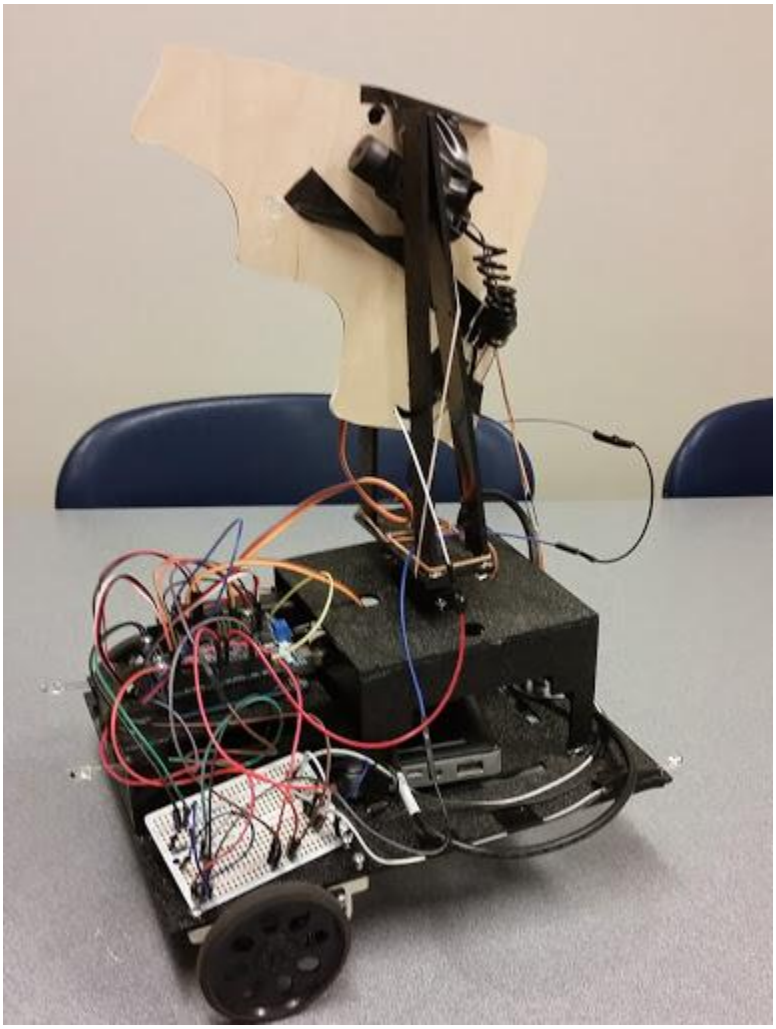
Trevor Craig

MECH 457

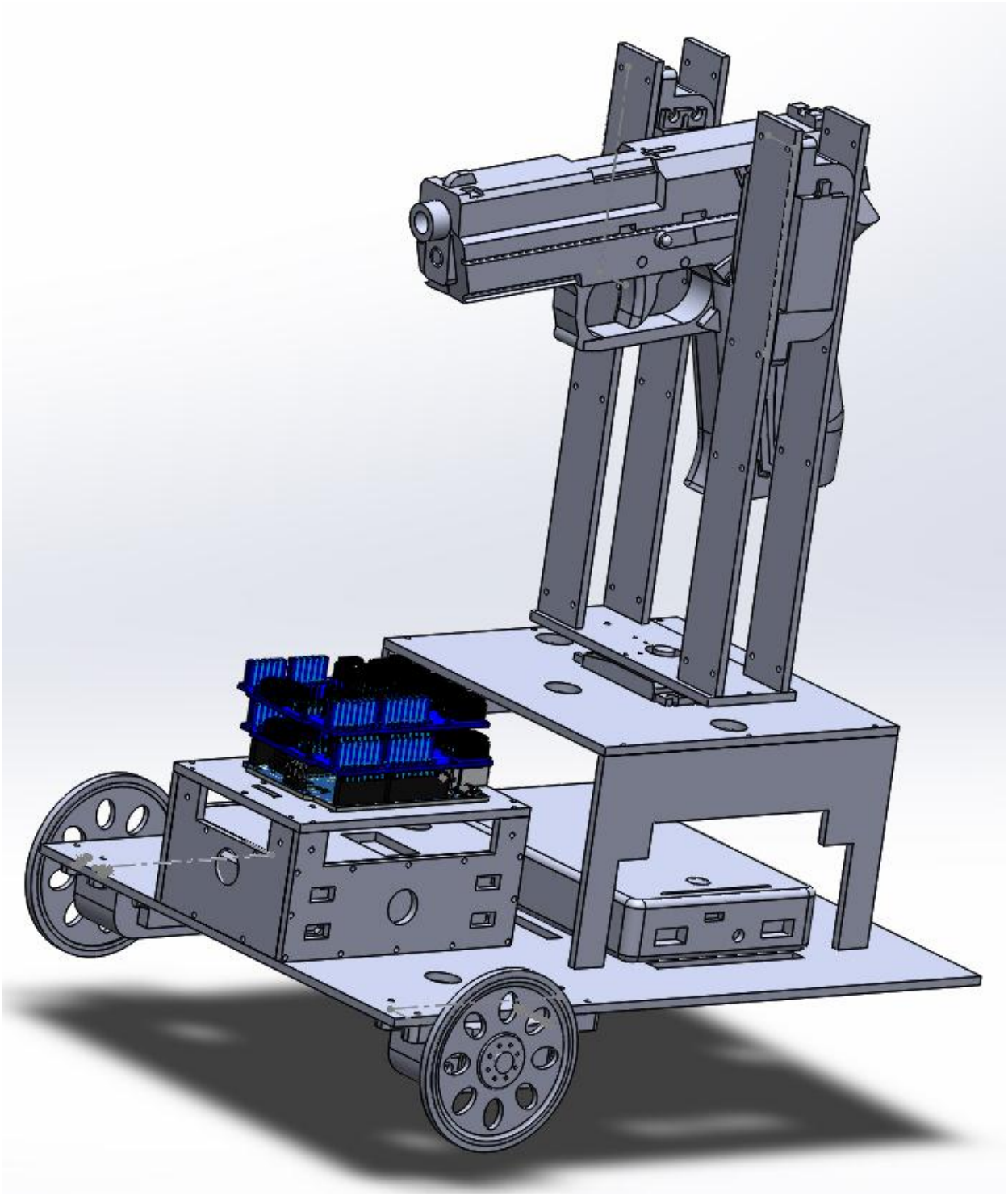
Mechatronics Project



Sentry Car

INTRODUCTION:

What is the Sentry Car? The Sentry Car is a small Arduino based car. The sentry car has the room and capability of allowing multiple sensors on board to accomplish a variety of tasks. The original idea of the sentry car was to be able to hold an airsoft gun and be able to aim it while driving the car. Although the airsoft gun was removed for safety, the car still accomplishes its intended purpose. Throughout the entire design process the ideas of usability, simple and effective controls, and reusability were always kept in mind. Everything that was designed is easily able to be redesigned to suit other purposes. All of the components listed in this document were created in SolidWorks and placed in a “loose” assembly that allows components to move with constraints, while creating a new part from the assembly. This part can then be saved as an STL for 3D printing. By using modular design concepts, it was easy to move components around and create a part that satisfied all the tolerances. The ending goal was to be able to have a robot chassis that I could provide online for anyone who wanted to continue the work or had a similar task. The main overall concept was to have multiple types of communication work together. In this project I2C, Serial over USB, Serial over Xbee, and sensors inputs were all used. This project is also split into 3 separate programming components: The Arduino Master Controller, Arduino Master Receiver, and the Raspberry PI Camera Control. All of this leads to a relatively expensive project, but one that was quite robust and fun to play with.



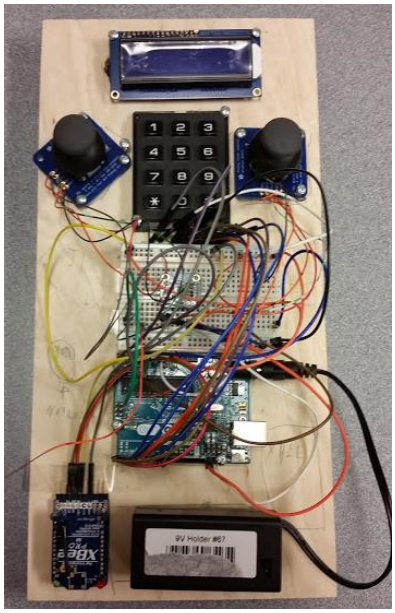
BILL OF MATERIALS

The first thing was to acquire all the parts required to build the Sentry Car. Below is a simple break down of all the components used within the car. Some of the parts are optional. The optional parts are the range of the Xbees and the Series of them, servos, LEDS, wiring, power sources, and servo shield. Almost all of these parts come from Adafruit and therefore should be easy for someone to be able to replicate the project. Parts can be found for cheaper elsewhere but you can't beat the simplicity of a single order.

Item	Cost
Arduino Uno R3	24.95
Arduino Uno R3	24.95
XBee Pro 60mW Wire Antenna - Series 1 (802.15.4)	37.95
XBee Pro 60mW Wire Antenna - Series 1 (802.15.4)	37.95
Adafruit 16-Channel 12-bit PWM/Servo Shield - I2C interface	17.50
Analog 2-axis Thumb Joystick with Select Button + Breakout Board	5.95
Analog 2-axis Thumb Joystick with Select Button + Breakout Board	5.95
3x4 Phone-style Matrix Keypad	7.50
i2c / SPI character LCD backpack	10.00
GA1A12S202 Log-scale Analog Light Sensor	3.95
Standard LCD 16x2 + extras - white on blue	9.95
Raspberry Pi 2	35.00
Adafruit Triple-Axis Accelerometer - ±2/4/8g @ 14-bit - MMA8451	7.95
4 x AA Battery Holder with On/Off Switch	2.95
9V battery holder with switch & 5.5mm/2.1mm plug	3.95
4GB SD card for Raspberry Pi preinstalled with Raspbian Wheezy - 2015-01-31	9.95
Raspberry Pi Camera Board	29.95
RAVPower 3rd Gen Deluxe USB Charging Hub	35.00
Continuous Rotation Servo - Futaba S148	14.00
Continuous Rotation Servo - Futaba S148	14.00
Standard servo - TowerPro SG-5010 - 5010	12.00
Standard servo - TowerPro SG-5010 - 5010	12.00
LEDS	1.00
Wiring	3.00
Materials	30.00
Total	397.35

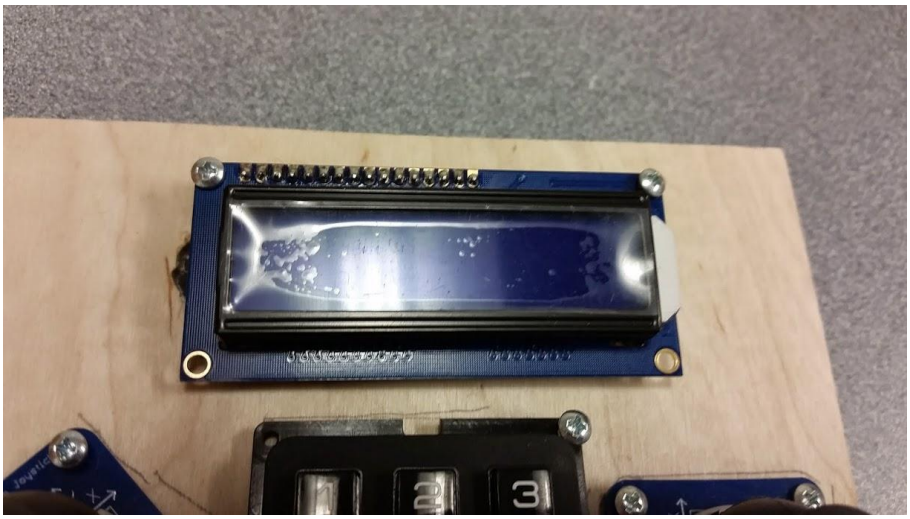
Item	Cost	Link
Arduino Uno R3	24.95	https://www.adafruit.com/products/50
Arduino Uno R3	24.95	https://www.adafruit.com/products/50
XBee Pro 60mW Wire Antenna - Series 1 (802.15.4)	37.95	https://www.sparkfun.com/products/8742
XBee Pro 60mW Wire Antenna - Series 1 (802.15.4)	37.95	https://www.sparkfun.com/products/8742
Adafruit 16-Channel 12-bit PWM/Servo Shield - I2C interface	17.50	https://www.adafruit.com/products/1411
Analog 2-axis Thumb Joystick with Select Button + Breakout Board	5.95	https://www.adafruit.com/products/512
Analog 2-axis Thumb Joystick with Select Button + Breakout Board	5.95	https://www.adafruit.com/products/512
3x4 Phone-style Matrix Keypad	7.50	https://www.adafruit.com/products/1824
i2c / SPI character LCD backpack	10.00	http://www.adafruit.com/products/292
GA1A12S202 Log-scale Analog Light Sensor	3.95	http://www.adafruit.com/products/1384
Standard LCD 16x2 + extras - white on blue	9.95	http://www.adafruit.com/products/181
Raspberry Pi 2	35.00	https://www.raspberrypi.org/raspberry-pi-2-on-sale/#raspberry-pi-2-on-sale
Adafruit Triple-Axis Accelerometer - ±2/4/8g @ 14-bit - MMA8451	7.95	http://www.adafruit.com/products/2019
4 x AA Battery Holder with On/Off Switch	2.95	http://www.adafruit.com/products/830
9V battery holder with switch & 5.5mm/2.1mm plug	3.95	http://www.adafruit.com/products/67
4GB SD card for Raspberry Pi preinstalled with Raspbian Wheezy - 2015-01-31	9.95	https://www.adafruit.com/products/1121
Raspberry Pi Camera Board	29.95	https://www.adafruit.com/products/1367
RAVPower 3rd Gen Deluxe USB Charging Hub	35.00	http://www.amazon.com/gp/product/B00MPIGPUY/ref=oh_aui_detailpage_o03_s00?ie=UTF8&psc=1
Continuous Rotation Servo - Futaba S148	14.00	https://www.adafruit.com/products/154
Continuous Rotation Servo - Futaba S148	14.00	https://www.adafruit.com/products/154
Standard servo - TowerPro SG-5010 - 5010	12.00	http://www.adafruit.com/products/155
Standard servo - TowerPro SG-5010 - 5010	12.00	http://www.adafruit.com/products/155
LEDS	1.00	http://www.adafruit.com/products/297
Wiring	3.00	http://www.adafruit.com/products/153
Materials	30.00	Can be bought/built anywhere

CONTROLLER



The controller is composed of an Arduino, XBEE, LCD screen, I2C backpack for the screen, 2 double potentiometer Joysticks, 9V battery holder, and keypad. The joystick is basically 2 10kΩ potentiometers. They take voltage in and are attached to ground and allow for a sensor out pin attached to one of the analog in pins for the Arduino. Each joystick uses 2 analog in pins. Although settings can be changed, my configuration was to have 1 joystick set to control the movement of the wheels and the other joystick to control the sentry pointing position. The joystick for wheel control was placed diagonally to allow both wheels to move forward when the user presses up, without having to code a center position. This once again followed the concept of keeping it simple minded controls, as anyone who has played a modern day video game should be able to drive the car. Also by placing the joystick in this configuration the user does not have to program into the program the balance or tilt of the steering. Although the joystick in not output an exact value it does provide more than enough accuracy to control the car and sentry.

On the controller side again, the LCD screen was set up in a way to allow the user to see important information about which mode they were currently in or values that were of interest to them. This was done using an updated version of the LiquidCrystal Library that allowed communication over I2C. A good reason to make this over I2C is to allow more pin availability for things like the keypad and other devices, otherwise 7 or more pins will be taken for the screen alone.

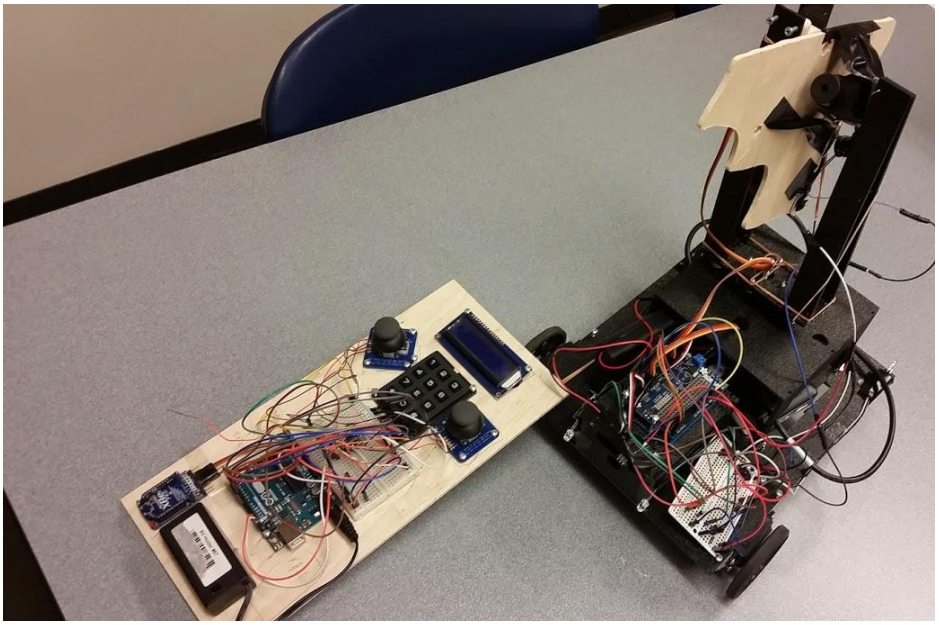


The controller also has a keypad that allows the user to navigate the menus and toggle on and off options, such as LEDS, motion control, camera control, and various other settings. Simple buttons could have been used but a keypad allows a more natural feel to the user and has more options. Wiring the keypad was a little bit of a pain as the internal wiring is a combination of resistors. In short a multimeter was needed to determine the pattern of the rows/columns in order for it to function. It is highly recommend using the keypad from adafruit instead of attempting to map your own keypad to work, although it does provide good insight into the internals of the keypad.

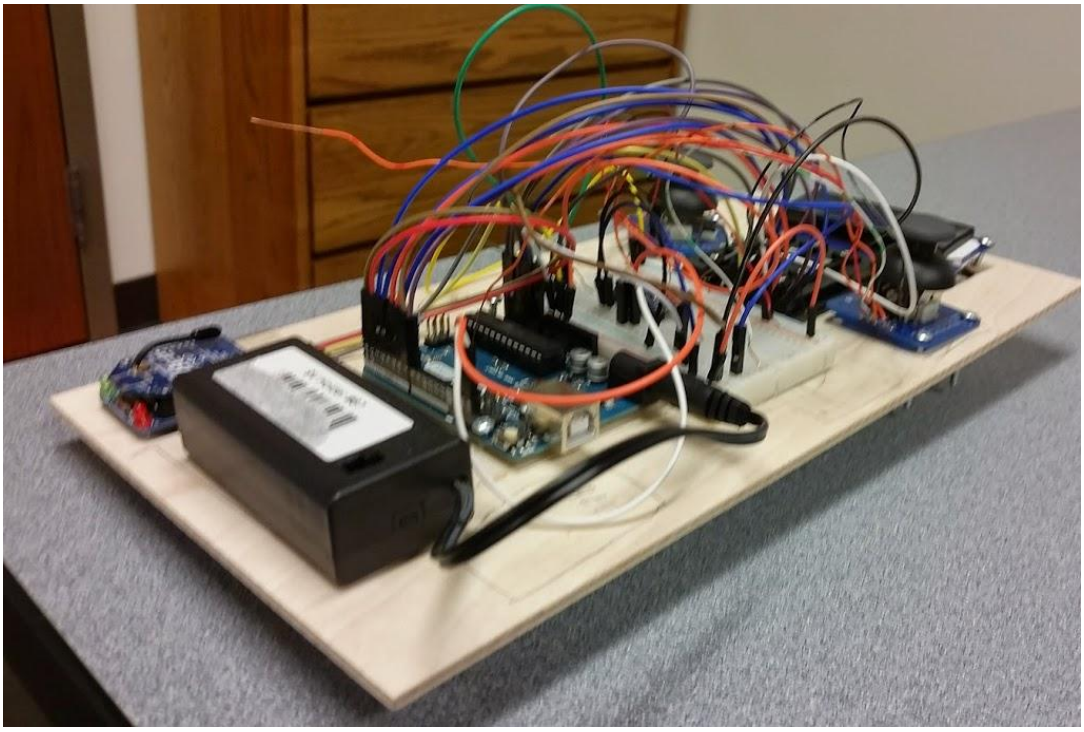
Also on the controller there is an accelerometer. The accelerometer is a really nice extra feature that allows the user to have another hand free while driving by using tilt controls for the sentry. The accelerometer sensor communicates over I2C with the Arduino controller. The values are then outputted in x, y, and z terms are remapped to turn the sentry gun to required motion. This was accomplished by placing the controller on its side and reading the values in the x,y, and z and the placing on the other side, as well as on its top and bottom. By doing this and recording them all, an accurate estimate at the required level of tilt was determined for the steering. This is technically an optional feature, but it would be really cool to implement the controller into a gun design and have the sentry gun react in parallel to the user's movement. For full documentation on soldering required and functions check this link or any associated documents with this file https://learn.adafruit.com/downloads/pdf/adafruit-mma8451-accelerometer-breakout.pdf. Care must be taken to match all the baud rates for serial and all sensors that communicate with sensors.

The baud rate is something that needs to be set experimentally to account for the distance and data transfer. The biggest and most important reason for this comes from the XBees. The XBees that are utilized in this report have shield that allows them to be hooked up to the computer and also the Arduino 5V line. These adapters were not included in the report as they may no longer exist on the market. To set or change the rate of the XBees an external program is necessary to communicate with them. The easiest and most straightforward way is through using a program called XCTU. While writing this report and looking deeper into the shield Adafruit recently brought this adapter back with better documentation, the previous details had been left quite vague. This would be placed in an appendix but due to the size of the documentation the link will be listed here and the documentation will also be kept with the files associated with this project. Here is the link to the PDF https://learn.adafruit.com/downloads/pdf/xbee-radios.pdf.

The rate of which this project used the XBees was at 19200, this was determined due to the speed of the project along with the highest rate before receiving too much interference. The pro version was chosen for their better range and they do perform exactly as one would expect. This project easily performed till it was unable to be seen accurately, around 100 feet or more. More testing could be done to see the full range. An important note, for accurate data transfer a delay should be utilized on the sending side not to overflow the receiving side with information causing data corruption. The larger the delay the more accurate the data should send but the less responsive the system is. A good standard is to make the system at about the same speed as the human reaction time, around 200 milliseconds; this creates an illusion that it is moving in real time.



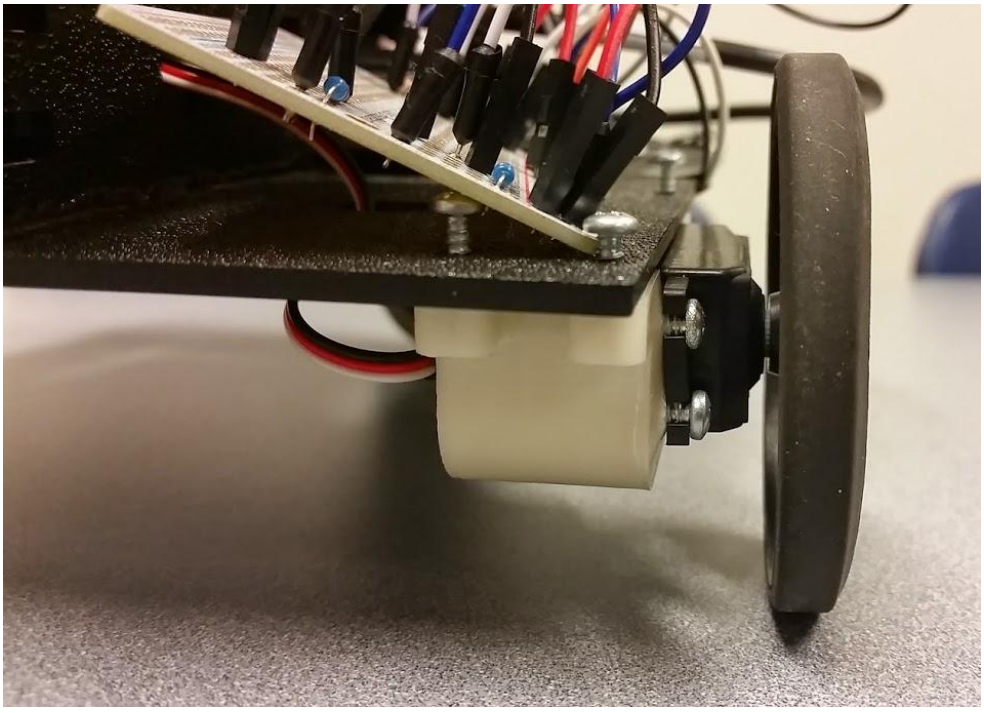
The sentry car is a dynamic system therefore things such as delays for button presses need to be avoided. The way this project handled it was using conditional variable changes with artificial counter delays. The conditional variable change is that under a certain condition such as a button press the system takes the variable associated with that action and places it into a variable as if the system is continually pressing the button. This allowed lights to be turned on or off and the system to be in different set modes. The artificial delay is for analog sensors, excluding the joysticks. This delay was put into place as a precaution against the Arduino not being able to keep up with all the variable changes, especially on the receiving end. This artificial delay only reads the data when a variable reaches a certain value, for example 10, this variable counts up every time through the loop and then resets when it reaches the set value. This technique was especially useful when using the light sensor. This was an easy, and non- resource intensive way to keep the program running without getting too slow. Using these techniques the program ran without problem.



The controller was chosen to show the wires involved and be a rugged wood frame. This allowed very easy access to all of the components for testing and allowed the project to show the work put into it. The frame is made from wood and number 4 screws. The wood was carved out using a Dremel as a quick and easy way for testing the controller in a more permeant form. There are holes throughout the controller frame that allow the wires to sneak behind the board, not to overwhelm the user. Due to time restraints and the fact that by keeping it open allowed it to be presented to kids easier for explanation, than seeing a finished controller that they may already use every day.

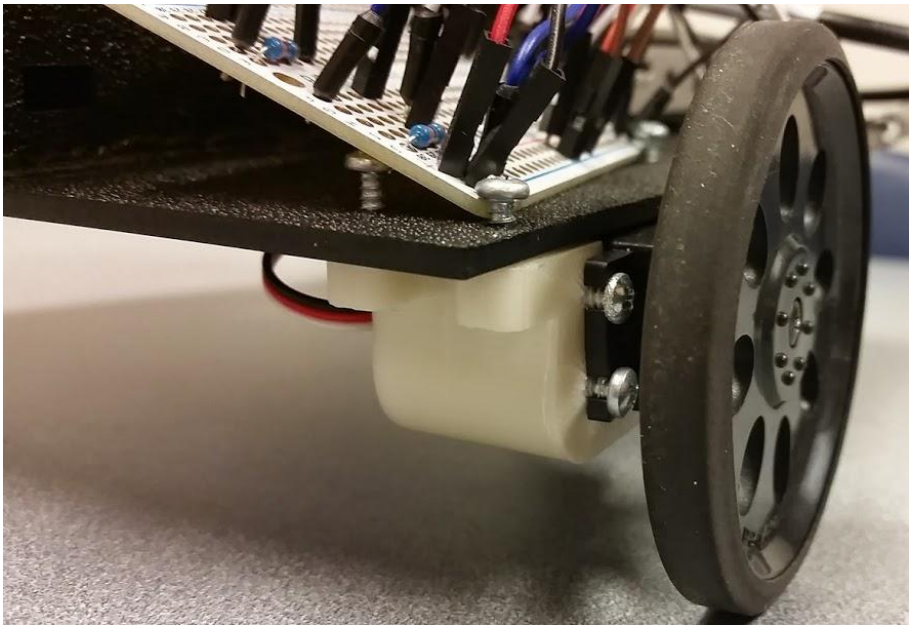
SENTRY CAR PARTS

The sentry car includes a lot of components, but no parts are as important as the servos. The servos control not only the servo but the driving as well. The unique balancing act between the 2 driving servos allows for near 0 point turns. The analog values from the servos are read in and then remapped using the Arduino map() function, which then returns an output for the servos to move after traveling over serial. The servomax and servomin positions are needed to be known for this function to return accurate results. To find this a test program was created to turn a potentiometer that was on the analog in pin telling the servo to move to that point, this can be found in appendix 4. To use the code, slowly turn the potentiometer until it reaches the max position and makes a small noise or stops. The value can then be read from the serial printed data. This number should be written down value for the min and the max, which can be found doing the same process. Repeating this process several times with the new serovmax and servomin in place, produce more accurate results.

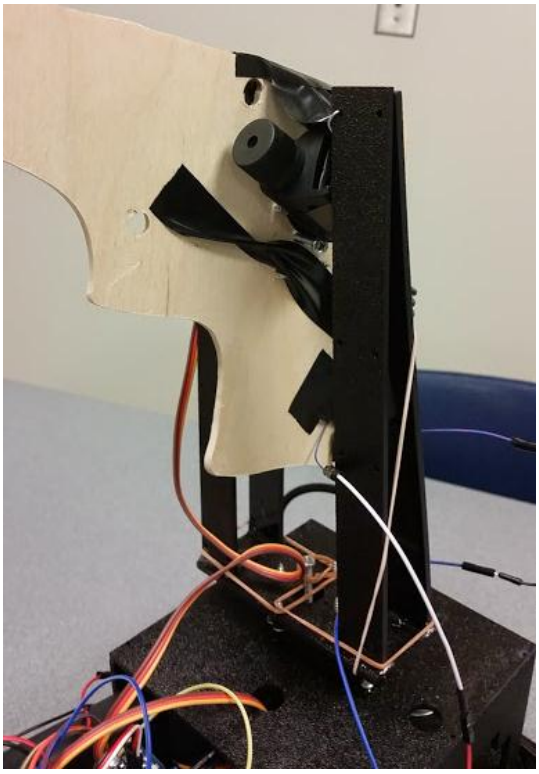


For the continuous servo finding the center position, where the wheels no longer move is preferred first. Finding the approximate top speed in both directions if found using the same process as before, but this time watch until the servo appears at top speed or starts making strange noises and scale back the values. Place the new servomax and servomin values into the

functions and iterate the process till the values are the closest to the true servo motion as possible.



Here is an example of how the code should look resvalue0=map(resvalue0, 0, 1023, SERVOMIN1, SERVOMAX1); the 0 to 1023 is for the analog input values range. Using a function that is adapted to print things more similar to C the data was easier to send as a “string”. The library that was created for this report is called ADAPRINT and found in appendix 5.1-5.2, is based off of an existing function but just placed into a more compact and library format. This allowed printing multiple variables easy and more similar to how other languages print data.



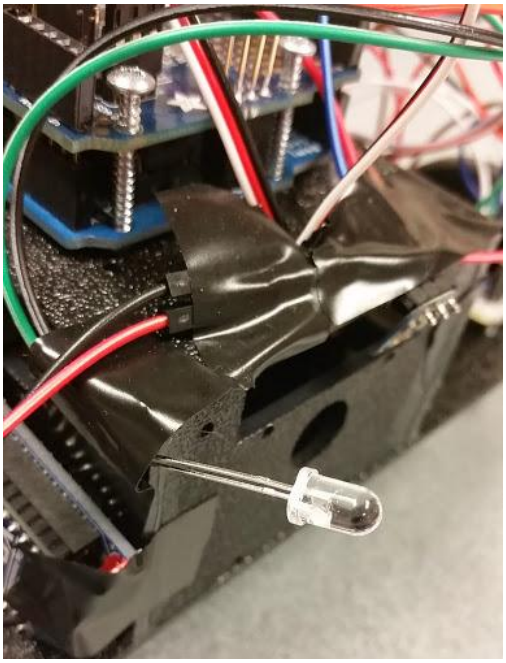
The servo moves after the other Arduino receives the data over serial and parses the commands looking for "," and then placing the data as integers into the variables listed. After the data is in the variable, the value for the servo must be set with the servo shield. Using the Adafruit_PWMServoDriver library and the function setPWM(channel, on, off) the user is able to control where the servo should be at. An example of this code is setPWM(pinNum1, 0, Variable). For best results the second argument should be a 0 and the third is the value read over serial. The first value is just what pins on the servo shield that are being used, which will vary. The reasoning of using a servo shield that communicated over I2C was to be able to save room on the board, allowing a standard power across all servos, and because there was 4 with the possibility for expansion in the future. Some servos in the future that would be added are the servo that controls the pull of the trigger and possible one to control the position of an IR transmitter to control televisions. For more information on how to solder the shield together or how to use some of the other functions, check the documentation that came with these files or this link https://learn.adafruit.com/downloads/pdf/adafruit-16-channel-pwm-slash-servo-shield.pdf .

The servos would be nothing without a way to attach them to the rest of the car. This was done using servo holders that were designed to fit standard size servos and bind them to the car. These servo holders were designed in SolidWorks and allow a slight variation on the sizes of the servos while still allowing the servos to be held firmly. The servo holders were 3D printed and are extremely strong. They are so effective and easy to implement and attach that several people have already expressed interest in purchasing a few for their own projects. These holders hold the 2 continuous servos to the frame and then one attaches the upper frame.

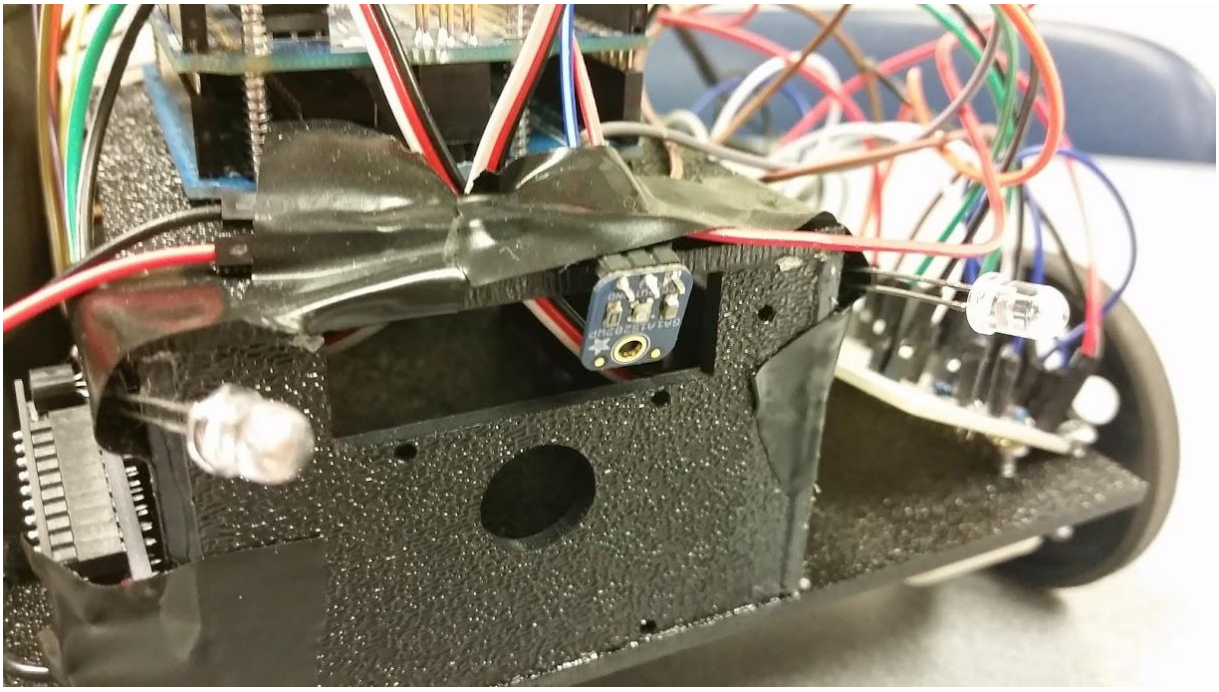
The frame itself went through much iteration before reaching the current frame options. The original frame was made out of cardboard and had no follower wheel. The next frame was made out of laser cut acrylic and held together with glue, screws, and force fits. This frame was the one that the current setup is in and has a caster wheel follower. This caster wheel allows the entire system to move at a faster speed and turn easier. This frame is exceptionally strong for how little it weighs and the small thickness of the material. As mentioned earlier, all of the components are modeled in SolidWorks so a 3D printed frame was created to allow for the real airsoft gun to be mounted to the sentry car. Due to the frame weighing so little and there being so much weight near the top of the sentry the sentry arm occasionally wiggles as it tries to place itself into one position. It will attempt to get to one location and miss it a little and then attempt to return to it and miss again due to the momentum. This will cause a shake that continues that was dampened with rubber bands and added weight. With the 3D printed material and extra weight the system does not appear to have as much of a problem. This could have been solved using transistors that cut power to the servos when they were not needed to be moved but this was not realized until

later into the project and therefore not explored. A larger power source did help a little with the shaking along with a capacitor on the power supply. Separating the logic power and the servo power also helped to achieve more reliable control.

As mentioned earlier the sentry car can be hard to see at a distance. To make the car easier to see at a distance lights were added to the car in the same way that quadcopters or planes reference their direction. There are two red lights that are placed on the back of the car and there are two green lights that are placed on the front of the car as headlights. By adding these lights it was easy to tell the direction of the car at far distances judging by seeing red or green lights. At a distance the 2 lights on both respective sides appear to be one and so by seeing both lights the car is turning or sideways, and by seeing red the car is moving away from the driver, green the car is moving towards the driver. This allowed an easy way to control the car from a distance.



The car does not always need lights on at all times so there is the option to turn them off and the option to auto turn on and off with accordance to the light intensity in the room. The light intensity as mentioned earlier makes us of the artificial delay. By using this technique, the data is not read in for every cycle of the program, this in turn allows for less chance that a shadow will turn on the lights by accident. The lights are bright enough to navigate a dark room using only the light generated from them. The sensor was placed in such a way to avoid a false positive of when the room is dark but the light sensor indicates light due to the headlights. The light sensor chosen is also already built to be a logarithmic light sensor which outputs on the analog read range. This was extremely handy as the wiring was becoming quite messy.



Also on the car is a laser from an airsoft gun. This laser shows the user accurately where the sentry is pointed at and can be toggled on or off. This feature is extremely useful when there is an actual airsoft gun on the system for not only aiming but for just overall accuracy of the systems tracking. The laser is mounted to a wooded plate that has been cut using a Dremel to fit the exact outline of an airsoft gun. There are identical plates that were constructed and drilled through to create a holder for the airsoft gun. The airsoft gun was confined between these two plates and with screws that had spacers so as not to scratch the gun. This was a tight fit that still allowed trigger pull from a string system attached to another servo. The airsoft gun is air compressed allowing the user not to create a reloading mechanism for the sentry and allowing a more rapid firing to be possible.

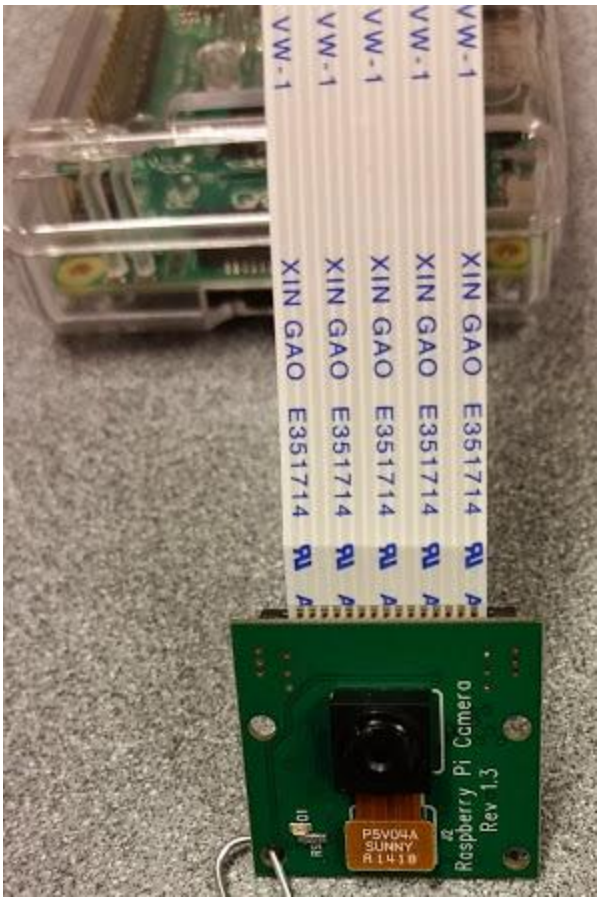
RASPBERRY PI

The Raspberry Pi 2 was chosen to handle the vision aspect of the car. The Raspberry Pi 2 is a large upgrade from the original Raspberry Pi and easily capable of performing as a microcomputer to handle the capturing process along with the processing needed on the images. The Raspberry Pi 2 was connected to the Arduino on the sentry car over USB and powered the Arduino, while receiving power from a USB power block. After pressing auto gun camera control on the Arduino controller it sent the code over XBees to the other Arduino which processed the data and then started serial communication with the Raspberry over the serial USB port. This is a strange process as the Arduino is prone to reset when a new serial communication is started. This was handled with delays and forcing the reset to go into the place it was before. After the communication was started the RPI took a picture and using the Open CV libraries, looked for the color red to target the sentry to. It then sent the approximate location in the image to the Arduino, along with data for where the servos needed to move to approach that point.

The programming language chosen for the task of handling the computer vision aspect of the project was Python using the Open CV. Python was easy to learn and fairly straightforward with a little experience from other programming experience. The original program that worked perfectly with the sentry car system crashed while presenting, but a close clone to the program can be found in appendix 3. When an Arduino does not have enough power to operate properly it is called a brown out, well the Raspberry Pi ended up drawing too much power to operate and froze. When the system was turned back on the code that had been operating for a few trials was

corrupted. All backups of the code were also found to be missing. Lesson learned don't let your Raspberry Pi draw too much power and always back up your code to a USB drive or equivalent.

The Raspberry Pi used the RPi Camera module to collect the pictures necessary for processing the images to find the targets to shoot. The basic idea of the program was to first connect over serial, then capture the image into ram. Capturing the image into ram allows for faster processing and also avoids ruining your SD card as rewriting images will eventually ruin an SD card. After the image was read in downscale it. Just because an image is high resolution does not mean that the higher resolution is necessary for the process. After downscaling apply a filter to the image by separating the color bands and thresh holding the minimum and maximum value. These values do change with light sensitivity so a wide range around the color that is to be targeted is preferred. To avoid this problem HSV method could be used in parallel but during this report it was hard to find accurate documentation and library support for the Raspberry Pi 2. Although this was not implemented, the light sensor could have been used to show the approximate light in the room and adjust the color scales to provide more accurate images for the auto targeting process.

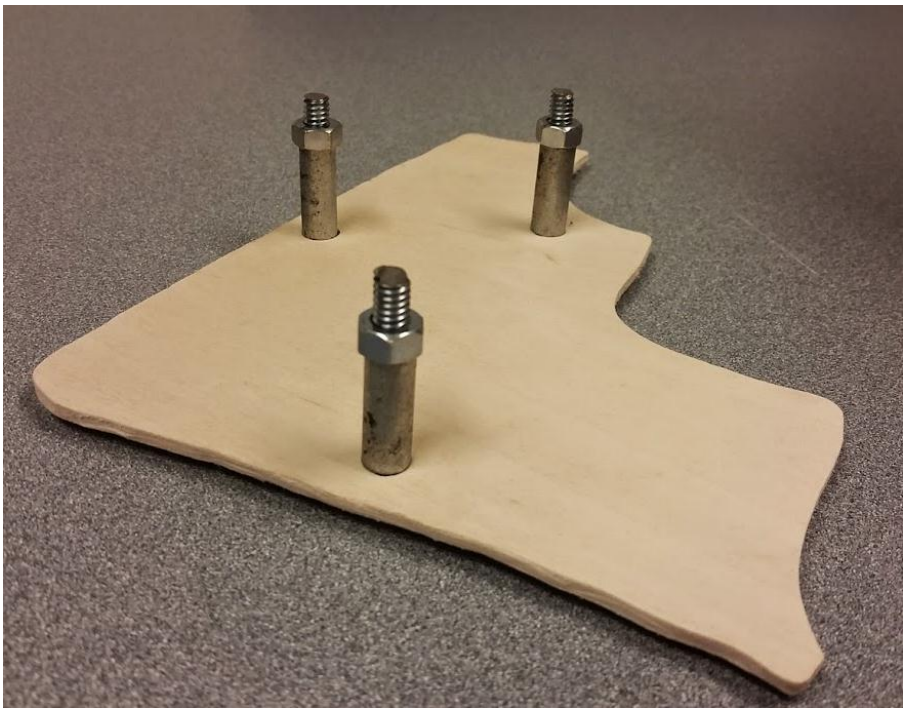


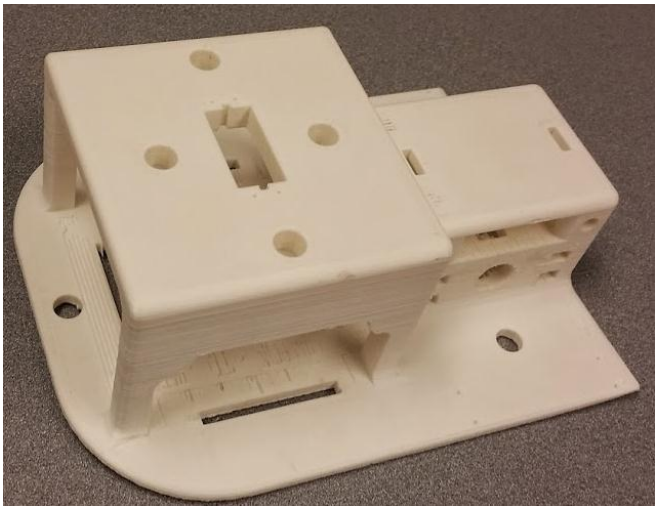
After the image is color thresheld it is time to find the location for the system to target. This is done by subwindowing the image to the expected shape of the object, for simplicity a simple square was used in this report. The squares travel across the rows and columns of the image, calculating the number of surviving color pixels and comparing it to the max number of pixels. If a square finds a new max then it logs those coordinates and continues. After one cycle of this the max pixel location is found and this is very accurately the object that was trying to be detected.

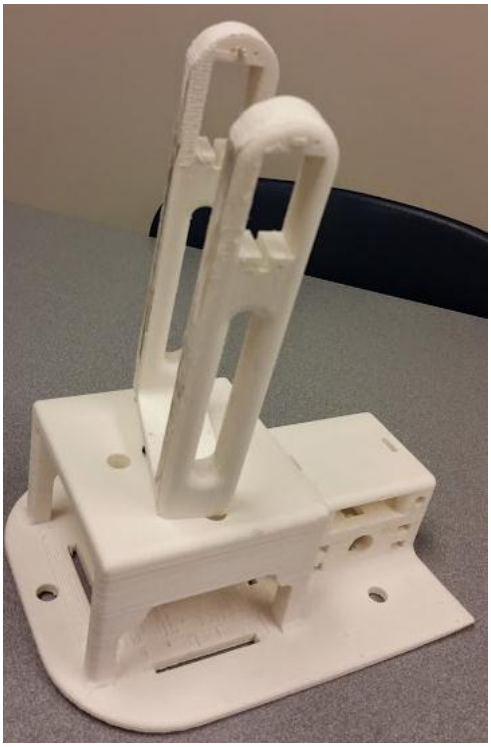
This happens because the color thresh holding was set to that color and after searching for the object in the frame it is the object of most density. Further filtering and blurring can be done to improve the results, but this process was already working fairly consistently. The data was then sent back to the Arduino where the results were then used to aim the sentry gun. The only data that was lost from the crash was the data calculating where the camera was located to the rest of the system, which makes aiming very difficult. The other part that was lost was part of the serial communication, it had been debugged but occasionally it can act sporadically. Due to finals week and the amount of work already into this project, it was deemed an acceptable loss and problem.

FUTURE DEVELOPMENT AND CHALLENGES

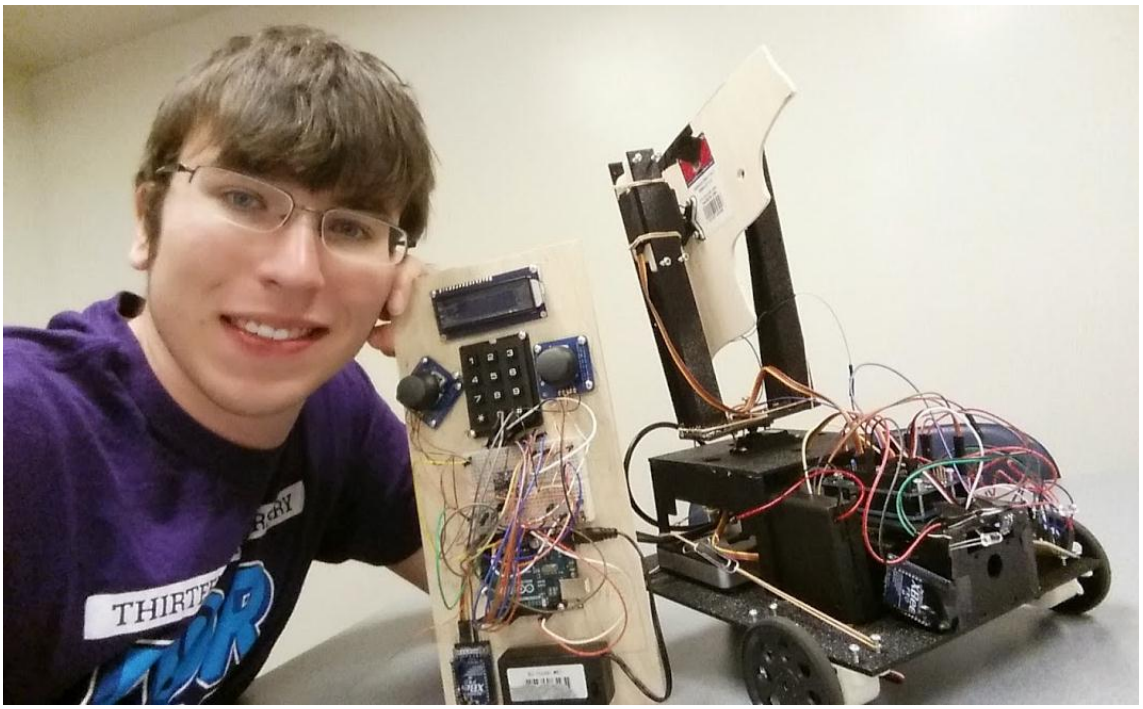
In the future development of this project it will be moved into the 3D printed body and the actual airsoft gun will be mounted into the project. This will be able to be done since the project will no longer be in a gun free zone. Another upgrade will be a more permeant controller that encases all the wires and includes a trigger for firing. Due to power restrictions a shield was removed that allowed music to be played while driving. This did not seem to have enough reason to be kept in the project as it did not really serve any function, but it would be nice to wire it up and drive to some nice beats. The servo's battery source will be upgraded to be LiPo batteries to allow longer life of the batteries and supply a little more power than the double a batteries or the USB power block. Some challenges in this project was being able to wire all of the powers together and still be able to run, this was eventually tackled by running from the Raspberry Pi 2 and Arduino connected. Another issue was having enough time to process all the data "simultaneously" this was accomplished by artificial delays and conditional variable changes, with delays on the sender. The largest problem in the project was the code crashing, which will be a constant reminder in future projects to always back up the code.







Some potential future uses for this project is being able to use the robot chassis for a different purpose as it was designed to fit a multitude of sensors and potential uses. The project could see a police force variant to assist in dangerous situations where it is too dangerous to send in a police officer. The project could also be used in airsoft and paintball wars as a robotic player. The project can also just be a fun toy to drive around. The project is meant to show what is possible and have a low entry level so that users can adapt the system to achieve what they want. This was a passion project and it turned out about how the envisioned project was expected. In the future even greater designs and systems will be created.



Appendix 1

Arduino Master Control Code with Comments

```
/*Sentry Main control system System
  This allows the sentry to read in
  multiple values at the same time and respond to them
  over xbees
  */
#include <Wire.h>
#include <ARDPRINTF.h>
#include <LiquidCrystal.h>
#include <Adafruit_MMA8451.h>
#include <Adafruit_Sensor.h>

Adafruit_MMA8451 mma = Adafruit_MMA8451();

//For KeyPresses
const int debouncer1 = 12;
int debouncer2 = 0;
int debouncer=0;

char home1[]= "HOME: 5 Help";
char home2[]= "Press *or #";
char Normal1[]= "Manuel";
char returnmessage[]= "Press 0 Home";
char AutoCamera[]= "Auto Gun Cam";
char AutoBalance[]= "Auto Gun Bal";
char lights[]= "Lights */#";
char lasers[]= "Laser */#";
char sounds[]= "Music Tracks";
char Help[]= "Help Press";
char Fire[]="Gun is Ready";
char Shoot[]="Gun Shooting";
char Next[]="Next";
char Previous[]="Previous";
int command;
int startnum=12345;
int endnum=67891;
 const int debounceTime = 20; //TRY TO TAKE THIS OUT

// Connect via i2c, default address #0 (A0-A2 not jumpered)
LiquidCrystal lcd(0);

// KeyPad Set Up
  const int numRows = 4; // number of rows in the keypad
  const int numCols = 3; // number of columns
  const char keymap[numRows][numCols] = {
    { '1', '2', '3' } ,
    { '4', '5', '6' } ,
```

```
    { '7', '8', '9' } ,
    { '*', '0', '#' }};
 // this array determines the pins used for rows and columns
  const int rowPins[numRows] = { 7, 2, 3, 5 }; // Rows 0 through 3
  const int colPins[numCols] = { 6, 8, 4 }; // Columns 0 through 2

//Car movement
  int potpin0=0; //analog pin used to connect the potentiometer
  int potpin1=1; //analog pin used to connect the potentiometer
  int resvalue0;//value from poteniometer
  int resvalue1;//value from poteniometer

// Senty Movment
  int potpin2=2; //analog pin used to connect the potentiometer
  int potpin3=3; //analog pin used to connect the potentiometer
  int resvalue2;//value from poteniometer
  int resvalue3;//value from poteniometer

// Depending on your servo make, the pulse width min and max may vary, you
// want these to be as small/large as possible without hitting the hard stop
// for max range. You'll have to tweak them as necessary to match the servos you
// have!
//Servo1 is a Parallax Continous, Motor Center 380
//Servo2 is a Parallax Continous, Motor Center 380
//Servo3 is a Stanard servo SG-5010 Tower Pro Digi hi Speed
//Servo4 is a Stanard servo SG-5010 Tower Pro Double Ball Bearings
//Servo5 is Tower pro micro servo 99 SG 90

#define SERVOMIN1  335 //this is the 'minimum' pulse length count (out of 4096)
#define SERVOMAX1  425 // this is the 'maximum' pulse length count (out of 4096)
#define SERVOMIN2  335 // this is the 'minimum' pulse length count (out of 4096)
#define SERVOMAX2  425 // this is the 'maximum' pulse length count (out of 4096)
#define SERVOMIN3  100 //this is the 'minimum' pulse length count (out of 4096)
#define SERVOMAX3  585 // this is the 'maximum' pulse length count (out of 4096)
#define SERVOMIN4  110 // this is the 'minimum' pulse length count (out of 4096)
#define SERVOMAX4  595 // this is the 'maximum' pulse length count (out of 4096)
//#define SERVOMIN5  125 // this is the 'minimum' pulse length count (out of 4096)
//#define SERVOMAX5  590 // this is the 'maximum' pulse length count (out of 4096)

//String readString, servo1, servo2, servo3, servo4, servo5;


void setup(void) {
  Serial.begin(19200);
    // set up the LCD's number of rows and columns:
  lcd.begin(16, 2);
```

```
  lcd.setBacklight(HIGH);
  // Print a message to the LCD.
  //lcd.print(home1);
    // set the cursor to column 0, line 1
  // (note: line 1 is the second row, since counting begins with 0):
  //lcd.setCursor(0, 1);
  // print the number of seconds since reset:
  //lcd.print(home2);

  //Set up for the Keypad to Work
    for (int row = 0; row < numRows; row++){
      pinMode(rowPins[row],INPUT); // Set row pins as input
      digitalWrite(rowPins[row],HIGH); // turn on Pull-ups
    }

    for (int column = 0; column < numCols; column++){
      pinMode(colPins[column],OUTPUT); // Set column pins as outputs for writing
      digitalWrite(colPins[column],HIGH); // Make all columns inactive
    }

    if (! mma.begin()) {//errors began after this was put in
      lcd.print("We never made it");
      while (1); }

  mma.setRange(MMA8451_RANGE_2_G);
}

void loop() {
  char key = getKey();

    if(command==0){
      command=777;}

   if (key=='1'){
    lcd.clear();
    lcd.print(lights);
     lcd.setCursor(0, 1);
     lcd.print(returnmessage);
    command=1;
  }

   if (key=='2'){
    lcd.clear();
    lcd.print(AutoCamera);
     lcd.setCursor(0, 1);
     lcd.print(returnmessage);
```

```
command=2;}

if (key=='3'){
lcd.clear();
lcd.print(AutoBalance);
 lcd.setCursor(0, 1);
 lcd.print(returnmessage);
command=3;}

if (key=='4'){
lcd.clear();
lcd.print(sounds);
 lcd.setCursor(0, 1);
 lcd.print(returnmessage);
command=4;}

if (key=='5'){
lcd.clear();
lcd.print(Help);
 lcd.setCursor(0, 1);
 lcd.print(returnmessage);
command=5;}

if (key=='6'){
lcd.clear();
lcd.print(lasers);
lcd.print("ON");
 lcd.setCursor(0, 1);
 lcd.print(returnmessage);
command=6;}

if (key=='7'){
lcd.clear();
lcd.print(lasers);
lcd.print("OFF");
 lcd.setCursor(0, 1);
 lcd.print(returnmessage);
command=7;}

if (key=='8'){
lcd.clear();
lcd.print(Fire);
 lcd.setCursor(0, 1);
 lcd.print(returnmessage);
command=8;}
```

```
   if (key=='9'){
   lcd.clear();
   lcd.print(Shoot);
    lcd.setCursor(0, 1);
    lcd.print(returnmessage);
   command=9;}

   if (key=='0'){
     lcd.clear();
    lcd.print(home1);
    lcd.setCursor(0, 1);
    lcd.print(home2);
    command=0;}

   if (key=='*'){
   lcd.clear();
   lcd.print(Next);
    lcd.setCursor(0, 1);
    lcd.print(returnmessage);
   command=10;}

   if (key=='#'){
   lcd.clear();
   lcd.print(Previous);
    lcd.setCursor(0, 1);
    lcd.print(returnmessage);
   command=11;}

  resvalue0=analogRead(potpin0);//Read the value of the poteniometer
  resvalue0=map(resvalue0, 0, 1023, SERVOMIN1, SERVOMAX1); //This is the value found
after calibration
  resvalue1=analogRead(potpin1);//Read the value of the poteniometer
  resvalue1=map(resvalue1, 0, 1023, SERVOMAX2,SERVOMIN2); //This is the value found
after calibration flipped for direction
   //pulselength = map(degrees, 0, 180, SERVOMIN, SERVOMAX);//used later for degrees if
wanted
    resvalue2=analogRead(potpin2);//Read the value of the poteniometer
    resvalue2=map(resvalue2, 0, 1023, SERVOMIN3, SERVOMAX3); //This is the value found
after calibration
    resvalue3=analogRead(potpin3);//Read the value of the poteniometer
    resvalue3=map(resvalue3, 0, 1023, SERVOMIN4, SERVOMAX4); //This is the value found
after calibration


  if(command==3){
     mma.read();
```

```
    resvalue2=map(mma.x, -4200, 4200, SERVOMIN3, SERVOMAX3);
    resvalue3=map(mma.y, -4200, 4200, SERVOMIN4, SERVOMAX4);}


  ARDPRINTF("%d,%d,%d,%d,%d", resvalue0, resvalue1,resvalue2,resvalue3,command);
  //Fixed the command thing so I should be able to send command codes over now Just fix code
above
  //ARDPRINTF("%d,%d,%d,%d", resvalue0, resvalue1,resvalue2,resvalue3);
   delay(200);
}

//The getkey function allows for us to get the value from the keypad
  char getKey(){
    char key = 0; // 0 indicates no key pressed

    for(int column = 0; column < numCols; column++){
      digitalWrite(colPins[column],LOW); // Activate the current column.

      for(int row = 0; row < numRows; row++){ // Scan all rows for a key press.

        if(digitalRead(rowPins[row]) == LOW){ // Is a key pressed?
          //delay(debounceTime); // debounce

            while(digitalRead(rowPins[row]) == LOW)
              ; // wait for key to be released !!!!!!!!This PAUEssss the program a BIT
          key = keymap[row][column]; // Remember which key was pressed.
        }
      }
      digitalWrite(colPins[column],HIGH); // De-activate the current column.
    }
    return key; // returns the key pressed or 0 if none
  }
```

Appendix 2

Arduino Master Receiver Code with Comments and Debugging Statements Commented Out

```
/*Sentry Main control system System
  This allows the sentry to read in
  multiple values at the same time and respond to them
  over xbees
  */
#include <Wire.h>
#include <Adafruit_PWMServoDriver.h>
#include <ARDPRINTF.h>

const int address=0x40;
//pulselength = map(degrees, 0, 180, SERVOMIN, SERVOMAX);
// called this way, it uses the default address 0x40
Adafruit_PWMServoDriver pwm = Adafruit_PWMServoDriver(address);
// Depending on your servo make, the pulse width min and max may vary, you
// want these to be as small/large as possible without hitting the hard stop
// for max range. You'll have to tweak them as necessary to match the servos you
// have!
//Servo1 is a Parallax Continous, Motor Center 380
//Servo2 Is a Parallax Continous, Motor Center 380
int servonum0 = 0; //This is the location where the pin is connected over I2c
int servonum1 = 15;
int servonum2= 4;
int servonum3= 11;
int sensorPin = A0;
int sensorValue = 0;  // variable to store the value coming from the sensor
int ledPin=8;
int LaserPin=9;
int fakedelay=10;
int fakedelaytime=0;
int oldservo3=0;
int oldservo4=0;
int iteration1=50;
int iteration2=50;
int auto1=0;
int auto2=0;
#define SERVOMIN3  100 //this is the 'minimum' pulse length count (out of 4096)
#define SERVOMAX3  585 // this is the 'maximum' pulse length count (out of 4096)
#define SERVOMIN4  110 // this is the 'minimum' pulse length count (out of 4096)
#define SERVOMAX4  595 // this is the 'maximum' pulse length count (out of 4096)

String readString, servo1, servo2, servo3, servo4, command;
String inputString = "";         // a string to hold incoming data
boolean stringComplete = false;

void setup() {
  Serial.begin(19200);
```

```
  pwm.begin();
  pwm.setPWMFreq(60);  // Analog servos run at ~60 Hz updates
  pinMode(ledPin, OUTPUT);
  pinMode(LaserPin, OUTPUT);
  inputString.reserve(200);
}

void loop() {

   //int startnum = Serial.parseInt();
   int servo1 = Serial.parseInt();
   int servo2 = Serial.parseInt();
   int servo3 = Serial.parseInt();
   int servo4 = Serial.parseInt();
   int command = Serial.parseInt();
   Serial.println("120");
   //int endnum = Serial.parseInt();
   //pulselength = map(degrees, 0, 180, SERVOMIN, SERVOMAX);//used later for degrees if
wanted
   pwm.setPWM(servonum0, 0, servo1);
   pwm.setPWM(servonum1, 0, servo2);
   if(command!=2){
     pwm.setPWM(servonum2, 0, servo3);
     pwm.setPWM(servonum3, 0, servo4);}

   if(command==2){
     int i=0;
     Serial.println("800");
      while(i==0){
       pwm.setPWM(servonum0, 0, 380);
       pwm.setPWM(servonum1, 0, 380);
        command=0;
        delay(7000);
         if (stringComplete) {
          //Serial.println(inputString);
            String myString=inputString;
            int commaIndex = myString.indexOf(',');
            //  Search for the next comma just after the first
            int secondCommaIndex = myString.indexOf(',', commaIndex+1);
            String firstValue = myString.substring(0, commaIndex);
            String secondValue = myString.substring(commaIndex+1, secondCommaIndex);
            String thirdValue = myString.substring(secondCommaIndex+1); //To the end of the
string
            int command = firstValue.toInt();
            int servo3 = secondValue.toInt();
            int servo4 = thirdValue.toInt();
```

```
        // clear the string:
        inputString = "";
        stringComplete = false;

      //if(command==777){
       // Serial.println("BOSS We did it");}
      // if(servo1<150){
       // Serial.println("Seems High");}
       servo3=map(servo3, 0, 292, SERVOMIN3, SERVOMAX3);
       servo4=map(servo4, 0, 389, SERVOMIN4, SERVOMAX4);
       pwm.setPWM(servonum2, 0, servo3);
        pwm.setPWM(servonum3, 0, servo4);
        //command=0;
        delay(3000);
         i=1;}}
/* Serial.println("800");}
  Serial.println("800");
  while(command!=777){
    pwm.setPWM(servonum0, 0, 380);
    pwm.setPWM(servonum1, 0, 380);
    int command=Serial.parseInt();
    int servo3 = Serial.parseInt();
    int servo4 = Serial.parseInt();}
  servo3=map(servo3, 0, 292, SERVOMIN3*.5, SERVOMAX3*.5);
  servo4=map(servo4, 0, 389, SERVOMIN4*.5, SERVOMAX4*.5);
  pwm.setPWM(servonum2, 0, servo3);
  pwm.setPWM(servonum3, 0, servo4);
  command=0;*/
  }


/*if(abs(oldservo3-servo3)>iteration1){
pwm.setPWM(servonum2, 0, servo3);}

if(abs(oldservo4-servo4)>iteration2){
pwm.setPWM(servonum3, 0, servo4);}

 oldservo3=servo3;
 oldservo4=servo4;*/

if(command==6){
  digitalWrite(LaserPin, HIGH);}

 if(command==7){
  digitalWrite(LaserPin, LOW);}
if(command!=1){
```

```
  if(fakedelay<fakedelaytime){
    sensorValue = analogRead(sensorPin);
    fakedelaytime=0;}

    if(sensorValue<250){
      digitalWrite(ledPin, HIGH);}

    if(sensorValue>250){
      digitalWrite(ledPin, LOW);}

     fakedelaytime++;
   }
   delay(50);
 if(command==1){
   digitalWrite(ledPin, HIGH);}
}

 void serialEvent() {
  while (Serial.available()) {
    // get the new byte:
    char inChar = (char)Serial.read();
    // add it to the inputString:
    inputString += inChar;
    // if the incoming character is a newline, set a flag
    // so the main loop can do something about it:
    if (inChar == '\n') {
      stringComplete = true;
    }
  }
}
```

Appendix 3
Raspberry Pi Camera Control with some debugging statements

```
#!/usr/bin/python
import serial
import os
import cv2
import cv
import numpy as np
import math
import time

arduino=serial.Serial('/dev/ttyACM0',19200)
while 1:
   #Read in the values from arduino
   comein=arduino.readline()
   comein=float(comein)
   haveidoneit=1
   if(comein>777):
      haveidoneit=0
   #print comein
   havedoneit=0
   if(haveidoneit==0):
      ##Resize with resize Command
      #FULL PICTURE IS 1944 2592
      #At .25 Image is 486 by 648
      def resizeImage(img):
         dst=cv2.resize(img,None, fx=.15, fy=.15, interpolation= cv2.INTER_LINEAR)
         return dst

      ##Take Image With Rasberry Pi Camera
      ## THis might need to be flipped if orientation is wrong
      os.system('raspistill -o /run/shm/image.jpg')


      ##Load image
      img= cv2.imread('/run/shm/image.jpg')

      img=resizeImage(img)
      image=img

      height, width, depth=img.shape
      print height, width, depth

      grey=cv2.imread('/run/shm/image.jpg',0) #0 for grayscale
      grey=resizeImage(grey)

      ## List of Boundaries
      #boundaries=[([17, 15, 100], [50, 56, 200])]
```

```
boundaries=[([0, 0, 125], [125,150, 255])]
#values are actually bgr
for (lower, upper) in boundaries:
   lower=np.array(lower, dtype="uint8")
   upper=np.array(upper, dtype="uint8")

#find the colors within the specified boundares
mask=cv2.inRange(image, lower, upper)
output=cv2.bitwise_and(image, image, mask=mask)
ret,thresh=cv2.threshold(output,0,255,cv2.THRESH_BINARY)

#Bright intensinity
radiusHigh=30
oldmax=0
newmax=0
maxcoordx=0
maxcoordy=0

paddedarray=cv2.copyMakeBorder(thresh,radiusHigh,radiusHigh,radiusHigh,radiusHigh,0)
padsize1, padsize2, depth3=paddedarray.shape
i=1
j=1
for i in range(1,(padsize1-radiusHigh)):
   for j in range (1,(padsize2-radiusHigh)):
      temp=paddedarray[i:radiusHigh+i, j:radiusHigh+j]
      oldmax=cv2.sumElems(temp)
      if(oldmax>newmax):
         newmax=oldmax
         maxcoordy=i-radiusHigh+radiusHigh*.5
         maxcoordx=j-radiusHigh+radiusHigh*.5
   j=1

command=777
message3=str(command)
message1=str(maxcoordy)
message2=str(maxcoordx)
message4=message3+','+message1
message5=message4+','+message2
message=message5+'\n'
print(message)
#arduino.write('Hi')
time.sleep(2)
arduino.write(message)
haveidoneit=1
```

Appendix 4
Program for Servo Setting Allows the User to use 2 servos but easier with 1

```
/*
  Servo control
  Be able to use servos over I2c
  Sends a messgae, over xbee and adjust the angle
  NOW IN COLOUR
  UPDATE: Arduino to arduino
  Made by Trevor Craig
  lincolnkite@live.com
*/


#include <Wire.h>
#include <Adafruit_PWMServoDriver.h>
#include <ARDPRINTF.h>

const int address=0x40;
int potpin0=0; //analog pin used to connect the potentiometer
int potpin1=1; //analog pin used to connect the potentiometer
int resvalue0;//value from poteniometer
int resvalue1;
//pulselength = map(degrees, 0, 180, SERVOMIN, SERVOMAX);
// called this way, it uses the default address 0x40
Adafruit_PWMServoDriver pwm = Adafruit_PWMServoDriver(address);

// Depending on your servo make, the pulse width min and max may vary, you
// want these to be as small/large as possible without hitting the hard stop
// for max range. You'll have to tweak them as necessary to match the servos you
// have!
//Servo1 is a Tower pro micro servo 99 SG 90
//Servo2 Is a Stanard servo SG-5010 Tower Pro

#define SERVOMIN1  335 //185this is the 'minimum' pulse length count (out of 4096)
#define SERVOMAX1  425 // this is the 'maximum' pulse length count (out of 4096)
//Stopped Servo is at 380

int servonum0 = 15; //This is the location where the pin is connected over I2c

void setup() {
  Serial.begin(9600);
  pwm.begin();
  pwm.setPWMFreq(60);  // Analog servos run at ~60 Hz updates
}

void loop() {
  resvalue0=analogRead(potpin0);//Read the value of the poteniometer
  resvalue1=analogRead(potpin1);//Read the value of the poteniometer
```

```
  resvalue0=map(resvalue0, 0, 1023, SERVOMIN1, SERVOMAX1); //This is the value found after calibration
  //resvalue0=map(resvalue0, 0, 1023, SERVOMAX1,SERVOMIN1); //This is the value found after calibration and direction to flip
  //resvalue1=analogRead(potpin1);//Read the value of the poteniometer
  //resvalue1=map(resvalue1, 0, 1023, SERVOMIN2, SERVOMAX2); //This is the value found after calibration
   //pulselength = map(degrees, 0, 180, SERVOMIN, SERVOMAX);//used later for degrees if wanted
   pwm.setPWM(servonum0, 0, resvalue0);
  // Serial.println(resvalue0);
   ARDPRINTF("%d,%d", resvalue0, resvalue1);
}
```

Appendix 5.1

Libraries for ADAPRINT cpp file

```
/*
   The function call is similar to printf: ardprintf("Test %d %s", 25, "string");

   To print the '%' character, use '%%'

   This code was first posted on http://arduino.stackexchange.com/a/201
         This code is adapted by Trevor Craig into a library
         Enjoy, this is my first library
*/

#include <ARDPRINTF.h>
#include "Arduino.h";
#define ARDBUFFER 16 //Buffer for storing intermediate strings. Performance may vary
depending on size.

int ARDPRINTF(char *str, ...) //Variadic Function
{
  int i, count=0, j=0, flag=0;
  char temp[ARDBUFFER+1];
  for(i=0; str[i]!='\0';i++)  if(str[i]=='%')  count++; //Evaluate number of arguments required to be
printed

  va_list argv;
  va_start(argv, count);
  for(i=0,j=0; str[i]!='\0';i++) //Iterate over formatting string
  {
    if(str[i]=='%')
    {
      //Clear buffer
      temp[j] = '\0';
      Serial.print(temp);
      j=0;
      temp[0] = '\0';

      //Process argument
      switch(str[++i])
      {
        case 'd': Serial.print(va_arg(argv, int));
                  break;
        case 'l': Serial.print(va_arg(argv, long));
                  break;
        case 'f': Serial.print(va_arg(argv, double));
                  break;
        case 'c': Serial.print((char)va_arg(argv, int));
                  break;
        case 's': Serial.print(va_arg(argv, char *));
```

```
                break;
        default:  ;
      };
    }
    else
    {
      //Add to buffer
      temp[j] = str[i];
      j = (j+1)%ARDBUFFER;
      if(j==0)  //If buffer is full, empty buffer.
      {
        temp[ARDBUFFER] = '\0';
        Serial.print(temp);
        temp[0]='\0';
      }
    }
  };

  Serial.println(); //Print trailing newline
  return count + 1; //Return number of arguments detected
}
```

Appendix 5.2

Libraries for ADAPRINT .h file

```
/*
   The function call is similar to printf: ardprintf("Test %d %s", 25, "string");

   To print the '%' character, use '%%'

   This code was first posted on http://arduino.stackexchange.com/a/201
        This code is adapted by Trevor Craig into a library
        Enjoy, this is my first library
*/

#ifndef ARDPRINTF_H
#define ARDPRINTF_H
#if ARDUINO >= 100
 #include "Arduino.h"
#else
 #include "WProgram.h"
#endif
#include <stdarg.h>
int ARDPRINTF(char *str, ...);
#undef ARDBUFFER
#endif
```