Principles of Engineering: Lab 2

Emily Lepert and Gracey Wilson

26 September 2017

1 Background and Procedure

In this lab, we built a 3 dimensional scanner using a pan/tilt mechanism with servo motors and an infrared distance sensor.

Materials Used:

- 1 solderless breadboard
- 1 Arduino
- 1 infrared distance sensor
- 2 servo motors

Our tasks were as follows:

- Design and build a circuit connecting the servos and infrared sensor to the Arduino
- Design and build a mount connecting the servos to the sensor so that the sensor can scan 3 dimensional objects
- Calibrate the sensor to correlate raw infrared sensor data to measurements of distance
- Write a program that moves the servos in panning and tilting motions in order for the sensor to scan a cardboard cut-out of a letter in 3 dimensions
- Visualize the data collected from the sensor's scanning of the cardboard letter in both 2 and 3 dimensions, creating a visual representation of the letter

2 Calibrating and Testing the Sensor

To calibrate our sensor, we took a set of measurements of the infrared sensor at different distances. We collected data from 5in to 55in in 5in increments 10 times and created a new data set that consisted of the averages of each reading. From there we used Mathematica's

Fit function to generate a line of best fit for the data as shown below. The curve equation is:

$$y = 112.1425508814386 - 1.0035935392387034 * x + 0.00405508173593060 * x^2$$

$$-7.416839407985343 * 10^{-6} * x^3 + 4.934357177555067 * 10^{-9} * x^4$$

where y is the distance from the sensor in inches and x is the reading from the infrared sensor.

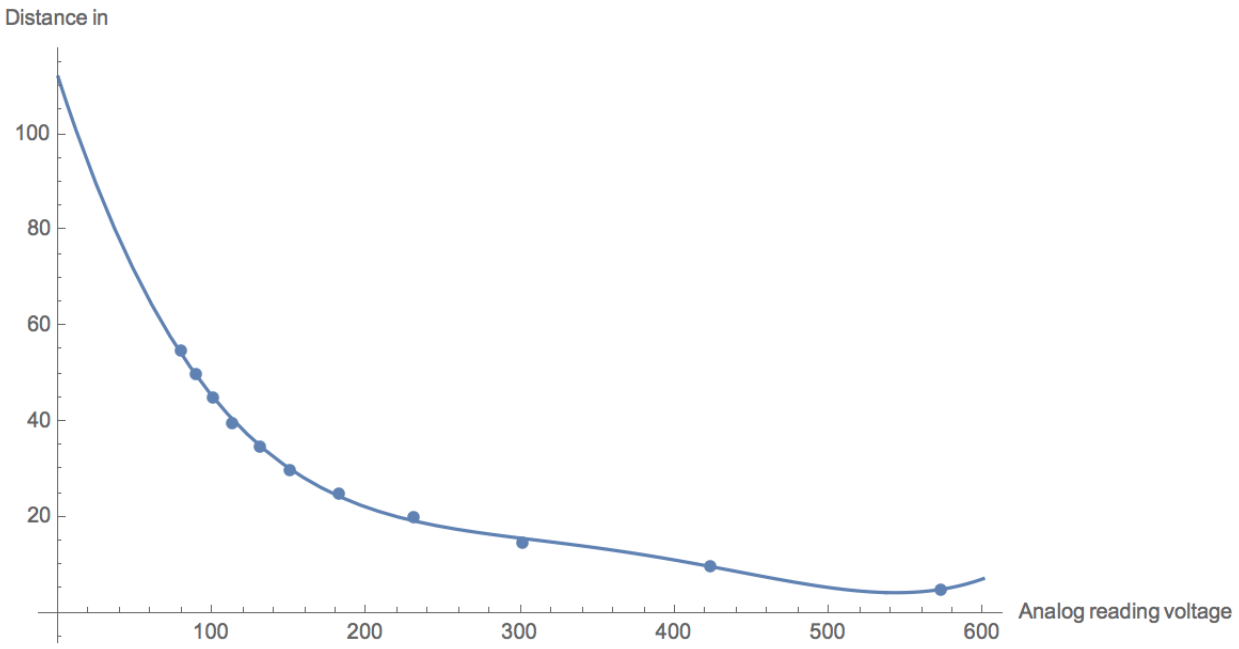

Figure 1: Curve fit for sensor calibration

Applying this same equation to a new set of readings taken from the infrared sensor, we get a new plot which depicts the predicted distance and actual distance for a given sensor reading. The readings on the plot below were not part of the readings used for the calibration.

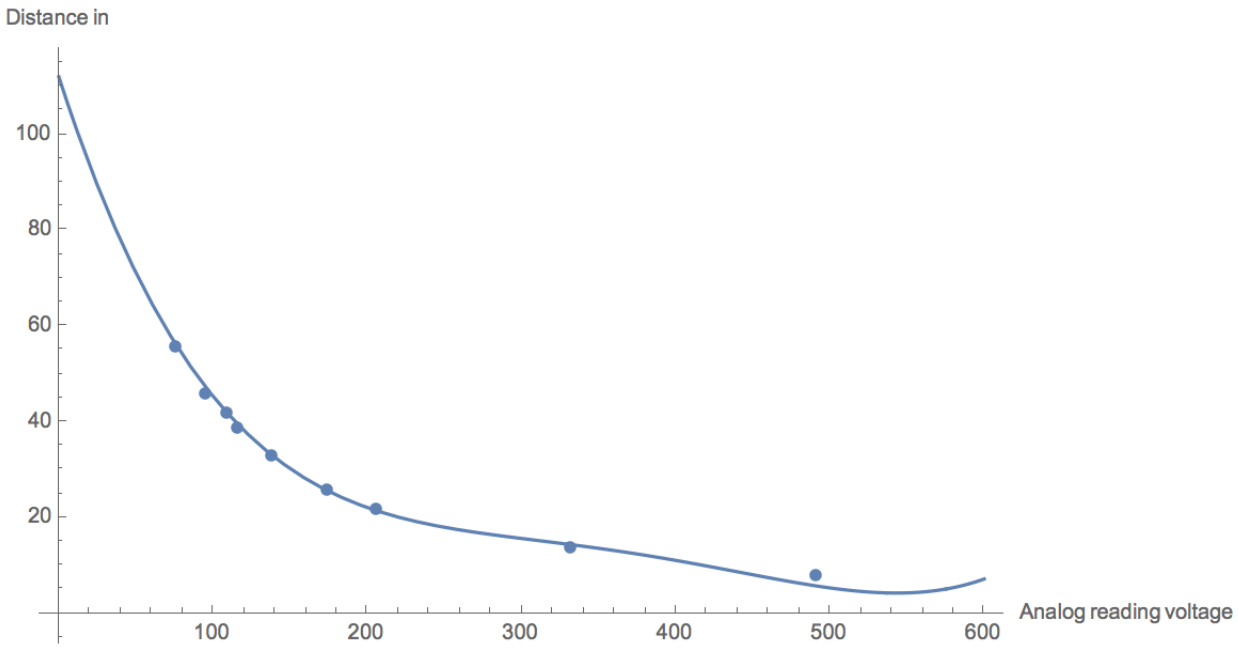

Figure 2: Error plot for sensor calibration

Our function fits this new data very well.

3 Mounting the Sensor

In order to make our sensor capable of scanning a 2-D cardboard letter, we needed to mount it to the servos. For the 2-D scan, we only needed to mount the sensor to one of the servos in order to pan left-to-right. As shown in Figure 3 however, we chose to mount both servos together in preparation for the 3-D scan, and simply only use one of the servos in the program (code shown in section 5.1.1). We built the prototype of the mount using cardboard and tape. The results of the 2-D scan are shown in section 5.1.3.

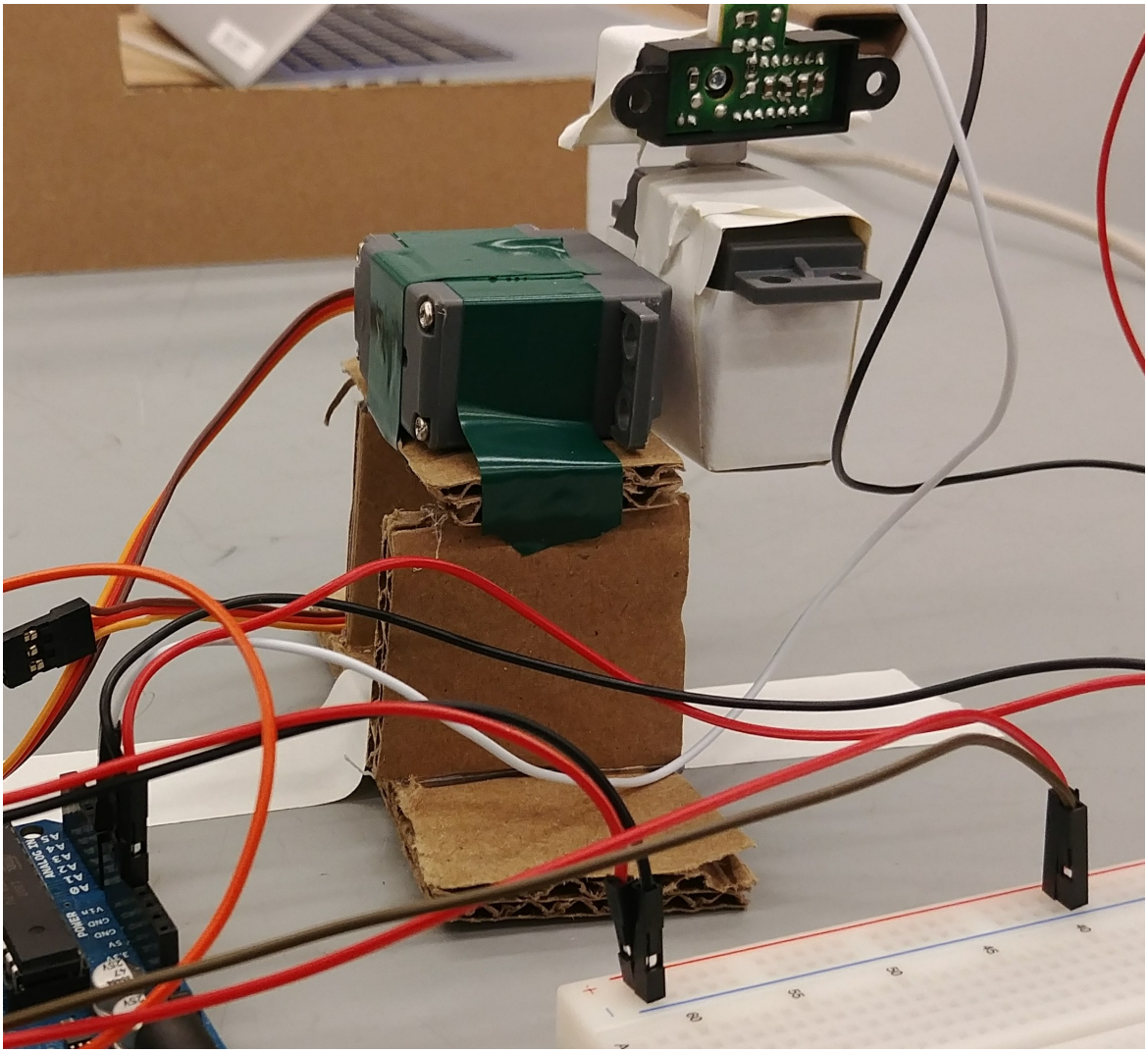

Figure 3: Prototype mount for 2-D sweep

Once we saw that the 2-D scan worked, we went on to program 2 servos to work together in a pan/tilt mechanism, sweeping the cardboard letter in 3 dimensions. We also used CAD-ing software and 3-D printers to design and print some pieces to hold the servos and sensor together more elegantly than tape and cardboard. The CAD models are shown in Figures 4-6.

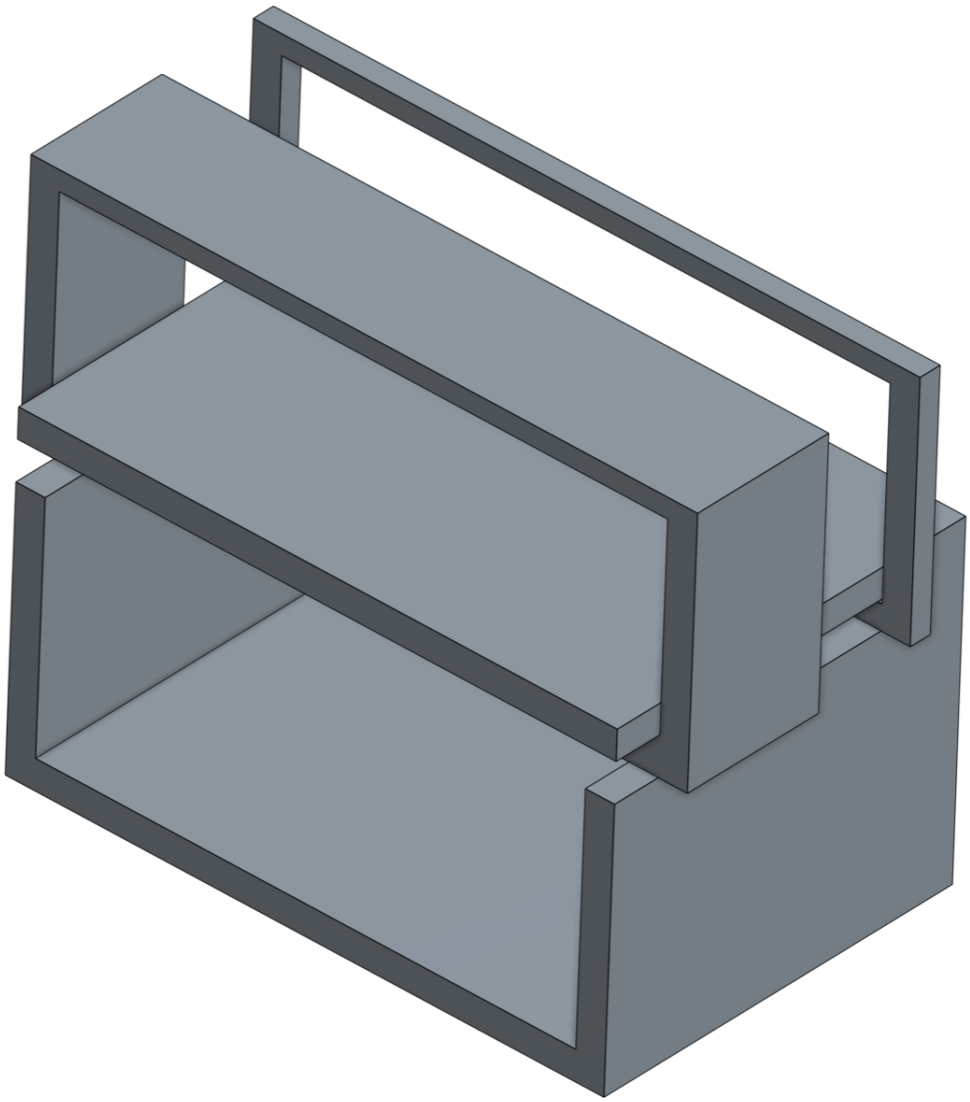

Figure 4: CAD model of the parts that elevate the tilting servo to the required height in order for the attached panning servo to be able to move.

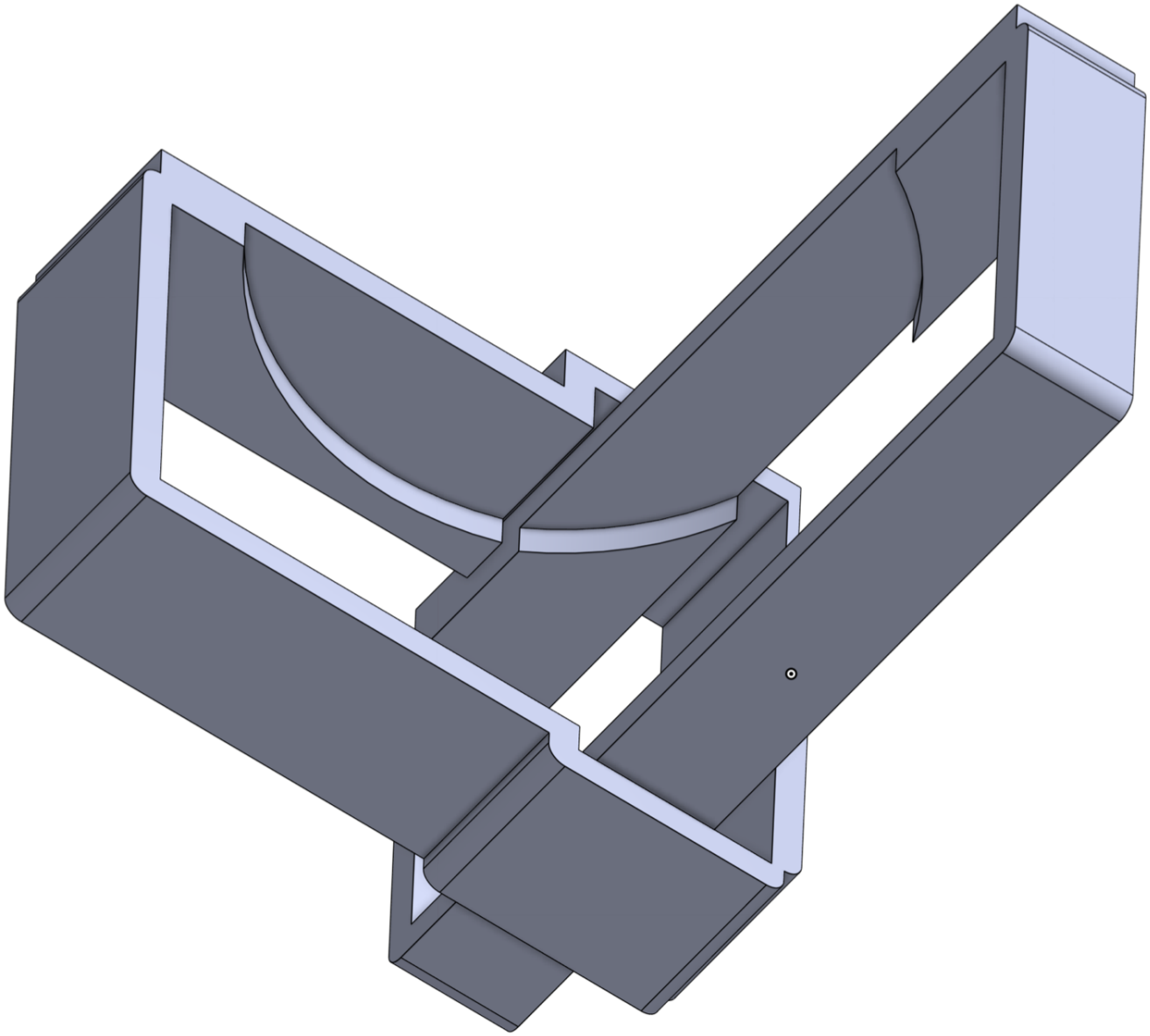

Figure 5: CAD model of the two bands that wrap around the panning servo and part of the plastic round servo attachment, which plugs into the tilting servo.

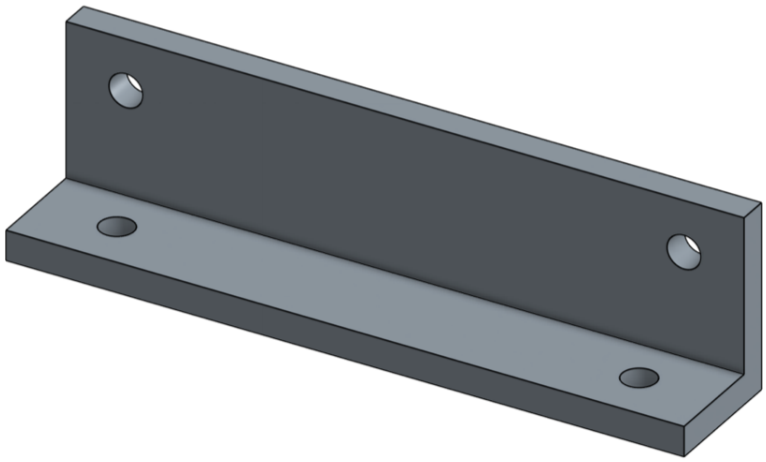

Figure 6: CAD model of the part that attaches the sensor to the panning servo using screws.

As with most design processes, our mechanical components required an iterative approach. We printed these CAD models, tested them to see how they fit the servos, tweaked dimensions, reprinted, and retested, several times for each of the parts. We had to account for diverse tolerances. Throughout the process, errors occurred in the measurements of the servos that we took using calipers, and in the inexactness of the 3-D printers. One example of where we accounted for that is visible in Figure 5. The vertical band (tilted to the left) leaves extra room when wrapping around the horizontal band (tilted to the right). This is the result of an earlier model which, when printed, did not fit snugly together as the CAD model showed, but actually did not fit at all. We left this extra room to account for a margin of error.

In the end, 4 of the 6 pieces were fully functional. We used a few pieces of electrical tape to replace the failed 3-D printed parts in 1) holding the tilting servo to the base piece, and 2) attaching the sensor to the panning servo. The final mounting structure is shown is Figures 7 and 8.

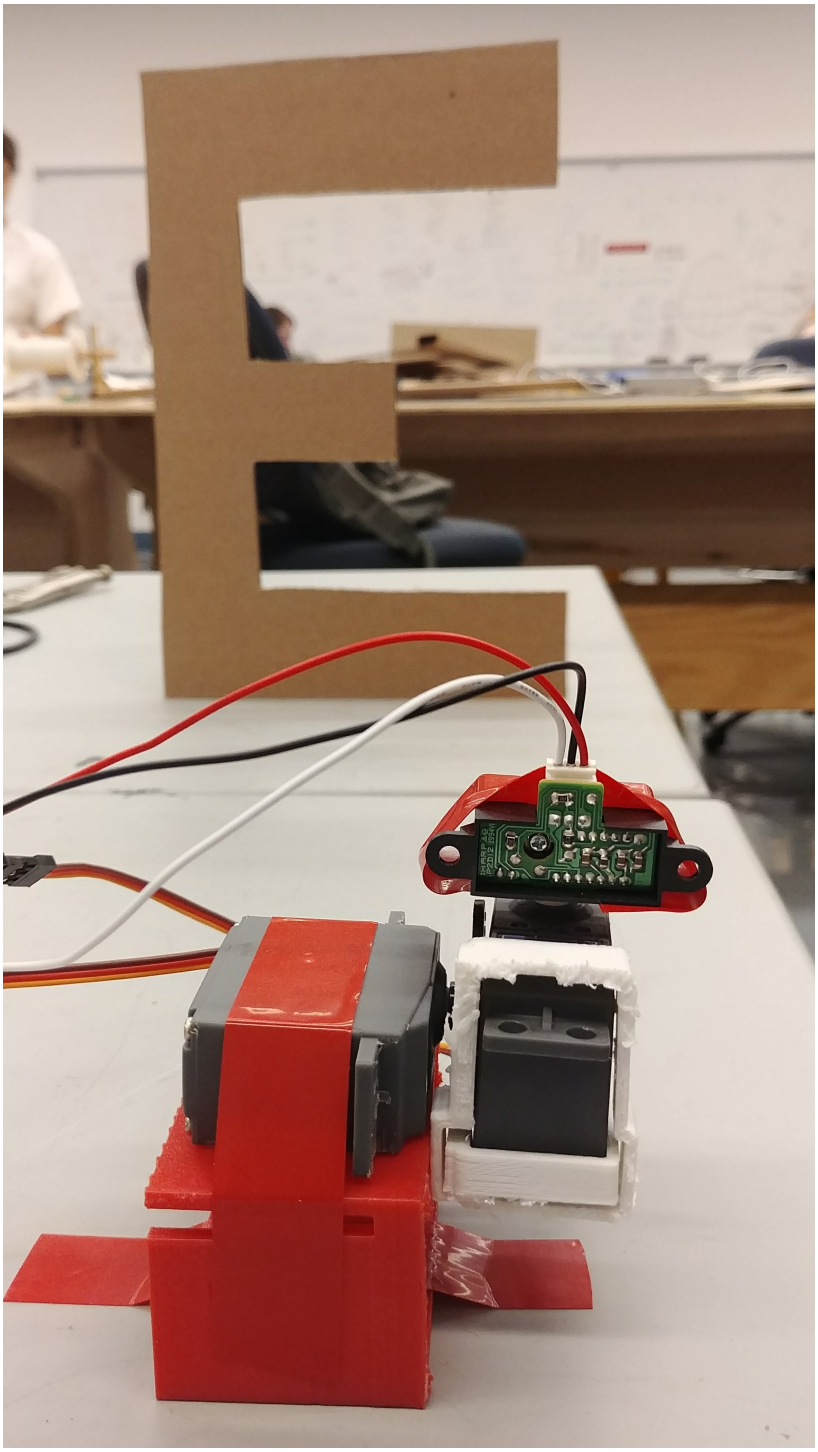

Figure 7: Final mount for 3-D sweep

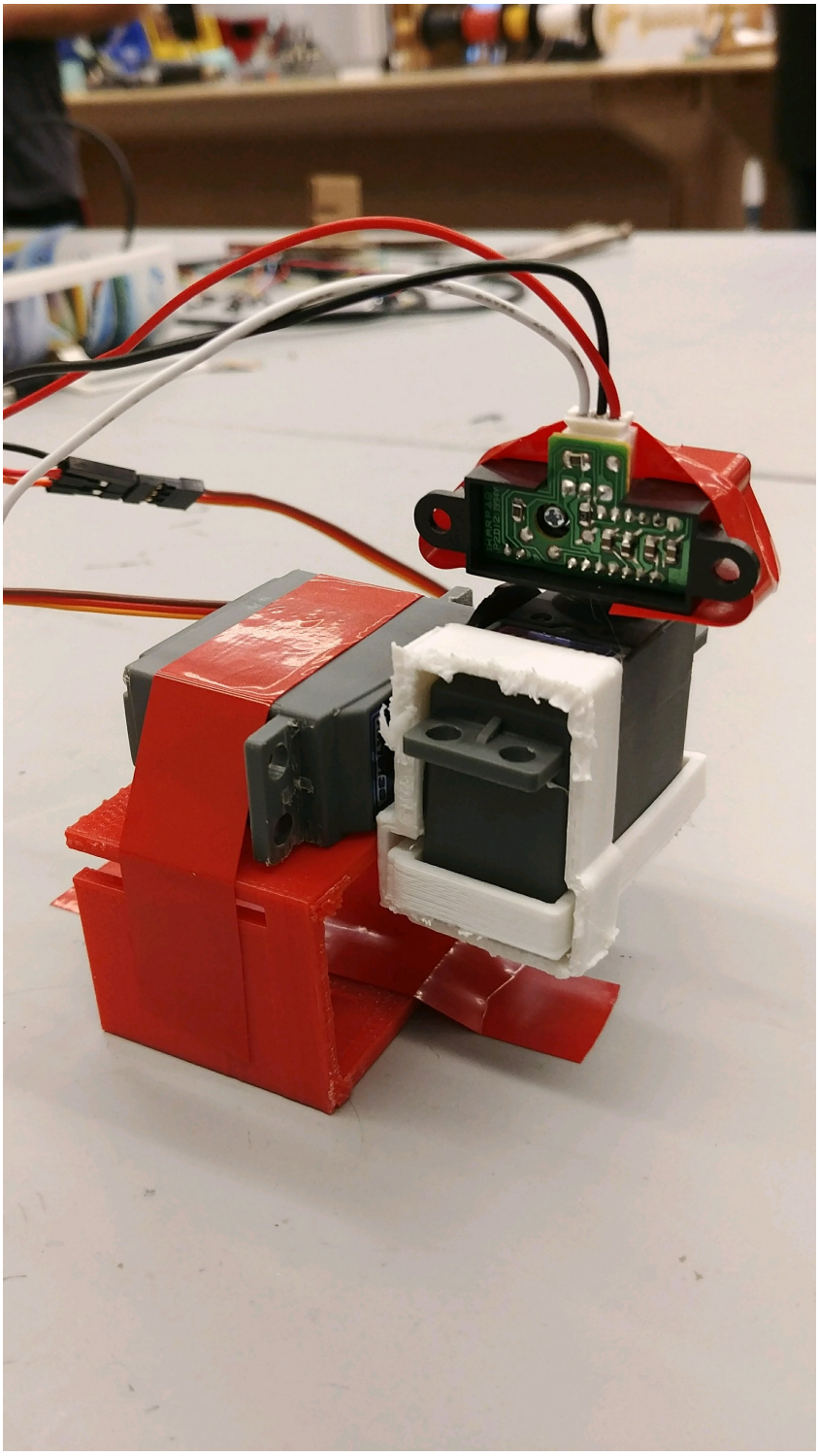

Figure 8: Close up of final mount for 3-D sweep

One thing to note is that our mechanical mount makes it so that our sensor is not actually at the center of the spherical coordinate system that we use to graph our letter. In hindsight, we could have compensated for that in our math by calculating where our sensor is along the

radius and angles of the sphere. Since we got a nice visualization anyway, we were satisfied with our output, but we would have perhaps been ever more successful had we compensated for the discrepancy in the location of the sensor.

4 Circuit Diagram

The circuit for this lab is quite simple. It consists of 2 servo motors, attached to the digital inputs on the Arduino, and 1 infrared sensor, attached to an analog input on the Arduino. Both the motors and sensor are all connected to ground and 5V as well.

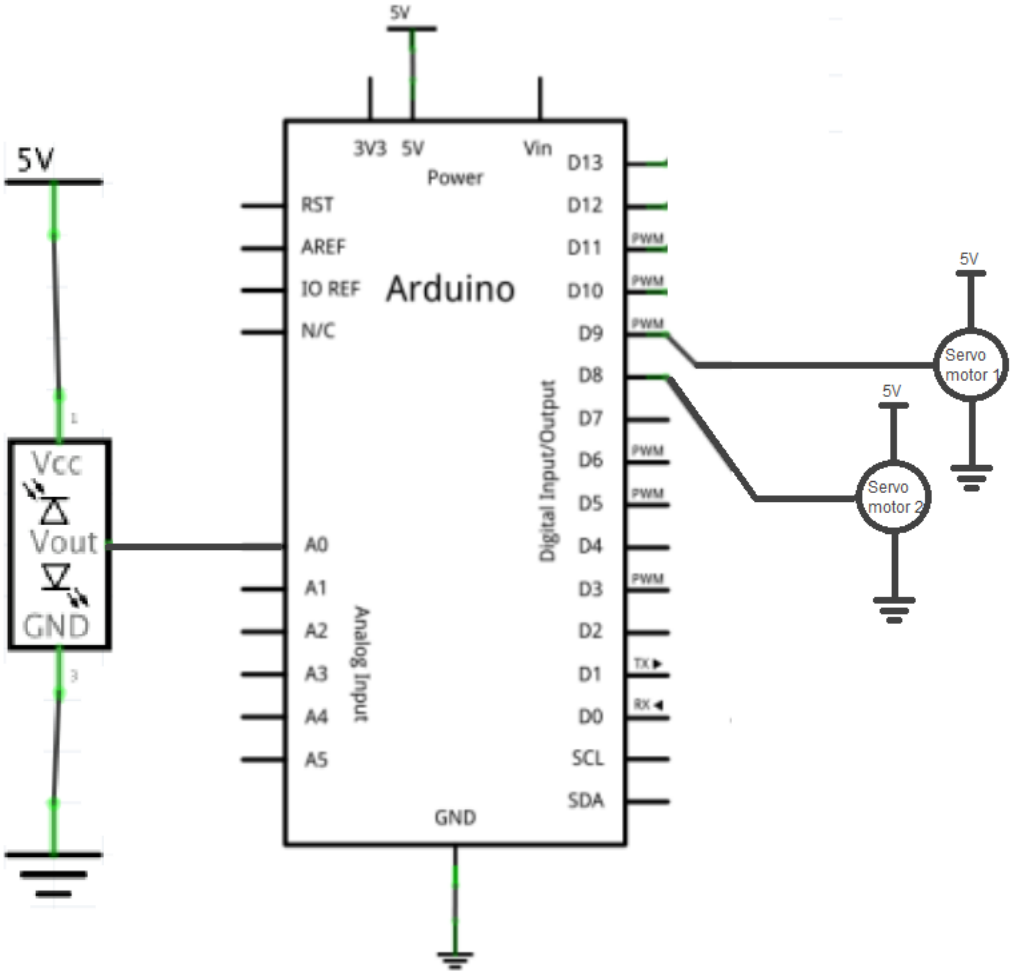

Figure 9: Circuit Diagram

5 Source Code and Visualizations

5.1 2-D Sweep

5.1.1 Arduino Code

Since we set up both of our sweeps to work with both servos, the 2-D sweep source code initializes objects and variables for both servos and the sensor.

```
#include <Servo.h>

Servo servo1;           //create 1st servo object to control tilting servo
Servo servo2;          //create 2nd servo object to control panning servo

int pos1 = 0;           //initialize variable to store 1st servo position
int pos2 = 0;           //initialize variable to store 2nd servo position
int ldeg2 = 0;         // leftmost degree for servo2
int rdeg2 = 90;        // rightmost degree for servo2

const int analogInPin = A0;  // Analog input pin that the potentiometer is
    attached to

int sensorValue = 0;         // value read from the sensor
```

In our setup function, we connect the servos to the servo objects, and start the scan with the scan function.

```
void setup() {
  servo1.attach(9);    //attaches servo1 on pin 9 to the servo object
  servo2.attach(8);    //attaches servo2 on pin 8 to the servo object
  Serial.begin(9600);
  scan();
  Serial.println("done");  //signal for our Python program to stop reading
      from serial monitor
}
```

Our scan function consists of setting the tilting servo motor (servo1) to a value that ensured that the panning servo (servo2) which is mounted on servo1, would point towards the letter. To create consistency between our trials, we decided to ensure that the tilting servo starts and ends in the same position (hence the two for loops). The first for loop sweeps the panning servo left-to-right without recording any data from the sensor. The second loop sweeps right-to-left and prints out the sensor's data to the serial monitor. Since we needed the position of the panning servo and the sensor reading, we print out "sensorValue" and "pos2" to the serial monitor. We include the "/" and ":" to make our Python program's parsing of the serial monitor information easier.

```
void scan(){
  servo1.write(45);  //set servo1's position
  for (pos2 = ldeg2; pos2 <= rdeg2; pos2 +=1){
    servo2.write(pos2);
    delay(15);
  }
  for (pos2 = rdeg2; pos2 >= ldeg2; pos2 -=1){
```

```
    servo2.write(pos2);
    sensorValue = analogRead(analogInPin);
    // print the results to the Serial Monitor:
    Serial.print(sensorValue);
    Serial.print("/");
    Serial.print(pos2);
    Serial.println(":");

    // wait 2 milliseconds before the next loop for the analog-to-digital
    // converter to settle after the last reading:
    delay(15);
  }
}
```

5.1.2 Python Code

We used Python to create a visual representation of our letter. We used the pyserial and MatPlotLib python libraries.

```
#!/usr/local/bin/python

from serial import Serial, SerialException
import matplotlib.pyplot as plt
```

Next we initialize our program's connection to the serial monitor.

```
cxn = Serial('/dev/tty.usbmodem1411', baudrate=9600)
```

We then used Python to read the values that the Arduino printed to the serial monitor, parse the data, and store it in lists to plot later on. When our program printed "done" to the serial monitor (the last thing the Arduino function prints after the scan is complete), the python program stops running the loop. The "/" and ":" embedded in the serial monitor output allows us to easily separate the sensor value from the servo position value.

coorz and coorx are arrays that store the values read from the serial monitors and are the values that go on to be plotted as r and theta respectively.

```
coorz = []
coorx = []

running = True

while running == True:
    cxn.write([1])

    result = str(cxn.readline())

    if result[2:-5] == 'done':
        running = False

    else:

        start_pos2 = result.index('/')
        end_pos2 = result.index(':')
```

```
18
19          r = int(result[2:start_pos2])
20          theta = int(result[start_pos2+1:end_pos2])
21
22          theta -= 30
23
24          coorx.append(theta)
25          coorz.append(r)
```

Once we collected all the data, we used matplotlib to plot our data.

```
fig = plt.figure()
axes = fig.add_axes([0.1, 0.1, 0.8, 0.8]) # left, bottom, width, height (range
    0 to 1)

surf = axes.plot(coorx, coorz, 'r-')
plt.xlabel("theta")
plt.ylabel("r")

plt.show()
```

5.1.3 2-D Sweep Graph

The plot generated from this is shown in Figure 10.

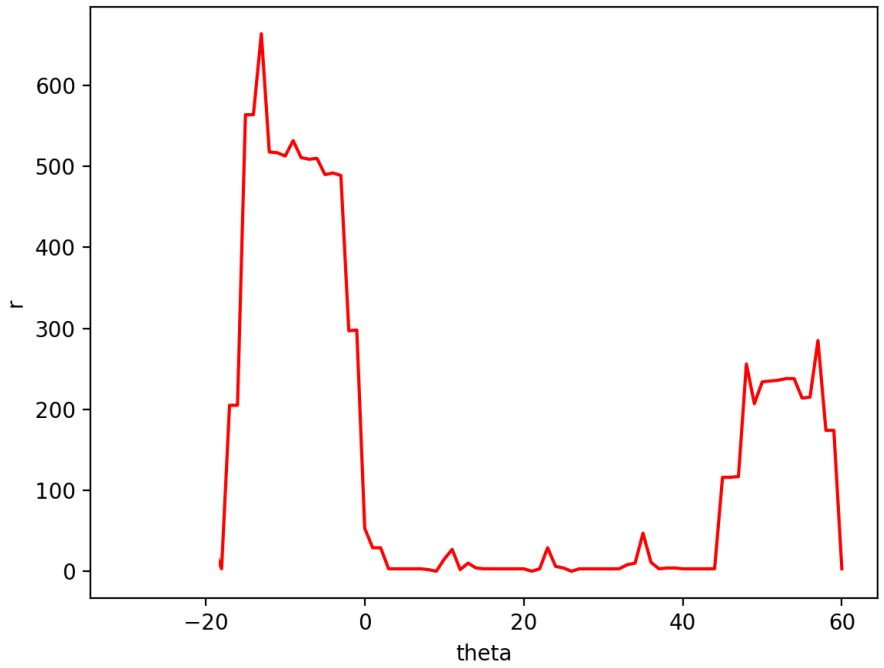

Figure 10: Sensor data for our 2-D sweep (where r is the sensor reading and theta is the servo position)

There are three distinct sections of the plot. The first is a section of low r values which indicates that the sensor was reading something close to it. The two sections on either side of the middle section have higher values of r indicating that the sensor was reading something further away from it. This far-close-far pattern indicates that the sensor was reading our letter when the servo was at the 0-45 position.

For this part, we were trying to get to a point where our sensor was showing distinct differences between the letter and non-letter sections. For this reason, the y axis of our plot is the raw sensor data rather than the data converted to a distance. Once we got our sensor data to be correct, we knew that any discrepancy later on would be due to our math.

5.2 3-D Sweep

5.2.1 Arduino Code

The set up for the 3-D sweep is nearly identical to the set up for our 2-D sweep with the addition of setting the maximum and minimum positions for our servo1 and modifying the values of the maximum and minimum positions for our servo2.

```
int hdeg1 = 70;        // highest degree for servo1
int ldeg1 = 10;        // lowest degree for servo1
int ldeg2 = 0;         // leftmost degree for servo2
int rdeg2 = 70;        // rightmost degree for servo2
```

For our 3-D sweep we needed both of our servos to move so we wrote a for loop for the tilting servo that contained a for loop for the tilting servo. We also added a print statement to the serial monitor because both the pan and tilt servos' positions were now important.

```
void scan(){
  for (pos1 = ldeg1; pos1 <= hdeg1; pos1 +=1) { //servo1 moves from 0 to deg
      degrees in steps on 1 degree
    servo1.write(pos1); //servo1 moves to pos1 position
    delay(15);                              //wait 15ms for servo to reach
       position

    for (pos2 =ldeg2; pos2 <= rdeg2; pos2 +=1){
      servo2.write(pos2);
      sensorValue = analogRead(analogInPin);   // read the analog in value

      // print the results to the Serial Monitor:
      Serial.print(sensorValue);
      Serial.print(";");
      Serial.print(pos1);
      Serial.print("/");
      Serial.print(pos2);
      Serial.println(":");

      // wait 2 milliseconds before the next loop for the analog-to-digital
      // converter to settle after the last reading:
      delay(15);
    }
  }

```

```
}
```

5.2.2 Python Code

Our Python code for the 3-D sweep is once again very similar to our 2-D sweep code. We used the same libraries as well as numpy. We added a third array for phi and a threshold value. We decided to use a scatter plot to visualize our letter which meant that there was going to be a lot of points on our graph that were not generated due to either noise or because the sensor was detecting things that were further away than the letter. By looking at our graph, we found a threshold value of 16 (for r) that eliminated a lot of the extraneous points on our graph.

```
import numpy as np

coorz = []
coorx = []
coory = []
running = True
threshold = 16

while running == True:
    cxn.write([1])

    result = str(cxn.readline())

    print(result)
    if result[2:-5] == 'done':
        running = False
```

We added an extra marker in the serial monitor output (";") to once again facilitate in differentiating between values. We also applied our equation to the sensor value to convert the value to an actual distance in inches.

```
    else:
        start_pos1 = result.index(';')
        start_pos2 = result.index('/')
        end_pos2 = result.index(':')

        r = int(result[2:start_pos1])
        theta = int(result[start_pos2+1:end_pos2])
        phi = int(result[start_pos1+1:start_pos2])

        r = 112.1425508814386 - 1.0035935392387034*r + 0.00405508173593060*r
            **2 - 7.416839407985343*10**(-6)*r**3 + 4.934357177555067*10**(-9)
            *r**4
```

In this program, before adding the values of r, theta, and phi to the array, we check to see whether r is below the threshold (whether it is part of the data that will describe the letter). If r is above the threshold, it we can ignore the point.

```
        if r < threshold:
            coorx.append(r)
```

```
            coory.append(theta)
            coorz.append(phi)
```

We also converted our spherical coordinates into Cartesian coordinates using these equations. We will go over why we prefer the spherical coordinates for this lab in the next section.

```
    x = r*np.cos(theta)*np.sin(phi)
    y = r*np.sin(theta)*np.sin(phi)
    z = r*np.cos(phi)
```

5.2.3 3-D Sweep Visualization

From the above code, we were able to generate this scatter plot.

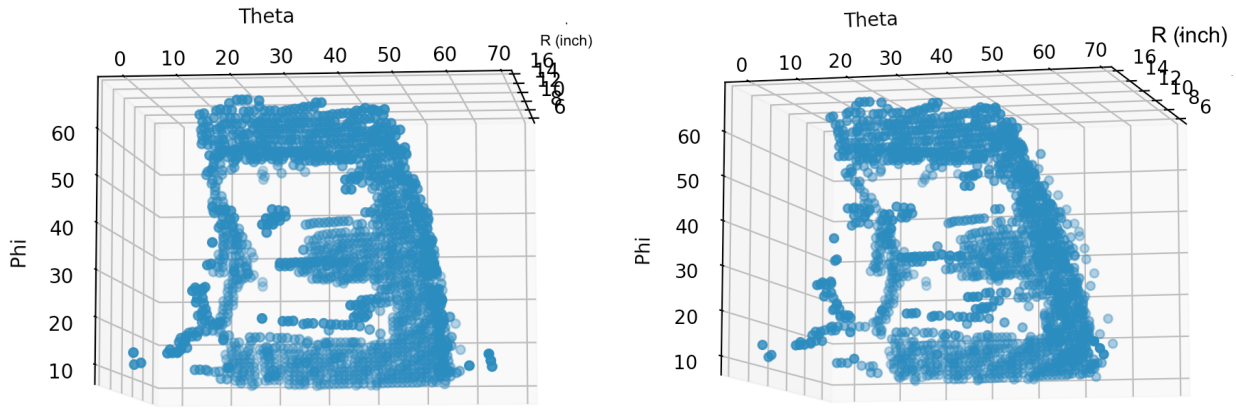

Figure 11: View of the E scatter plot from the front and slightly to the side

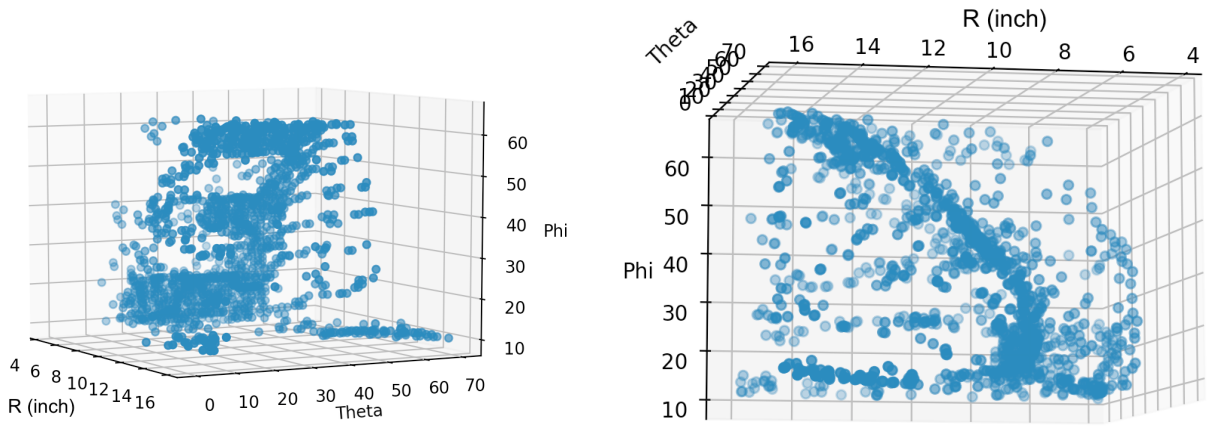

Figure 12: View of the E scatter plot from the side

We plotted r vs theta vs phi. Figure 11 shows that we can distinctly see the letter E in our scatter plot. The letter is displayed backwards due to the orientation of the graph.

Figure 12 shows the extraneous points that remain on our plot even after implementing the threshold. To reduce the appearance of all of these points, we could have made our threshold dependent on r, theta, and phi. We can also see that the E is curved in our scatter plot. The curve is due to the fact that our sensor was positioned toward the bottom of the E such that the distance between the sensor and the bottom of the E was smaller than the distance between the sensor and the top of the E. Because our letter was small enough, the distortion was not extreme.

Because the distortion was not too bad, using r, theta, and phi to plot our letter worked out fine. We also plotted the letter using Cartesian coordinates (see code) to see what the difference would be.

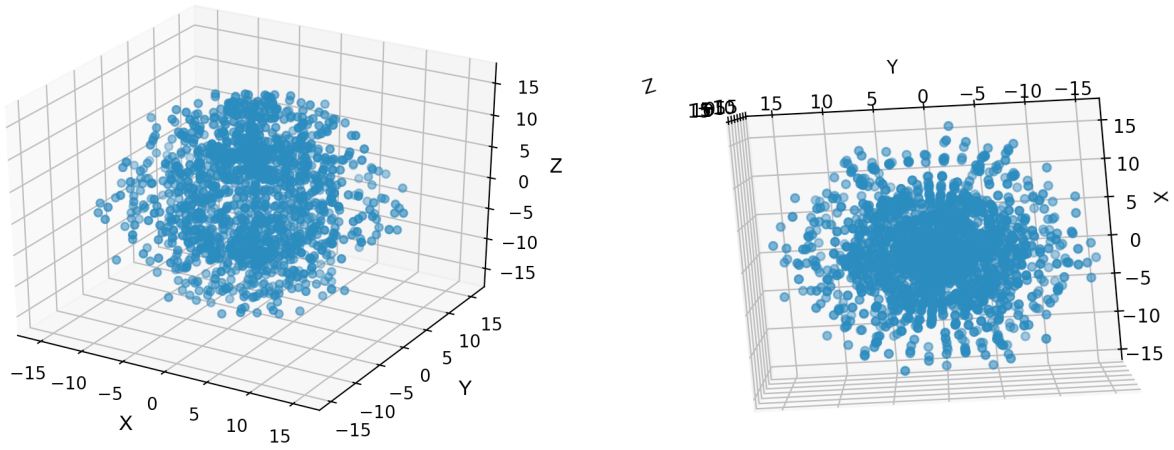

Figure 13: View of the E scatter plot in Cartesian coordinates

We thought we would need to convert our spherical coordinates to Cartesian coordinates because it would visually look better on our graph. However, it turned out that plotting the spherical coordinates produced a much clearer visualization of our letter. Awesome NINJA Mackenzie Frackleton helped us understand that for the purpose of this lab (which is to visualize a physical letter in virtual space), using spherical coordinates is acceptable.

6 Reflection

This lab was one of our first experiences with the true purpose of POE: integrating elements of mechanical, electrical and computer engineering design. We practiced CAD-ing and 3-D printing when creating our mount, became more familiar with servos, infrared sensors and Arduinos when making our circuit, and practiced coding both when programming the servos' movement in Arduino and when visualizing our data in Python.

One challenge we experienced during this lab which we had not faced before was interfacing between 2 forms of code: Arduino and Python. We had to ask Python to read values from Arduino's serial monitor and familiarize ourselves with the technical constraints associated with that process. One such constraint we learned about along the way is that the Arduino program was not able to run while the Python program was running, so we had

to exit out of the graph produced by our Python program each time we wanted to run the Arduino program once more.

Overall, we gained some valuable knowledge that will be applicable to systems we will work with in the future - and we hope to adapt that much quicker when faced with the next challenge.

We are grateful to have benefited from the expertise of the teaching team during class and the patient assistance of NINJAs outside of class. Specifically we learned about how to make our plots clearer from NINJA Mackenzie Frackleton and received lab report edits from NINJA William Lu. Thanks to all for another great learning experience!

7 Appendix

Full Source Code: 2-D Sweep - Arduino

```
/* Sweep across cardboard letter 2D
 *  POE Lab 2 - DIY 3-D Scanner
 *  Emily Lepert and Gracey Wilson
*/

#include <Servo.h>

Servo servo1;          //create 1st servo object to control the tilt servo
Servo servo2;         //create 2nd servo object to control the pan servo

int pos1 = 0;          //initialize variable to store 1st servo position
int pos2 = 0;          //initialize variable to store 2nd servo position
int ldeg2 = 0;        // leftmost degree for servo2
int rdeg2 = 90;       // rightmost degree for servo2

const int analogInPin = A0;  // Analog input pin that the potentiometer is
    attached to

int sensorValue = 0;        // value read from the sensor

void setup() {
  servo1.attach(9);   //attaches servo1 on pin 9 to the servo object
  servo2.attach(8);   //attaches servo2 on pin 8 to the servo object
  Serial.begin(9600);
  scan();
  Serial.println("done");  //signal for python code to stop reading from
      serial monitor
}

void scan(){
  servo1.write(45);
  for (pos2 = ldeg2; pos2 <= rdeg2; pos2 +=1){
    servo2.write(pos2);
    delay(15);
  }
  for (pos2 = rdeg2; pos2 >= ldeg2; pos2 -=1){
    servo2.write(pos2);
```

```
    sensorValue = analogRead(analogInPin);
    // print the results to the Serial Monitor:
    Serial.print(sensorValue);
    Serial.print("/");
    Serial.print(pos2);
    Serial.println(":");

    // wait 2 milliseconds before the next loop for the analog-to-digital
    // converter to settle after the last reading:
    delay(15);
 }
}
 void loop(){
 }
```

2-D Sweep - Python

```
#!/usr/local/bin/python

from serial import Serial, SerialException
import matplotlib.pyplot as plt

# The Serial constructor will take a different first argument on
# Windows. The first argument on Windows will likely be of the form
# 'COMX' where 'X' is a number like 3,4,5 etc.
# Eg.cxn = Serial('COM5', baudrate=9600
#
# NOTE: You won't be able to program your Arduino or run the Serial
# Monitor while the Python script is running.
#cxn = Serial('/dev/tty.usbmodem1411', baudrate=9600)


cxn = Serial('/dev/tty.usbmodem1411', baudrate=9600)

coorz = []
coorx = []

running = True

while running == True:
# try:
    cxn.write([1])

    result = str(cxn.readline())

    if result[2:-5] == 'done':
        running = False

    else:

        start_pos2 = result.index('/')
        end_pos2 = result.index(':')

        r = int(result[2:start_pos2])
```

```
        theta = int(result[start_pos2+1:end_pos2])
        theta -= 30

        coorx.append(theta)
        coorz.append(r)

fig = plt.figure()
axes = fig.add_axes([0.1, 0.1, 0.8, 0.8]) # left, bottom, width, height (range
    0 to 1)

surf = axes.plot(coorx, coorz, 'r-')
plt.xlabel("theta")
plt.ylabel("r")

plt.show()
```

3-D Sweep - Arduino

```
/* Sweep across cardboard letter 3-D
 *  POE Lab 2 - DIY 3-D Scanner
 *  Emily Lepert and Gracey Wilson
*/

#include <Servo.h>

Servo servo1;          //create 1st servo object to control a servo
Servo servo2;       //create 2nd servo object to control 2nd servo

int pos1 = 0;          //initialize variable to store 1st servo position
int pos2 = 0;          //initialize variable to store 2nd servo position
int hdeg1 = 70;       // highest degree for servo1
int ldeg1 = 10;       // lowest degree for servo1
int ldeg2 = 0;        // leftmost degree for servo2
int rdeg2 = 70;       // rightmost degree for servo2

const int analogInPin = A0;  // Analog input pin that the potentiometer is
    attached to

int sensorValue = 0;        // value read from the sensor

void setup() {
  servo1.attach(9);   //attaches servo1 on pin 9 to the servo object
  servo2.attach(8);   //attaches servo2 on pin 8 to the servo object
  Serial.begin(9600);
  scan();
  Serial.println("done");
}

void scan(){
  for (pos1 = ldeg1; pos1 <= hdeg1; pos1 +=1) { //servo1 moves from 0 to deg
      degrees in steps on 1 degree
    servo1.write(pos1); //servo1 moves to pos1 position
    delay(15);                          //wait 15ms for servo to reach
        position
```

```
34
35      for (pos2 =ldeg2; pos2 <= rdeg2; pos2 +=1){
36        servo2.write(pos2);
37        sensorValue = analogRead(analogInPin);    // read the analog in value
38
39        // print the results to the Serial Monitor:
40        Serial.print(sensorValue);
41        Serial.print(";");
42        Serial.print(pos1);
43        Serial.print("/");
44        Serial.print(pos2);
45        Serial.println(":");
46
47        // wait 2 milliseconds before the next loop for the analog-to-digital
48        // converter to settle after the last reading:
49        delay(15);
50      }
51    }
52
53 }
54
55 void loop() {
56 }
```

3-D Sweep - Python

```
1  #!/usr/local/bin/python
2
3  from serial import Serial, SerialException
4  import numpy as np
5
6  import matplotlib.pyplot as plt
7
8  # The Serial constructor will take a different first argument on
9  # Windows. The first argument on Windows will likely be of the form
10 # 'COMX' where 'X' is a number like 3,4,5 etc.
11 # Eg.cxn = Serial('COM5', baudrate=9600
12 #
13 # NOTE: You won't be able to program your Arduino or run the Serial
14 # Monitor while the Python script is running.
15 cxn = Serial('/dev/tty.usbmodem1421', baudrate=9600)
16
17 coorz = []
18 coorx = []
19 coory = []
20 running = True
21 threshold = 16
22
23 while running == True:
24     cxn.write([1])
25
26     result = str(cxn.readline())
27
28     print(result)
```

```
    if result[2:-5] == 'done':
        running = False

    else:

        start_pos1 = result.index(';')
        start_pos2 = result.index('/')
        end_pos2 = result.index(':')

        r = int(result[2:start_pos1])
        theta = int(result[start_pos2+1:end_pos2])
        phi = int(result[start_pos1+1:start_pos2])
        r = 112.1425508814386 - 1.0035935392387034*r + 0.00405508173593060*r
            **2 - 7.416839407985343*10**(-6)*r**3 + 4.934357177555067*10**(-9)
            *r**4

        x = r*np.cos(theta)*np.sin(phi)
        y = r*np.sin(theta)*np.sin(phi)
        z = r*np.cos(phi)

        if r < threshold:
            coorx.append(x)
            coory.append(y)
            coorz.append(z)


fig = plt.figure()
ax = fig.gca(projection='3d')

surf = ax.scatter(coorx, coory, coorz)

ax.set_xlabel('r')
ax.set_ylabel('Theta')
ax.set_zlabel('Phi')

plt.show()
```