47, 1, 0, 0, 0, 0, 0, 0};

private double[] maxBackward = {7.1, 7.12, 8.62, 10.19, 10.31, 10.6, 10.7, 10.12, 10, 8.7, 6.87, 5.27, 4.0, 3.2, 0};
private double[] midBackward = {7.1, 7.12, 7.31, 8.08, 8.16, 8.24, 7.82, 6.37, 5, 4.35, 3.44, 2.64, 2.0, 1.6, 0};
private double[] minBackward = {7.1, 7.12, 6, 5.96, 6, 5.87, 4.93, 2.62, 0, 0, 0, 0, 0, 0};

private int[] ASmaxForward = {20, 20, 20, 20, 30, 34, 42, 50, 53, 60, 72, 80, 85, 90, 100};
private int[] ASmidForward = {61, 67, 68, 79, 80, 86, 106, 109, 117, 123, 116, 113, 112, 109, 100};
private int[] ASminForward = {70, 73, 78, 95, 109, 125, 145, 145, 145, 145, 142, 129, 118, 116, 100};

private int[] ASmidBackward = {145, 145, 145, 137, 134, 129, 118, 103, 94, 87, 86, 88, 94, 91, 100};
private int[] ASminBackward = {145, 145, 145, 133, 129, 122, 104, 79, 58, 58, 64, 74, 86, 93, 100};

// robot reach

private double[] ASmaxForward = {20, 20, 20, 20, 30, 34, 42, 50, 53, 60, 72, 80, 85, 90, 100};
private double[] ASmidForward = {61, 67, 68, 79, 80, 86, 106, 109, 117, 123, 116, 113, 112, 109, 100};
private double[] ASminForward = {70, 73, 78, 95, 109, 125, 145, 145, 145, 145, 142, 129, 118, 116, 100};

private double[] ASmidBackward = {145, 145, 145, 137, 134, 129, 118, 103, 94, 87, 86, 88, 94, 91, 100};
private double[] ASminBackward = {145, 145, 145, 133, 129, 122, 104, 79, 58, 58, 64, 74, 86, 93, 100};

// arm frame per height measurement

// arm shoulder per height measurement

// arm frame per height measurement

84
private int[] AFmaxForward = {105, 106, 102, 91, 91, 91, 91, 91, 91, 91, 91, 91, 91, 90};
private int[] AFmidForward = {149, 152, 152, 149, 152, 152, 158, 148, 150, 144, 122, 112, 111, 108, 90};
private int[] AFminForward = {185, 185, 185, 185, 185, 185, 185, 175, 154, 148, 137, 143, 126, 109, 110, 90};

private int[] AFmaxBackward = {50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 90};
private int[] AFmidBackward = {50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 54, 63, 75, 75, 90};
private int[] AFminBackward = {50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 61, 82, 91, 90};

// arm hand per height measurement
private int[] AHmidForward = {49, 49, 49, 49, 49, 58, 68, 62, 66, 62, 66, 74, 81, 93, 108};
private int[] AHminForward = {20, 20, 20, 20, 20, 20, 20, 20, 32, 75, 78, 84, 93, 108};

private int[] AHmidBackward = {184, 181, 175, 162, 162, 162, 158, 166, 178, 166, 143, 135, 130, 130, 108};
private int[] AHminBackward = {184, 182, 207, 210, 210, 210, 210, 210, 210, 210, 210, 189, 147, 136, 139, 127, 108};

public SharedResources()
{
    X = 0;
    Y = 14.25;
    Z = 0;
    AB = startAB;
    AS = startAS;
    AF = startAF;
    AH = startAH;
    AG = startAG;

    prevAB = startAB;
    prevAS = startAS;
    prevAF = startAF;
    prevAH = startAH;
    prevAG = startAG;

    curAB = startAB;
    curAS = startAS;
    curAF = startAF;
    curAH = startAH;
    curAG = startAG;

    index = 0;
    rightClickEvent = false;
    keyDown = false;
    isForward = false;
}
// determining amounts to adjust servos per scroll wheel increment and
// set boolean for whether new destination point is => previous point
public void calculateServoAdjustments()
{
    double AB_adjust = ((double)AB - (double)prevAB) / scrollWheelFactor;
    if (Math.Abs(AB_adjust) < 1)
    {
        if (AB_adjust < 0)
        {
            AB_adjust = -1;
        }
        else
        {
            AB_adjust = 1;
        }
    }
    if (prevAB <= AB)
    {
        // then must increment to reach end point
        AB_less = false;
    }
    else
    {
        // must decrement to reach end point
        AB_less = true;
    }

    double AS_adjust = ((double)AS - (double)prevAS) / scrollWheelFactor;
    if (Math.Abs(AS_adjust) < 1)
    {
        if (AS_adjust < 0)
        {
            AS_adjust = -1;
        }
        else
        {
            AS_adjust = 1;
        }
    }
    if (prevAS <= AS)
    {
        // then must increment to reach end point
        AS_less = false;
    }
    else
    {
        // must decrement to reach end point
        AS_less = true;
    }

    double AF_adjust = ((double)AF - (double)prevAF) / scrollWheelFactor;
    if (Math.Abs(AF_adjust) < 1)
    {
        if (AF_adjust < 0)
        {
            AF_adjust = -1;
        }
        else
        {
            AF_adjust = 1;
        }
    }
    else
    {
if (prevAF <= AF)
{
   // then must increment to reach end point
   AFLess = false;
}
else
{
   // must decrement to reach end point
   AFLess = true;
}

AHadjust = ((double)AH - (double)prevAH) / scrollWheelFactor;
if(Math.Abs(AHadjust)<1) {
   if (AHadjust < 0)
   {
      AHadjust = -1;
   }
   else
   {
      AHadjust = 1;
   }
}
if (prevAH <= AH)
{
   // then must increment to reach end point
   AHLess = false;
}
else
{
   // must decrement to reach end point
   AHLess = true;
}

public void adjustServos(int direction)
{
   if (direction == 0)
   {
      curAB += (int)ABadjust;
      if (ABLess)
      {
         if (curAB <= AB)
         {
            curAB = AB;
         }
      }
      else
      {
         if (curAB > AB)
         {
            curAB = AB;
         }
      }
   }
curAS += (int)ASadjust;
if (ASLess)
{
    if (curAS <= AS)
    {
        curAS = AS;
    }
} else
{
    if (curAS > AS)
    {
        curAS = AS;
    }
}
curAF += (int)AFadjust;
if (AFLess)
{
    if (curAF <= AF)
    {
        curAF = AF;
    }
} else
{
    if (curAF > AF)
    {
        curAF = AF;
    }
}
curAH += (int)AHadjust;
if (AHLess)
{
    if (curAH <= AH)
    {
        curAH = AH;
    }
} else
{
    if (curAH > AH)
    {
        curAH = AH;
    }
}
else
{
    curAB -= (int)ABadjust;
    if (!ABLess)
    {
        if (curAB <= prevAB)
        {
            curAB = prevAB;
        }
    } else
    {

if (curAB > prevAB)
{
    curAB = prevAB;
}

curAS -= (int)ASadjust;
if (!ASLess)
{
    if (curAS <= prevAS)
    {
        curAS = prevAS;
    }
    else
    {
        if (curAS > prevAS)
        {
            curAS = prevAS;
        }
    }
}

curAF -= (int)AFadjust;
if (!AFLess)
{
    if (curAF <= prevAF)
    {
        curAF = prevAF;
    }
    else
    {
        if (curAF > prevAF)
        {
            curAF = prevAF;
        }
    }
}

curAH -= (int)AHadjust;
if (!AHLess)
{
    if (curAH <= prevAH)
    {
        curAH = prevAH;
    }
    else
    {
        if (curAH > prevAH)
        {
            curAH = prevAH;
        }
    }
}

public void resetToLastPosition()
{
    AB = prevAB;
    AS = prevAS;
AF = prevAF;
AH = prevAH;
AG = prevAG;
curAB = prevAB;
curAS = prevAS;
curAF = prevAF;
curAH = prevAH;
curAG = prevAG;
}

public int getPrevAS()
{
    return prevAS;
}

public int getPrevAF()
{
    return prevAF;
}

public int getPrevAH()
{
    return prevAH;
}

public int getPrevAB()
{
    return prevAB;
}

public int getPrevAG()
{
    return prevAG;
}

public void determineMinMaxRangeValues()
{
    // set starting point servo positions for next motion
    setPrevServos();
    index = 0;
    double min_y;
    double max_y;
    double min_x;
    double max_x;
    double displacement;
    double tempScale;
    double[] arrayMinX;
    double[] arrayMaxX;

    // finding the correct height index for determining minimum and maximum
    // range on the x-axis
    while(Y >= height[index] && index < height.Length - 1)
    {
        index++;
    }
}
// determining which list to use depending on whether the arm will move in
// front of or behind itself
if (isForward)
    {
        if (X < midForward[index])
        {
            arrayMinX = minForward;
            arrayMaxX = midForward;
        }
        else
        {
            arrayMinX = midForward;
            arrayMaxX = maxForward;
        }
    }
else
    {
        if (X < midBackward[index])
        {
            arrayMinX = minBackward;
            arrayMaxX = midBackward;
        }
        else
        {
            arrayMinX = midBackward;
            arrayMaxX = maxBackward;
        }
    }

    // if the index is maximum, then minimum, maximum and range for x-axis
    // is zero
if (index == height.Length)
    {
        minX = 14.25;
        maxX = 14.25;
        XRange = 0;
    }
else
    // otherwise there is a range
    {
        // determine minimum x-value at a particular height
        max_y = height[index];
        //if(index != 0 )
        //  min_y = height[index - 1];
        //else
        //   min_y = height[index-1];

        max_x = arrayMinX[index];
        //if(index != 0 )
        //  min_x = arrayMinX[index-1];
        //else
        //   min_x = arrayMinX[index-1];

        // difference of two minimum values / difference of two height values
        tempScale = (max_x - min_x) / (max_y - min_y);
        if (tempScale == 0)
            minX = min_x;
else // increments of x per each y increase from minimum y = new minX
define

minX = ((Y - min_y) * tempScale) + min_x;

// determine maximum x-value at a particular height
max_x = arrayMaxX[index];
//if(index != 0)
// min_x = arrayMaxX[index-1];
//else
// min_x = arrayMaxX[index-1];

/*
 tempScale = (max_x - min_x) / (max_y - min_y);
 if (tempScale == 0)
  maxX = max_x;
 else
  maxX = min_x + ((Y - min_y) * tempScale);
*/

// determine total range for given height
XRange = maxX - minX;

// determine displacement from minimum x value
displacement = X - minX;

// determine scale for calculating new servo values
displacementScale = displacement / XRange;
arrayMaxServoAH = AHmidForward;
} else {
    arrayMinServoAS = ASmidForward;
    arrayMaxServoAS = ASmaxForward;
    arrayMinServoAF = AFmidForward;
    arrayMaxServoAF = AFmaxForward;
    arrayMinServoAH = AHmidForward;
    arrayMaxServoAH = AHmaxForward;
}
else {
    if (X < midBackward[index]) {
        arrayMinServoAS = ASminBackward;
        arrayMaxServoAS = ASmidBackward;
        arrayMinServoAF = AFminBackward;
        arrayMaxServoAF = AFmidBackward;
        arrayMinServoAH = AHminBackward;
        arrayMaxServoAH = AHmidBackward;
    } else {
        arrayMinServoAS = ASmidBackward;
        arrayMaxServoAS = ASmaxBackward;
        arrayMinServoAF = AFmidBackward;
        arrayMaxServoAF = AFmaxBackward;
        arrayMinServoAH = AHmidBackward;
        arrayMaxServoAH = AHmaxBackward;
    }
}

// ********** ARM BASE CALCULATIONS **********
double tempAngle = angle;
if (tempAngle > 97 && tempAngle < 263) {
    tempAngle = 360 + (tempAngle - 180);
}
if (tempAngle >= 0 && tempAngle <= 97) {
    AB = (int)(startAB - tempAngle);
}
else {
    AB = (int)(startAB - (tempAngle - 360));
}

if (Y < 14.25) {
    // ********** ARM SHOULDER CALCULATIONS **********
    // getting max arm shoulder value
    max_servo = arrayMaxServoAS[index];
    //if(index != 0)
    min_servo = arrayMaxServoAS[index - 1];
    else
        min_servo = arrayMaxServoAS[index];
// determines a scale for servo adjustment to height
tempScale = (max_servo - min_servo) / (max_y - min_y);

if (tempScale == 0)
    maxServo = max_servo;
else
    maxServo = ((Y - min_y) * tempScale) + min_servo;

// getting min arm shoulder value
max_servo = arrayMinServoAS[index];
if (index != 0)
    min_servo = arrayMinServoAS[index - 1];
else
    min_servo = arrayMaxServoAS[index];

tempScale = (max_servo - min_servo) / (max_y - min_y);
if (tempScale == 0)
    minServo = min_servo;
else
    minServo = min_servo + ((Y - min_y) * tempScale);

AS = (int)(((maxServo - minServo) * displacementScale) + minServo);
if (AS < 20)
    AS = 20;
else if (AS > 145)
    AS = 145;

// ************* ARM FRAME CALCULATIONS *************
// getting max arm frame value
max_servo = arrayMaxServoAF[index];
if (index != 0)
    min_servo = arrayMaxServoAF[index - 1];
else
    min_servo = arrayMaxServoAF[index];

tempScale = (max_servo - min_servo) / (max_y - min_y);
if (tempScale == 0)
    maxServo = max_servo;
else
    maxServo = ((Y - min_y) * tempScale) + min_servo;

// getting min arm frame value
max_servo = arrayMinServoAF[index];
if (index != 0)
    min_servo = arrayMinServoAF[index - 1];
else
    min_servo = arrayMaxServoAF[index];

tempScale = (max_servo - min_servo) / (max_y - min_y);
if (tempScale == 0)
    minServo = min_servo;
else
    minServo = min_servo + ((Y - min_y) * tempScale);
AF = (int)(((maxServo - minServo) * displacementScale) + minServo);
if (AF < 50)
    AF = 50;
else if (AF > 185)
    AF = 185;

// *********** ARM HAND CALCULATIONS ***********
// getting max arm hand value
max_servo = arrayMaxServoAH[index];
//if (index != 0)
min_servo = arrayMaxServoAH[index - 1];
//else
//    min_servo = arrayMaxServoAH[index];

tempScale = (max_servo - min_servo) / (max_y - min_y);
if (tempScale == 0)
    maxServo = max_servo;
else
    maxServo = ((Y - min_y) * tempScale) + min_servo;

// getting min arm hand value
max_servo = arrayMinServoAH[index];
//if (index != 0)
min_servo = arrayMinServoAH[index - 1];
//else
//    min_servo = arrayMaxServoAH[index];

tempScale = (max_servo - min_servo) / (max_y - min_y);
if (tempScale == 0)
    minServo = min_servo;
else
    minServo = min_servo + ((Y - min_y) * tempScale);

AH = (int)(((maxServo - minServo) * displacementScale) + minServo);
if (AH < 20)
    AH = 20;
else if (AH > 210)
    AH = 210;

} else {
    AS = startAS;
    AF = startAF;
    AH = startAH;
}

// sets booleans for how the servos must adjust from the starting point in order to get the end point
calculateServoAdjustments();

public void setServos(int AB, int AS, int AF, int AH, int AG) {
    curAB = AB;
    curAS = AS;
    curAF = AF;
curAH = AH;
curAG = AG;
}
public int getAS()
{
    return AS;
}
public int getAF()
{
    return AF;
}
public int getAH()
{
    return AH;
}
public int getAB()
{
    return AB;
}
public int getAG()
{
    return AG;
}
public int getcurAS()
{
    return curAS;
}
public int getcurAF()
{
    return curAF;
}
public int getcurAH()
{
    return curAH;
}
public int getcurAB()
{
    return curAB;
}

// sets the starting servo positions for the next arm movement
public void setPrevServos()
{
    prevAB = curAB;
    prevAS = curAS;
    prevAF = curAF;
    prevAH = curAH;
    prevAG = curAG;
}
// resets all servo values for prev, current, and end positions
public void resetServos()
{
    AB = startAB;
    AS = startAS;
    AF = startAF;
    AH = startAH;
    AG = startAG;

    prevAB = startAB;
    prevAS = startAS;
    prevAF = startAF;
    prevAH = startAH;
    prevAG = startAG;

    curAB = startAB;
    curAS = startAS;
    curAF = startAF;
    curAH = startAH;
    curAG = startAG;
}

double getRange()
{
    return XRange;
}

double getDisplacementScale()
{
    return displacementScale;
}

double getMinX()
{
    return minX;
}

double getMinX()
{
    return maxX;
}

void setRange(int range)
{
    XRange = range;
}

void setMinX(int minX)
{
    this.minX = minX;
}

void setMaxX(int maxX)
{
    this.maxX = maxX;
}
// sets the X value based on angle A and B for a triangle \( A^2 + B^2 = C^2 \)
public void setX(double X)
{
    radianX = X;
    if (X == 0)
        this.X = X;
    else
    {
        double A = radianX - centerX;
        double B = radianZ - centerZ;
        this.X = (Math.Sqrt(A * A + B * B)) / scale;
    }
    if (this.X > 11.5)
        this.X = 11.5; // this may be changed to set a max for a particular width
}

public void replaceY(double Y)
{
    this.Y = Y;
}

public void setY(double Y)
{
    if (Y == 0)
        this.Y = Y;
    else
        this.Y = ((Y - centerY) * -1) / scale;
    if (this.Y > 14.25) // this may be changed to set a max for particular x value
        this.Y = 14.25;
    else if (this.Y < 0)
        this.Y = 0;
    Console.WriteLine("Y: " + this.Y);
}

// sets the Z value
public void setZ(double Z)
{
    radianZ = Z;
    if (Z == 0)
        this.Z = Z;
    else
        this.Z = ((Z - centerZ) * -1) / scale;
}

public double getX()
{
    return X;
}

public double getY()
{
    return Y;
}

public double getZ()
{
    return Z;
}
```java
public int getRadianX()
{
    return (int)radianX;
}

public int getRadianY()
{
    return (int)radianZ;
}

public void adjustAG(bool direction)
{
    if (direction)
        AG += 3;
    else
        AG -= 3;

    curAG = AG;
}

// sets that a right click event is occurring
public void setRightClickEvent(bool setEvent)
{
    rightClickEvent = setEvent;
}

public bool getRightClickEvent()
{
    return rightClickEvent;
}

public void setCenterX(int cX)
{
    centerX = cX;
}

public void setCenterY(int cY)
{
    centerY = cY;
}

public void setCenterZ(int cZ)
{
    centerZ = cZ;
}

// resets the angle to zero
public void resetDegrees()
{
    angle = 0;
}

// sets the angle for determining Arm Base Rotation
public void setDegrees()
{
    if (Z == 0 && X == 0)
angle = 0;
else
{
    radians = Math.Atan2((radianZ - centerZ), (radianX - centerX));
    angle = ((radians * (180 / Math.PI)) * -1) - 90;
    if (angle < 0)
    {
        angle += 360;
    }
}

// determining whether the robot is reaching in front of itself
if (angle >= 0 && angle <= 97 || angle >= 263 && angle <= 360)
{
    isForward = true;
}
else
{
    isForward = false;
}

// returns whether true or false whether the robot is reaching behind itself
public bool isFor()
{
    return isForward;
}
public double getDegrees()
{
    return angle;
}

// sets whether the left button is pressed or released
public void setKeyDown(bool flag)
{
    keyDown = flag;
}

// returns whether left mouse button is up or down
public bool getKeyDown()
{
    return keyDown;
}
Appendix B: User Manual

II. Overview

The purpose of the User Manual is to show the user how to install the RobotControl software, connect to the robotic arm, and manipulate it. This manual will provide step-by-step instructions.

II. Installing the Software

To install the software, simply double click the application file titled, ‘setup.’ On execution, the setup program will verify that you have all the required software necessary to install it. If you do not, the setup program will prompt you to download it. Follow the step by step instructions to download the required software.

Once completed, a security warning will display, stating that the publisher cannot be verified. Click the install button.
Upon successful installation, the application will be added to your list of programs accessible from the start menu and will automatically start up.

III. Connecting to the Robotic Arm

To connect to the robotic arm, simply click the button in the upper left with the text, ‘Connect.’ Upon successful connection, blue text will appear above the button, stating, ‘Connected.’ This will cause the ‘Connected’ button to be disabled and the ‘Disconnect’ button will be enabled. Other buttons will also be enabled. These include the ‘Reset Button’, the ‘Last Position’ button, and the ‘Send Servo Commands’ button. Also, after several seconds, the black box in the upper center will display a video feed showing the robotic arm in real time. A successful connection will also cause the robot to reset itself to its start position which is vertically upright and facing the camera.

(Figure 2.4) Video Stream when connected to the robot.
IV. Disconnecting from the Robotic Arm

To disconnect from the robotic arm, simply click the button below the ‘Connect’ button with the text, ‘Disconnect.’ Upon successful disconnection, the video display box will turn black and the robot will no longer be visible. Also, the ‘Disconnect’ button will become disabled. Disconnecting will cause the ‘Connect’ button to become enabled. It will also cause the remaining buttons on the display to be disabled. These include the ‘Reset Button’, the ‘Last Position’ button, and the ‘Send Servo Commands’ button.

![Figure B.2](image) Connect and Disconnect buttons

V. Plotting a Destination for the Robotic Arm

There are two ways to plot a destination for where the robotic arm will move. Either by using the text boxes along the left hand side under the heading ‘Current Servos,’ along with the ‘Send Servo Commands’ button directly beneath, or by plotting a destination on the map in the lower center of the interface.

V.1. Text Box Servo Control

The text boxes allow a user to simply type in different numerical values for each of the five joints on the robot. The ‘AB’ text box is for the arm base of the robot. This allows the user to rotate the base left or right up to 97 degrees. The ‘AS’ text box is for the arm shoulder of the robot. This is located just above the arm base and allows the user to reposition the shoulder forwards or backwards from its resting position standing straight up. The ‘AF’ text box controls the arm frame of the robot. This is elbow of the robot and is the next joint above the arm shoulder. The arm frame can rotate forwards and backwards from its start position. The ‘AH’
text box controls the arm hand of the robot. This is the next joint above the arm frame. It allows the user to rotate the hand forward or backwards from the start position. The ‘AG’ text box controls the arm gripper. This is the top most joint of the robot arm and controls how wide or far apart the opposite sides of the gripper are from each other. When all values are selected, the user simply needs to click the ‘Send Servo Commands’ button. The robot will immediately reposition itself to the new location.

(Figure B.3) ‘AB’, ‘AS’, ‘AF’, ‘AH’, ‘AG’ and ‘Current Servos’ label

V.2. Destination Plotting using the Map Control

The map in the lower center of the interface allows the user to plot destination points after which they can use the mouse to move to and from the plotted point. Initially, without any mouse interaction, a radial map is displayed. This map allows the user to select the arm base rotation as well as the distance out from the center of the robot base that the robot shall reach to. The radial map has eleven rings that grow in size from the center outward. Each ring from the center point represents a one inch change in distance out that the robot can attempt to reach to. The center 2.25 inches of the radial map appears in a gray color and represents the diameter of the robot’s base. The map also includes two red lines that start in the center and travel to the outside edge of the map. The larger portion of the map quadranted off by the red lines designates areas in front of the robot and the smaller portion represents reaches behind the robot. The radial map also displays number along the upper edge that denotes varying degrees from the robot’s starting position. To select a point on the map, simply use the left mouse button and click any desired point on the map.
While the left mouse button is pressed, a height map appears on the screen. This map allows the user to select the height from the ground to which the robot shall move. The height map includes a twenty-two by twenty-two grid of squares, where each square designates one inch. Also, drawn on the map is a plotted graph depicting the limits of the robot reach for a given height. In the center of the map is a vertical line in green that displays height values in inches. At the base of the vertical line, the map is numbered from one to eleven moving from the center out to the right. To the right of center, the map displays numbers from negative one to negative eleven. These numbers are used to determine the maximum reach possible for any given height.

To select the height at which the robot will move, simply release the left mouse button over the desired height value. One thing to take note of, if the reach out that was chosen exceeds the limitation at the selected height the program will set the reach value to the maximum value for that height.
On release of the left mouse button, the text boxes to the right of the map will be filled with the selected values chosen. They are denoted by the label ‘End Position.’ The ‘X’ text box shows the distance out in inches that the robot will reach. The ‘Y’ text box will display the chosen height value in inches. The ‘R’ text box will show the rotation in a 360 degree radius that the robot will rotate to.
VI. Moving the Robotic Arm to the Destination after Map Selection

After selecting points on the map interface, simply scroll the wheel forward to move to the destination point. The faster the scroll, the quicker the robot will move to the new location. By moving the scroll wheel backwards, the robot will move incrementally towards the current start location. At anytime between locations, the user may choose to plot a new destination on the map interface. When a new selection occurs, the position the robot was at when the selection was made becomes the new starting coordinate and the selected coordinate becomes the latest end coordinate.

To move the gripper, simply click and hold down the right mouse button. While the mouse button is pressed the gripper will close or open at a constant speed. Each press of the gripper changes the direction that the joint will rotate.

(Figure B.7) Mouse Control
VII. Other features

VII.1. ‘Reset Position’ Button

In the upper right of the interface is a button titled, ‘Reset Position.’ When this is pressed, the robot returns to its original starting position immediately, standing vertically to its highest reach and facing the camera. The arm is now reset to its original state when the program loads. Any selected positions previously made are removed. The ‘Current Servos’ text boxes display the start position joint values and the ‘End Position’ text boxes display the beginning values of the robot with an ‘X’ value of zero, a ‘Y’ value of 14.25 inches, and a rotation, R, of zero degrees.

VII.2. ‘Last Position’ Button

Just below the ‘Reset Position’ button is the ‘Last Position’ button. When pressed, the robot returns to the current start position immediately. The latest end point selection is removed completely and allows for a new destination coordinate to be plotted. The ‘Current Servos’ text boxes display the rotation values for each of the five joints at the start position and the ‘End Position’ text boxes are now blank.
(Figure B.9) Complete GUI