Mini Project 2: DIY 3D Scanner

Lilo Heinrich & Yehya Albakri, PIE Section 1

Goal

In this project, we created a circuit that scans a stationary object placed in front of it using servo motors to actuate an Infrared (IR) sensor. To accomplish this, we created parts to mount the servos and IR sensor onto, constructed the circuit and scanner, and wrote python scripts and Arduino code to collect, save, and visualize data. For our final test, we placed a cardboard cutout of the letter “A” at a specified distance, performed a scan, and filtered our data by distance to see whether the letter was recognizable.

Construction

The construction of our 3D scanner is simple; it consists of a bracket that links the shafts of the servos orthogonally, and a base, rendered in Figures 1(a) and 1(b) respectively. Both parts were 3D printed for accuracy and reliability. The servos were mounted using screws to the horn, and the horns were screwed to the bracket.

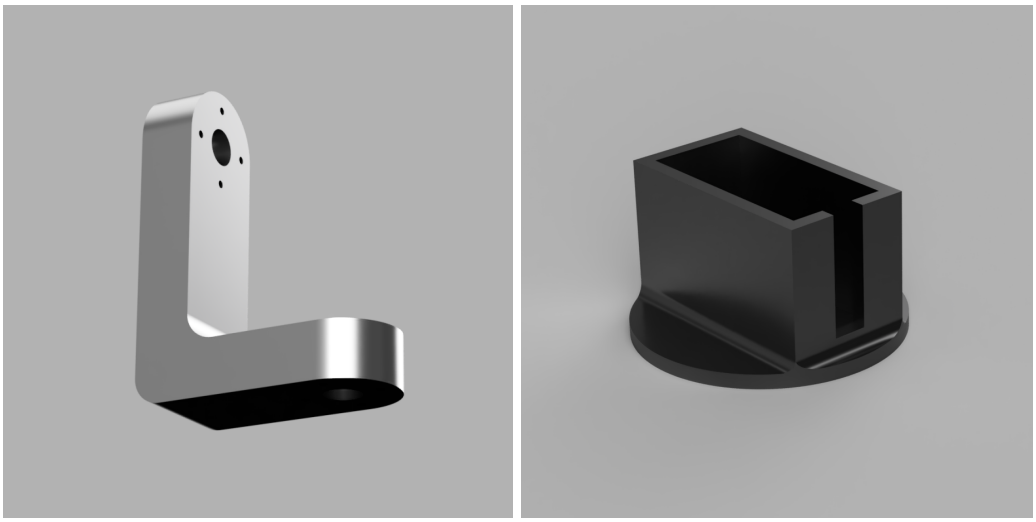

Figure 1: Render of Parts (a) Bracket (b) Base

We attached the IR sensor as close to the axis of rotation of the servos as we could, to keep the perspective of the IR sensor relatively constant and make data interpretation easier. However, there was still a deviation that we couldn’t account for since the sensor cannot fit directly on the axes of rotation. We used double-sided tape to mount the sensor directly on the servo to minimize the distance from the axes of rotation. Figure 2 shows a picture of the final scanner.

Figure 2: Photo of Final Scanner

Circuit

Shown in Figure 3, our circuit connects two servos to digital pins nine and ten and the IR sensor to analog pin zero. There is a pull-down resistor between the analog pin and ground to ensure that the voltage remains low by clearing any remaining charge in the wire. This pull-down resistor was probably not essential to the functioning of our circuit, but also not detrimental.

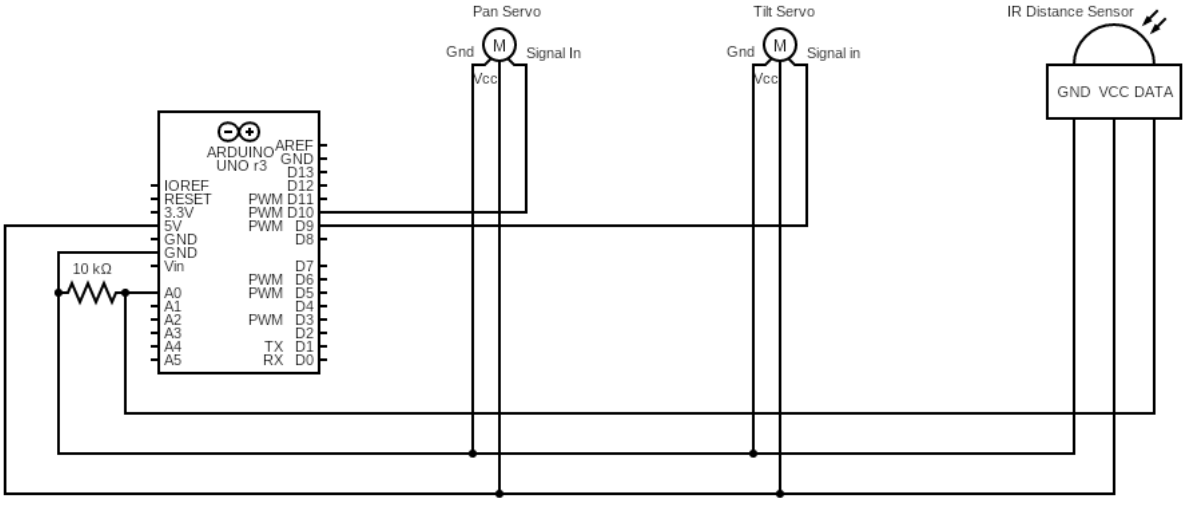

Figure 3: Final Circuit Diagram

Calibration

To make a calibration plot for our sensor, we taped the IR sensor and a measuring tape onto a work table, held up a large piece of cardboard in front of the sensor at different distances, and recorded the analog readings from the IR sensor off of the serial monitor. We took data at one inch increments from one to twenty-four inches, then two inch increments from twenty-six to sixty inches. The reason for increasing the interval of measurement was that the difference between readings was getting smaller as distance increased, and that our intended use range was twenty-four inches and below.

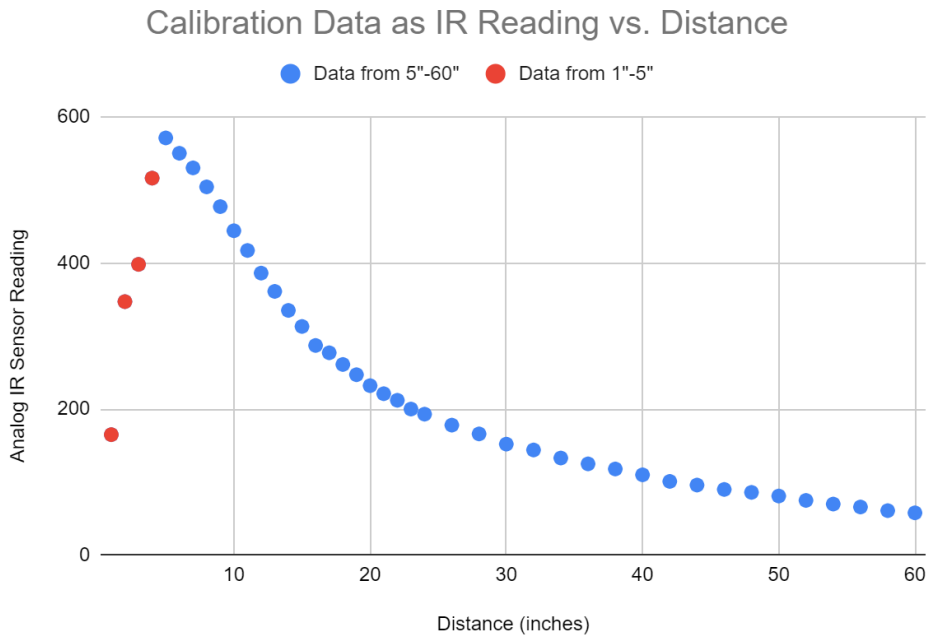

Figure 4: Calibration Data as IR Reading vs. Distance

Figure 4 shows the calibration data as IR sensor reading by distance in inches. Analog readings from the Arduino are given on a range from zero to 1023, and represent the analog voltage measured from zero to 5V. In our data collection, we found that for the data points below five inches, the IR sensor was reading values decreasing as we got closer to the sensor rather than increasing as in the rest of the data. Therefore, we concluded that five inches was near the edge of the functional range of the sensor and disincluded it from the data to train our calibration function. On Figure 4 the disincluded data points can be seen colored in red rather than the rest of the data shown in blue. Our finding about the minimum and maximum range of the IR sensor was further corroborated by the specifications sheet, which described a minimum distance measuring range of 20 cm (7.9 inches) and maximum range of 150 cm (59 inches) [1].

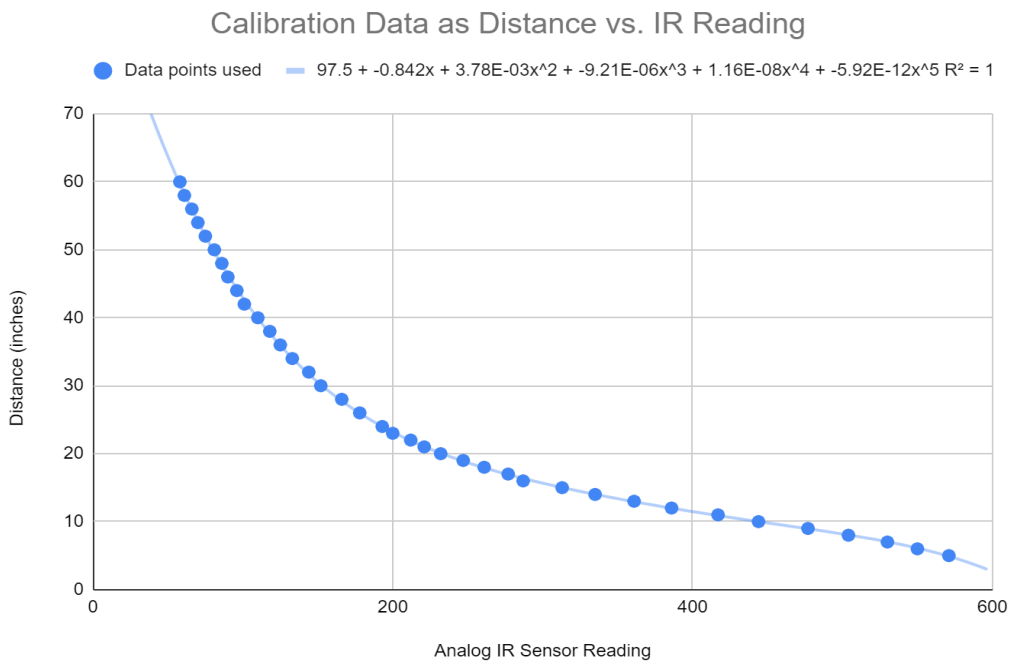

Figure 5: Calibration Data with Fit Function

In Figure 5, we flip the axes to plot distance by IR reading, disincluding data points collected below five inches. The reason for changing the axes is that in our code, the IR reading is the input variable (x) and the distance is the output variable (y), so fitting our function to this plot will give us a function that we can use directly. We found that a fifth degree polynomial function fit the data best, with an r-squared value of nearly exactly one. This final polynomial function is given at the top of Figure 5. Normally, high-degree fit functions create problems with overfitting, but since in this case we are merely trying to characterize the IR sensor, more accuracy is better than less. We thought that underfitting would likely cause increased error in the results anyway, so overfitting would not harm our model's accuracy.

We did not test the accuracy of our calibration curve at distances not included in our calibration routine, as we calibrated for the full range of the distance sensor. We collected forty-two total data points spanning from one to sixty inches (2.5 to 152 cm), greater than the specified range of the sensor (20 to 150 cm). Additionally, our calibration function looks like it would continue to give a good approximation of distance even at values higher than the maximum tested distance. Because of these factors, we found it unnecessary to test the error outside our functional range.

Code

For this project, we wrote three code files. The first was an Arduino script to sweep by servo angle and print the analog reading at each combination of pan and tilt angle to the serial port. The full Arduino code is provided under Appendix A. To perform the sweep, the panning servo sweeps in one degree increments until it reaches the maximum pan angle, then the tilting servo is incremented by one degree, and finally the panning servo sweeps back in the opposite direction

towards its minimum angle. This process is repeated until the maximum tilt angle is reached and the scan is finished. Note that the minimum and maximum angles are arbitrary values set based on the area that we wanted to scan and distance between the sensor and the plane being scanned.

The second file was a python script for data collection, provided in Appendix B. This script reads the serial port, then uses the calibration function to calculate the distance in inches from the IR readings using the function *convert_ir_to_dist()*, reproduced below. Lastly, the program saves the data (pan angle, tilt angle, IR reading, distance in inches) to a csv file.

Code Excerpt from Appendix B:

```
05 # coefficients of our calibration function
06 coeff = [97.4512,-0.841732,0.00377858,-0.00000921155,1.1576*10**-8,-5.9236*10**-12]
07
08 def convert_ir_to_dist(ir, coeff):
09     dist = 0.0
10     for i in range(len(coeff)):
11         dist += coeff[i] * ir**i
12     return dist
```

The final file included under Appendix C reads the data from a csv, converts each data point from spherical to cartesian coordinates, and creates a plot to visualize the data. The spherical to cartesian conversion is performed by the function *compensate_angle()* found on lines 5-12, and the math used in this function will be described below.

Math

In our coordinate system, x is the horizontal axis, y is the vertical axis, and z is the depth. To derive the equations we initially used, we visualized our robot standing at coordinates $(0,\ 0,\ z)$, where z is the distance between the scanner and the object we are scanning. To estimate the conversion from spherical to cartesian, we used the IR sensor distance, and pan and tilt angles to calculate the corresponding horizontal and vertical deviations of the viewpoint from the origin.

These are the trigonometric functions we used to calculate the x, y, and z values from pan (θ), tilt ($\boldsymbol{\phi}$), and distance (r):

$$x = r \cdot sin(\theta) \tag{1}$$

$$y = r \cdot sin(\boldsymbol{\phi}) \tag{2}$$

$$z = r \cdot cos(\theta) \cdot cos(\boldsymbol{\phi}) \tag{3}$$

Framing the problem as shown below in Figure 6, the distance measured by the IR sensor is represented by the red line, the x distance by the green line, and y distance by the blue line. The IR sensor measured distance makes up the hypotenuse of a right triangle with both the x- and y-distances. Using the sine of the angle (pan and tilt respectively) multiplied by the IR sensor distance, we can calculate the length of the opposite sides of the triangles to find x- and y-distance, described mathematically by Equations (1) and (2).

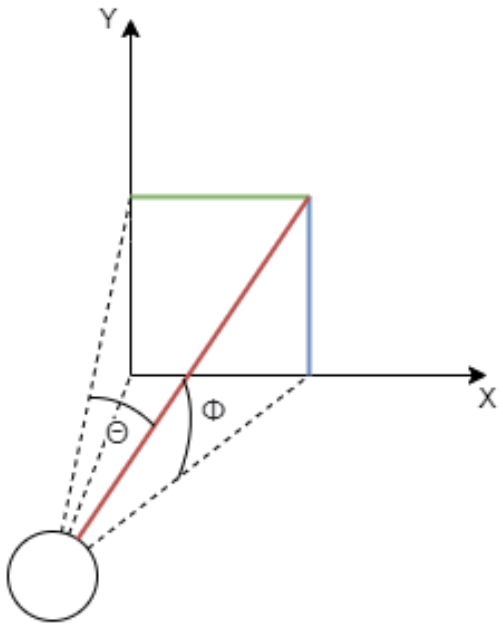

Figure 6: Diagram Relating X-Y Plane to Pan (θ) and Tilt (ϕ) Angles

Additionally, following a similar method, we found the z-distance by using the cosine of the pan angle to find the length of the adjacent side of the right triangle formed with the x-distance line. We then used that resulting adjacent side length as the hypotenuse for another triangle formed with the y-distance line (a triangle therefore oriented in the y-z plane), and using the cosine of the tilt angle to calculate the length along the z-axis, giving Equation (3).

Results

To test the functionality of our scanner, we created and scanned a cardboard cutout of the letter "A" that was eighteen inches tall and wide. Figure 7 shows the results of a test where we placed the "A" also at eighteen inches distance from the IR sensor, and scanned the letter on one axis (using only one actuating servo). We swept across a range of sixty degrees using the panning servo, scanning to capture distance on the x (horizontal) and z (depth) axes. The servo was pointed towards the bottom half of the letter, with the "legs" of the A in the sensor's signline.

Looking at Figure 7 below, there are two groupings of points at z-values around fifteen to twenty inches, one grouping on the left of the x-axis and one grouping on the right of the x-axis. The rest of the points are all greater than thirty inches away from the origin, indicating that they were too far away to be part of the cardboard letter. This result of two groupings of points that are close to eighteen inches on the z-axis is exactly what we expected to see in this one-servo scan.

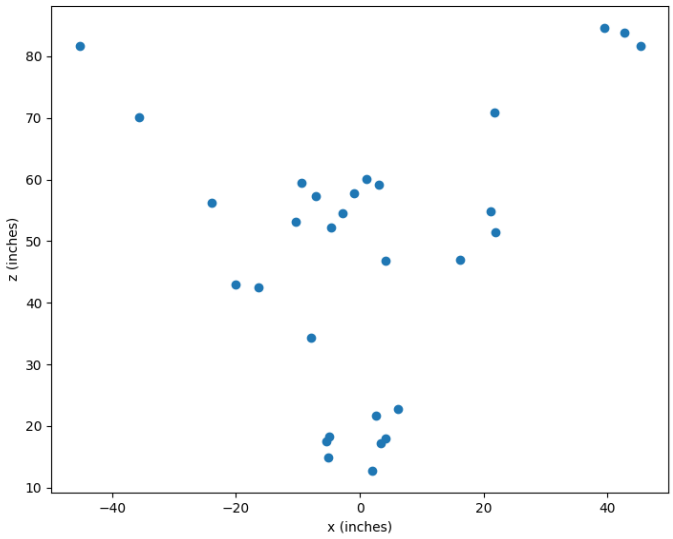

Figure 7: Plot of X vs. Z Data for Tilt (ϕ) of zero degrees

The results of our final test are shown in Figure 8, where the shape of a capital letter "A" is clearly visible, proving that our scanner works! To create the plot in Figure 8, we first collected data with the "A" placed at a distance of eighteen inches away from the sensor. The next step we took was to filter the data by z-distances between seventeen and nineteen inches, giving a small error margin on either side. Lastly, we filtered on y-values greater than negative four inches in order to disinclude data points showing the table directly below the "A". We represented our three-dimensional data using a two-dimensional plot of x vs. y, effectively "flattening" the z-axis into the page, a method that worked because z-distance was filtered on a meaningful range.

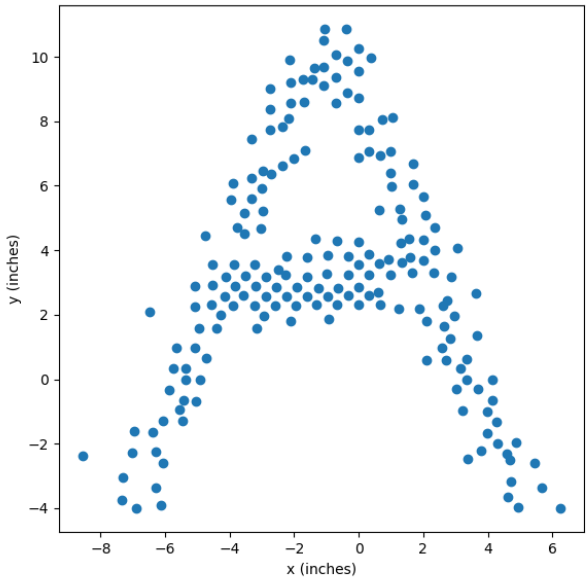

Figure 8: Plot of X vs. Y Filtered Data

Reflection

Lilo: This project was both frustrating and fun. I worked mainly on the code and the math during this mini-project, playing to my strengths, while Yehya worked on all areas of the project, but especially taking the lead on the mechanical side. We spent a good amount of time debugging python and arduino code, spent a couple of hours revisiting trigonometry and being confused by 3D coordinate systems, and made many unsuccessful plots, so it was quite a process to get our scanner working. Thankfully our hours of work paid off and we were able to achieve a good result in the end, clearly recognizing the cardboard letter in our final plot. If we had more time, I would address sources of error to increase the accuracy, including figuring out why the sensor reads outliers sometimes, and revisiting the coordinate transformation math.

Yehya: This project was great to work on. I thought it was a great mix of combining prior knowledge with new experiences. It was nice to have the freedom to choose how to approach this project from a mechanical and software perspective. Although we were able to arrive at our desired results, we had some difficulty getting there in terms of the math. We had found some conversions from spherical to cartesian coordinates online, but we could not get them to work. Instead, we derived our own equations that ended up working pretty well. For the mechanical design part, we tried our best to have the sensor as close as possible to the axes of rotation so that our math would be more accurate. Otherwise, it was very straightforward. If we had more time on this project, something we would probably work on is finding better ways to account for curvature of the scan and deviations off the axes of rotation.

Sources

[1] datasheetspdf.com/pdf-file/815274/SharpElectrionic/GP2Y0A02YK0F

Appendix A: Arduino Code

```
01 #include <Servo.h>
02
03 Servo tilt_servo;
04 Servo pan_servo;
05
06 int inputPin = A0;
07 int pan_interval = 50;
08 int tilt_interval = 50;
09 int pan_increment = 1;
10 int tilt_increment = 1;
11 int pan_angle = 0;
12 int tilt_angle = 0;
13 bool pan_direction = true;
14 bool tilt_direction = true;
15 bool tilt_changed = false;
16 int min_pan_angle = 60;
17 int min_tilt_angle = 50;
18 int max_pan_angle = 120;
19 int max_tilt_angle = 120;
20 bool set_start_pos = true;
21 bool done = false;
22
23 void setup() {
24   tilt_servo.attach(9);
25   pan_servo.attach(10);
26   Serial.begin(9600);
27 }
28
29 void loop() {
30   if (set_start_pos){
31     tilt_servo.write(min_tilt_angle);
32     pan_servo.write(min_pan_angle);
33     set_start_pos = false;
34   }else if (not done){
35     tilt_changed = false;
36     if (pan_angle >= max_pan_angle){
37       pan_angle = max_pan_angle;
38       pan_direction = false;
39       check_tilt();
40     }else if (pan_angle <= min_pan_angle) {
41       pan_angle = min_pan_angle;
42       pan_direction = true;
43       check_tilt();
44     }
45
46     if (tilt_changed) {
47       tilt_angle += tilt_increment;
48       tilt_servo.write(tilt_angle);
49       delay(tilt_interval);
50       read_IR();
51     }
52
53     if (pan_direction) {
54       pan_angle += pan_increment;
55     } else {
56       pan_angle -= pan_increment;
57     }
58     pan_servo.write(pan_angle);
59     delay(pan_interval);
60     read_IR();
61   }
62 }
```

```
63 void check_tilt() {
64   if (tilt_angle >= max_tilt_angle){
65     tilt_angle = max_tilt_angle;
66     tilt_direction = false;
67     done = true; // exit because scan is finished
68   }else if (tilt_angle <= min_tilt_angle) {
69     tilt_angle = min_tilt_angle;
70     tilt_direction = true;
71   }
72   if (tilt_direction) {
73     tilt_changed = true;
74   }
75 }
76
77 void read_IR(){
78   int sum = analogRead(inputPin);
79   Serial.print(pan_angle);
80   Serial.print(",");
81   Serial.print(tilt_angle);
82   Serial.print(",");
83   Serial.println(sum);
84 }
```

Appendix B: Python Data Collection

```
01 import serial
02 import math
03 import pandas as pd
04
05 # coefficients of our calibration function
06 coeff = [97.4512, -0.841732, 0.00377858, -0.00000921155, 1.1576*10**-8, -5.9236*10**-12]
07
08 def convert_ir_to_dist(ir, coeff):
09     dist = 0.0
10     for i in range(len(coeff)):
11         dist += coeff[i] * ir**i
12     return dist
13
14 arduinoComPort = "COM13"
15 baudRate = 9600
16
17 # open the serial port
18 serialPort = serial.Serial(arduinoComPort, baudRate, timeout=1)
19
20 tilt, max_tilt_angle = 50, 120
21
22 d = {'pan':[], 'tilt':[], 'ir':[], 'dist':[]}
23 df = pd.DataFrame(data=d)
24
25 # main loop to read data from the Arduino, then display it
26 while tilt < max_tilt_angle:
27   # ask for a line of data from the serial port, the ".decode()" converts the
28   # data from an "array of bytes", to a string
29   lineOfData = serialPort.readline().decode()
30
31   # check if data was received
32   if len(lineOfData) > 0:
33     # data was received, convert it into integers
34     pan, tilt, ir = (float(x) for x in lineOfData.split(','))
35     # pan: 30-150, tilt: 60-120, ir: 60-600
36
37     dist = convert_ir_to_dist(ir, coeff)
38     # print the results
39     print(str(pan), end="")
40     print("," + str(tilt), end="")
41     print("," + str(ir) + ",", end="")
42     print(round(dist, 2))
43
44     # if dist < 24 and dist > 6:
45     df_add = {'pan':pan, 'tilt':tilt, 'ir':ir, 'dist':dist}
46     df = df.append(df_add, ignore_index=True)
47
48 df.to_csv('mp2_data.csv', index=False)
```

Appendix C: Python Data Visualization

```
01 import math
02 import matplotlib.pyplot as plt
03 import pandas as pd
04
05 def compensate_angle(pan, tilt, dist):
06     pan_radians = - math.radians(pan)
07     tilt_radians = - math.radians(tilt)
08
09     x = dist * math.sin(pan_radians)
10     y = dist * math.sin(tilt_radians)
11     z = dist * math.cos(pan_radians) * math.cos(tilt_radians)
12     return float(x), float(y), float(z)
13
14 data = pd.read_csv('mp2_data.csv')
15 df = pd.DataFrame(data=data)
16 print(df)
17
18 pan_center_angle = 90
19 tilt_center_angle = 90
20 df["pan_centered"] = df["pan"] - pan_center_angle
21 df["tilt_centered"] = df["tilt"] - tilt_center_angle
22
23 x, y, z = [], [], []
24 for i in range(len(df)):
25     x_i, y_i, z_i = compensate_angle(df["pan_centered"][i], df["tilt_centered"][i],
df["dist"][i])
26     if z_i < 19 and z_i > 17 and y_i > -4:
27         x.append(round(x_i, 2))
28         y.append(round(y_i, 2))
29         z.append(round(z_i, 2))
30
31 fig = plt.figure(figsize=(12,12))
32 ax = fig.add_subplot()
33 plt.scatter(x, y)
34 ax.set_xlabel('x (inches)')
35 ax.set_ylabel('y (inches)')
36 ax.set_aspect('equal', adjustable='box')
37 plt.show()
```