Table of Contents

[1. Project Overview 2](#_Toc441809229)

[2. Project Requirements & Achievements 2](#_Toc441809230)

[2.1 Rover Assembly 2](#_Toc441809231)

[2.2 Arm Sequence 3](#_Toc441809232)

[2.3 Programming the ALD5 Arm Using Arduino Uno 4](#_Toc441809233)

[2.4 XYZ Arm Program 8](#_Toc441809234)

[2.5 Object Recognition 15](#_Toc441809235)

[2.6 Pixels to Distance Conversion Explanation 15](#_Toc441809236)

[2.7 Rover’s Motor Controller Connections and Hardware Configuration 19](#_Toc441809237)

[2.8 Differential or Tank Mode Test Program: Controlling Rover Movement and Speed 23](#_Toc441809238)

[2.9 Leveraging Pulse Wave Modulation (PWM) to control DC Motor Movement and Speed 23](#_Toc441809239)

[2.10 RoverFollow Program: Tracking and Following an Object 28](#_Toc441809240)

[2.11 The HELPER Project Version 1.0 Features: 40](#_Toc441809241)

[3. Work Distribution & Project Schedule 41](#_Toc441809242)

[3.1 Work Distribution 41](#_Toc441809243)

[3.2 Project Meeting Schedule 41](#_Toc441809244)

[3.3 Task Schedule 42](#_Toc441809245)

[4. Future Steps 43](#_Toc441809246)

[4.1 Interfacing the Arm and PixyCam 43](#_Toc441809247)

[4.2 Integrating IR Sensors with Rover 44](#_Toc441809248)

[5. Process 45](#_Toc441809249)

[5.1 Team Experience 45](#_Toc441809250)

[5.2 Process Tools and Activities 45](#_Toc441809251)

[5.3 Engineering Tools and Activities 46](#_Toc441809252)

[5.4 Interesting Challenges 47](#_Toc441809253)

[6. Reflection 48](#_Toc441809254)

[6.1 Learning Outcomes 48](#_Toc441809255)

[6.2 Things to do differently 49](#_Toc441809256)

[7. References 50](#_Toc441809257)

# 1. Project Overview

Pollution levels in present day ecosystems have become one of the greatest threats to wildlife and humanity as a whole. Litter originating from houses, pedestrians, cars, and a myriad of other places have negatively impacted society not just aesthetically, but also economically, causing millions of dollars to be spent annually for litter clean-up. Although mankind is making an effort to change its ways, the rate of pollution exceeds the rate at which it is being cleaned up.

Human labor is incredibly expensive and inefficient when it comes to repetitive busy work such as picking up trash. The repetitiveness in picking up trash makes litter clean-up a task that an autonomous robot could do efficiently. The Handy Environmental Litter Pick-up & Eradicator Robot (HELPER) Project aims to address the waste problem by designing a robot autonomously identify garbage objects, move towards them, pick the items up and place them in a trash container.

Being a large scale proposal and task, several versions of the HELPER project are going to be constructed, with the current version being the first iteration mainly aiming for separate Lynxmotion arm/rover functionality with the use of a Pixy camera.

# 2. Project Requirements & Achievements

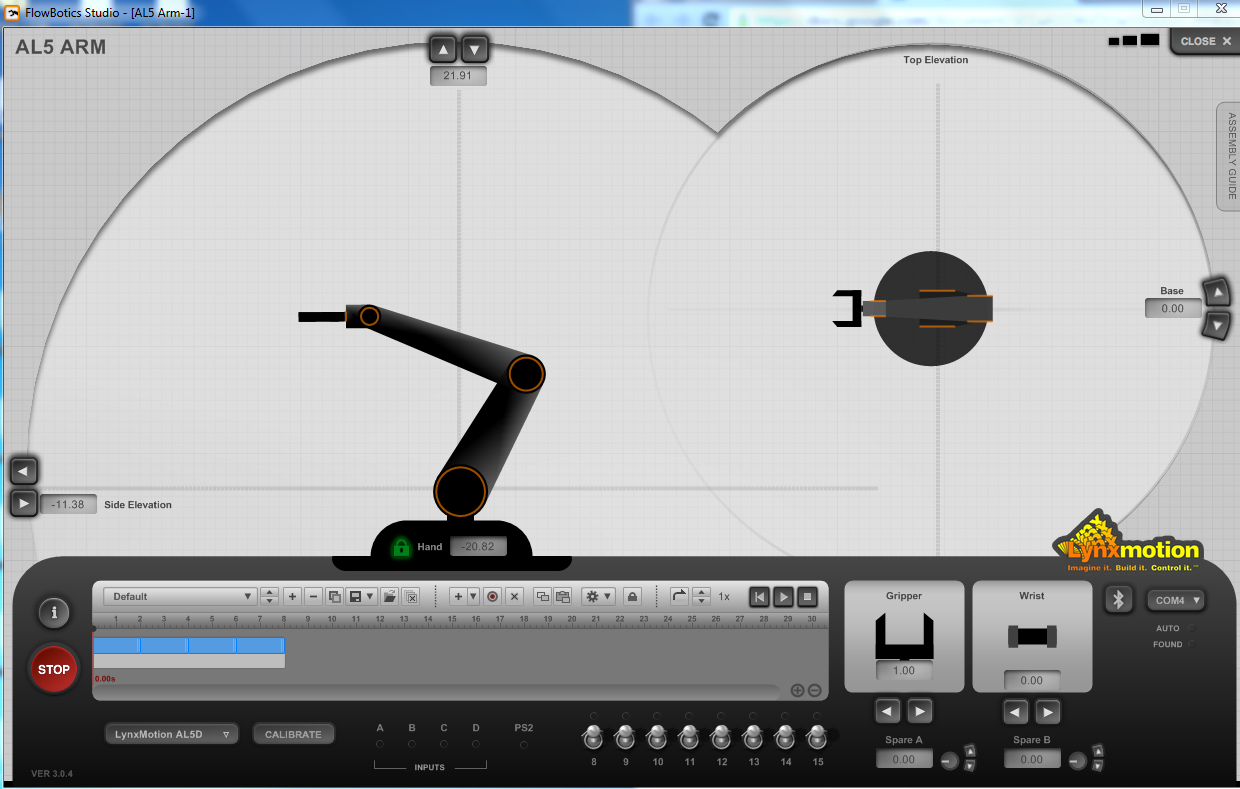
The main goal of the HELPER project Version 1.0 is to have the Lynxmotion ALD5 arm and rover successfully working separately; the arm should be able to pick up objects from various locations while the rover should autonomously follow an object. To reach this goal, several tasks had to be accomplished. The project requirements, achievements, and explanations are appropriately outlined below.

### 2.1 Rover Assembly

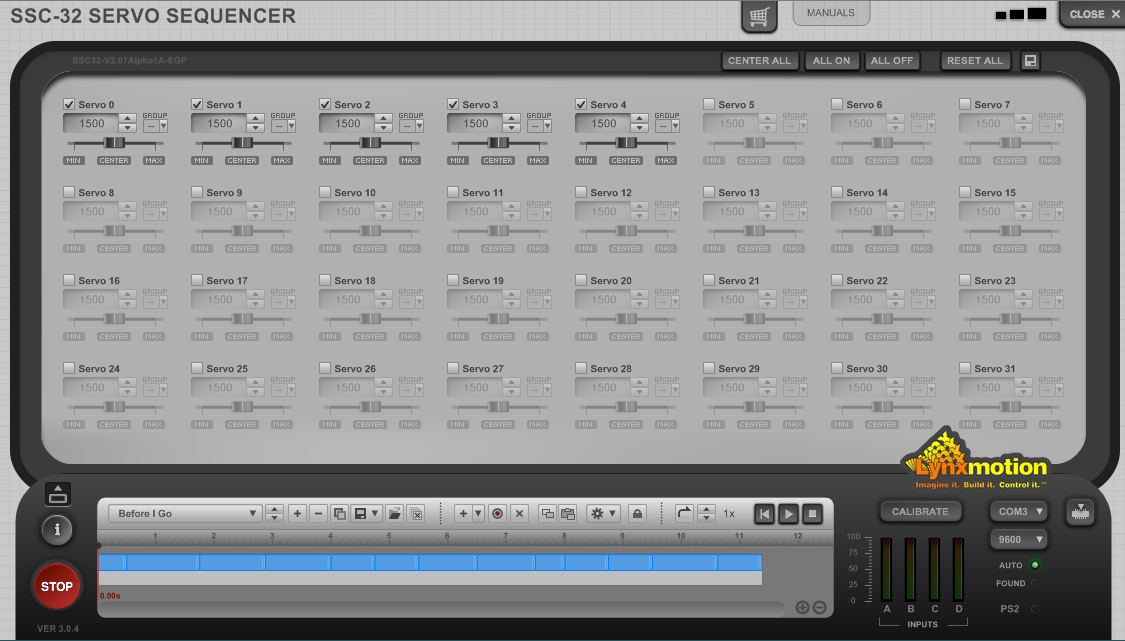
To transition from an idea to a tangible product, the necessary hardware to construct the robot was ordered. To choose the right vendor, a document was created to compare all of the different platforms available. Being one of the most durable and resilient platforms, Lynxmotion was the company chosen to supply the required hardware. Once the rover and ALD5 parts arrived, the online manuals and assembly directions were obtained. The team then formed two groups where one group focused on rover assembly and the other on constructing and manually calibrating the ALD5 arm.

### 2.2 Arm Sequence

In order to program the arm to move, the SSC-32U Servo board is directly connected to a computer, via USB, running the *FlowBotics Studio* graphical software. By utilizing the software’s virtual arm to create and record sequences, the ALD5 arm can be programmed to execute those sequences accordingly. A sequence was initially created to move the arm to predetermined positions and pick up objects at fixed location. *FlowBotics* *Studio* also provides a sequencer that takes advantage of sliders for quick changes and text boxes where the user can explicitly define the position of the servos. This sequencer also provides users complete calibration and baud rate control, which can be chosen as opposed to the graphical arm sequencer which only has two baud rate value options.



**Figure 2.1: Using *FlowBotics Studio*’s Virtual Arm Graphical Sequencer to Program ALD5 Arm**



**Figure 2.2: Using *FlowBotics Studio*’s Sequencer to Program ALD5 Arm**

### 2.3 Programming the ALD5 Arm Using Arduino Uno

In order to be able to program the ALD5 arm using the Arduino software, the Arduino Uno needed to be interfaced with the SSC-32U servo controller board so that they can communicate; the two boards used serial communication to transfer data.

After researching the serial command format required to position a servo using the SSC-32U controller along with an Arduino Uno, a program was written to successfully move each servo to predetermined positions.

In serial communication, there is a transmission line (TX), and a receiving line (RX). The signals from Arduino to the SSC-32U controller is sent via the Arduino’s transmission pin to the SSC-32U controller’s receiving pin. To position a servo, the SSC-32U controller must receive a serial command in the following format: #<ch> P <pw> S <spd> T <time> <cr>

1. <ch>: is the PWM channel to which the servo is connected to the SSC-32U controller
2. <pw>: is the desired pulse width in microseconds. This is the position that the servo will go to.
3. <spd>: servo movement speed in microseconds per second
4. <time>: the time in microseconds that the servo should take to travel from the current position to the desired position.
5. <cr>: Carriage return. The carriage return segment of the command can be achieved by using Arduino’s Serial.println() command to communicate the string instruction to the SSC-32U controller.

The requested SSC-32U controller command format can include all of the shown values; however, the only required values are: “#<ch> P <pw> <cr>”.

In order to understand how the position parameters affect the movement of each servo on the arm, a test code was written to manipulate each servo. It is important to note that a position of 1500µs, each servo is theoretically centered. The following are the findings:

|  |  |  |
| --- | --- | --- |
| **Table 2.3: Base Servo Channel #0** | | |
| Position < 1500 | Position = 1500 | Position > 1500 |
|  |  |  |

|  |  |  |
| --- | --- | --- |
| **Table 2.4 Shoulder Servo Channel #1** | | |
| Position < 1500 | Position = 1500 | Position > 1500 |
|  |  |  |

|  |  |  |
| --- | --- | --- |
| **Table 2.5: Elbow Servo Channel #2** | | |
| Position < 1500 | Position = 1500 | Position > 1500 |
|  |  |  |

|  |  |  |
| --- | --- | --- |
| **Table 2.6: Wrist Servo Channel #3** | | |
| Position < 1500 | Position = 1500 | Position > 1500 |
|  |  |  |

|  |  |  |
| --- | --- | --- |
| **Table 2.7: Grip Servo Channel #4** | | |
| Position < 1500 | Position = 1500 | Position > 1500 |
|  |  |  |

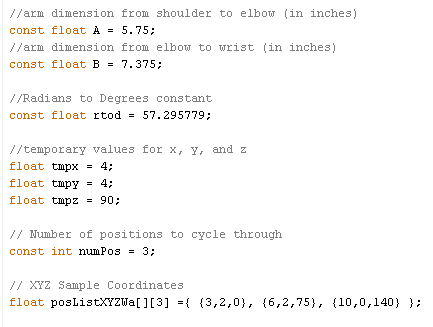
### 2.4 XYZ Arm Program

In the beginning stages of the HELPER project, the ALD5 arm was programed through the *FlowBotics Studio* software that communicated a predetermined sequence to the arm for it to imitate. This approach however, was not practical because the arm position and placement were not easily modified by the programmer. To create a more versatile program, an Arduino was interfaced with the SSC-32U controller and an XYZ positioning program was created. The main goal of the XYZ positioning program is to move the arm to a given XYZ position in a polar, three dimensional coordinate system. The first coordinate is the radius from the center point outward, followed by the height of the arm, and lastly, the rotational position. The extra flexibility of this program allows the arm to move to different positions by simply changing the coordinate values.

There are four main parts of the XYZ Code: the setup, Arm, pickUp, and loop function.

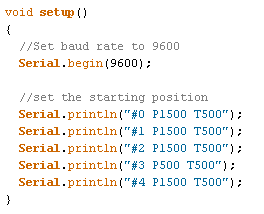
#### 2.4.1 Understanding Essential Variables

In order to fully understand the code, the purpose of the variables and constants used need to be known. The variables declared can be viewed in Figure 2.8 below.



**Figure 2.8: Variable Declarations in XYZ Program**

#### 2.4.2 The setup() Function



**Figure 2.9: Code within the setup Function**

The SSC-32 controller is shipped with a default baud rate of 9600. In order for the Arduino to communicate serially with the SSC-32 controller, the data rate in bits per second needs to be set to match that of the controller’s for successful serial data transmission. Once the baud rate is set in the code, the arm moves to a predetermined position.

#### 2.4.3 The Arm Function

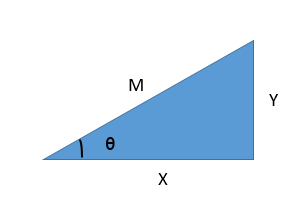


**Figure 2.10: Complete Code within the Arm Function**

The Arm(float x, float y, float z) function is the most complex out of all the functions in the XYZ program because it contains the inverse kinematics to control the arm. Diagrams will be used to explain the arithmetic behind the kinematics in the function. The following is a closer look of the function.

https://lh6.googleusercontent.com/wP8R58hZ-ZHc0A6lEFDhoVkTu6LX2M3obZ1WEAmApgI6l0WJqCO6eLGg8jeW3PxrIfH3fIe-2STPBZd44M7vI9deOT1puxwu2TAelfxh15ITDxiRpvJwoQh4c69X-kJfMZ54YpewdAxkmIJJ

Thinking of a two dimensional coordinate system, a right triangle can be created with the X and Y coordinates:



Pythagorean Theorem

**Figure 2.11 Right Triangle Created with X,Y Coordinate along the Pythagorean Theorem**

In the code, M is equal to. Solving for M in the Pythagorean Theorem equation, verifies that M is the hypotenuse (a non-negative value) of the right triangle.



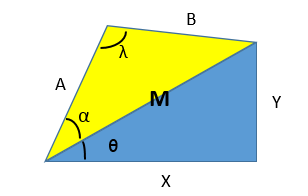
Using trigonometric functions:

Shows that by taking the inverse tangent of in the code, a value for theta (in the diagram is obtained and stored in A1 such that.

Due to the fact that an error is obtained when any number is divided by zero, the X coordinate cannot be zero thus explaining the value check of X within the Arm function.



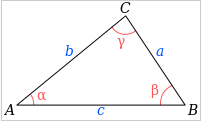
At this point, it is essential to understand the law of cosines relating the side lengths of a triangle to the cosine of one of its angles. This law is frequently used to compute unknown angles if all three sides of a triangle are known. The triangle made by the arm has a known shoulder to elbow length and a known elbow to wrist length defined as constants A and B (refer to section 2.3.1 for actual constant values). In this triangle, the length of the third side (the hypotenuse) is the same length M as the hypotenuse of the right triangle made by the X and Y coordinates.



**Figure 2.12 Right Triangle Created with X,Y Coordinate & Triangle Created by Arm Lengths A & B.**

Since the triangle made by the arm (yellow) has three known sides (A, B, and M), the law of cosines equation can be used to obtain the value of alpha.

Law of Cosines



**Figure 2.13 Law of Cosines Diagram and Equations**

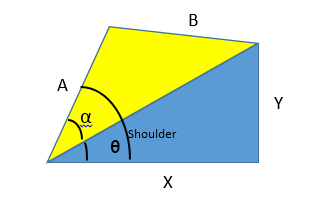
The equation for γ derived from the law of cosines shows that the angle measurement obtained from the formula is the angle opposite the side that has the negative squared value.

Following this logic, the function statement: gives value of the angle opposite to side B in the yellow triangle created by the arm dimensions A, B, and M. That means that variable A2 is equal to alpha in the two-triangle diagram. .

Additionally, the function statement: gives value of angle lambda, opposite to side M such that in the two-triangle diagram.



Lastly, the two angles and are added to form the shoulder angle of the arm relative to the X axis.



**Figure 2.14 Addition of Angles**

At this point in the code, the angle values to set the servos are known and the complex arithmetic is for the most part finished.



**Figure 2.15 Arm Code Involving Mapping the Angles and Serially Writing to the SSC-32U Controller**

The rest of the Arm function simply turns the Elbow and Shoulder angle values from degrees to radians. Note that neither of the values can be a negative value, otherwise there will be an error; this means that the X and Y coordinate need to be chosen in such a way that none of the calculations result in erroneous numbers.

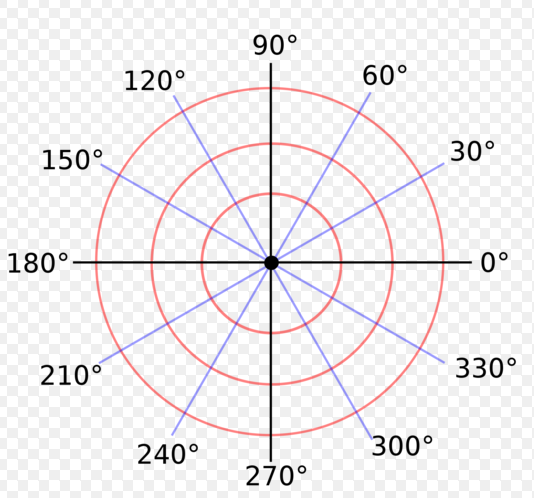
Lastly, the angle values are converted to microseconds (units the SSC-32U controller understands) through the map function which is then followed by arm placement at the calculated position through serial communication.

#### 2.4.4 The pickUp Function

#### 

**Figure 2.16 PickUp Function**

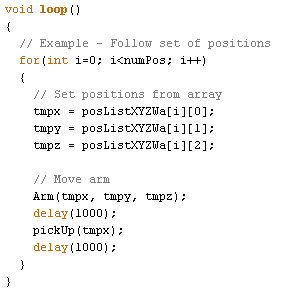
The XYZ code uses a polar coordinate system like the one displayed in Figure 2.17:



**Figure 2.17 Two Dimensional Polar Coordinate System**

The X dictates the radius from the center of the coordinate system, Y indicates the height that the arm is placed above that axis (the three-dimensional segment of the polar coordinate system), and Z indicates the rotation angle. For presentation purposes, three circular axis were chosen. The pickUp code directs the arm how to pick up an object that is on the ground at that XYZ position. Predetermined discrete values had to be manually calculated for each circular path so that the shoulder, elbow, wrist, and gripper will move accurately enough as to pick up the object without altering its resting location.

#### 2.4.5 The loop Function

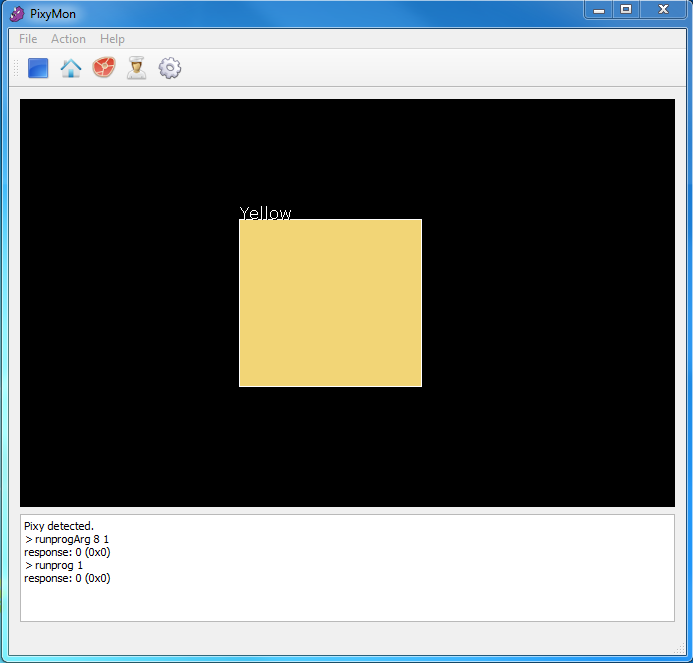


**Figure 2.18 Loop Function for XYZ Arm Program**

The loop function of the XYZ positioning program is constantly executed while power is supplied. The program runs a “for” loop that retrieves different coordinate values for the arm to move to and pick up an object.

### 2.5 Object Recognition

The Pixy camera uses a hue-based color filtering algorithm to detect objects. Through the *PixyMon* software package, the Pixy camera was successfully configured to learn and follow bright solid colored objects.



**Figure 2.19 Pixy Camera Recognizing an Object**

### 2.6 Pixels to Distance Conversion Explanation

The PixyCam, with its ability to center its position with an object, is incapable of detecting depth. The PixyCam can determine the height and width of the colour signature it reads using the number of pixels in the frame for which it occupies as a unit of measure. The camera can also find the center of a colour signature it detects and returns the x and y positions in pixels. Each frame is rendered with a 320 by 200 resolution. The origin (0, 0) starts in the top left corner and increases left to right, from 0 to 319, and top to bottom, 0 to 119.

#### 2.6.1 Obstacles and Solutions

The camera’s calculated area of a colour signature is heavily dependent on the distance the object is from the lens of the camera; if the object is close to the camera, it will occupy more pixels in the frame than if it is farther away. If the size or area of the object is known, a mapping between the number of pixels per inch and distance from the camera can be formulated. This mapping was created with a one inch by one inch yellow square taped to an “L” bracket to guarantee the same general placement within the frame, and with bright tape, as the camera recognizes bright colour signatures more consistently compared to a darker colour. Since the square was one inch in both height and width, the number of pixels the PixyCam returned as the height and width were pixels per inch. At first, the height and width readings provided by the camera were used to give an average number of pixels per inch because their readings, theoretically, should have been the same. With the use of *PixyMon* the team was able to see exactly what the camera saw and what area it registered as the target colour signature. The camera did have a tendency to determine the height with greater accuracy and consistency than the width of the detected object; therefore the final tests were run using only the height.

Distance measurements were originally meant to be taken from the lens of the camera to the target using a tape measure, but proved to be far more difficult. Collecting a large enough sample of data and manually inputting this data into a spreadsheet would take far more time than granted. To determine the distance of an object or an obstacle in a more automated fashion, the team employed the use a Sharp infrared sensor. The Sharp Infrared sensor is capable of determining the distance of an object 5 centimeters to 80 centimeters away. As simple as integrating an IR sensor to Arduino was, formulating a curve for measuring distance was far more complicated. The best angle to approach the curve was to divide the curve into intuitive sections where each section could be considered mostly linear. After searching the web for examples of how these sensors had been integrated into other projects, a Sharp Arduino library was found and added to the list of libraries being used. This simplified a large portion of the programming, as all that was needed was a function call and a conversion from centimeters to inches.

With the new found ability to take a large set of data, formatting was the final concern. Arduino cannot export directly to any spreadsheet program directly, so the needed information was printed from Arduino serially in Comma Separated Values format (CSV). CSV is a raw grid system format where columns are separated using commas and rows are distinguished with a new line. Once the data was collected, it was copied as a whole, pasted into a text file, and saved with the extension \*.CSV.

#### 2.6.2 Explanation of Pixels to Distance Conversion Code

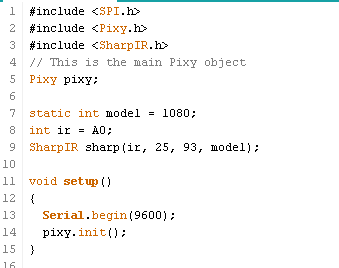
**Figure 2.20: Variable Declarations and setup function within the Distance Conversion Program**

Figure 2.20 shows the variable declarations and setup function. This program interfaces with the PixyCam unit and with a Sharp IR sensor. Under the libraries, a Pixy object is declared to access information processed by the camera. The Sharp library requires four pieces of information to create an object. The first argument is the pin number. IR sensors provide analog readings, therefore it connected to analog input A0. The SharpIR library provides averaged distances of the number of readings specified in the second argument; Figure 2.20 specifies that the sensor should take twenty five readings each time. The third parameter is the acceptable tolerance between measurements. SharpIR averaging function filters the measurements taken in each round, discarding outliers before averaging the data. “93” was used in an example of the library, and for simplicity, was reused. The final parameter is the sensor model number. Sharp sells many different variations of its infrared sensors and requires this parameter to know which distance-to-analog-voltage curve to refer to. The setup function executes only once on startup, and runs two main functions. The first initiates serial communication with the computer with a baud rate of 9600, followed by the Pixy initialization function that initiates SPI serial communication between the microcontroller and the PixyCam unit.



**Figure 2.21: Code within Loop Function within the Distance Conversion Program**

Figure 2.21 shows the loop function used to gather large amounts of data. The beginning of the function declares an integer variable called “blocks” which holds the number of blocks detected by the camera obtained from the “pixy.getBlocks()” function call. If there are blocks detected, Arduino enters the first “if” statement where a counter is incremented. The program was designed this way because printing serially every frame would bog down the Arduino. Every fifty iterations that a block is detected, the block’s distance from the camera is taken, converted to inches, and printed, followed by a comma to signify a new column, the height of the colour signature recognized, and the finally, the newline character. This program would produce an output shown below:

5.51,38

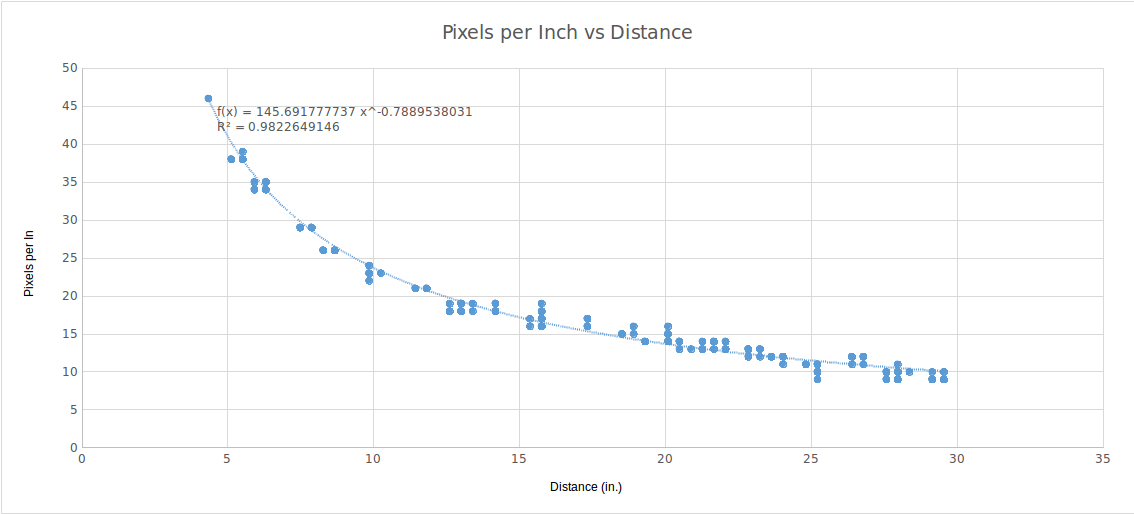
5.52,38

5.51,38

6.3,35

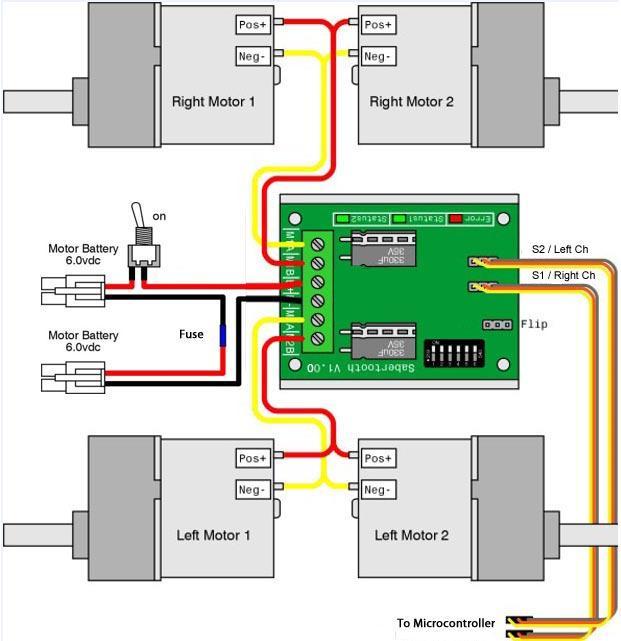
6.3,34

After importing the CSV file into Microsoft Excel, the data was plotted and the best fit curve was drawn. Several options were available to create the curve of best fit, but the best options were exponential and regression equations. The exponential equation was a lot longer and went to the third power, therefore it was decided that the regression curve was simpler to implement and had a correlation value of over 98%. Figure 2.2 shows the data, the curve and the formula. It is important to note that there are over 500 points plotted but many coincide and are overlaid on top of one another.

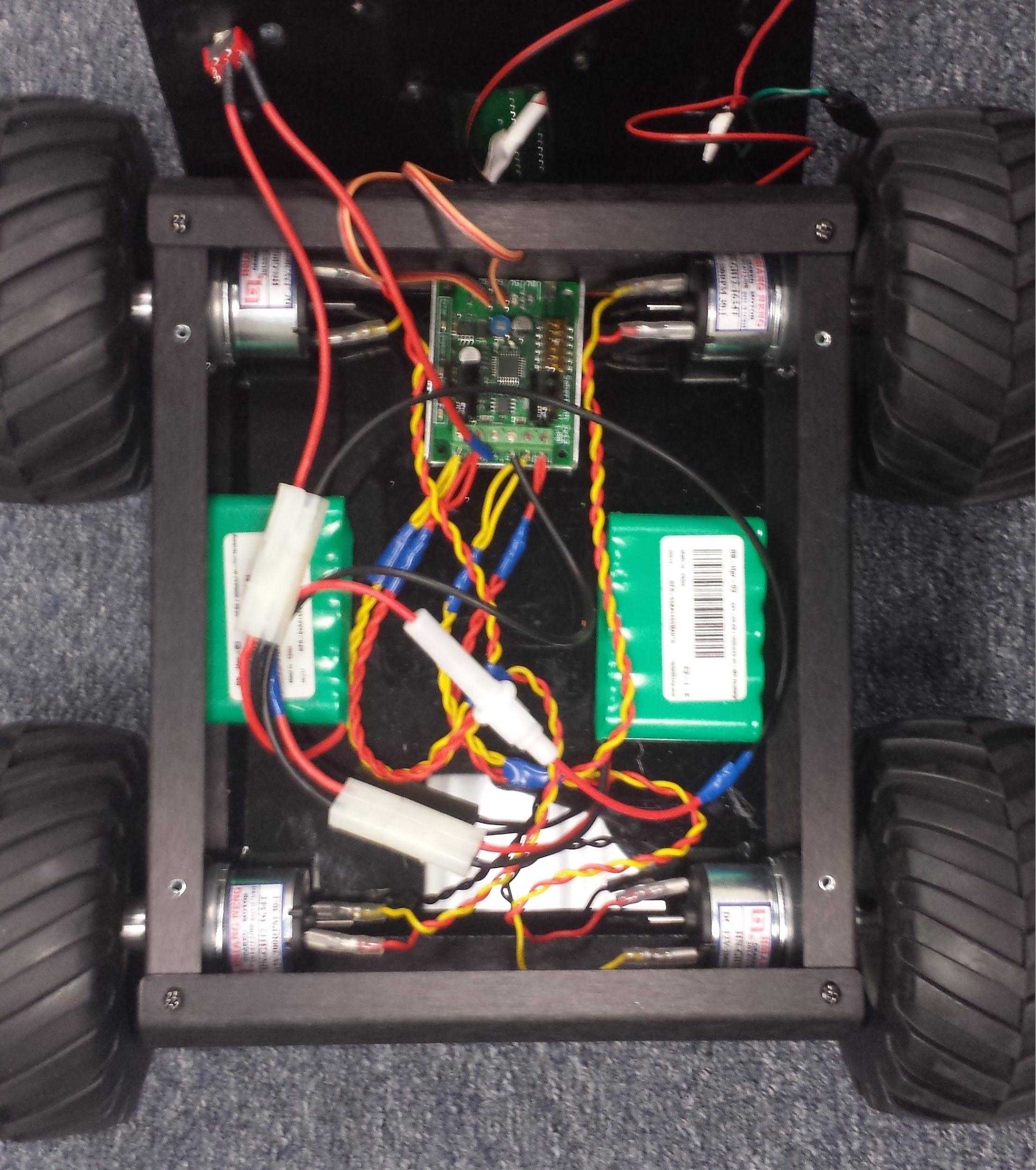


**Figure 2.22: Pixels per Inch vs Distance**

### 2.7 Rover’s Motor Controller Connections and Hardware Configuration



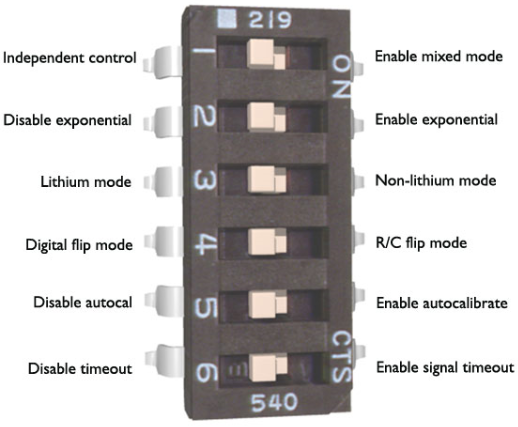
**Figure 2.23 Rover’s Motor controller Wiring Schematic**



**Figure 2.24 Actual Rover Wiring**

The Sabertooth's red Error LED will light to indicate overheating or current limit. The green Status1 LED will glow dimly when power is applied, and brightly when it is receiving pulses from the microcontroller. The green Status2 LED will flash out the detected number of lithium cells when lithium mode is enabled. Two BAT-03 6V/ 1600mAh Nickel-metal hydride (Ni-MH) Rechargeable Battery packs are used in series to power the Motor controller and the DC motors. Therefore the Status2 LED indicator is not of importance to the setup.

The rover moves on four HN-GH12-1634T 12V DC Plastic Gear head motors. Each motor requires 0.386 Amperes for maximum efficiency, meaning the Motor controller should not produce current much higher than 4 x 0.386= 1.544 Amperes. The fuse indicated in Figure 2.23 is used to ensure current does not exceed 2 Amperes. If it does, the fuse will burn, sever the power supply connection to the motor controller and thereby cut the power to the DC motors.



**Figure 2.25 Sabertooth 2x12 R/C DIP Switch Options Referenced in Table 2.26**

|  |  |  |  |
| --- | --- | --- | --- |
| **Table 2.26 Sabertooth 2x12 R/C DIP Switch Configuration** | | | |
| **DIP Switch** | **Position** | **Action** | **Explanation** |
| 1 | Off | Independent Control | The signal fed to the S1 input directly controls the right channel -- the two motors on the right side of the rover connected to the M1A and M1B output connections. The signal fed to S2 controls the left channel -- the two motors on the left side of the rover connected to the M2A and M2B output connections. The left and right channels are controlled independently of each other. |
| 2 | Off | Disable Exponential | Linear control mode. The Throttle and turning speed response relationship to the input signals is linear. This linear relationship is further discussed in Section 2.10.5. |
| 3 | On | Non-Lithium Mode | The switch must be in the up position when using NiCd, NiMH, or lead-acid batteries per Dimensions Engineering Specification. Two 6V/1600mAh NiMH rechargeable battery packs are used to power the motor controller and the four DC motors. |
| 4 | Off | Digital flip mode | Flip Control Switch. The flip channel of the motor controller is not used because the rover is not invertible; therefore, this switch is set to the OFF position. |
| 5 | On | Enable auto-calibrate | The Auto calibration feature reads the neutral position of the DC motors at power up and afterwards maps the highest throttle signal received thus far to 100% forward drive, and the lowest signal is mapped 100% reverse drive. After the rover is powered to full throttle and full reverse, the driver will have a proper calibration of what the throttle settings of the rover is. This is handy if throttle response is not symmetrical between forward and reverse drive. Turning off the auto-calibration mode will map a pulse width of 1000µs to full forward drive, 1500µs to rest state, and 2000µs for full reverse drive. Through experimentation with Differential\_or\_Tank\_Mode\_Test program, it was found that sending a signal of 1500µs often resulted is slow reverse motion of the right channel’s motors if the auto calibration mode was disabled. Enabling the auto calibration mode allowed for symmetrical control of the throttle responses of both motors using any standard Servo pulse width. With the auto calibration feature enabled, both channels’ motors did not rotate with a pulse width of 1500µs. |
| 6 | Off | Disable Timeout | This is a feature for Remote Controlled Vehicles and is not used in this project. The timeout failsafe will cause the motor controller to shut down the motors if it misses ten signals in a row from the transmitting device, preventing the vehicle from running away or crashing. This happens if the vehicle is encounters heavy radio interference and has lost signal; however as the rover is being controlled from a microcontroller, interference is not a problem. Disabling this feature gives greater control over the autonomous behavior of the rover. |

### 2.8 Differential or Tank Mode Test Program: Controlling Rover Movement and Speed

The Arduino Integrated Development Environment (IDE) was used to program the rover’s BotBoarduino microcontroller. A test program designed for an obsolete microcontroller was transposed and modified to make use of the board's three built-in push buttons to manually manipulate and control the speed and direction of the rover’s movement. The Differential\_or\_Tank\_Mode\_Test program was vital in testing the rover’s motor control. This program tests the motion of the rover using the A, B, and C push buttons on the BotBoarduino. The "A" button controls the right channel throttle. The "C" button controls the left channel throttle. The "B" button resets the speed and direction of both channels. Pressing the A button increments the right channel to full forward, then decrements to full reverse in 10% increments. Pressing the C button increments the Left channel to full forward, and then decrements to full reverse in 10% increments. Pressing the B button will reset the speed and direction of both left and right channels. The Sabertooth 2x12 R/C motor controller has two modes of operation. The default is the mixer mode.This is where one channel provides throttle for forward and reverse speed control, and the other channel provides steering control for turning. In this mode the left channel is used for throttle control and the right channel is used for steering control. This mode requires the Sabertooth's DIP Switch 1 to be flipped to the "On" or "Mixing Mode" position. The motor controller’s DIP Switch configuration is discussed in Table 2.26. The alternative mode of operation is independent channel control. This is where the left and right channels are controlled independently to steer like a tank. Because independent channel control provides finer control over the rover’s motion, the team decided to use this mode for the project.

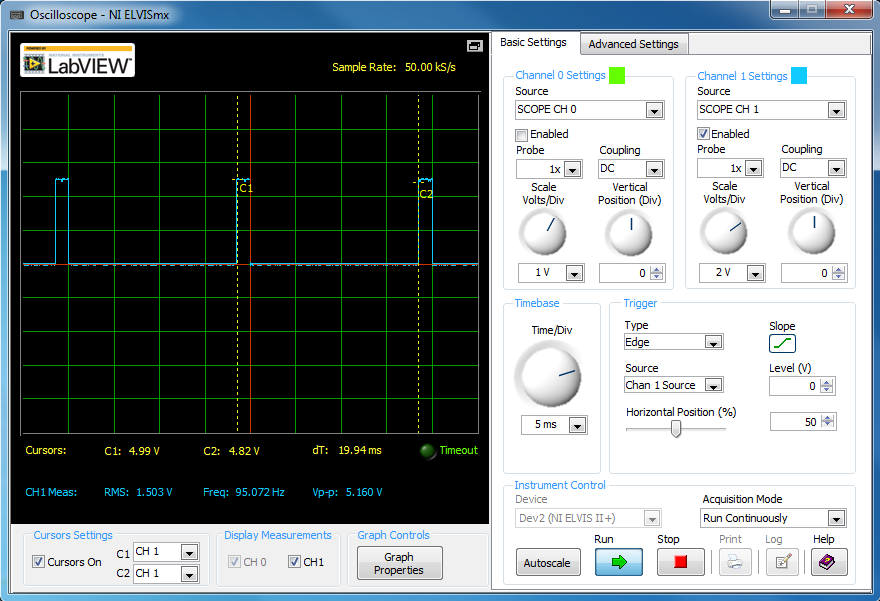
### 2.9 Leveraging Pulse Wave Modulation (PWM) to control DC Motor Movement and Speed

Microcontrollers are discrete devices whose output peripherals can be either logic LOW (0V) or logic HIGH (5V). The world does not operate in either on or off states, instead it is analog. Rather than just being on or off, lighting or brightness may need dimming, buzzers need to cycle through a range of sound frequencies, motors require control of speed, and servos need to move to certain positions. Atmel AVR microcontrollers like the ATmega328P-PU used in Arduino based boards do not have Digital-to-Analog Converters (DAC) to convert digital numbers into variable voltages. Pulse Width Modulation (PWM) is the closest solution to imitating analog control. By switching an output on and off at any frequency, the average amounts of time that the output is high can be obtained. The average voltage and current fed to an electrical load is controlled by the amount of time the switch between the power supply and load is on. The switching frequency has to be much higher than would affect the load, otherwise the response of the device being powered will be jerky. This project requires control of the speed of the DC motors of the rover. If a DC motor was connected to a battery, with time, it will eventually rotate at full speed. This response latency can be leveraged to control the motors’ speeds with PWM. If the power is repeatedly switched on and off a few times in a second, the motor’s rotation speed would increase and slow down in a non-continuous manner. By increasing the switching frequency, the resultant response becomes smoother. The speed of the motor can be changed by adjusting the duty cycle, which is the percentage of one period (the time it takes for a signal to complete an on-and-off cycle) in which the rectangular signal is active.

The Atmel AVR microcontrollers use various timers for producing modulated waveforms. There are four modules involved in producing a PWM signal: the clock, pre-scaler, timer, and comparator. The BotBoarduino ATmega328P-PU microcontroller is configured to use the external crystal clocked at 16 MHz. The pre-scaler serves to divide the clock frequency by a power value of 2 so as to slow down the counting process in the timer and clock cycle length in general. The Differential\_or\_Tank\_Mode\_Test and RoverFollow programs use the microcontroller’s 16-bit Timer/Counter unit. This timer was configured to use a pre-scale value of 8. The counter is incremented every <pre-scaler>/<clock frequency> seconds. Essentially, the timer adds one to its value every = 0.5µs. The comparator unit is provided a comparator value that is used to change the duty cycle of the PWM.

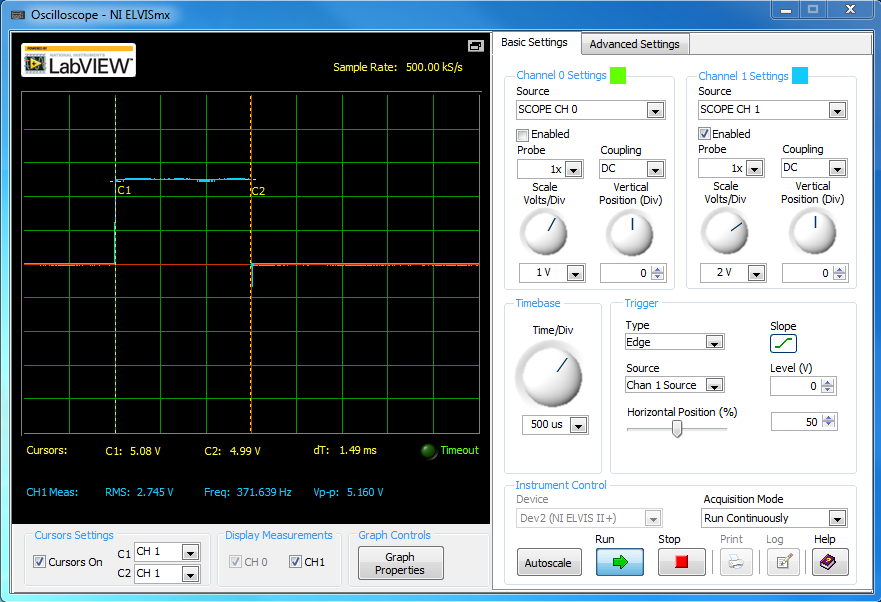
The timer mode of operation used is referred to as *Normal Mode*. In this mode, the timer count is always incrementing. The TOP for a counter is the highest value in the count sequence. The MAX for a counter is the maximum value a counter can hold before an integer overrun. The 16-bit counter overruns when it passes its MAX 16-bit value of 65535. In *Normal Mode* the TOP value is fixed to this same MAX value. The 16-bit timer allows for 16 bits of accuracy granting 65536 different comparator values or duty cycles. An 8-bit timer would only allow for 256 possible duty cycle values. 16-bit timers are therefore capable of generating waveforms of higher accuracy and flexibility. In *Normal Mode*, the output compare unit is used to generate interrupts at a provided time in Arduino’s Servo library.

The Sabertooth 2x12 R/C (Remote Control) Motor Controller was specifically optimized for radio controlled vehicles, however, it can be manipulated for autonomous movement. The “H-bridge” electronic circuit in the Sabertooth motor controller can supply the switching and control the current required to operate a DC motor forward and backwards. The motor controller serves as an interface between the servo pulses and an H-bridge. It is manufactured in such a way that it expects PWM signals exactly like a Servo does. This was not clearly documented in the Dimension’s Engineering Documentation; it was assumed the Motor Controller can be controlled with PWM signals of any duty cycle, as documentation did not say that this was not possible and most motor controllers can operate in such a fashion. The Remote Control (RC) PWM used to remote control servos acts very differently from the similar-sounding PWM DC motor speed control. Because the most common device a radio receiver controls is a servo, the signals the radio receiver produces is referred to as “R/C servo signals”. In order to control R/C components like the Sabertooth 2x12 R/C motor controller, the microcontroller must replicate the signals it expects to receive. Although the microcontroller’s PWM hardware module is used to generate servo control signals, and the pulse width is modulated, it is not correct to call the generated signals PWM signals because that implies the relevance of the duty cycle of the signal. The frequency of the pulse train does not affect the servo position if the pulse width remains the same. On this premise, the duty cycle does not necessarily affect servo position if the frequency of the pulse train does not. Therefore the rate at which pulses are sent to the Motor controller or the servo not particularly important, but the pulse width, that is the amount of time the signal is logic HIGH, is.

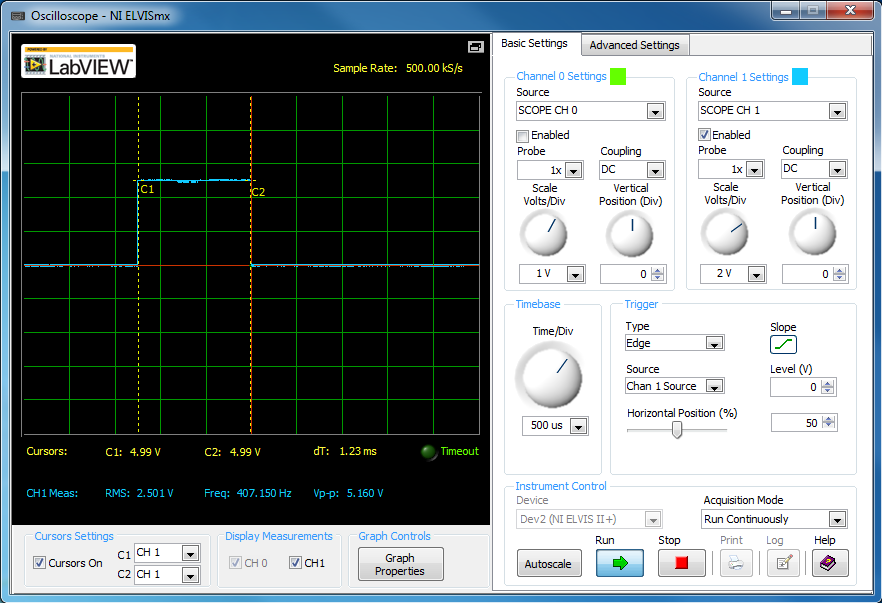


**Figure 2.27 1500µs Servo pulse train test generated by the Differential\_or\_Tank\_Mode\_Test Program. Note the distance between pulses is 20 milliseconds.**

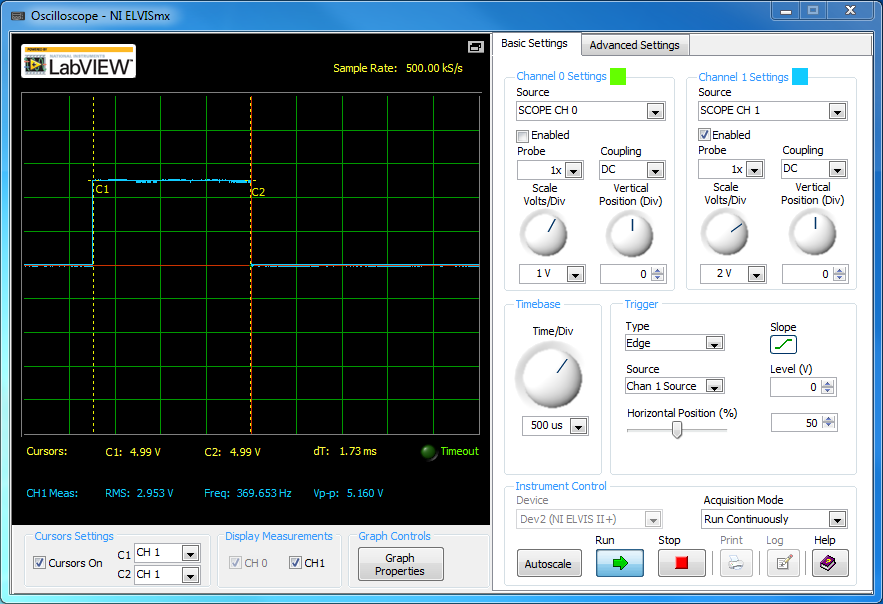
The typical pulse period for a servo is around 20 milliseconds as shown in Figure 2.27, which corresponds to a frequency of 50Hz. Depending on the width of the pulse a servo will move to a given location. The rover drive program uses the Arduino Servo library to generate PWM signals. The Servo library sets the pulse frequency at 50Hz. The pulse width is generally the most important part of the servo control interface. There is no standard correspondence between pulse width and servo position or DC motor motion. There is no standard minimum or maximum pulse either. Sabertooth Motor Controller documentation implicitly states that without signal calibration, a pulse width of 1000µs is used for full reverse, 1500µs for stop, and 2000µs for full forward motion. Through experimental testing it was found that a pulse width of 1000µs produces full forward motion and a pulse width of 2000µs produces full reverse motion. A pulse width of 1500 µs is considered neutral position as shown in Figure 2.28, at which a servo is positioned at 90 degrees or a DC motor is not rotated in either direction. Figure 2.29 displays a servo pulse of width 1250µs and Figure 2.30 displays a servo pulse of width 1750µs.



**Figure 2.28:** **1500µs Servo pulse generated by the Differential\_or\_Tank\_Mode\_Test Program.**



**Figure 2.29:** **1250µs Servo pulse generated by the Differential\_or\_Tank\_Mode\_Test Program.**



**Figure 2.30:** **1750µs Servo pulse generated by the Differential\_or\_Tank\_Mode\_Test Program.**

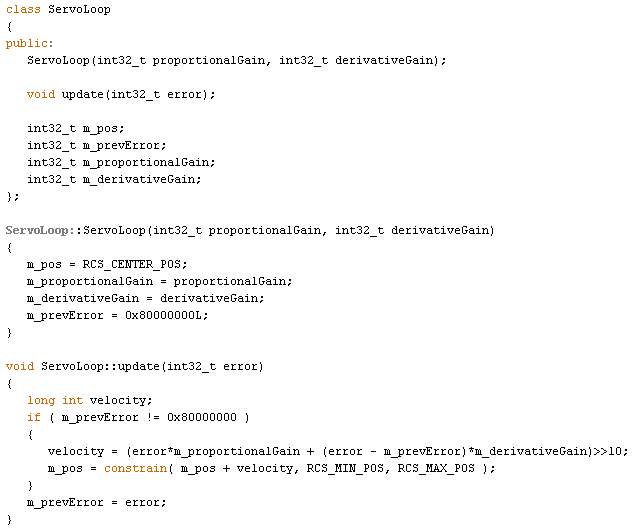
The Sabertooth motor controller expects high pulses between 1 and 2 milliseconds. The minimum pulse width of 1ms per 20ms represents a 5% duty cycle. The maximum pulse width of 2ms per 20ms represents a 10% duty cycle, hence a 5% duty cycle fluctuation between the minimum and maximum pulse widths. Only 5% of the timer’s comparator values will ultimately make any difference in how the DC motors behave. Out of the microcontroller’s 16-bit timer with = 65,536 different values, only 5% of the total of 65,536= 3,276 can be used. For Servo motors this is 3,276 different positions. Because the upper range of the pulse width between 1.5ms to 2ms controls reverse motion, and lower range of the pulse width between 1 ms to 1.5ms controls forward motion, the number of speed settings is actually half of the number of possible pulse widths the 16-bit timer can generate: . 1,638 different speed settings is certainly more than ample for DC motors. This speed control granularity is smaller than 1 microsecond. There are a total of 1000 microseconds between 1ms and 2ms. Arduino’s Servo library allows control of the active pulse width by a granularity of 1 microsecond. From the software perspective, there’s access to 500 different speed settings for both forward and backwards motion. The configuration of the microcontroller’s 16-bit timer as used in both the Differential\_or\_Tank\_Mode\_Test and RoverFollow programs is given in Table 2.31.

|  |  |
| --- | --- |
| **Table 2.31: ATmega328P-PU’s 16-bit Timer Waveform Generation Configuration** | |
| External Clock Speed | 16 MHz |
| Pre-Scaler | 8 |
| Comparator Mode | Non-inverting mode |
| Timer Mode of Operation | Normal Mode |
| Pulse Period | 20ms |
| Update of Comparator Register | Immediate |
| TOP Counter Value | 0xFFFF (65535) |

### 2.10 RoverFollow Program: Tracking and Following an Object

The RoverFollow program uses the Pixy camera module’s image processing capability to scan for, detect, and track an object with a pan/tilt servo mount mechanism. The program guides the rover to follow tracked objects using the panning servo’s position.

#### 2.10.1 The ServoLoop Class



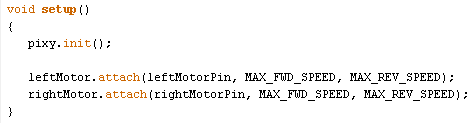
**Figure 2.32 ServoLoop Class of RoverFollow Program**

The ServoLoop class and its functionality was ported from the Pixy pan/tilt code and modified. The source code is shown in the FIGURE 2.32. This class defines four member variables that are used to describe the parameters associated with the movement of panning and tilt servos that command the camera’s horizontal and vertical motion. The m\_pos member variable holds a value between 0 and 1000, which the Pixy module maps to a range of pulse widths that vary from approximately 30 to 120 degrees in the unit circle, respectively. Each instance of the ServoLoop class created with the ServoLoop constructor initializes the starting position to 90 degrees. The proportionalGain and derivativeGain parameters dictate how fast the servo moves to a position. The update method calculates a servo position based on the measured error passed and the last measured error stored in the prevError member variable, dictating the camera’s tracking motion. The error value is based on the Pixy camera’s pixel positioning. The Pixy camera has a 200x320 pixel resolution. In this window an object detected at the leftmost corner has an X value of 0, and a value of 319 at the rightmost corner. An object detected at the top has a Y value of 0, and value of 199 at the bottom. This error value is the pixel offset from a center value. The error value for the Pan servo is the offset of a tracked object’s y-axis position from the Y center, which is 99. The error value for the Tilt servo is the offset of a tracked object’s x-axis position from the X center, which is 159.

The ServoLoop constructor initializes the prevError to a flag value. Because new servo positions are calculated based on a previous error measurement, the first time a ServoLoop instance calls the update function, the passed error value is saved in the prevError variable. All subsequent calls will calculate new servo positions. The velocity variable will hold the position offset to add to the current servo position value. This new position value is constrained to the range [0, 1000]. A Pan Servo position value in the range [0, 500) will move camera leftward from the servo’s neutral x-axis center at 500. A Tilt servo value in this same range will move the camera upward from the servo’s y-axis center at 500.

The variable velocity may seem a misnomer since it is really a position offset. It is the nature of servos however, that the greater the offset between the current and new position is, the faster the servo will move to the new position. The same position can be achieved by multiple calls to update function will smaller position offsets, and that will result in a more gradual movement towards the final destination. The velocity is proportional to the error value. The velocity in relation to error values can be proportionally controlled by the proportionalGain parameter. The higher the proportional gain is, the greater the position offset and ultimately the faster the servo’s motion will be. The sensitivity to change in error values can be proportionally controlled by the derivativeGain parameter. The higher the derivative gain value is, the more responsive the servo will be to changes in an object’s change in position.

#### 2.10.2 The Setup Function

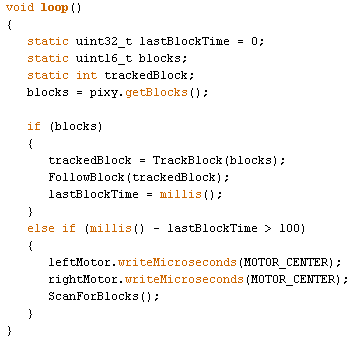


**Figure 2.33 Setup Function within the RoverFollow Program**

The setup function is the first function executed upon powering the BotBoarduino microcontroller. The pixy object is an instance of the Pixy class. The init API call initializes Serial Peripheral Interface (SPI) communication between the BotBoarduino ATmega328P-PU microcontroller as the master and the Pixy camera module as the slave. The Pixy camera module is connected to the BotBoarduino via the In-Circuit Serial Programming (ICSP) pins, a set of six lines, of which three are important for SPI: the MISO (Master In Slave Out) line for sending data to the master (microcontroller), the MOSI line for sending data to the slave (Pixy camera module), and the SCK (Serial Clock) line for clock pulses that synchronize data transmission generated by the master. Through SPI communication, the microcontroller receives processed image data from the pixy camera module, and the microcontroller sends servo positioning commands to the pan/tilt servo package the Pixy camera is mounted on.

The Servo class defines methods for manipulating servo motors connected to digital pins on boards running on Arduino. The Rover runs on four DC motors, which are controlled in a Tank Style manner. As discussed previously, the Sabertooth motor controller expects Servo pulses for rover motion control. The leftMotor Servo class object corresponds to the left channel of the rover, which consists of the two DC motors on the left side of the rover. The rightMotor Servo class object corresponds to the right channel of the rover, which consists of the two DC motors on the right side of the rover. The attach API call will configure the provided leftMotorPin and rightMotorPin digital pins on the BotBoarduino to behave as output signal pins. MAX\_FWD\_SPEED and MAX\_REV\_SPEED are microsecond values passed as parameters to the attach API that set the minimum and maximum active pulse width of the Servo signals sent to the Sabertooth Motor controller. As aforementioned, the Sabertooth 2x12 R/C motor controller is configured to interpret a pulse width of 1000µs as maximum forward motion, 1500µs pulse width as neutral or rest, and a pulse width of 2000µs as maximum reverse motion. The MAX\_FWD\_SPEED value must therefore be less than 1500 and more than, or equal to 1000; the MAX\_REV\_SPEED value must be more than 1500 and less than, or equal to 2000. The team experimented with various pulse width values within these ranges through Differential\_or\_Tank\_Mode\_Test program in order to observe the speed response of the DC motors. As it was observed that the rover is very fast, the MAX\_FWD\_SPEED and MAX\_REV\_SPEED values were capped around the middle of their respective ranges: MAX\_FWD\_SPEED is defined as 1300 and MAX\_REV\_SPEED is defined as 1700. This provides 200 speed settings for both forward and backwards motion. The actual difference in rover speed between the speed settings diminishes as the MAX\_FWD\_SPEED and MAX\_REV\_SPEED pulse widths are approached.

#### 2.10.3 The loop Function



**Figure 2.34 Loop Function within RoverFollow Program**

The loop function is effectively the main function of the RoverFollow program. It runs continuously after setup. The most important method used in Pixy’s arduino library is getBlocks(). This function returns the number of objects Pixy has detected. These are objects with distinct color signatures that the Pixy camera module was configured to recognize through the PixyMon software. The function copies information about the objects the Pixy cam has recognized, into the blocks array member of the pixy object instance. There is one array member for each detected object. The number of blocks the Pixy module returns is configurable through PixyMon, with the default being 1000. However, the maximum number of objects that Pixy can store in the block array is capped inside the Pixy library to be 130.

If the Pixy camera detects one or more blocks, the TrackBlock routine is called to return the index of the object with the largest pixel area for a particular signature. The FollowBlock routine is then called to make the rover follow the object returned by the TrackBlock routine.

millis() is an Arduino function that returns the number of milliseconds since the BotBoarduino began running the current program. The LastBlockTime variable holds the last time that Pixy detected an object. If no blocks have been detected for at least 100 milliseconds, the Rover’s motion will be ceased, and the ScanForBlocks routine will be called to scan for objects. The writeMicroseconds method sets the pulse width of the calling Servo object in microseconds. The MOTOR\_CENTER value is always defined to be 1500. Sending a signal with a pulse width of 1500µs to the left and right motor channels will bring the DC motor rotation to a complete stop.

#### 2.10.4 The TrackBlock Routine



**Figure 2.35 Code within TrackBlock Function in RoverFollow Program**

This routine tracks an object by use of the Pixy pan/tilt Servos and is inspired from Pixy’s pan/tilt Arduino program. Pixy’s region growing algorithm attempts to determine which pixels of an image are part of the detected objects and will inscribe this region inside distinct rectangular blocks. The first time this routine is called, the “for” loop logic will search through the blocks array for the object with the greatest area; regardless of its hue signature. The area of objects is calculated by multiplying the height and width in pixels of the detected object. The signature of the tracked object is saved so that in all subsequent calls to the routine, the “for” loop logic will search through the blocks array for objects with that signature and track the one with the greatest area. Because the pixel area of an object varies with distance, it is not accurate to say that the closest or the largest object is always being tracked, though this may well be the case.

To track the object, how far off center the object is in pixels relative to the X\_CENTER and Y\_CENTER of the image must be found in order to center the camera on it. The Pixy’s image center point in coordinates is defined by X\_CENTER and Y\_CENTER at (159, 99). Each block has two member variables x and y that hold the XY coordinates of the tracked object’s center. The pan servo horizontally positions the camera in accordance to the object’s x pixel position offset from the X\_CENTER value, and stores this position offset in the variable panError. The value range of the panError variable is [-159,159]. If the pan error is negative, the object is to the left of the image center. If the pan error is positive, the object is the right of the image center. The tilt servo vertically positions the camera in accordance to the object’s y pixel position offset from the Y\_CENTER value, and stores this position offset in the variable tiltError. The value range of the tiltError variable is [-99, 99]. If the tilt error is negative, the object is below the center point of the image. If the tilt error is positive, the object is above the center point of the image. The positions of the servos are set by update method discussed earlier.

#### 2.10.5 The FollowBlock Routine



**Figure 2.36 FollowBlock Function in RoverFollow Program**

The purpose of this routine is to mobilize the rover to follow a tracked object provided by the TrackBlock function. The following algorithm works on negative feedback. The speed and direction of the rover is proportional to the area in pixels of the tracked object. As aforementioned, the area of the object is calculated by multiplying the tracked block’s width and height pixel values. To smooth out the speed response of the rover relative to the area of the tracked object, the rolling average of the object’s area is calculated over 8 routine calls and saved in the local static variable size. The size of the object is linearly mapped to a speed value in a derived Speed range. The map function accepts five parameters x, in\_min, in\_max, out\_min, and out\_max. The x value is mapped from the range of values [in\_min, in\_max] to the range of values [out\_min, out\_max]. The area of a tracked object cannot be 0, it must have at dimensions of at least 1 pixel in width and 1 pixel in height. Because the camera’s image resolution is 320x200, the maximum possible pixel area is 64000. The map function does not constrain values to within the output range, however, the size of an object will always fall in the range of the minimum pixel area of 1 (MIN\_PIXEL\_AREA) to the maximum Pixel area of 64,000 (MAX\_PIXEL\_AREA).

A following distance that the rover will actively maintain between itself and the object being followed must be set. To compute the area PixyCam will try to maintain, the Pixels per Inch vs Distance formula mentioned in Section 2.6 is utilized and shown below:

In this formula, *y* is pixels per inch, and *x* is distance in inches. Using a 6.25 inch ball for the rover to follow, the distance was set to approximately 18 inches, producing the following:

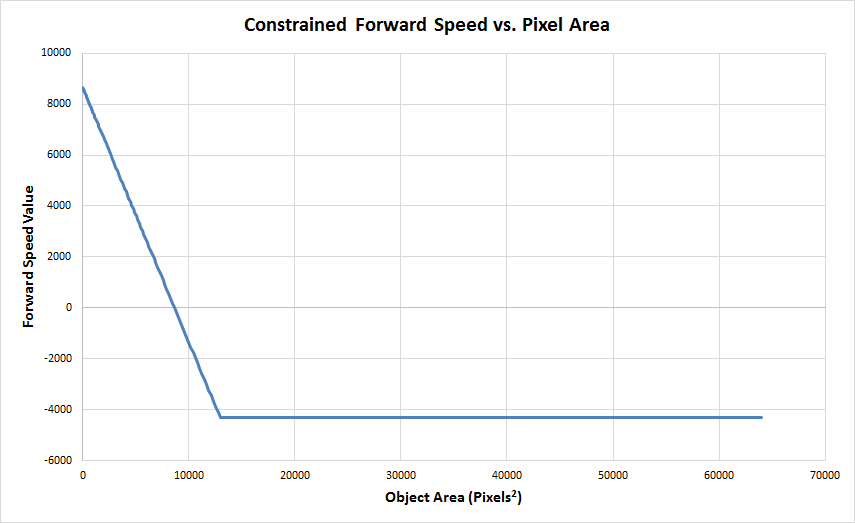
With 14.89 pixels per inch for an object 18 inches away, the last part of the process is to determine the area the PixyCam will maintain once it is tracking an object. The first step is to determine the actual size of the object it will track. A 6 ¼ inch diameter bright yellow *Volt* dodgeball is used as the object for the PixyCam to track. Computing the number of pixels that will span the diameter of the ball is basic arithmetic, as is computing the area of the rectangular box Pixy will identify as the ball; both of which are shown below. The rover will maintain the area of the object at 8,649 pixels squared. This area corresponds directly to how large the ball will look to the PixyCam when it is 18 inches away.

Using a mapping function, a linear speed response curve was created with a zero speed intersection at an area of 8649. The mapping function is given below, where x is the value to map. in\_min and in\_max are the lower and upper bound of the values’ current rage, respectively; out\_min and out\_max are the lower and upper bound of the value’s target range, respectively.

By setting y to 0 and substituting known values in the mapping function, the proportional relationship between the out\_max and out\_min bounds that define the speed value range can be found. The calculations are provided below.

The out\_max and out\_min values were set to -55351 and 8648, respectively. These bound values may be computed in a simple way to set the zero speed point at any given pixel area, as is done in the FollowBlock routine. HIGHEST\_SPEED\_VAL is the out\_min value and is defined to be the desired stand pixel area to stop at minus 1: STAND\_STILL\_AREA - 1. LOWEST\_SPEED\_VAL is the out\_max value and is calculated by subtracting the Pixy Camera’s image resolution MAX\_PIXEL\_AREA at 64000 from STAND\_STILL\_AREA.

The variable forwardSpeed holds the result of the mapping. The speed value is constrained between the range [-HIGHEST\_SPEED\_VAL/2, HIGHEST\_SPEED\_VAL] which is set to [-4324, 8648] in the program. Ultimately this permits the rover to have full forward throttle response, and half the full backward throttle response. Refer to Figure 2.37 to view the linear relationship between constrained forwardSpeed values and Pixel area. The very subtle nuances that make the line seem somewhat jagged is due to rounding down the mapped forward speed values. The forwardSpeed variable is of datatype integer and will truncate the decimal portion of floating type value. Figure 2.37 was constructed through Microsoft Excel to model this behavior.



**Figure 2.37 Constrained Forward Speed vs Pixel Area Graph**

The result of the linear response implementation is that forward movement speed is directly proportional to how far away the object is from the distance associated with STAND\_STILL\_AREA; this is the follow distance. Reverse movement speed is directly proportional to how much closer the object is from the following distance. After the throttle speed of rover’s motion is computed, the steering differential, or how fast to turn in one direction or the other, is computed and used to adjust the left and right channel motor speeds. Steering differential is directly proportional to the position of the panning servo. How far off center the pan servo is positioned is stored in the followError variable. The follow error will always be a value in the range [-500, 500]. The error will be less than 0 if the tracked object is located left of the center, and greater than 0 if the object is located to the right of the center. If the tracked object has a smaller pixel area than STAND\_STILL\_AREA, the rover will move forward and turn left or right according to the speed offset provided by the differential value. If on the other hand the size of the tracked object is greater than STAND\_STILL\_AREA, then the object is closer than desire; the forwardSpeed value will always be negative in this case. How the rover moves in this scenario is dependent on the value of followError. As shown in Figure 2.39, the Pan Servo has 120 degree range of motion. Mapping a followError offset of 100 from the range [0,1000] to [0, 120], will provide a degree tolerance from the center of 90 degrees, which can be used to determine whether the rover should move backwards or rotate in proportion to the Pan servo’s position. This tolerance can be computed as follows: degrees. If the Pan servo’ position is more than 12 degrees off center, the Rover will be rotated like tank in the direction of the object; otherwise the rover will move backwards until the tracked object has a pixel area of STAND\_STILL\_AREA.

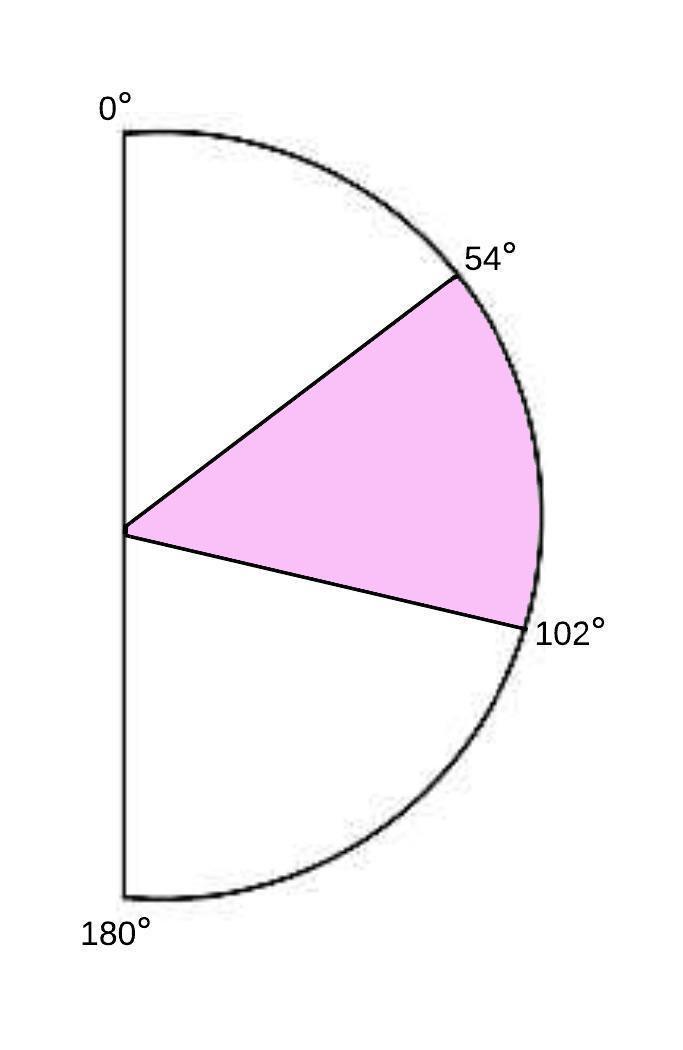
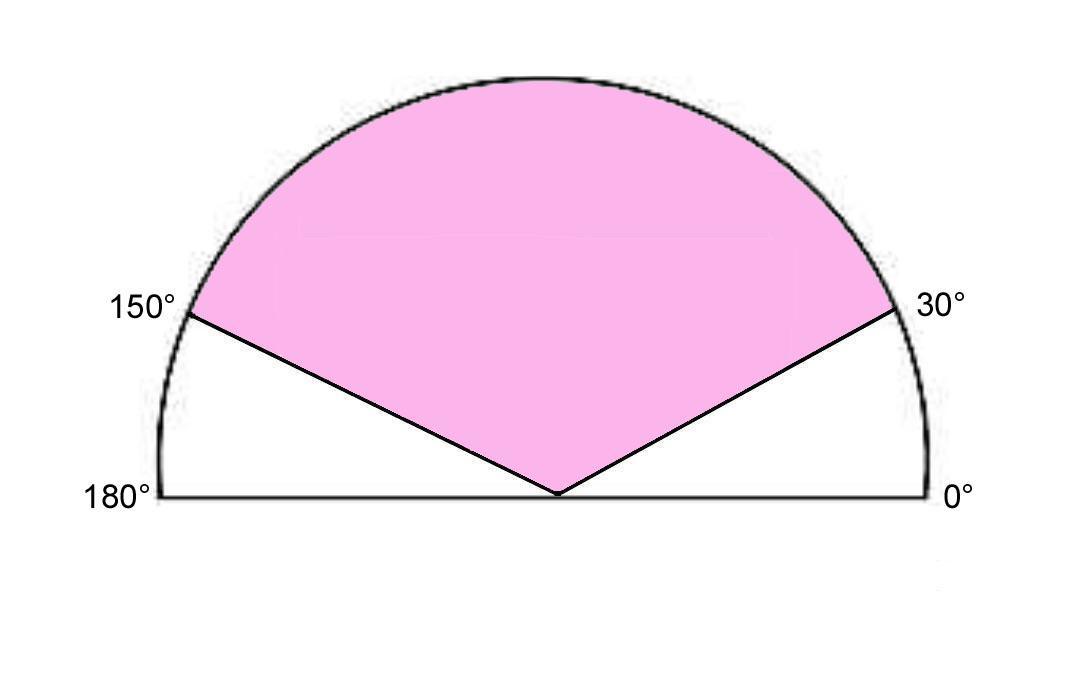
The final leftSpeed and rightSpeed values are constrained to the range [-HIGHEST\_SPEED\_VAL, HIGHEST\_SPEED\_VAL], defined as [ -8648, 8648] in the program. This is done in order to map the speed values in a symmetrical fashion to a pulse width value within the pulse width range of [MAX\_REV\_SPEED, MAX\_FWD\_SPEED] defined as [1700, 1300] that ends up controlling the DC motor motion.

#### 2.10.6 The ScanForBlocks Routine



**Figure 2.38 ScanForBlocks Function RoverFollow**

This routine pans the camera back and forth, and up and down at random until the Pixy camera module has detected at least 1 block. The Pan servo’s position is adjusted periodically every 20 milliseconds by an increment of SCAN\_INCREMENT, which is defined as 10 in this program. Increasing the SCAN\_INCREMENT value will increase the panning speed, whereas decreasing it will decrease the panning speed. Note that the panning motion the user observes is accomplished through repeated calls to this subroutine.

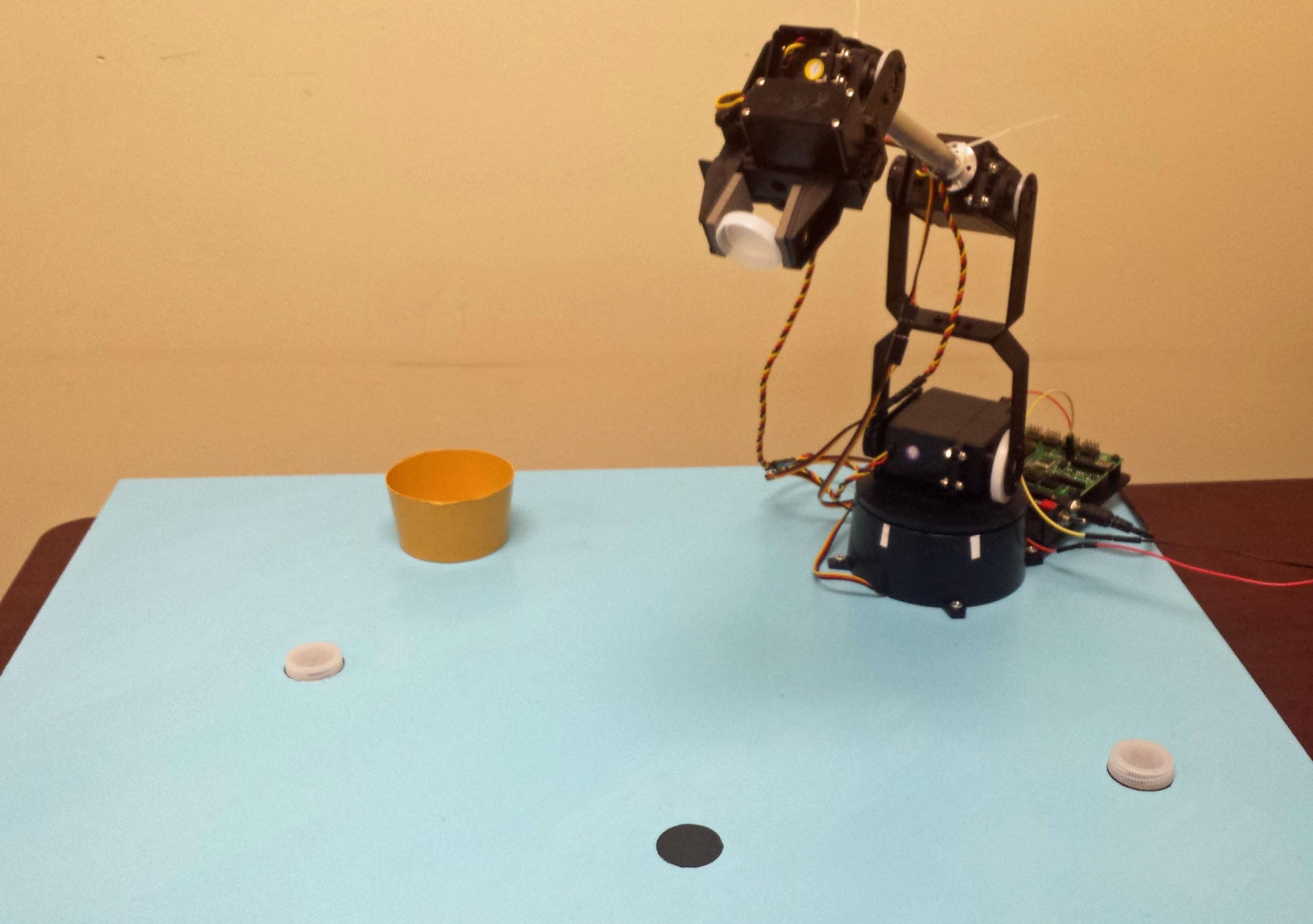


**Figure 2.39: Pan Servo range of motion Figure 2.40 Tilt servo Range of Motion**

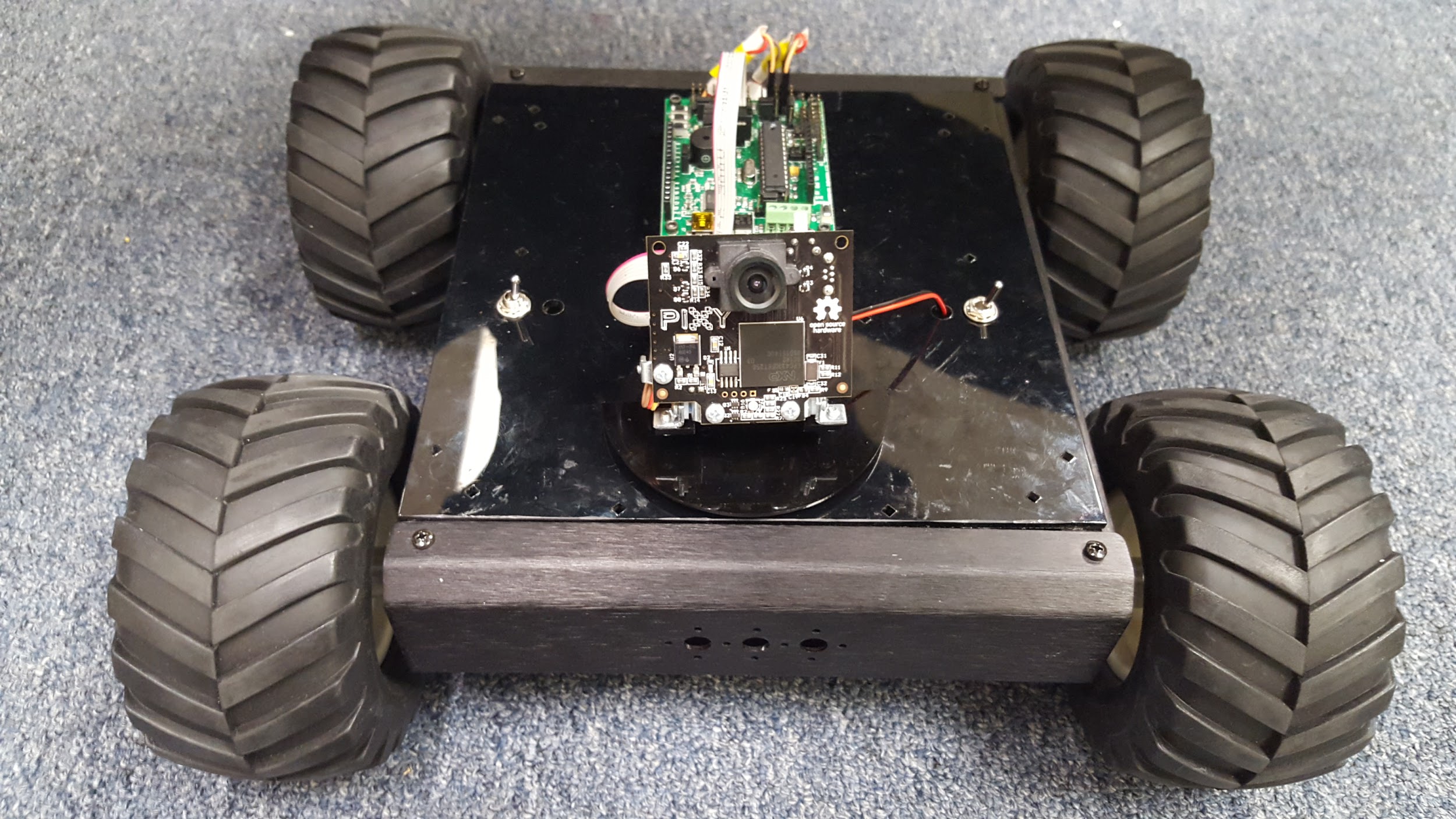
RCS\_MAX\_POS is 1000 and corresponds to about roughly 150 degrees. RCS\_MIN\_POS is 0 and corresponds to about roughly 30 degrees as shown in Figure 2.39. The camera will always be panned left first, before it is panned to the right. The camera is panned all the way to leftmost position or the rightmost position before switching direction. When the camera has reached its left or right panning limit, the tilt servo position is randomly adjusted between 200 and 600. The range of motion of the Pan and TiltServos is 60 degrees to the left and right from the neutral position of 90 degrees. The position values 200 and 600 directly correspond to degrees and degrees. This range of motion is displayed in Figure 2.40.

Before the tilt servo is set to its new position and the pan servo starts panning in the opposite direction, the rover is rotated for a random duration of 250 to 500 milliseconds. If the camera has reached its leftmost panning limit, the rover is rotated counterclockwise, otherwise the rover is rotated clockwise. The rover is rotated in a tank style manner, such that the left and right channel motors are moving in opposite directions at the same speed; this prevents the rover from straying from its current location.

### 2.11 The HELPER Project Version 1.0 Features:



**Figure 2.41 Arm Presentation Setup**



**Figure 2.42 The Tracking Rover**

# 3. Work Distribution & Project Schedule

### 3.1 Work Distribution

Throughout the creation of the HELPER Project Version 1.0, all three team members contributed in all areas. However, due to time limitations, a divide and conquer approach was taken. Dary Cabrera’s efforts were focused on image guided rover movement. Gabriela Rotver’s efforts were dedicated to robotic arm movement. Sheik Rahaman provided assistance to both members in a variety of physical tasks, and work with the camera. The written report and PowerPoint presentation for the HELPER project was created and written by all three team members.

### 3.2 Project Meeting Schedule

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| January 2016 | | | | | | |
| Su | M | T | W | Th | F | S |
|  |  |  |  |  | 1 | 2 |
| 3 | 4  Group Meet  9 am – 5 pm | 5 | 6  Group Meet  9 am – 5 pm | 7 | 8  Group Meet  9 am – 5 pm | 9 |
| 10 | 11  Group Meet  9 am – 5 pm | 12 | 13  Group Meet  9 am – 5 pm | 14 | 15  Group Meet  9 am – 5 pm | 16 |
| 17 | 18  Group Meet  9 am – 5 pm | 19 | 20  Group Meet  9 am – 5 pm | 21 | 22  Group Meet  9 am – 5 pm | 23 |
| 24 | 25  Group Meet  8 am – 9 pm | 26  Group Meet  8 am – 9 pm | 27  Group Meet  8 am – 9 pm | 28  Group Meet  8 am – 9 pm | **29**  **FINAL PRESENTATION** | 30 |
| 31 |  |  |  |  |  |  |

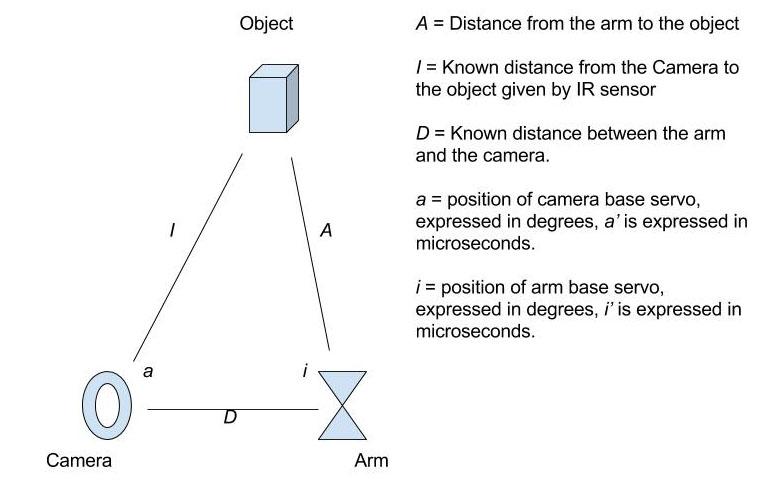
### 3.3 Task Schedule

|  |  |
| --- | --- |
| **Table 3.1 Project Task Schedule** | |
| **Date** | **Tasks** |
| 01/04/16 | * Finish the arm sequence using *FlowBotics Studio* software. * Interface SSC-32U controller with Arduino. * Read documentation to understand how the commands are formatted and how they affect each servo movement in the arm. * Experiment with the pulse width modulation (PWM) techniques available for Arduino programming. |
| 01/06/16 | * Finish and cleanup transposed test program for the rover’s BotBoarduino microcontroller to have it use the push buttons to manually manipulate and control the speed and direction of the rover’s movement. * Focus on writing the Interim Status report content, updating the current achievements and challenges. |
| 01/08/16 | * Work with the PixyMon software to have the camera recognize an object. * Finish writing and proofread the interim status report. * Turn in interim status report * Study the law of cosines and how it can be used to build the XYZ code for the arm. |
| 01/11/16 | * Work with PixyMon to have the Pixy camera track an object. Have the pan/tilt servos center on the object. * Study the law of cosines and how it can be used to build the XYZ code for the arm. * Start focusing on an XYZ code for the arm. (Kinematics equations specifically) for the arm. |
| 01/13/16 | * Program an algorithm to have the Pixy camera scan for an object if it does not immediately see one within a set duration of time. * Finish and polish Arm function placing the arm in a given XYZ position. |
| 01/15/16 | * Purchase materials to fasten the arm to a wooden board. * Paint the board to make it professional for the presentation. * Start working on the pickUp function requiring shoulder, elbow, and wrist position modification. Manually find the appropriate change in position values to be used for two different X coordinates. * Start working on an algorithm that uses the position of an object detected by the Pixy camera and the panning servo’s position to have the rover follow an object. |
| 01/18/16 | * Keep working on algorithm that will have the rover follow an object being tracked by the Pixy camera unit. * Experiment with and construct various Excel Sheets and Graphs that map the pixel area of an object to a speed value. * Drill the holes into the wood and insert the arm to stabilize. * Finish working on the first part of the pickUp function requiring shoulder, elbow, and wrist position modification. Manually find the appropriate change in position values to be used for another X coordinate. |
| 01/20/16 | * Finish and program the algorithm that will have the rover follow an object being tracked by the Pixy camera unit. * Construct a final Excel Sheet and Graph that linearly maps the pixel area of an object to a speed value. * Completely finish the pickUp function by changing the gripper, elbow, and base values to have the arm pick up the object from the floor on the rotational axis at the current position. Finally take the object and place at a fixed spot to place item in a small trash bin. |
| 01/22/16 | * Clean up the final versions of the codes and comment for the final presentation. * Make sure the hardware and presentation code works successfully. Run several trials. |
| 01/25/16 | * Start working on the final report. Focus on the achievements and challenges during the creation of the HELPER project Version 1.0. |
| 01/26/16 | * Keep working on final report. Make sure to sufficiently explain the code and include diagrams and graphs. |
| 01/27/16 | * Proofread and cleanup final written report. * Work on PowerPoint Presentation. |
| 01/28/16 | * Proofread and cleanup PowerPoint presentation. * Make sure all demonstrations are successfully working. * Practice the presentation. |
| 01/29/16 | * Final presentation 12:00 pm * Turn in Final Report |

# 4. Future Steps

### 4.1 Interfacing the Arm and PixyCam

In the future, prior to interfacing the arm and the rover together, the camera needs to be utilized to locate specific objects for the arm to pick up. A key drawback in this is that the PixyCam, with its ability to center its position with an object, has is incapable of detecting depth. Incorporating an infrared sensor will provide depth detection. With the implementation of an infrared sensor directly under the lens of the camera, the arm relinquishes the need for the camera to provide an approximate distance. The only requirement for the camera is that it must center objects it tracks so the IR sensor points directly at the object. This produces the Triangle shown in Figure 4.1.



**Figure 4.1: Position of Camera, Arm, and Object**

When *I, D* and *a* are known, *A* and *i* can be solved for using the law of cosines. In the XYZ program, *A* would be the x coordinate, and *i* can be mapped to microseconds to directly control the arm’s rotational servo. This would turn the coordinate system into a two parameter system; x designates distance to the object and y designates height.

### 4.2 Integrating IR Sensors with Rover

Currently, the rover has been outfitted to track and follow an object, but in the future, with the integration of IR sensors, the rover will have autonomous movement about a room. IR sensor readings can be used to implement an obstacle avoidance algorithm to safely move about without human interaction. This is a key safety feature for the rover as well as anyone around. If the rover were to run into an obstacle at full velocity, it could be severely damaged, or hurt someone if it ran into him/her.

# 5. Process

### 5.1 Team Experience

Working as a team has allowed each individual in the group to be part of something bigger by collaborating on ideas and tasks leading to a successful completion of the HELPER project Version 1.0. Of course, it is inevitable that schedules conflict in a group of three people; which complicates arranging group meeting times. Nevertheless, the irreplaceable benefit of working as a team involves the fact that whenever one member is struggling with a task, the rest of the team is ready to get involved and help out as much as possible because everyone is working towards a main goal.

### 5.2 Process Tools and Activities

Process tools and activities are considered methods used along the creation of the project that were not necessarily learned in a classroom setting. The following are the process tools and activities used during the creation of the HELPER Project Version 1.0.

#### 5.2.1 Need Finding

To develop a potential idea to undertake, the team had to go through the need finding process to gain meaningful insight and inspire an impactful and valuable project design. Observation of the environment and people in it led to the creation of a list of perceived problems. For each problem, worthwhile product ideas emerged. However, only few were conceivable to carry out in a semester long project. During the need finding process, the idea for a robot that autonomously picks up litter developed.

#### 5.2.2 Developing and Scheduling Tasks

Seeing that a robot that autonomously moves and picks up litter is a large-scale project, it was decided to take the project in iterations where the current team will focus on separate rover/arm autonomous movements and future Nova students taking the CENG 4900 Capstone Design course will take off where the last group left off and focus on making the rover and arm work together.

To successfully complete the main goal of having a rover and arm autonomously working separately to form the HELPER Project Version 1.0, smaller tasks needed to be developed and properly scheduled. The process activity of developing realistic tasks that aid in reaching the main goals was undoubtedly a learning experience for the team.

In the beginning days of the HELPER Project Version 1.0, a mistake that was constantly made was underestimating the amount of time a task would take. By the final month of the project, a tentative schedule was outlined in Table 3.1 with a contingent task schedule requiring a realistic amount of time to be assigned depending on the activity.

#### 5.2.3 Collaboration Tools Used

As a team, usage of a collaboration tool is essential for everyone to work together on code or on the reports. For this project, Google Drive and Google Docs were the tools used for each member to have access to all of the files being used and for everyone to be able to concurrently contribute in the report write up. For future projects, another process technique that can be used to increase efficiency will be parallel programing.

One of the most beneficial tools used were forums. The collaboration and quick assistance from experts in the field of robotics helped obtain answers to several questions that arose during the project.

#### 5.2.4 Reading, Comprehending, and Filtering through Technical Documentation

Each component used to carry out the HELPER Project has various pages of extremely detailed technical documentation. In order to adequately comprehend and find specific information without getting overwhelmed by the excess amount of pages, speed reading techniques such as word clustering and skimming were used. Through skimming, important words from a block of text are extracted and the rest are ignored. Clustering is the ability to cluster several words together and jointly process them. In addition to these techniques, the table of contents of said documents as well as the finding function available for most documents/web pages helped in finding the specific bits of much needed information.

### 5.3 Engineering Tools and Activities

Engineering tools and activities are considered methods and instruments used along the creation of the project that were learned in a classroom setting. The following are the engineering tools and activities used during the creation of the HELPER Project Version 1.0.

#### 5.3.1 Programming Embedded Systems

Several engineering courses at Nova provided indispensable lessons that greatly assisted in the completion of the project. Important labs involving activities such as designing and implementing microcontroller systems with appropriate interfacing to external devices provided the knowledge required to interface and program the Arduino Uno with the SSC-32U servo controller for the arm as well as the interfacing of BotBoarduino microcontroller with the Sabertooth 2x12 R/C motor controller. Knowledge about the serial communication process also aided in properly interfacing and sending data to and from the different devices/controllers.

#### 5.3.2 Debugging Problems/Issues

Granted that not everything is going to work on the first try when a program is executed, being aware of useful debugging techniques helped in realizing a prompt discovery of what was causing a problem. Engineering techniques such as how to properly use and read a multimeter aided in finding hardware issues. If the hardware is working properly, knowing how to use an oscilloscope helps in determining whether the signals/values generated by a program are correct. During the project, the oscilloscope was regularly used to validate that a program was generating the appropriate servo signals corresponding to various speed values. These debugging techniques greatly assisted in determining what was and what was not the root of an issue.

Beyond the physical tools that can be used for the purposes of debugging, there are several coding techniques that also assist in finding the issues within the program code. Arduino’s Serial class provides API functionality for communication between the Arduino board and a computer. By using several serial statements to print out the current state of the code, a programmer can see what part of the code is executing and what is not. This technique was used as a guide to find where the problems started in a program file with an overwhelming amount code.

### 5.4 Interesting Challenges

As the team progressed through the project, there were several obstacles that needed deep thought to be overcome. Some challenges are still face today.

#### 5.4.1 Lack of BotBoarduino Programming Support and Library Code Explanatory Comments

The Lynxmotion BotBoarduino is a new microcontroller for the rover that is Arduino Duemilanove compatible, but does not have repositories or test code that exercise the functionality of the microcontroller or movement of the rover. The test code and sample code that does exist is written in a different language for the older Lynxmotion Bot Board II microcontroller. This code has to be fully understood in order to port it to the BotBoarduino.

The BotBoarduino is an Arduino Duemilanove compatible microcontroller and is designed to be programmed through the open-source Arduino IDE. Through its library base, the Arduino platform is an interface with Atmel ATmega328P Microcontroller functionality. The centerpiece of the functionality used to control the Rover’s movement is Servo Pulse Wave Modulation. The Arduino library provides APIs to generate PWM signals for Servos, but the configuration of various parameters that control various characteristics of the signals is hidden from the programmer inside Arduino’s core library and Servo library. The source code of these libraries is not sufficiently documented for programmers that wish to understand how it works rather than how to use it. The ATmega328P-PU microcontroller datasheet had to be studied to understand how Arduino library functionality interfaces with it.

#### 5.4.2 Constructing Signals to control DC Motor Direction and Speed Control

As discussed in detail in section 2.9, lack of detailed documentation for signal inputs to the Sabertooth motor controller led the team astray to assume that the DC motors motion can be controlled by adjusting the duty cycle of a pulse train. This was not the case. Much time was invested to making this approach work, but to no avail.The team found that the Remote Control Signals the Motor controller expects are actually Servo signals. In order for the motor controller to properly control the rotation of the DC motors, the microcontroller must generate and send servo signals.

#### 5.4.3 Limited Hardware

When working on an extensive project, extra hardware is frequently required to debug an issue when a component is not behaving as expected. Due to budget restraints, an extra motor controller was not available until late into the project. This was a challenge because if an issue persists even when the controller is replaced, it would quickly reassure the programmer to look for the problem within the code because it’s not the hardware. This makes for more efficient debugging because it is a quick way to pinpoint the cause of a problem.

Additionally, the fact that only one Pixy camera was available meant that out of the rover or the arm, only one can be chosen to integrate the camera. This was an issue because both the arm and rover were being programmed and improved in parallel. When the time came for the arm to start using the camera to integrate it with the code, the rover needed it to accomplish guided rover movement.

#### 5.4.4 Arm Calibration Problem

Even though the Lynxmotion ALD5 arm is powerful and sturdy, it has a tendency to behave erratically when it is used over a prolonged period of time. This was a challenge because the arm needed to be constantly “on” in order to see how the frequent code changes affected its position.

When testing the arm to pick up objects at specified positions, the arm oftentimes de-calibrated. Although it would only slightly de-calibrate, it would be enough for the arm to miss the item when attempting to pick it up. After some rest, the arm would act normally; however, this presents a challenge and risk for when the arm is presented to the faculty.

# 6. Reflection

### 6.1 Learning Outcomes

By the final week of the project, there are several lessons acquired from the capstone design experience. By having to come up with a project idea, need finding practices were learned and carried out with the aim of determining what could be a useful product that the team can create within the given time lapse.

To take on such a large scale project would have been unrealistic, which highlighted the importance and effectiveness of agile methodology. Taking the project in iterations, the team learned the significance of effective planning by having realistic goals, designing feasible tasks that add value to the project, and planning for the tasks to be properly scheduled to get them done by the deadline.

Problems and difficulties that arose during the project creation process portrayed the value of considering all of the different venues and approaches that could have been taken. There is great emphasis on doing enough research on the different platforms, components, and software so that a minimum amount of complications are encountered (i.e. such as lacking important documentation explaining the main components used).

Efficient research skills as well as proficiency techniques on reading and understanding technical documentation were used to obtain the technical knowledge required to execute an effective design. Additionally, the abundant value and assistance obtained from public forums supplying expert assistance was fully recognized once the team started looking at those sources for help. The technical knowledge gained from the literature research was applied to various methods, tools, and techniques used throughout the project.

Functioning effectively in a multi-disciplinary team provided valuable experience to each individual. The teamwork required to carry out a successful project cultivated the importance of communication and group collaboration. By participating in the project report, all members practiced the art of writing technical documentation, adhering to a specified format using proper style, grammar, and spelling.

### 6.2 Things to do differently

In every project, there are tasks that could have been done differently in order to increase productivity and efficiency. For the capstone design project, there are several activities that, if the team could go back, would be carried out in a different manner.

One of the learning outcomes for this project was the significance of planning. Lack of proper planning in the beginning of the semester was the cause of haste and commotion to get the project finished in time. The absence of a proper budget and an unclear goals were both originators of a feeling of disorganization and unproductivity. It was not until after a time extension was given that adequate and productive planning took place and proper requirements with realistic tasks were defined. Because of this, one of the main things that would be done differently is taking enough time to plan ahead from the beginning to define adequate objectives and to find a proper budget.

Along with planning ahead, spending more time on researching the components that are going to be ordered would be another thing to do different. Completing extensive research on the available hardware will help make sure the type of controllers (and any other component) chosen have enough documentation available for proper understanding of the technical details to carry out the project.

In hindsight, paired programming could have halved the time spent overcoming the learning curve and the debugging process. Additionally, by having one person write the code while the other is observing, the design quality of the program’s logic potentially increases.

The task of writing a final capstone report for the project was not extremely intimidating until the team actually started writing it. To avoid becoming overwhelmed with the amount of content that has to be written to complete the report, logging daily activities and documenting achievements as the project progresses would be beneficial to incorporate from the beginning of the project.

# 7. References

Arduino. (2016). Servo Library. Retrieved January 28, 2016, from <https://www.arduino.cc/en/Reference/Servo>

Atmel. (2015, November). ATMEL 8-BIT MICROCONTROLLER WITH 4/8/16/32KBYTES IN-SYSTEM PROGRAMMABLE FLASH DATASHEET. Retrieved January 28, 2016, from <http://www.atmel.com/images/atmel-8271-8-bit-avr-microcontroller-atmega48a-48pa-88a-88pa-168a-168pa-328-328p_datasheet_complete.pdf>

Bobblick. (2002). Servo Pulse to PWM. Retrieved January 28, 2016, from <http://bobblick.com/techref/projects/sv2pwm/sv2pwm.html>

CMUcam. (n.d.). The Latest Pixy Stuff. Retrieved January 28, 2016, from <http://cmucam.org/projects/cmucam5/wiki/Latest_release>

Dimension Engineering. (n.d.a). Sabertooth dual 12A motor driver for R/C. Retrieved January 28, 2016, from <https://www.dimensionengineering.com/products/sabertooth2x12rc>

Dimension Engineering. (n.d.b). Sabertooth R/C DIP Switch Settings. Retrieved January 28, 2016, from <https://www.dimensionengineering.com/tours/sabertooth_rc/SabertoothRCDips_long.htm>

Jan. (11, February 9). Servo control interface in detail. [Web log comment] Retrieved January 28, 2016, from <https://www.pololu.com/blog/17/servo-control-interface-in-detail>

Lynxmotion. (2015, August). Lynxmotion SSC-32U USB Servo Controller Board. Retrieved January 28, 2016, from <http://www.lynxmotion.com/images/data/lynxmotion_ssc-32u_usb_user_guide.pdf#page=24>

Parent-Charette, S. (2015, June 4). How to Choose a Lynxmotion Robotic Arm. Retrieved January 28, 2016, from <http://www.robotshop.com/blog/en/lynxmotion-robotic-arms-15845>

Richling, M. (2009). R/C Control Signals. Retrieved January 28, 2016, from <http://www.mitchr.me/SS/batteriesRequired/RCcontrolTheory/>

Simmons, D. (2012, March 27). BotBoarduino Manual. Retrieved January 28, 2016, from <http://www.lynxmotion.com/images/html/build185.htm>

RobotShop. (2015, October 4). Simple moves code for Lynxmotion Botboarduino robot arm. Retrieved January 28, 2016, from <http://www.robotshop.com/forum/post-p119738>

Webbot. (2008, November 30). Driving a servo via PWM | Member Robot Tutorials. Retrieved January 28, 2016, from <http://www.societyofrobots.com/member_tutorials/node/231>

Wolfram MathWorld. (n.d.). Law of Cosines. Retrieved January 28, 2016, from <http://mathworld.wolfram.com/LawofCosines.html>