PoE Lab 2: 3D Scanner

Evan Cusato and Benjamin Ziemann

December 27, 2018

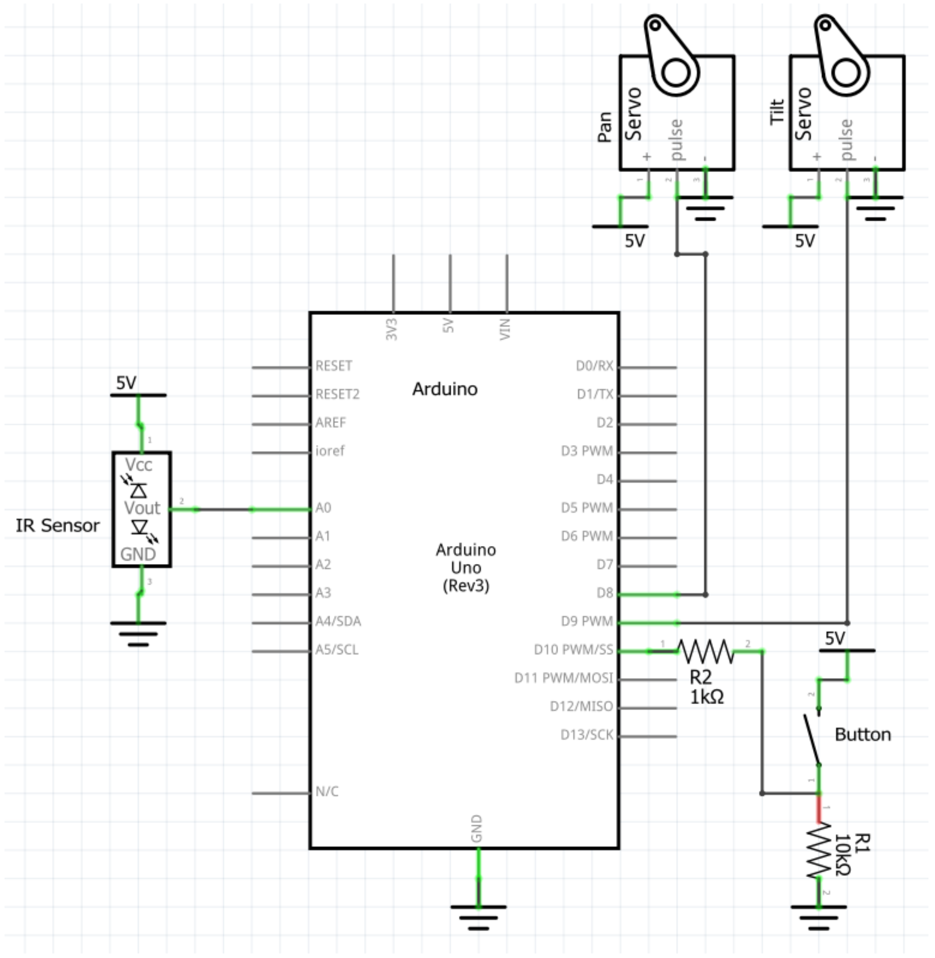

Figure 1: Final Circuit diagram–Two servos and IR sensor attached to the Arduino with a button to manually initiate the scan.

1 Part 1: Setup

1.1 Hardware Testing

We tested the servos by connecting the power wire to the Arduino's 5 volt pin, the ground wire to the Arduino ground pin, and the pulse wire into one of the PWM capable digital IO pins, as seen in figure 1. We then used the example ”sweep” sketch to ensure we had it wired correctly. After that we experiemented with different Servo.write() positions to find an appropriate sweep range on both the panning and tilting servos for an object the size of the letter we were using. The resulting range can be seen in our final Arduino sketch in Appendix 1.

We tested the distance sensor by hooking it up to the Arduino as shown in Figure 1. The values passed to the analog input were displayed in the serial monitor. We placed a sheet at regular distances from the sensor and monitored the output. It was consistent with what we expected to see as the output voltage.

1.2 Distance Sensor Calibration

Using the sensor system depicted in Figure 1 and described above, we recorded the voltage output at increments of 10 cm in the sensor's effective range of 20cm-150cm. The data points matched the values listed in the sensor data sheet. Since the scanner would not need to work effectively at a range greater than 70cm, we only used the points between 20cm and 70cm for the calibration curve depicted in Figure 2. To find the calibration curve, we took the calibration data points and used the Matlab fit command to find an accurate function, the exponential function listed below. The implementation of this command is depicted below.

$$f(x) = ae^{bx}$$

$$a = 134.1$$

$$b = -0.003899$$

We then plotted this curve with a data set of random distance and output to find our error graph, depicted in Figure 3. The values generated by the sensor do not correspond directly to an exponential function over the sensor's entire operational range (we did try a calibration curve of all the calibration data points, which was less accurate), so we optimized our sensor calibration for values below 70cm, which was the range of the backdrop we would be scanning. Therefore, the curve becomes less and less representative the if the distance is significantly beyond 70cm.

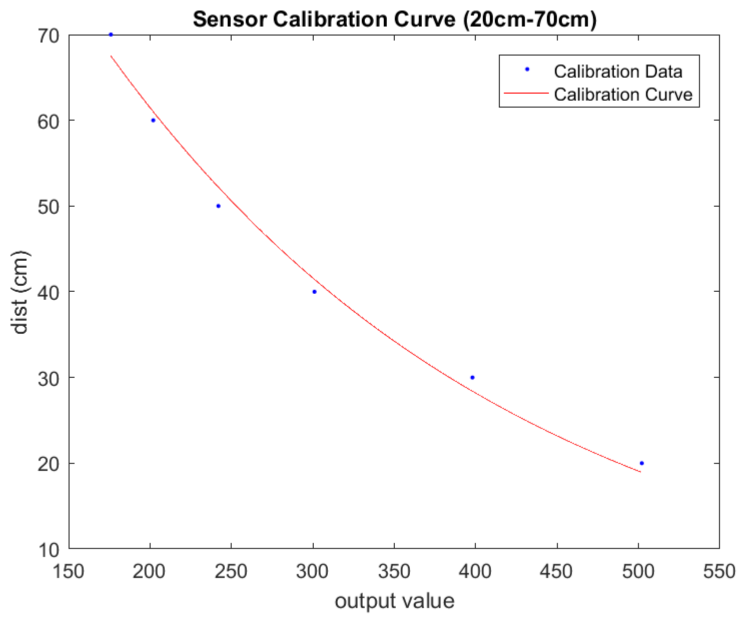

Figure 2: Sensor Calibration Curve taken for values relevant to scan (20cm-70cm).

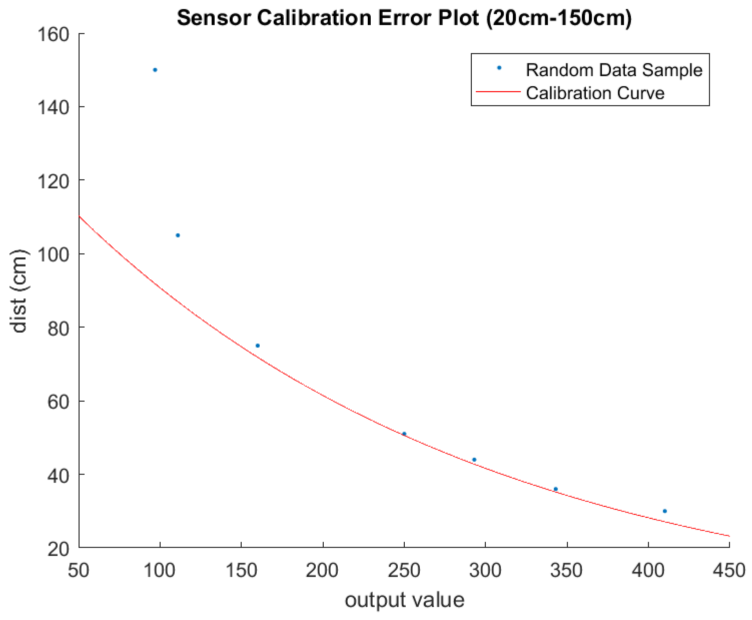

Figure 3: Sensor Calibration Curve plotted with random sample of distances (20cm-150cm).

1.2.1 Fit Command Implementation & Error Graph Generation:

```
y = Distance1; %Distances data was recorded at
x = AnalogRead1; %Voltages recorded
f = fit(x,y,'exp1') %Fit command implemented on an exponential curve
plot(f,x,y)

title('Sensor Calibration Curve (20cm-70cm)');
ylabel('dist (cm)');
xlabel('output value');
legend('Calibration Data','Calibration Curve');

a =       134.1; %Values of coefficients in calibration curve
b =    -0.003899;
fx = a*exp(b*AnalogRead2) %Calibration Curve
figure;

Dist=[30, 36, 44,51,75,105,150] %Random Distance values and corresponding voltages
Val=[410, 343, 293,250,160,111,97]

scatter(Val,Dist,'.') %Error graph
hold on;
plot(f)  %Calibration Curve

title('Sensor Calibration Error Plot (20cm-150cm)');
ylabel('dist (cm)');
xlabel('output value');
legend('Random Data Sample','Calibration Curve');
```

Code Sample 1: Matlab Calibration Curve Fitting

1.3 Mount Construction

We designed the the two horn-mounted servo components with SOLIDWORKS and 3D printed them to the tolerances necessary for our purposes. Given our design and that the steps listed in the lab document were "suggested steps," we decided to not assemble a one-servo scanner and moved straight to the final two-servo hardware design, depicted in Figure 4.

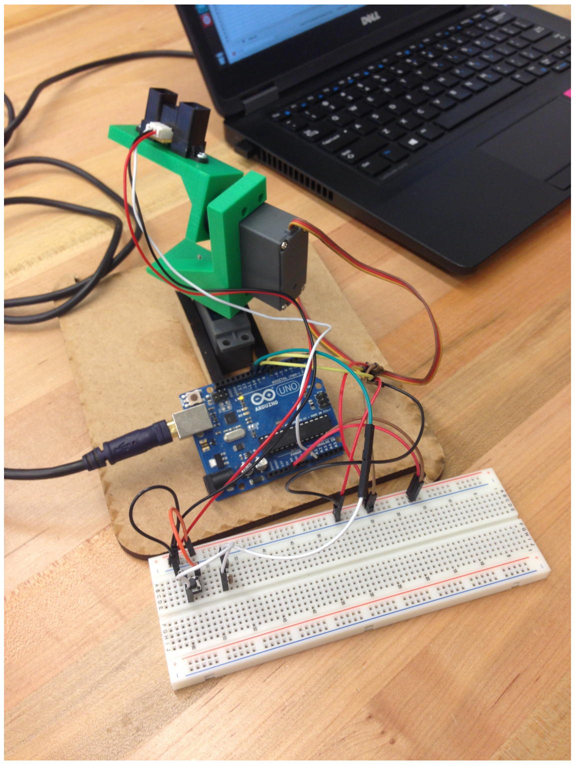

Figure 4: Our completed sensor mount.

1.4 Data Collection System

Our full code files can be seen in the appendix. It is broken down into a few key components across the files.

The Arduino sketch, which can be seen in full in appendix 1, is used to take in IR sensor data and relay that via serial to our python script running on the computer. The key components of the sketch are the double for loop used to sweep both servos and the actual sending of the data.

```

if(digitalRead(button)){
    //Start command to python script
    Serial.println("Start");

    for(posY = 30; posY <= servoDegY; posY += servoDegY/resolutionY){
      tilt.write(posY);

      for (posX = 0; posX <= servoDegX; posX += servoDegX/resolutionX) {
        pan.write(posX);
        sendData(posY,posX);
        delay(100);
      }
    }

    //Stop commadn to python script
    Serial.println("Stop");

    //Reset position for next run
    tilt.write(30);
    pan.write(0);
  }
```

Code Sample 2: Arudino double for loop in main loop() to sweep both pan and tilt servos

In the main loop of the program there is a constant check for a button press to begin the process of scanning. This time the button does not need a debounce as the process of scanning takes about half a minute so there is no fear of multiple inputs for a reasonable human press. This sends a start command to the python script via serial. The outer loop causes the pan servo to move a single degree while the inner loop sweeps the tilting servo, creating an array of points that are sent to the python script, along with the their associated distance in the next function, sendData(). At the end of these loops a stop command is sent so the python script knows when to stop looking for serial data.

```
void sendData(int x, int y){
    //Take three distances and average them
    int take1 = analogRead(IR);
    delay(5);
    int take2 = analogRead(IR);
    delay(5);
    int take3 = analogRead(IR);
    float avg = (take1+take2+take3)/3.0;

    //Send to python, let it do distance calculations
    Serial.println(String(x) + "&" + String(y) + "&" + String(avg));

}
```

Code Sample 3: Arduino sendData() function to write to serial

The sendData() function takes an average of three distances at the given x,y position. We take an average to ensure the data is good and it is not affected by the movement of the servos. We then send that data in the form x&y&distance to the python script via serial communication. We had it take this form because it is easy to parse out the values later in the python script.

The python script, seen in full in appendix 2, takes care of the parsing, conversion, and saving of the data received from the Arduino.

```
def getAxis():
    """
    Breaks down serial output and writes to CSV
    """

    #Setup
    output = ""
    axis = []
    currAxis = ''

    while(True):
        #Output takes the form x&y&analogValue
        output = ser.readline().strip()
        if output == "Stop":
            break

        #Break output into easy to use list
        axis = output.split("&")

        distance = calcDistance(axis[2])

        #IR value limits to eliminate noise
        #Background
      if distance > 60:
        distance = 60
        #Too close to function
      elif distance < 0:
        distance = 0

        #Write to csv
        f.write(axis[0] + ',' + axis[1] + ',' + str(distance) + "\n")

        #Reset
        axis =[]
```

Code Sample 4: Python script parsing the received Arduino data P

The getAxis() function takes in data via the serial port and then parses it. Because of the string we sent from the Arduino, we can use Python's convenient split() function to use the s as marker as to where are individual values begin. The raw IR value is then converted into a distance in centimeters, as described in Code Sample 1. A limiting range is than placed on this distance to filter out any noise (see Obstacles section 2.5 for more information) before all three values are written to a line in a .csv file for later use by Matlab. This process then repeats until a stop command is received in the serial port, indicating the scan is done, breaking the while loop and ending this function.

```
if __name__ == "__main__":
  f = open("dataLab2.csv",'a')
    #Clear any existing data
    f.seek(0)
    f.truncate()
    #Header
    f.write("x" + ',' + "y" + ',' + "dist" + "\n")

    #Wait for button press / start command
    while(True):
        out = ser.readline().strip()
        if out == "Start":
            break

    getAxis()
    f.close()
```

Code Sample 5: Python script of .csv file setup and start of scanning

The python script, when called, runs some preliminary setup of the .csv file we will write to. If the file already exists it clears it out and adds a convenient header. It then waits for the start command to begin reading data. After the .csv file has been populated, the data is then moved over to Matlab and processed.

2 Part 2: Recording Data

2.1 Our Letter & General Setup

We used the negative of a "Z" so we could raise the bottom of the letter off the tabletop for more accurate readings. In all cases, the letter is positioned at 30cm and the backdrop is positioned at 60cm. We set distances greater than 60cm to 60 cm. The value spiked above 60cm because the scanner would occasionally sweep outside the backdrop.

2.2 Single Servo

For the single servo test, we decided to use a scatter plot and have the tilt servo sweep through a 30°arc. The result is displayed in Figure 5. The two drops to 60cm near 25°and 10°indicate areas where the letter was cut out of the sheet and the signal bounced off the backdrop instead of the front panel. The spikes to 30cm at 30°and 18°indicate points where the signal was returned from the sheet the letter was cut from. The intermediate values are places where the scanner was receiving returns from both the front sheet and backdrop in various ratios. At the bottom most value, the sensor picked up the tabletop.

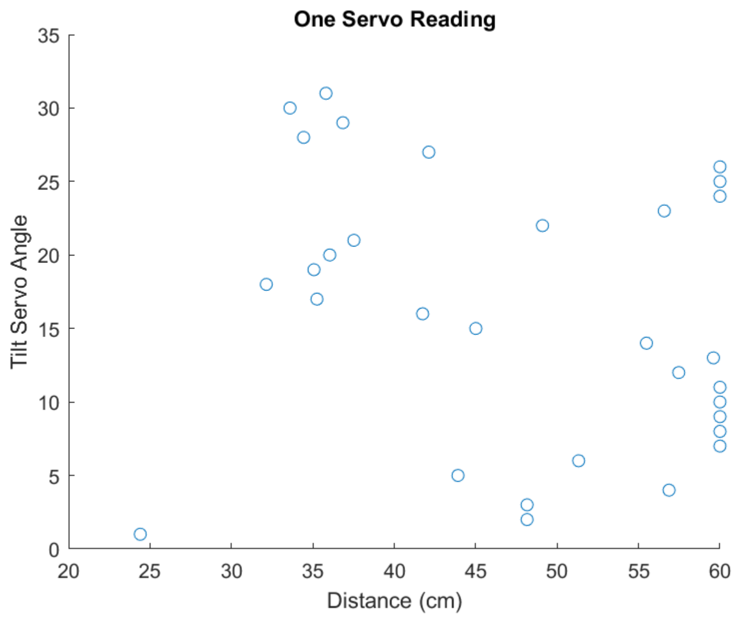

Figure 5: Scatter plot of single servo distance sweep.

2.3 Double Servo

The double servo configuration allowed us to sweep a 2D space and scan our whole object. We programmed the Arduino to take a full tilt sweep for each step of the pan servo. A 3D scatter plot of the location and distance values is displayed in Figure 6. This was an excellent data validation tool, because we had a number of problems with out heat map. Since we used a negative of our letter, the raised sections of our graph are where the signal returned off the backboard and the valleys around the letter outline are where the signal returned from the board the letter was cut from. The dense stream of points on the lower right section is a result of the plot prospective and is actually the sensor registering the backdrop over the edge of the sheet the negative was cut from.

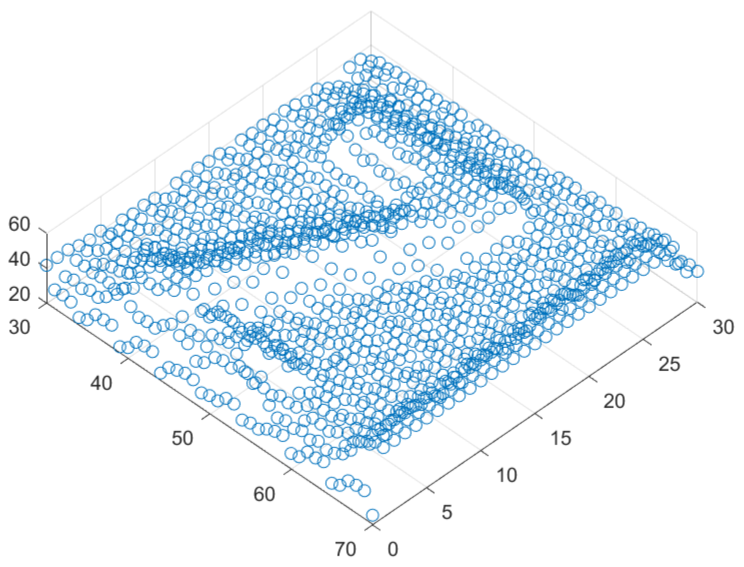

Figure 6: 3D Scatter Plot of pan and tilt servo positions and distance readings.

2.4 Data Processing

The Arudino passed the x, y, and distance values to a python script running on a laptop. Given that we we only send a single string per data point, we parsed that string by breaking it up with key characters. After parsing the raw values we put it through our equation from section 1.2 to calculate a distance in centimeters. Each of these data points is then written into a .csv file.

After the python code generated a .csv file, we input the file into Matlab and used it to generate a heat map, like the one displayed in Figure 7, which represents our final scan. In order to generate the heat map, the distance data had to be reformatted from a vector to a matrix of dimensions x by y in order to be used by the heat map. We also had to rescale the heat map color scale to clearly display the distance.

```
mat6=vec2mat(dist10, 31); %Take distance vector and turn into matrix of x by y
    dimensions (41x31)
mat6t=transpose(mat6); %Transpose matricies to reorient letters
hm = HeatMap(mat5t); %Default heat map plot

ax = hm.plot; % 'ax' will be a handle to a standard MATLAB axes.
colorbar('Peer', ax); % Turn the colorbar on
caxis(ax, [0 60]); % Adjust the color limits
%Output: heat map with color limits from 0 to 60
```

Code Sample 6: Heat Map Creation

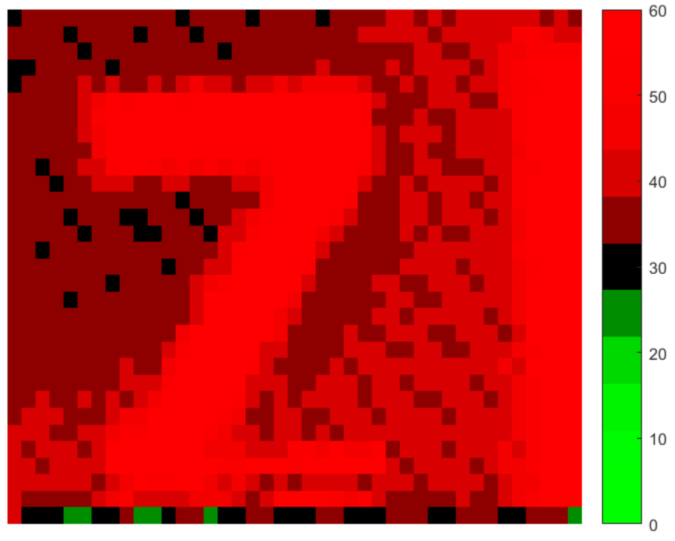

Figure 7: Final heat map of two servo scan

2.5 Obstacles

```
#Break output into easy to use list
        axis = output.split("&")

        distance = calcDistance(axis[2])

        #IR value limits to eliminate noise
        #Background
      if distance > 60:
        distance = 60
        #Too close to function
      elif distance < 0:
        distance = 0

        #Write to csv
        f.write(axis[0] + ',' + axis[1] + ',' + str(distance) + "\n")
```

Code Sample 7: Distance limiting to eliminate noise

One obstacle we had was dealing with the noise due to our function not fitting our calibration data at several points. Knowing the general range our object would be away from the scanner, we had it the code ignore anything registered to be greater than 60cm away. Additionally if the sensor were to see the table below it, with our function it would produce very negative numbers, skewing the range. Because of this we also set a lower limit of 0.

```python
f = open("dataLab2.csv",'a')
#Clear any existing data
f.seek(0)
f.truncate()
#Header
f.write("x" + ',' + "y" + ',' + "dist" + "\n")

#Wait for button press / start command
while(True):
    out = ser.readline().strip()
    if out == "Start":
        break

getAxis()

...


while(True):
        #Output takes the form x&y&analogValue
        output = ser.readline().strip()
        if output == "Stop":
            break
```

Code Sample 8: Python script start and stop checks to know when to read from the serial port

Another problem we ran into was when to begin and end taking data. To combat this we added a start and stop command through serial communication that the code looks for to actually begin taking data. These commands are then told to be sent through a physical button press by the user, seen in the schematic of figure 1. For a while this process had us stumped until we realized each use of the "readLine()" function "consumed" a line of the serial output. Because we had several print statements that utilized this function, we found ourselves missing data points and these stop and start points. The way around this was to simply read the line once, set it as a variable, then perform all necessary checks on that variable before reading from the serial port again.

2.6 Possible Next Steps

If we were to take this project further something we could improve is fitting a function through a larger range of our calibration points. Knowing our object would be between 20cm and 60cm from our scanner, we built our function around that range. Finding a function with a better fit to all data points would give our scanner a better range so it could more accurately scan more complex and larger objects. One way this could be accomplished is by creating a piecewise calibration function. The data processing code would include an if statement to indicate if the sensor output is greater than or below a certain value. Based on that question, an equation would be selected which would approximate that section of the curve.

Additionally we could add a rotating base to our object and update our code accordingly, allowing us to fully visualize objects in 3D. This would involve additional hardware, possibly a stepper motor or continuous rotation servo so as to keep track of the base's position as well as finding a function in either matplotlib or Matlab to plot it.

3 Reflection

3.1 Ben

I was really glad with the outcome of the project as well as the continued trend of these projects having easily recognizable, real life applications. I especially enjoyed this project because of the integration of multiple systems and programming languages, each with their own specialties and hope this continues. It helps me learn many new aspects of mechanical, electrical, and software, gives a feeling of legitimacy to the project while also inspiring me to think about other projects I could do with these fairly easily accessible and cheap parts. Finally I'm also very interested in the further applications of this and am considering taking it further in my own time.

3.2 Evan

I am excited to be expanding the utility of the Arduino beyond what can be accomplished with the hardware and IDE alone. I am beginning to understand how powerful a tool it can be when combined with other programming languages. My goal is to concretely understand all aspects of these labs and be confident in my ability to use the skills required to complete them in the future. That being said, I have also never programmed in Python before and a portion of this lab requires Python code implementation to take the values from the Arduino and convert them into a .csv file which can be used by Matlab. My inability to fully understand and contribute to this section of the project was frustrating. Since my partner is an experienced Python user it was not an impediment to completing the lab, but it has been an impediment to my understanding of the lab. Nevertheless, I still feel my skills with microcontrollers and accompanying software are improving, and I am still extremely interested in learning more about this area of engineering.

4 Appendix

4.1 Appendix 1: Arduino Sketch

```
#include <Servo.h>

//Tilt pan numbers
int resolutionX = 30; //How many divisions per axis
int servoDegX = 30; //Degrees servo can rotate through

int resolutionY = 70; //How many divisions per axis
int servoDegY = 70; //Degrees servo can rotate through

//Start positions
int posX = 0;
int posY = 30;

//Sensor port
int IR = 0;

//Servo objects
Servo pan;
Servo tilt;

//button pin
```

```
int button = 10;

void setup() {
  // Make sure the baud rate is the same as in sendReceive.py
  Serial.begin(9600);

  //Setup servos
  pan.attach(8);
  tilt.attach(9);
  pan.write(posX);
  tilt.write(30);

  //Setup button
  pinMode(button, INPUT);
}

//Sends angle and distance data to python
void sendData(int x, int y){
    //Take three distances and average them
    int take1 = analogRead(IR);
    delay(5);
    int take2 = analogRead(IR);
    delay(5);
    int take3 = analogRead(IR);
    float avg = (take1+take2+take3)/3.0;

    //Send to python, let it do distance calculations
    Serial.println(String(x) + "&" + String(y) + "&" + String(avg));

}

void loop() {
  if(digitalRead(button)){
    //Start command to python script
    Serial.println("Start");

    for(posY = 30; posY <= servoDegY; posY += servoDegY/resolutionY){
      tilt.write(posY);

      for (posX = 0; posX <= servoDegX; posX += servoDegX/resolutionX) {
        pan.write(posX);
        sendData(posY,posX);
        delay(100);
      }
    }

    //Stop commadn to python script
    Serial.println("Stop");

    //Reset position for next run
    tilt.write(30);
    pan.write(0);
  }

}
```

4.2 Appendix 2: Python Script

```
"""
PoE Lab 2 - 3D Scanner
Evan Cusato and Benjamin Ziemann

Recieves position and IR values from serial, converts to distance,
and writes it into CSV
"""

import serial
import binascii
import math

#Read from serial port
ser = serial.Serial('/dev//ttyACM0',9600)

def getAxis():
    """
    Breaks down serial output and writes to CSV
    """

    #Setup
    output = ""
    axis = []
    currAxis = ''

    while(True):
        #Output takes the form x&y&analogValue
        output = ser.readline().strip()
        if output == "Stop":
            break

        #Break output into easy to use list
        axis = output.split("&")

        distance = calcDistance(axis[2])

        #IR value limits to eliminate noise
        #Background
      if distance > 60:
        distance = 60
        #Too close to function
      elif distance < 0:
        distance = 0

        #Write to csv
        f.write(axis[0] + ',' + axis[1] + ',' + str(distance) + "\n")

        #Reset
        axis =[]

def calcDistance(aVal):
    """
    Rough mathematical function to fit IR values to calibration data
    """
    aVal = float(aVal)
    a = 134.1
    b = -.003899
    calced = a*math.exp(b*aVal)
    return calced

if __name__ == "__main__":
    f = open("dataLab2.csv",'a')
    #Clear any existing data
    f.seek(0)
    f.truncate()
    #Header
    f.write("x" + ',' + "y" + ',' + "dist" + "\n")

    #Wait for button press / start command
    while(True):
        out = ser.readline().strip()
        if out == "Start":
            break

    getAxis()
    f.close()
```

4.3 Appendix 3: MatLab Script

```
mat4=vec2mat(dist9 , 31); %Take distance vector and turn into matrix of x by y
    dimensions (41x31)
mat5=vec2mat(VarName3, 31);
mat6=vec2mat(dist10 , 31);

mat5t = transpose(mat5); %Transpose matricies to reorient letters
mat6t=transpose(mat6);



hm = HeatMap(mat6t); %Default heat map plot

ax = hm.plot; % 'ax' will be a handle to a standard MATLAB axes.
colorbar('Peer', ax); % Turn the colorbar on
caxis(ax, [0 60]); % Adjust the color limits
%Output: heat map with color limits from 0 to 60

figure;
scatter(mat6t(:,15),1:31); %Single Servo Plot
xlabel('Distance (cm)');
ylabel('Tilt Servo Angle');
title('One Servo Reading')

figure;
scatter3(x9,y9,dist10);
```