Mini Project 2

Jonas Kazlauskas, Ruby Eisenbud
PIE Section 2
September 30, 2021

Abstract

Using two servo motors and an infrared distance sensor, our goal was to scan a cardboard letter K and plot points to recreate its shape in a 3D plotter. We successfully calibrated the infrared distance sensor and determined an equation to approximate the relationship between the sensor values and the distance from the object in reality. We then designated one servo motor as Tilt and the other as Pan, and designed a mounting setup where the axes of rotation of the servo motors were stacked vertically, allowing for simpler distance calculations in our code. In Arduino, we wrote a program to control the angle of Tilt and Pan to collect distance data across the entire area of our cardboard letter. These values were sent using serial to a Python program, which converted them to cartesian coordinates and plotted the results. We also designed a base to hold the servo setup upright, and used recycled 3D prints as a base for the letter. We explain the details of our process and results in the sections below.

Methodology

Sensor Calibration

We began by calibrating our infrared sensor by taking readings from a known distance between 20-130 cm. We chose this range to fit within the rated range of the sensor, as well as the distance we expected to place our letter when scanning. For each known distance, we took the average of three readings from the sensor and plotted it as shown in Figure 1. After plotting each point, we found an equation to fit the data. We found that $y=7605x^{-0.868}$ fit the data well, where x is the analog voltage reading and y is the actual distance in cm. We tested a few possible fit lines, and found an exponential equation to fit best with an R^2 value of 0.998. This equation allows us to record an analog voltage and map it onto an equivalent centimeter distance.

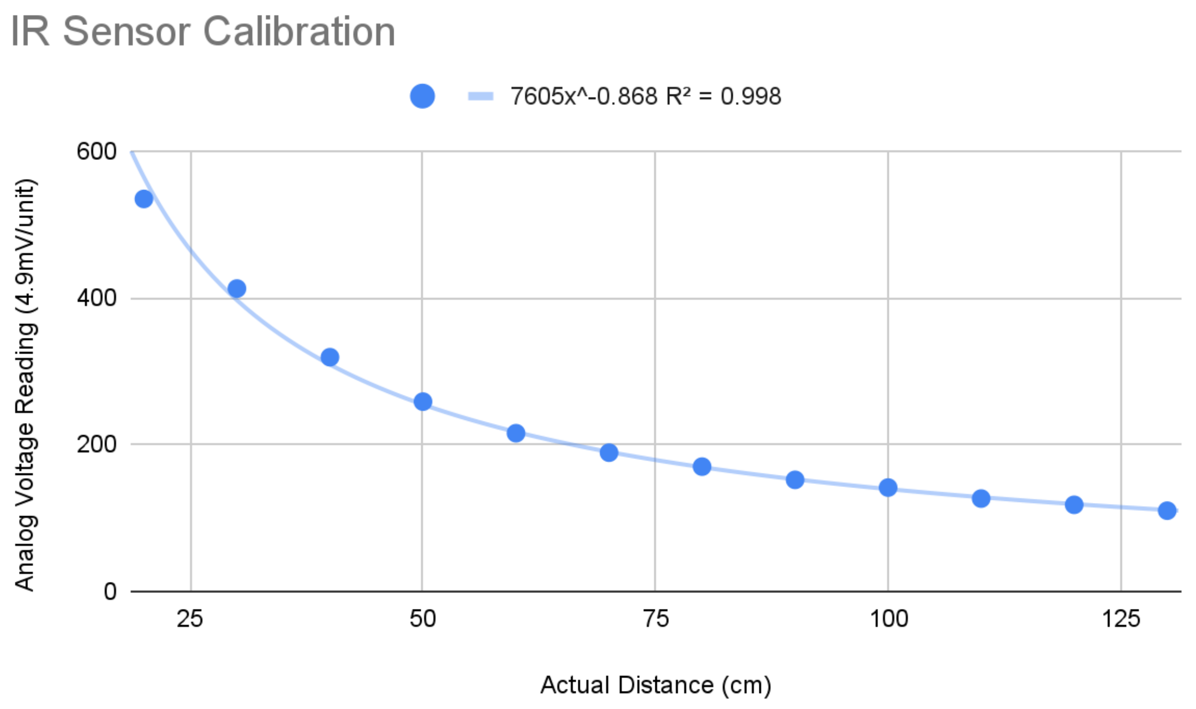

Figure 1. Calibration data points plotted with a line of best fit. The equation for the line is shown above, and has an R^2 value of 0.998.

Next, we validated our calibration curve equation from above by taking new sensor readings from known distances and recording the average of three readings. We then plugged in the known distances to our calibration equation to find our predicted analog voltage reading. Lastly, we subtracted the actual sensor readings from our prediction to find the error which is shown in Figure 2. We observed errors in both the positive and negative directions. This suggested that our error was random and not systematic. However, we also see a grouping in the 30-100 range that trends on the positive error side. This points to a potential issue with our calibration, and requires further investigation.

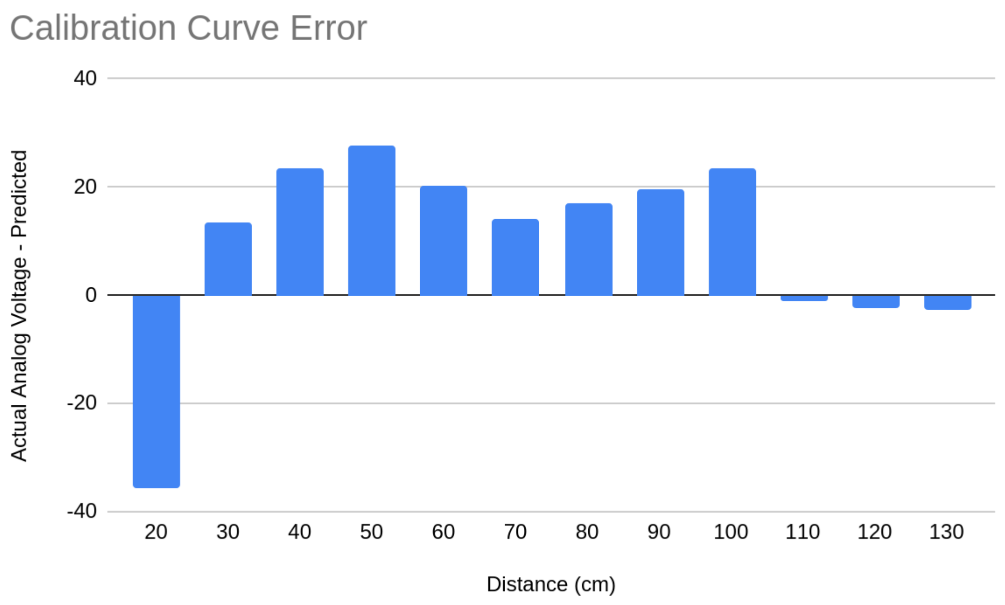

Figure 2. The error between our sensor data and predicted analog voltage reading from our calibration curve.

The equation shared above and included in our code was our second calibration equation. Our first equation was a quadratic that had a lower R^2 value than the current equation. It also showed higher consistent errors that were visible in our 3D plot. Because of this error, we repeated our sensor calibration with a more consistent setup, which yielded our current calibration equation shown above.

Pan-Tilt Mechanism Iterations

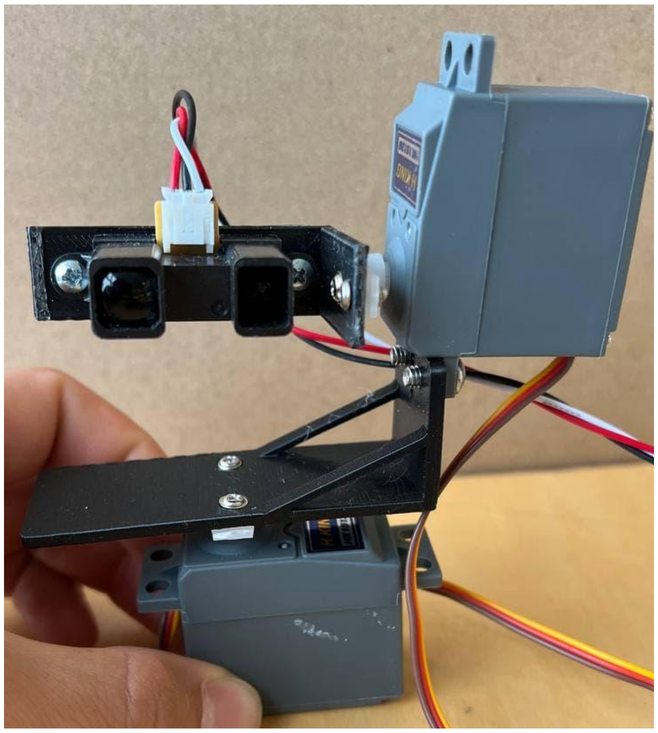
Figure 3. Pan tilt mechanism used for testing a pan scan of our letter. The bottom servo was the only servo that was run.

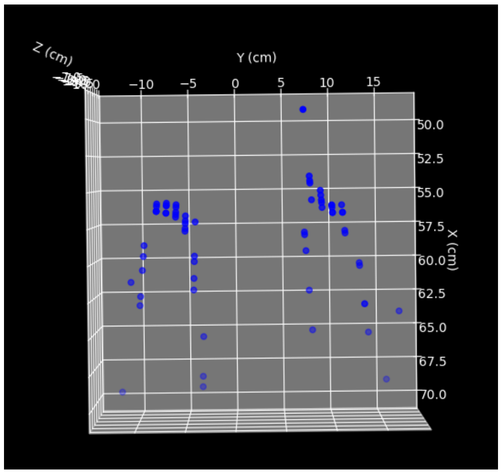

Figure 4. The top view of a scan using a single servo pan of our letter K. We can see there are two groupings of points when the scan is reading the legs of our letter K. There are some additional points in front of the letter from our sensor reading the floor.

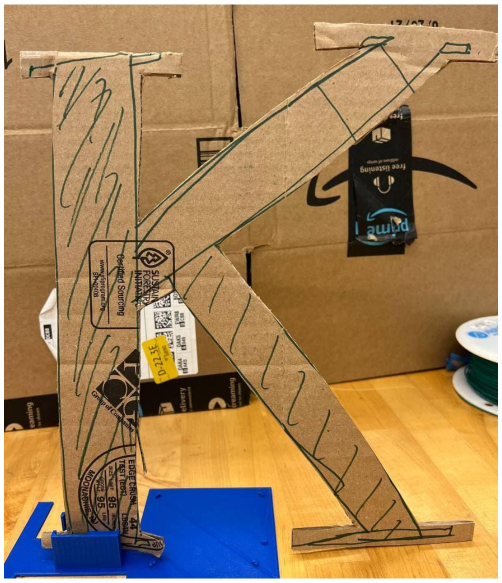

Figure 5. Our cardboard letter K.

Our first design of the pan tilt mechanism was created to minimize the amount of 3D printed material needed by stacking the servos. While this design was effective, the axes of rotation of the IR sensor were not aligned, which meant that any panning or tilting would also be translating the sensor. It is possible to account for this change using math, but we decided to redesign our mechanism to instead align the axes of rotation. Our final setup included this change, which can be seen in Figures 6 and 7. A close up of this mechanism is displayed in Figure 3.

Once we mounted the servos in the stacked configuration (final design), we found that the weight of the top servo made the setup unbalanced; it would not stay upright without us holding it. To keep the setup standing upright, we designed a base with a slot to hold the bottom servo. This design also used the breadboard and Arduino board as a counterweight, since the 3D printed servo slot alone would not be heavy enough to oppose the weight of the top servo. This design went through two printed iterations; we changed the design after the first print since our measurements did not properly leave room for the parts we wanted to attach to the base. The final design is shown in Figures 6 and 7.

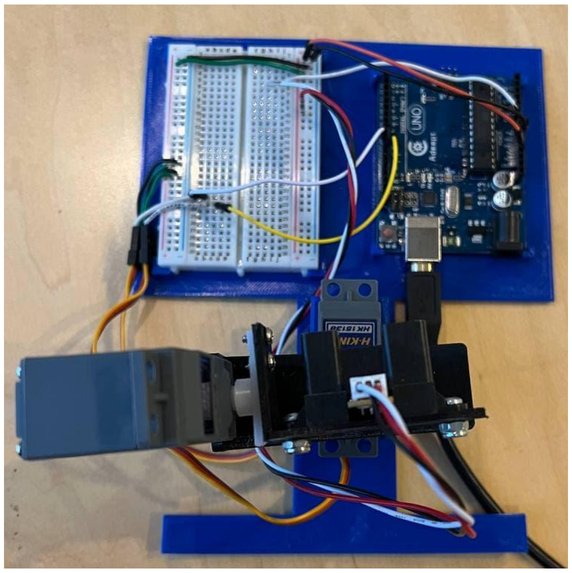

Figure 6. Top down view of our pan tilt mechanism. This shows the vertical alignment of rotational axes.

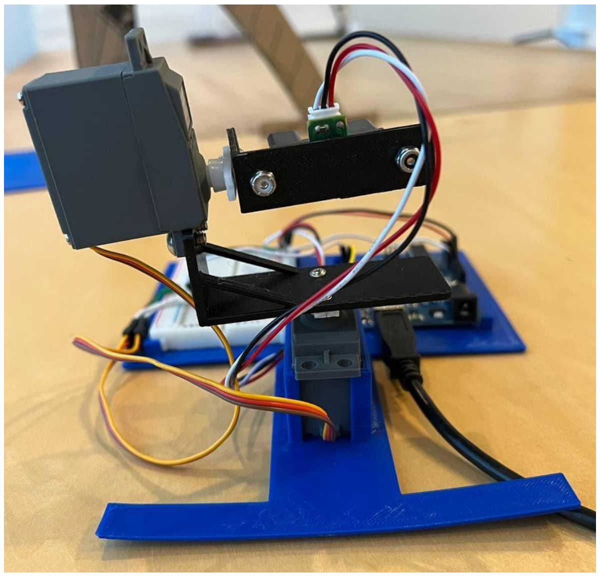

Figure 7. Back view of our pan tilt mechanism

Arduino -- Controlling servos & reading IR sensor

The first step in creating a 3D plot of our cardboard letter was scanning the letter and collecting distance data, which we did using Arduino. We wrote a program that gradually tilts up, pans a set step size to the side once it reaches a maximum tilt angle, gradually tilts down to a minimum tilt angle, pans by the same step size, and repeats until a maximum pan angle is reached. The sensor is read and the reading is sent via serial to our python code for every servo increment. We defined a function for each of these actions to easily call in the void loop (explained in detail below).

- setServoInit(): Sets the initial position for both the Pan and Tilt servos. Called in the void setup().

```
void setServoInit() {
  // Set servo initial positions
  panServo.write(startPan);
  delay(1000);
  tiltServo.write(startTilt);
  delay(3000);
}
```

- getReading(): Collects five consecutive readings from the infrared sensor pin and takes the average of those values. Returns the average of the values. Called within both the tiltUp() and tiltDown() functions.

```
float getReading() {
  // Get 5 sensor readings and return their average
  float rawRead1 = analogRead(sensorPin);
  float rawRead2 = analogRead(sensorPin);
  float rawRead3 = analogRead(sensorPin);
  float rawRead4 = analogRead(sensorPin);
  float rawRead5 = analogRead(sensorPin);
  return (rawRead1 + rawRead2 + rawRead3 + rawRead4 +
rawRead5)/5;
}
```

- sendReadings(): Takes infrared sensor reading and associated Pan and Tilt angles formats the readings. This method allows us to transfer the data from Arduino to Python to be used for 3D plotting. We call this function within both the tiltUp() and tiltDown() functions.

```
void sendReadings(int reading, int pan, int tilt) {
  // Send a sensor reading with pan and tilt over serial
  Serial.print(reading);      Serial.print(",");
  Serial.print(pan);          Serial.print(",");
  Serial.println(tilt);
}
```

- tiltUp(), tiltDown(): Both functions use a for loop to increment the tilt angle in the desired direction (either up or down). With each iteration of the for loop, the angle is increased or decreased by one step size (defined at the beginning of the code), the value is sent to the servo, there is a defined delay, and then the Arduino reads the infrared sensor pin using the getReading() function and sends it via serial using the sendReadings() function. The for loop ends when the desired value (endTilt for tiltUp() and startTilt for tiltDown()) is reached. We call these functions in void loop().

```
void tiltUp() {
  // Tilt sensor up, sending sensor readings at each position
  for (int pos=startTilt; pos<= startTilt + endTilt;
pos+=tiltStep) {
      tiltServo.write(pos*multiplier);
      delay(tiltDelay);
      sendReadings(getReading(), panPos, pos);
      }
}
```

```
void tiltDown() {
  // Tilt sensor down, sending sensor readings at each position
  for (int pos=startTilt + endTilt; pos>= startTilt;
pos-=tiltStep) {
      tiltServo.write(pos*multiplier);
      delay(tiltDelay);
      sendReadings(getReading(), panPos, pos);
      }
}
```

We used these functions in the void loop() to scan the letter. The servo alternated between tiltUp() and tiltDown(), panning by a set increment with each change of direction (up or down). Since the getReading() and sendReadings() functions are called in the tilt functions, distance data was regularly sent via serial.

Python -- Converts data sent from Arduino and creates 3D plot

In order to create a 3D plot of our letter, we took the data that was collected using the Arduino and sent it to a Python script using serial communication. We used a python library called PySerial to receive serial data from the Arduino. We also used the matplotlib library to visualize the data. After receiving the data, we convert it from spherical to cartesian coordinates, and save it in a text file. Lastly, the data is plotted live in a 3D scatter plot. We defined the following functions to perform these actions. We used the provided course example and examples regarding live plotting[1] with matplotlib to get started.

- convert_raw_reading(): Takes the distance data received from the serial port as an input and converts it into distances in centimeters using our calibration curve equation.
- calibrate_pan_tilt(): Takes the pan and tilt angle data received from the serial port as an input and calculates the difference between the actual angle and our defined axes. The function defines new pan and tilt angles as these differences, which are used in the calc_coords() function to convert from spherical to cartesian.
- calc_coords(): Since the distance detected by the infrared sensor is the radial distance from the sensor to each point on the letter based on the position of the pan and tilt angles, we need to convert from spherical to cartesian coordinates to be able to create a 3D plot of the letter. This function does this conversion and returns the distance to each point as (x, y, z) coordinates.

A second python script reads the data in the text file and plots the points that correspond to the letter (ignoring points that represent the wall or the floor). We successfully created this plot by comparing each coordinate to the allowable y and z values. While plotting, we noticed that large outliers would often skew our plot. This is due to our exponential calibration curve that occasionally places points 1000+ cm away in the X direction. To account for this, we included a

[1] https://pythonprogramming.net/live-graphs-matplotlib-tutorial/

filter to ignore any points with an X distance farther than 70 cm. We also used the same filtering to remove any points below a z value of -10 to prevent our plot from displaying the ground.

We confirmed that these values were reasonable by looking at our real life setup. In the code, if the coordinate is within the acceptable range, the point is added to the plot. Since data is being sent from the Arduino to Python via the serial port in real time, this script creates the plot as the servos scan the letter.

Results

With the sensor calibrated, mounts printed, and code operating as expected, we began scanning. Based on a tilt angle range of 30 degrees and the size of our letter, we determined an optimal distance to place our letter from the sensor. This distance is ~57 cm, which ensured that our sensor tilt would capture the letter completely.

After our initial scans, we noticed that objects appeared slanted in the x-direction. Since the objects were vertically upright and all distances in the horizontal direction were the same (perpendicular to the face of the letter), we recognized that there was an error somewhere in our reading, conversion, and/or plotting. We reviewed our mathematical conversion from spherical to cartesian coordinates, increased the delay time between each angle reading to give the servo time to move from one position to the next, and also recalibrated our infrared sensor. By plotting points from our calibration to generate a new calibration equation and using that to convert the raw sensor readings to distances, we saw less curvature in our 3D plot but still more than expected.

After talking about it with peers, we decided to follow their suggestion of visually determining the amount our angle needs to change to move 90 degrees. We found that 87 degrees moved our angle ~90 degrees, so we multiplied our calculated angle values by 87/90 to carry the proportion throughout our code. We tested this method by scanning a flat surface which is shown in Figure 8.

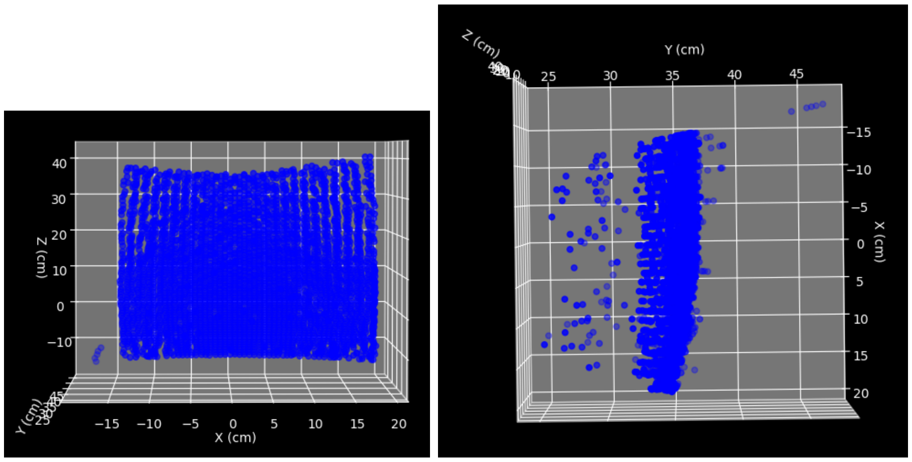

Figure 8. 3D scan of a flat piece of cardboard. While there is some error, curvature is minimized relative to initial results.

Using this new method, we scanned our letter and created a 3D plot. While it is still slanted in the x-direction, it is a major improvement from our first plot.

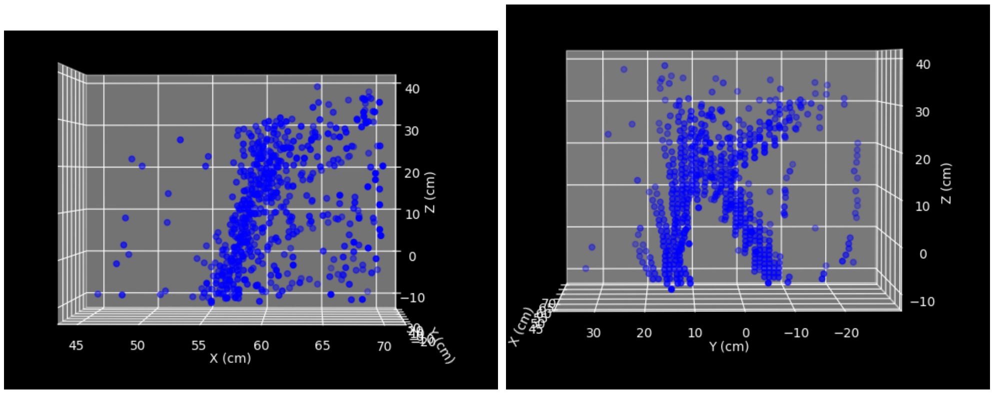

Figure 9. 3D plot in the yz plane shows points forming a letter K. Some points representing the floor that were not excluded are displayed as the line at ~Z=-5cm.

We recognize that there are possible sources of error in our calibration curve and scanning setup. For example, we could have had a more precise method for setting up the distance between our sensor setup and the letter, and aligning the y-axis of the scanner with the center of the letter. We could also improve our pan tilt mechanism to ensure that the sensor is stable during movement.

Full source code can be found here: https://github.com/jonaskaz/3D-Scanner

Reflections

Jonas: I really enjoyed this project. I felt like it was at a great level of difficulty in terms of stretching myself while utilizing skills I already had. I typically work on E:C tasks, but I wanted to expand my abilities on this project. I was able to CAD using onshape for the first time and create a few iterations of our pan tilt mechanism. The designs we had were relatively simple to CAD, but a great way for me to get familiar with using onshape and to work on my design/mechanical skills. 3D printing was a really great medium, as it allowed me to go through 4 iterations of the design very quickly. Working with Arduino and Python together was a good learning experience. I have worked a fair amount in Python, and a little bit with arduino and it was interesting to combine the two. This gave me an idea of their differences and strengths. I have not done any plotting in Python before, and was proud to get the live 3D plotting to work. In terms of the overall project timeline, I think we did a pretty good job. I tried to work on it a lot in the beginning, knowing that we would likely run into issues. This gave us a bit of time at the end, but I still hoped we would have worked more on it earlier. In the future I want to schedule more time as a group to work, rather than both working when we can.

Ruby: This project presented opportunities for me to practice my prototyping and design skills and be exposed to connecting hardware and software beyond Arduino. While I have more practice with rapid prototyping than coding, it has been a while since I used the 3D printer. I've been creating parts in Solidworks for Mechanical Design, but for this project I got to practice

CAD using OnShape (so we could collaborate and easily share files) and make that part real, giving me the chance to experience what worked and what didn't and iterate to make it better. In terms of the coding, ModSim captures most of my experience using Python. Adding comments to the code and asking Jonas questions to guarantee that I understood the logic of our code helped my understanding. Initially I was unsure of how we were going to do this project, but I can feel myself building up a comfort level with talking through technical challenges with peers and learning new concepts on the go. I'm enjoying the process of putting all the pieces together (software, electrical, and mechanical) to get a final setup that works.

Appendix A: Arduino code

```
#include <Servo.h>

Servo panServo;
Servo tiltServo;

int startPan = 80;
int panPos = startPan;
int startTilt = 35;
int endPan = 30;
int endTilt = 65;
int panStep = 1;
int tiltStep = 1;
int tiltDelay = 50;
int panDelay = 20;
int sensorPin = A0;
float multiplier = 0.96666;

void setServoInit() {
  // Set servo initial positions
  panServo.write(startPan);
  delay(1000);
  tiltServo.write(startTilt);
  delay(3000);
}

void setup() {
  panServo.attach(10);
  tiltServo.attach(9);
  Serial.begin(9600);
  setServoInit();
}
```

```
float getReading() {
  // Get 5 sensor readings and return their average
  float rawRead1 = analogRead(sensorPin);
  float rawRead2 = analogRead(sensorPin);
  float rawRead3 = analogRead(sensorPin);
  float rawRead4 = analogRead(sensorPin);
  float rawRead5 = analogRead(sensorPin);
  return (rawRead1 + rawRead2 + rawRead3 + rawRead4 + rawRead5)/5;
}


void sendReadings(int reading, int pan, int tilt) {
  // Send a sensor reading with pan and tilt over serial
  Serial.print(reading);      Serial.print(",");
  Serial.print(pan);          Serial.print(",");
  Serial.println(tilt);
}

void tiltUp() {
  // Tilt sensor up, sending sensor readings at each position
  for (int pos=startTilt; pos<= startTilt + endTilt; pos+=tiltStep) {
      tiltServo.write(pos*multiplier);
      delay(tiltDelay);
      sendReadings(getReading(), panPos, pos);
      }
}

void tiltDown() {
  // Tilt sensor down, sending sensor readings at each position
  for (int pos=startTilt + endTilt; pos>= startTilt; pos-=tiltStep) {
      tiltServo.write(pos*multiplier);
      delay(tiltDelay);
      sendReadings(getReading(), panPos, pos);
      }
}

void loop() {
  while (panPos - 2*panStep >= endPan) {
      // Pan the sensor while tilting up and down
      tiltUp();
      panPos-=panStep;
      panServo.write(panPos*multiplier);
      delay(panDelay);
      tiltDown();
      panPos-=panStep;
      panServo.write(panPos*multiplier);
      delay(panDelay);
  }
  panPos = startPan;
  setServoInit();
  delay(5000);
}
```

Appendix B: Python code

config.py

```
DATA_FILEPATH = "scanner/data/data.txt"
ZERO_TILT_DEGREES = 143
ZERO_PAN_DEGREES = 142
MAX_X_DISTANCE = 70
MIN_Z_DISTANCE = -10
```

read_data.py

```
import serial
import math

#Import global variables, including file location for data from serial
monitor and
#angle value where pan and tilt are zero for conversion from spherical to
cartesian coords.
from config import DATA_FILEPATH, ZERO_TILT_DEGREES, ZERO_PAN_DEGREES

arduinoComPort = "/dev/ttyACM0"

baudRate = 9600

serialPort = serial.Serial(arduinoComPort, baudRate, timeout=1)


def run():
      while True:
      lineOfData = serialPort.readline().decode("utf8")  #Read data from
serial port
```

```
      data = lineOfData.split(",")
      if len(data) == 3:  #If 3 data points are detected, clean and store
data
            clean_data = [int(d) for d in data]
            print(clean_data)
            coords = calc_coords(*clean_data)
            with open(DATA_FILEPATH, "a") as f:  #Send data to text file
                  f.write(coords)

def convert_raw_reading(raw_reading):
      return 7605*(raw_reading**-0.868)

def calibrate_pan_tilt(pan, tilt):
      pan = math.radians(ZERO_PAN_DEGREES-pan)
      tilt = math.radians(ZERO_TILT_DEGREES-tilt)
      return pan, tilt

#Convert from spherical to cartesian coordinates (sensor gives distance
from sensor to each point on letter,
#not distances in x, y, and z directions)
def calc_coords(raw_reading, pan, tilt):
      """
      Returns string in format X,Y,Z
      """
      distance = convert_raw_reading(raw_reading)
      pan, tilt = calibrate_pan_tilt(pan, tilt)
      z = math.cos(tilt) * distance
      r = math.sin(tilt) * distance
      x = math.sin(pan) * r
      y = math.cos(pan) * r
      return str(x) + "," + str(y) + "," + str(z) + "\n"

if __name__ == "__main__":
      run()
```

live_plot.py

```
import matplotlib.pyplot as plt  #Import relevant libraries
import matplotlib.animation as animation
from matplotlib import style

#Import global variables, including file location for data from serial
```

```
monitor and
#maximum acceptable x distance and minimum acceptable z distance
#(to eliminate data points representing the wall behind the letter and
floor below).
from config import DATA_FILEPATH, MAX_X_DISTANCE, MIN_Z_DISTANCE

#Defining 3D plot space
style.use("dark_background")
fig = plt.figure()
ax = plt.axes(projection="3d")

def animate(i):
      graph_data = open(DATA_FILEPATH, "r").read()
      lines = graph_data.split("\n")
      xs, ys, zs = [], [], []  #Sorting data to be read and plotted
      for line in lines:
      if len(line) > 1:
            x, y, z = line.split(",")

            if float(x) > MAX_X_DISTANCE:  #if statements to check if point
is within acceptable distance range
                  continue
            elif float(z) < MIN_Z_DISTANCE:
                  continue
            xs.append(float(x))
            ys.append(float(y))
            zs.append(float(z))
      ax.clear()
      ax.set_label("3D IR Scan")  #Labeling plot and axes
      ax.set_xlabel("X (cm)")
      ax.set_ylabel("Y (cm)")
      ax.set_zlabel("Z (cm)")
      ax.scatter3D(xs, ys, zs, color="blue")

#Plotting points representing letter as distance values are read from text
file
ani = animation.FuncAnimation(fig, animate, interval=1000)
plt.show()
```