3D Scanner

Mark Goldwater and Erika Serna

September 25, 2018

1 Setup

1.1 Hardware Setup

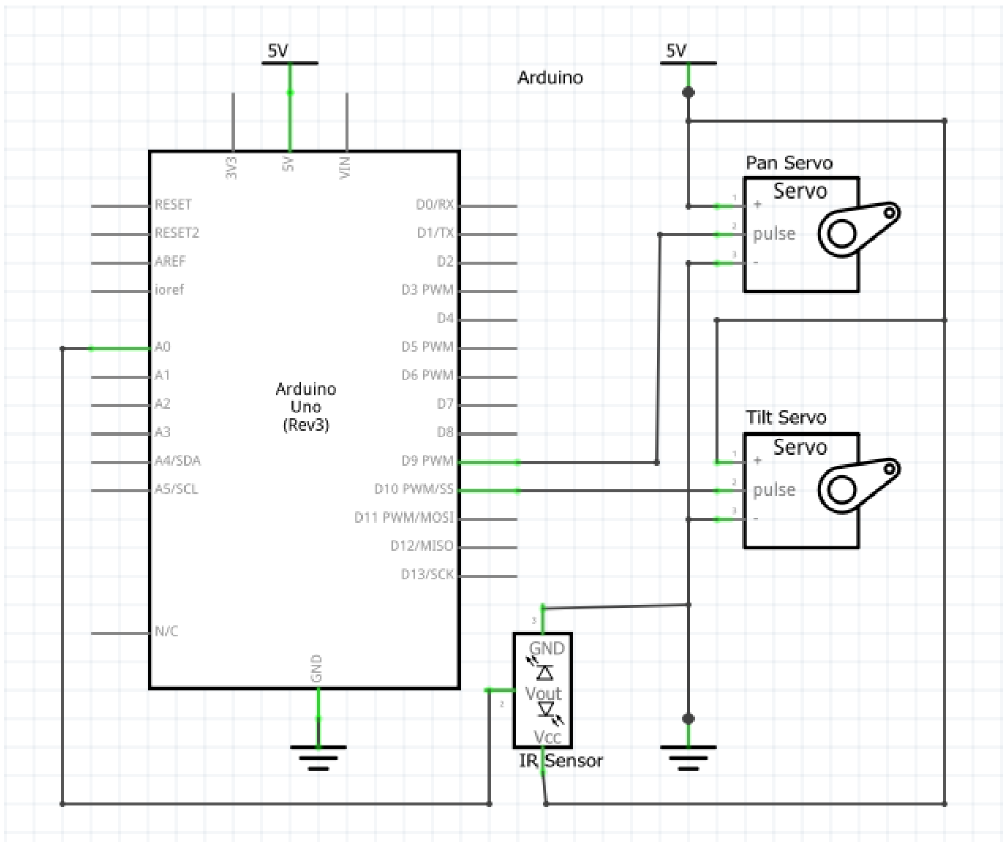

Figure 1: We built our 3D Scanning circuit using an Arduino Uno (R3), one servo for panning the 3D scanner, one servo for tilting it, and an IR sensor for taking distance measurements.

The circuit for this lab, which is shown in Figure 1, was fairly simple, and my partner and I got it set up quickly. We connected each of the servos to power and ground and each signal to a PWM (Pulse Width Modulation) digital pin. This way we could use Arduino code to write signals with different duty cycles to the servos in order to make them turn a desired amount of degrees. The IR sensor was simply connected to power and ground as well as an analog pin. Throughout the scan we read this analog pin in order to collect data from the IR sensor that we could convert to a distance.

1.2 Building the Scanner

After we had created a circuit that we knew would work, we began to design the hardware which would hold the two servos and the IR sensor. We decided to 3D print our 3D scanner because we believed that it would allow us the most freedom and provide the most reliability in its fabrication. While designing the scanner in SolidWorks, we had in mind that one of the outcomes we wanted from our design was for the IR sensor to pan and tilt on its own axes. In other words, if one pictures lines coming out of the shafts of both the pan and tilt servo, we wanted these lines to divide the IR sensor symmetrically. This way the pan and tilt of the sensor would not be augmented in terms of the arc length that the IR sensor traverses.

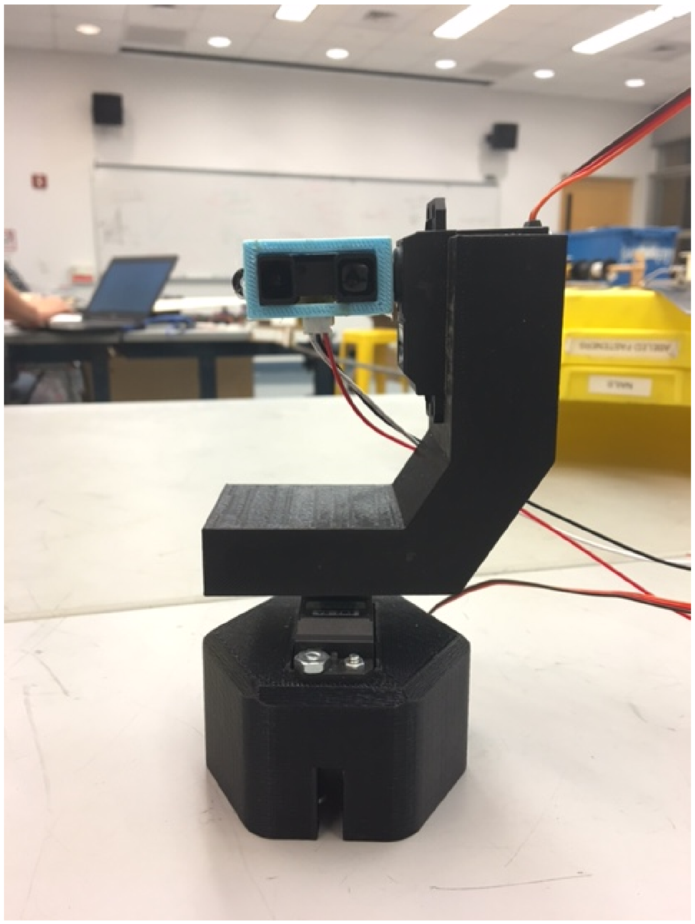

Figure 2: The 3D scanner was printed in three parts: the Base, the Arm, and the IR Sensor Holder. The Base holds the tilt servo, the Arm is connected by the horn of the tilt servo and holds the pan servo, and the IR Sensor Holder is connected by the pan servo without a horn and houses the IR sensor.

As can be seen in Figure 2, my partner and I designed a base which housed the pan servo. This servo had a circular servo horn on it which was press fitted into the arm which held the tilt servo.

We made a small casing for the IR sensor which allowed it to be press fitted right into the tilt servo which was itself secured in the arm of the servo by friction alone. Moreover, we measured the parts out such that the IR sensor would lie exactly over the center of the pan servo. So, since it was attached at the center of its side directly to the tilt servo, we achieved our goal of having the sensor pan and tilt in its own reference frame.

1.3 IR Sensor Calibration

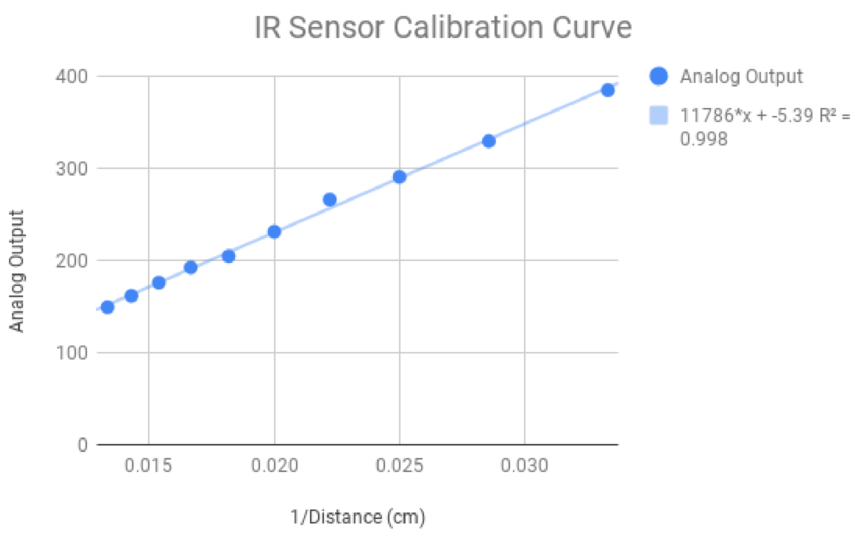

Figure 3: Calibration curve made linear.

The last step in our preparation to code the 3D scanner was to create a calibration curve (Figure 3) that correlates the value we get by reading the analog value from the IR sensor to its distance from the object the IR beam is hitting. We did this by pointing the IR sensor at a white piece of paper and measuring the output in 5 cm distance intervals from 30 cm to 75 cm. We recorded the analog read as voltage (by multiplying it by $\frac{5}{1023}$) in order to record a value that fluctuated less as we were measuring it. Then, before graphing the data, we converted it back to the analog read value by multiplying the voltage value by this fraction's reciprocal ($\frac{1023}{5}$).

When we graphed raw distance vs. analog output, the data resembled a downward curve which slowly flattened, so in order to make the data linear we graphed the reciprocal of the distance in centimeters vs. the analog output. From this we used linear regression to obtain a line of best fit with the equation $Output = 11786(\frac{1}{Distance}) - 5.39$.

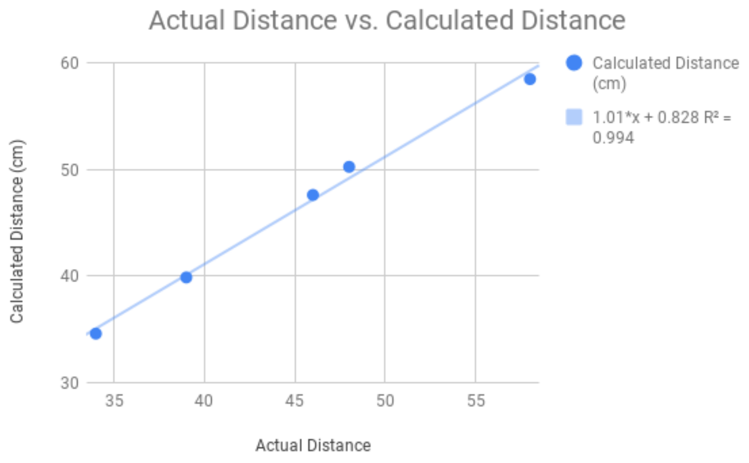

Figure 4: This graph shows the actual distance of the sensor and the calculated distance using the calibration curve we derived above.

In Figure 4 there is a graph which correlates the actual distance of the sensor and the distance we calculated using the calibration curve (Figure 3). Ideally this curve should be $y = x$ because in an ideal world, the calculated value would be the same as the actual. However, there is some error because the line of best fit for this graph is $calculated = 1.01(actual) + 0.828$. This is fairly close to $y = x$. So at this point, we can be confident that when we start using our calibration curve to convert the IR sensor value to a distance, we will get values that are accurate enough to create a representation of a recognizable letter.

2 Software

The software to actually carry out the 3D scan was written in Arduino C and was pretty simple. It scanned the letter in columns by setting a fixed pan angle, changing the tilt angle in order to scan down the letter, and then incrementally raising the pan angle by one degree. We initially did the opposite of this (scanning in rows), but the image was always slightly skewed. After reviewing the IR sensors documentation, we saw that one is supposed to move the sensor in the direction of the line that divides the emitter and the receiver on the sensor. After this change the sensor gave us a clear image of our letter 'E'. In the Arduino we initially set the variable **bool running = true** so that we could set it to false after one iteration of the 3D scan. We also used **define** to establish the variables **MAX_POS_PAN, MAX_NEG_PAN, MAX_POS_TILT,** and **MAX_NEG_TILT** to set the area that the scanner would take data for. The following is the main loop we used in the Arduino code to carry out the 3D scan:

```
void loop() {
  // put your main code here, to run repeatedly:
  if (running) {
    for (int i = 90 + MAX_POS_PAN; i >= 90 + MAX_NEG_PAN; i--) {
      panServo.write(i);
      for (int j = 90 + MAX_POS_TILT; j >= 90 + MAX_NEG_TILT; j--) {
        tiltServo.write(j);
        IRVal = analogRead(IRPin);
        Serial.print(IRVal);   Serial.print(",");
        Serial.print(i);       Serial.print(",");
        Serial.println(j);
        delay(300);
      }
    }
    running = false;

    // short delay after scanning and recenter mount
    delay(500);
    tiltServo.write(90);
    panServo.write(90);
  }
```

As can be seen in the above code, once the script initiated and the loop function was called for the first time, it entered the if statement which checked if **running** was **true** and then began the double for loop which carried out the actual scan. The outer loop would traverse from the pan angle range, writing the newest angle to the pan servo before entering the tilt servo for loop. In this loop, the newest tilt angle is written to the servo, and the IR sensor value is read and assigned to the variable **IRVal**. Then, the IRVal, pan angle (in degrees), and tilt angle (in degrees) is printed to the serial monitor for python to read and process.

Because the tilt for loop is on the inside, the 3D scanner traverses over every tilt angle for every pan angle. So it is scanning in columns. We initially scanned in rows; however, when we did this our data came back skewed. After looking at the documentation, we realized that the IR sensor is supposed to move along the line that splits the emitter and receiver. After making this change the quality of our results increased dramatically.

After the double for loop has completed, **running** is set to **false** so that the scanner does not run again. Then, after a short delay of half a second. both servos are centered again by writing 90 degrees to each of them.

3 Processing Scan Data

3.1 Reference Frame of the IR Sensor

Before we go into how we processed the data from the IR sensor, let's establish the coordinate system that we used to represent its reference frame. Our IR sensor was mounted so that the emitter and receiver were horizontal. We established the direction that they pointed as z, to the right of the sensor (with the emitter and receiver facing away from you) was the x direction, and the direction above the sensor was the y.

So, in order to convert from spherical coordinates, which are based off of the distance read by the IR sensor and the pan and tilt angle, to Cartesian coordinates we used the following equations

$$x = r * cos(\theta_2)sin(\theta_1) \tag{1}$$

$$y = r * sin(\theta_2) \tag{2}$$

$$z = r * cos(\theta_2)cos(\theta_1) \tag{3}$$

where r is the distance in centimeters that the IR sensor reads, θ_1 is the pan, and θ_2 is the tilt. Now with this and our calculated r from the calibration curve shown in Figure 3, we are able to calculate the Cartesian components of the vector that is the IR beam from the sensor to whatever it hits.

3.2 Reading Data From the Serial Monitor in Python

```
# connect to serial port
arduinoComPort = '/dev/ttyACM0'

# Set the baud rate
baudRate = 9600

# open the serial port
serialPort = serial.Serial(arduinoComPort, baudRate, timeout=1)

# read one line of data from serial monitor
lineOfData = serialPort.readline().decode()
```

Above is Python code that reads data one line at a time from the serial monitor, which is printed in the Arduino code. Through the **Comm. Port**, Python is able to access the data that the IR sensor records as well as the associated pan and tilt angles of the scanner. The **baud rate** sets the rate at which the information is transferred. **SerialPort** makes the data accessible through the Python code, and **lineOfData** reads one line of data from the serial monitor at a time.

3.3 Top-Down View Plot

Below is the main part of the code to plot the top-down view of the letter which can be seen in Figure 5. This data was taken by panning the scanner across the letter, which sat in front of a cardboard wall, while keeping the tilt angle at zero.

```
X, Y, Z = list(), list(), list()

while True:
  # read one line of data from serial monitor
  lineOfData = serialPort.readline().decode()

  # check if data was received
  if len(lineOfData) > 0:
    # data was received, convert it into 3 floats
    val, theta1 = (x for x in lineOfData.split(','))

    theta1 = radians((int(theta1) - 90))
    r = ((float(val) + 5.39) / 11786) ** -1

    print(theta1)

    # store respective points in their lists
    X.append(r*cos(theta2)*sin(theta1))
    Y.append(r*sin(theta2))
    Z.append(r*cos(theta2)*cos(theta1))

    if (theta1 == radians(-20)):
      break

fig = plt.figure()
ax = fig.add_subplot(111, projection=None)

ax.scatter(X, Z, c='r', marker='o')

ax.set_xlabel('Horizontal Distance From IR Sensor (cm)')
ax.set_ylabel('Vertical Distance From IR Sensor (cm)')
ax.set_title('Top Down View of Sweep Across Letter')

plt.show()
```

Essentially what we do here is read each line of the serial monitor (one at a time) and only save the IR sensor value and theta1 (the pan) since theta2 (the tilt) can be assumed to always be zero. Then r and the x, y, z Cartesian coordinates are calculated and stored in their own lists. Then keeping in mind the reference frame established in **Section 3.1**, plotting x vs. z would give us a top down view of the letter after sweeping through the range of pan angles only with the tilt angle at constant zero.

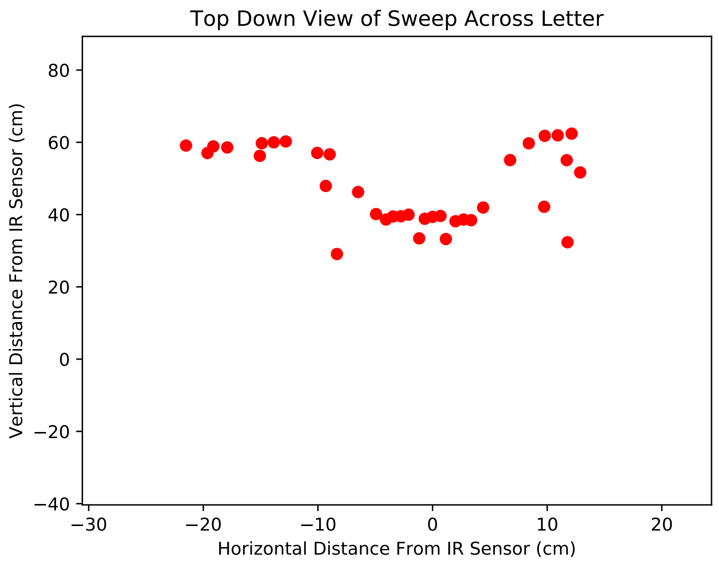

Figure 5: Here a clear wall can be seen behind the outfacing bump in the graph which was caused by the IR beam scanning the top of the 'E'. The measurement values in this image are also quite accurate. The letter was about 40 cm away while the wall was 60 cm away.

3.4 Heatmap

Below is the code that processed the data from the full 3D Scan. The logic behind it is very similar to that of the code for the single sweep scan. The main difference was that each column that we scanned was stored in a numpy array with the following code

```
z.append(r*cos(theta2)*cos(theta1))
```

and then after each column was scanned it was added to a larger numpy array (Z) after the horizontal row vector the values were stored in were transposed. This was all done with the following line of code.

```
Z = np.hstack((Z, np.array(z)[np.newaxis].T))
```

After the large Z matrix was created, it represented each "pixel" of the area we scanned in the correct orientation. So we graphed it in a heat map with the following code.

```
fig, ax = plt.subplots()
im = ax.imshow(np.divide(Z, np.max(Z)))

#We want to show all ticks...
ax.set_xticks(np.arange(len(thetas1)))
ax.set_yticks(np.arange(len(thetas2)))

#... and label them with the respective list entries
ax.set_xticklabels(thetas1)
ax.set_yticklabels(np.multiply(thetas2, -1))

# label axes
ax.set_xlabel('Pan Angle of Sensor Relative to Letter (Degrees)')
ax.set_ylabel('Tilt Angle of Sensor Relative to Letter (Degrees)')

divider = make_axes_locatable(ax)
cax = divider.append_axes('right', size='5%', pad=0.05)
cbar = fig.colorbar(im, cax=cax, orientation='vertical')
cbar.ax.set_ylabel('Normalized Distance From Sensor')

# Rotate the tick labels and set their alignment.
plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
         rotation_mode="anchor")

ax.set_title("3D Scan of Cardboard Letter 'E'")
fig.tight_layout()
plt.show()
```

This code may look like a lot but all it does is graph the distances values in the Z matrix as a heat map (after normalizing them), set the x axis labels to the pan angles, the y axis labels to the tilt angles, labels the axes, creates a color bar key, and titles the graph.

The code calculates the r value in cm and only the Z distance (perpendicular distance from scanner to the surface it is scanning) because this is the most important component to make the heat map. In the script, as the scanner scanned down each column of its field of view, the values were stored in a numpy array. Then, when the current column was not being scanned, we transposed it and stuck it to the right side of the rest of data. This was done with every column until the end of the scan. Then this 2D matrix was plotted with matplotlib in order to create the heatmap shown in Figure 6.

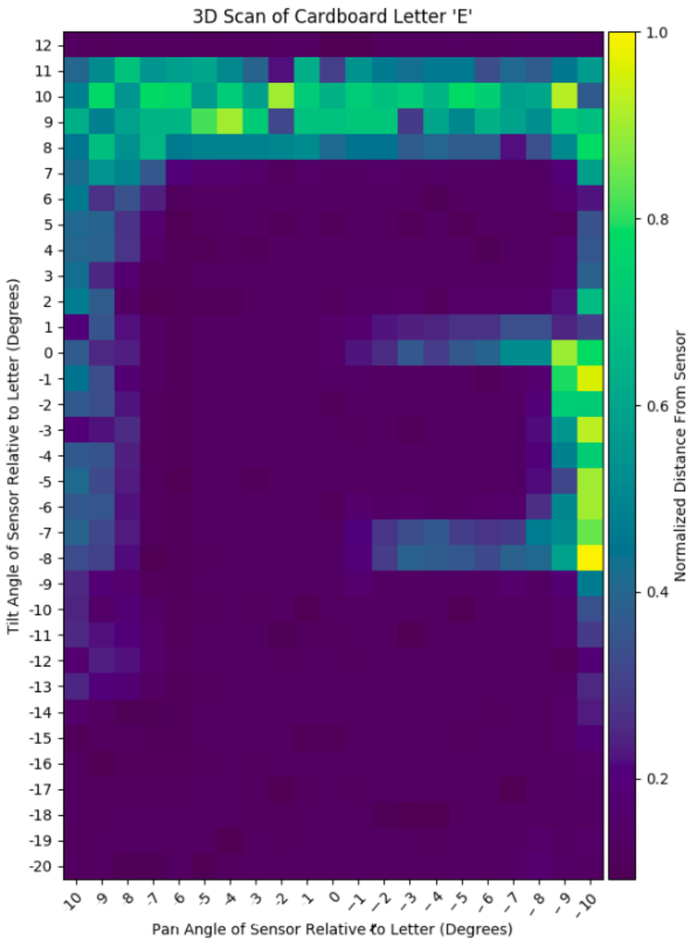

Figure 6: Here a clear wall can be seen behind the outfacing bump in the graph which was caused by the IR beam scanning the top of the 'E'. The measurement values in this image are also quite accurate. The letter was about 40 cm away while the wall was 60 cm away.

The x and y axes of the heat map are labeled with the pan and tilt angles respectively for that section of the scan. The distance (represented by the color of each pixel). We decided not to use x and y Cartesian distances in the heat map because when the beam went past the letter, what it hit was so far that the distance reading was useless. Moreover, we decided to normalize the color coding of the heat map in order to reduce the range from 0 - 2000 to a more presentable 0 - 1.

4 Reflection

We spent a considerable amount of time working on this project. Overall it can be said that this project had some parts that were definitely time consuming but that the process of how the 3D scanner works was reasonably easy to understand once running.

The CAD for the parts and the 3D print is what took most of our time. We edited the CAD for the Arm several times to assure that our IR Sensor lined up with our bottom servo. We spent more than 5 hours printing.

To be more time efficient, we did different tasks and then gave a brief explanation to the other partner and vice-versa as to be on the same page. This method worked moderately well but felt a bit a disjointed.

5 Appendix

5.1 3D Scanner Arduino Source Code

```
#include <Servo.h>

// angle constants
#define MAX_POS_PAN 10
#define MAX_NEG_PAN -10
#define MAX_POS_TILT 20
#define MAX_NEG_TILT -12

Servo panServo;
Servo tiltServo;

// set IR sensor and servo pins
const int IRPin = A0;
const int panServoPin = 9;
const int tiltServoPin = 10;

int IRVal;
bool running = true;

void setup() {
  // begin serial monitor
  int baudRate = 9600;
  Serial.begin(baudRate);

  // attach servos to pins
  panServo.attach(panServoPin);
  tiltServo.attach(tiltServoPin);

  // center servos and pause
  panServo.write(90);
  tiltServo.write(90);
  // wait 1 second before the scanning starts
  delay(1000);
}

void loop() {
  // put your main code here, to run repeatedly:
  if (running) {
    for (int i = 90 + MAX_POS_PAN; i >= 90 + MAX_NEG_PAN; i--) {
      panServo.write(i);
      for (int j = 90 + MAX_POS_TILT; j >= 90 + MAX_NEG_TILT; j--) {
        tiltServo.write(j);
        IRVal = analogRead(IRPin);
        Serial.print(IRVal);   Serial.print(",");
        Serial.print(i);       Serial.print(",");
        Serial.println(j);
        delay(300);
      }
    }
    running = false;

    // short delay after scanning and recenter mount
    delay(500);
    tiltServo.write(90);
    panServo.write(90);
  }
}
```

5.2 Top-Down Plot Python Source Code

```
import serial
from math import *
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt


# connect to serial port
arduinoComPort = '/dev/ttyACM0'

# Set the baud rate
baudRate = 9600

# open the serial port
serialPort = serial.Serial(arduinoComPort, baudRate, timeout=1)

# initialize sets of x, y, z points
X, Y, Z = list(), list(), list()

theta2 = 0
```

```
# main loop to read data from the Arduino, process it, and plot it
while True:
  # read one line of data from serial monitor
  lineOfData = serialPort.readline().decode()

  #TODO convert angles from [0, 180] to [-90, 90]

  # check if data was received
  if len(lineOfData) > 0:
    # data was received, convert it into 3 floats
    val, theta1 = (x for x in lineOfData.split(','))

    theta1 = radians((int(theta1) - 90))
    r = ((float(val) + 5.39) / 11786) ** -1

    print(theta1)

    # store respective points in their lists
    X.append(r*cos(theta2)*sin(theta1))
    Y.append(r*sin(theta2))
    Z.append(r*cos(theta2)*cos(theta1))

    if (theta1 == radians(-20)):
      break

fig = plt.figure()
ax = fig.add_subplot(111, projection=None)

ax.scatter(X, Z, c='r', marker='o')

ax.set_xlabel('Horizontal Distance From IR Sensor (cm)')
ax.set_ylabel('Vertical Distance From IR Sensor (cm)')
ax.set_title('Top Down View of Sweep Across Letter')

plt.show()
```

5.3 Heat Map Python Source Code

```
import serial
from math import *
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
from mpl_toolkits.axes_grid1 import make_axes_locatable
import matplotlib.pyplot as plt

# connect to serial port
arduinoComPort = '/dev/ttyACM0'
```

```
# Set the baud rate
baudRate = 9600

# open the serial port
serialPort = serial.Serial(arduinoComPort, baudRate, timeout=1)

# initialize sets of x, y, z points
X, Y, z = list(), list(), list()

thetas1 = np.arange(-10, 11)
thetas2 = np.arange(-12, 21)

# numpy array for distance axis
Z = np.array([])

zCounter = 0
rowCounter = 0

# main loop to read data from the Arduino, process it, and plot it
while True:
  # read one line of data from serial monitor
  lineOfData = serialPort.readline().decode()

  # check if data was received
  if len(lineOfData) > 0:
    # data was received, convert it into 3 floats
    val, theta1, theta2 = (x for x in lineOfData.split(','))

    # convert angles to -90 to 90 rather than 0 to 180
    theta1 = radians((int(theta1) - 90))
    theta2 = radians((int(theta2) - 90))

    # calculate IR sensor distance
    r = ((float(val) + 5.39) / 11786) ** -1
    print(r)

    # store respective points in their lists
    #X.append(int(r*cos(theta2)*sin(theta1)))
    #Y.append(int(r*sin(theta2)))
    z.append(r*cos(theta2)*cos(theta1))

    zCounter += 1
    if zCounter == 33 and Z.size != 0:
      Z = np.hstack((Z, np.array(z)[np.newaxis].T))

      if (rowCounter == 20):
        break
      else:
        rowCounter += 1

      z = list()
      zCounter = 0

    elif zCounter == 33:
      Z = np.array(z)[np.newaxis].T

      rowCounter += 1

      z = list()
      zCounter = 0

fig, ax = plt.subplots()
im = ax.imshow(np.divide(Z, np.max(Z)))

#We want to show all ticks...
ax.set_xticks(np.arange(len(thetas1)))
ax.set_yticks(np.arange(len(thetas2)))
#... and label them with the respective list entries
ax.set_xticklabels(thetas1)
ax.set_yticklabels(np.multiply(thetas2, -1))

# label axes

ax.set_xlabel('Pan Angle of Sensor Relative to Letter (Degrees)')
ax.set_ylabel('Tilt Angle of Sensor Relative to Letter (Degrees)')

divider = make_axes_locatable(ax)
cax = divider.append_axes('right', size='5%', pad=0.05)
cbar = fig.colorbar(im, cax=cax, orientation='vertical')
cbar.ax.set_ylabel('Normalized Distance From Sensor')

# Rotate the tick labels and set their alignment.
plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
         rotation_mode="anchor")

ax.set_title("3D Scan of Cardboard Letter 'E'")
fig.tight_layout()
plt.show()
```