POE Lab Report: 3D Scanner

Dennis Chen & Jennifer Wei

September 2015

1 3D Scanner Description

We built a 3D scanner using two servos and an infrared distance sensor to create a pan/tilt mechanism. With an Arduino, we controlled the servos and sensor and transmitted the servo angles and sensor distances to our laptop, where we processed and visualized the data.

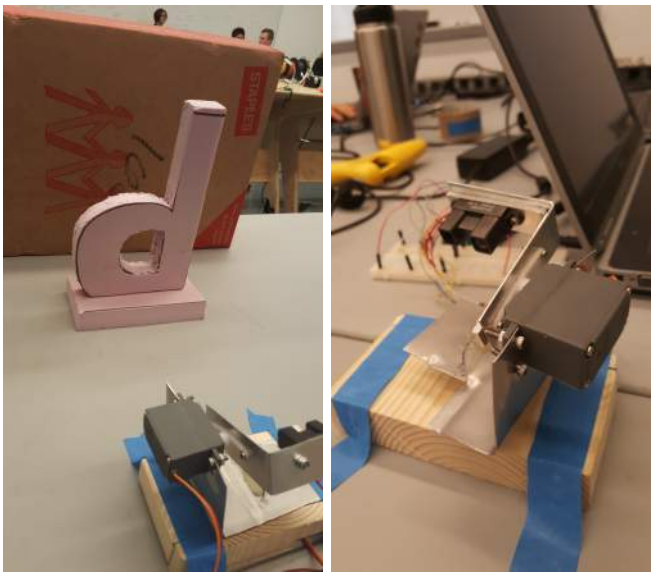

Figure 1: To the left is our 3D scanner scanning the foam letter *d* we made. To the right is a close-up of our 3D scanner.

2 Testing Description

Before building the scanner, we first tested the sensor to figure out the relation between the analog voltage reading of the sensor output and the actual distance between an object and the sensor.

To do this, we mounted the sensor to the edge of the lab table and set up a tape measure perpendicular to the table edge. We then held a large piece of cardboard at specific distances away from the sensor, read the analog sensor outputs with an Arduino, and converted those to voltage readings (max of 5V). We did this for the distances (in cm) of: 5,10,15,20,25,30,35,40,45,50.

Distance (cm)	Arduino Analog Output (max 255)	Voltage Reading (max 5V)	Calculated Distance (cm)
5	100	1.960784314	Did not include
10	112	2.196078431	Did not include
15	145	2.843137255	16.47799557
20	135	2.647058824	19.4554524
25	124	2.431372549	22.99683627
30	107	2.098039216	29.14070006
35	94	1.843137255	34.53795224
40	81	1.588235294	40.73985339
45	73	1.431372549	45.07275812
50	66	1.294117647	49.27295392
NEW DISTANCES	Output (max 255)	Voltage (max 5V)	Predicted Distance (cm)
17	144	2.823529412	2.80774032
22	128	2.509803922	2.490242271
27	114	2.235294118	2.208646762
55	61	1.196078431	1.127925385
60	55	1.078431373	1.000380075

Figure 2: Table of data used for calibrating infrared sensor.

We plotted distance vs voltage datapoints and fit an exponential trendline to the data in order to get an equation for calculating distance based on the sensor output. We chose an exponential equation for our function relating the sensor output to the distance of an object from the sensor because exponential functions asymptotically reach a small value as the distance between an object and sensor goes to infinity, which matches the expected behavior of the system.

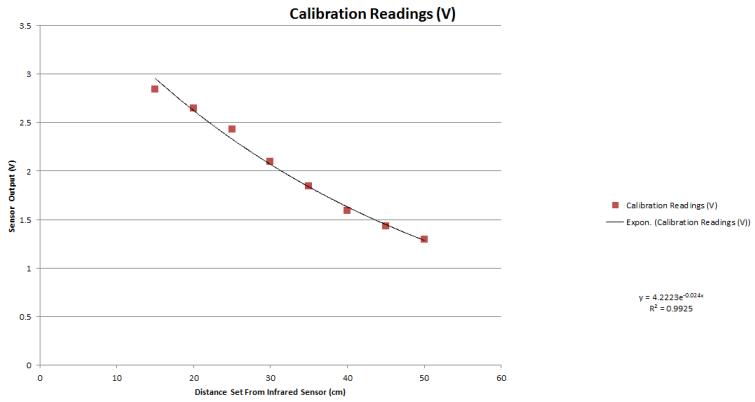

Figure 3: Plot of calibration data and the equation used to fit data.

To test our distance calculating equation, we plugged in a new set of distances (in cm): 17,22,27,55,60 - and compared predictions to the actual data. Our predictions agreed well with the experimental data collected.

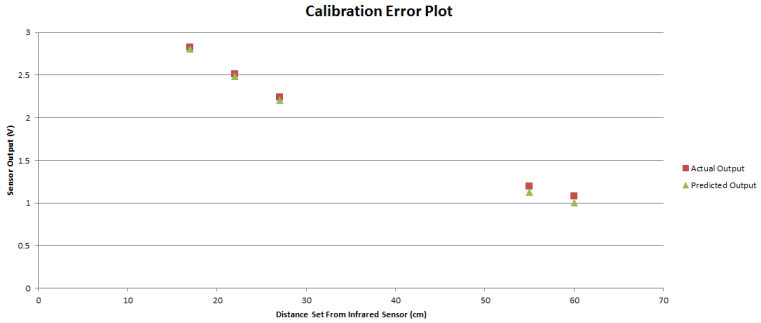

Figure 4: Plot of predicted vs actual measurements for new distances.

3 One Servo Scan

After building our scanner mounts, we conducted a basic, one servo scan by holding one servo at a constant angle while rotating the other servo about the vertical, or Z, axis. We chose to have our scanner scan a single line across the center of our foam *'d'*, as well as a portion of a box we placed behind our shape for reference. Our setup and the scan target (depicted as a red line) is shown in figure 5.

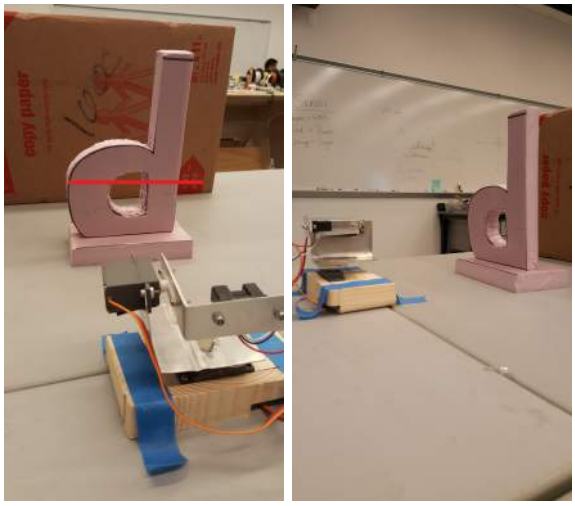

Figure 5: One servo scan setup. On the left, the approximate line that the scanner was set up to scan is marked in red. The picture on the right offers a side view of the same setup.

We collected data by printing out the angle of the servo and the reading from the infrared sensor to the serial monitor, and pasting the data into csv files that we then parsed in Python. To convert between polar and Cartesian coordinates and to get a top down 2-D visualization, we used the following formulas:

$$x = rcos(\theta) \tag{1}$$

$$y = rsin(\theta) \tag{2}$$

Because we mounted our sensor so that it was in-line with the axes of rotation of the servos, we did not have to do any additional compensations in our calculations.

A visualization of the single servo scan data is shown in figure 6. The two sections of the blue line furthest from the origin represent the box in the background behind the foam letter, and the two sections of the line that are closest to the origin are the two sides of the *'d'* around the middle of the hole in the letter. To confirm that this was actually the case, we measured the size of the hole in our letter, which was 5cm. This closely matches the size of the hole in the visualization plot, which is about 5cm wide.

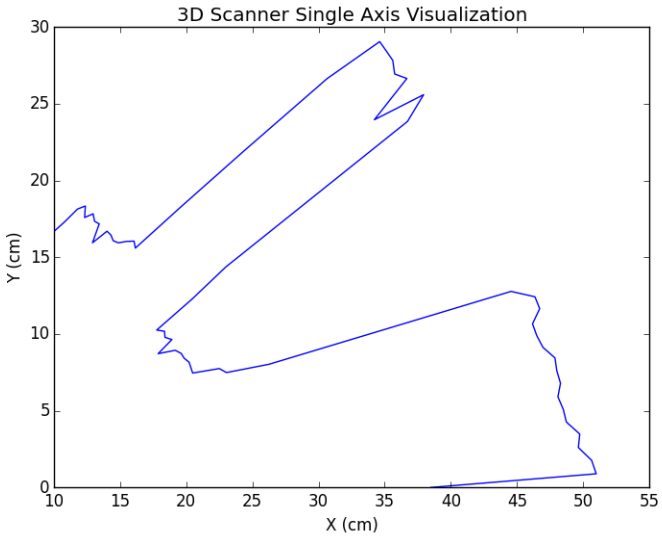

Figure 6: A 2-D visualization of the one servo scan.

4 Two Servo Scan

We performed a two servo scan as well, scanning the distance sensor in both the vertical and horizontal axes. The setup was identical as in the one servo case (shown again in figure 7), except that we wrote our Arduino code so that both servos would move.

Figure 7: Setup for the two servo scan.

We printed sets of servo angles and the measurements for each of those configurations and then parsed and visualized the data in Python. Before doing any conversion from polar to Cartesian coordinates, we first created a surface plot of the values read.

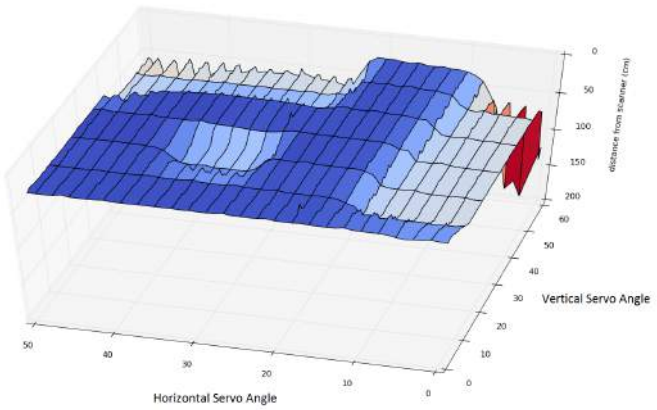

Figure 8: A 3-D visualization of the two servo scan data before converting to cartesian coordinates.

In order to get a visualization with less distortion, we then converted the polar coordinates to Cartesian ones using the following set of equations:

$$x = rcos(\theta)cos(\phi) \tag{3}$$

$$y = rcos(\theta)sin(\phi) \tag{4}$$

$$z = rsin(\theta) \tag{5}$$

Because we mounted our sensor so that it was in-line with the axes of rotation of the servos, we did not have to do any additional compensation in our calculations. After doing calculations and adding offsets to some of the angles to rotate our visualization in space so that it could be easily visualized on a single plane, we created a heatmap showing our two servo scan results.

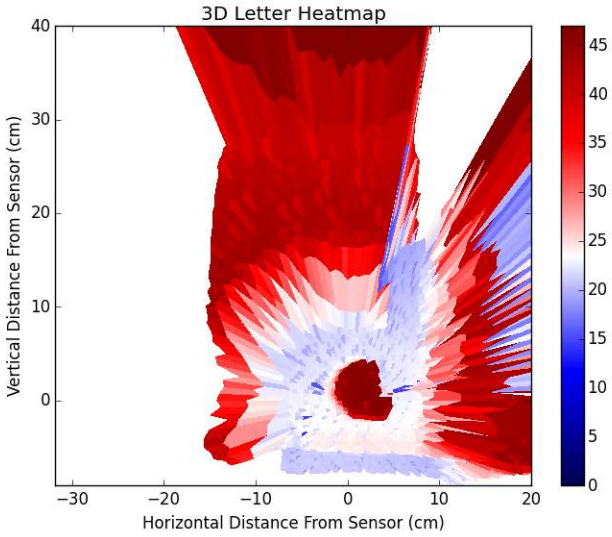

Figure 9: A 3-D visualization of the two servo scan data in cartesian coordinates.

The foam d shape is clearly visible in our plot, along with the little platform that we built for it to sit on! The calculated dimensions of the d in the plot matched up well with the actual dimensions of the foam letter - the hole in the d is about 5cm wide, and the base that the d sits on is about 17cm wide.

5 Code

5.1 Arduino

To collect data, we wrote two Arduino scripts - one for the one-axis sweep and the other for the two-axis sweep.

Instead of using the `loop` function, we wrote a `runScan` (7.3, Line 20). Based on the location of our *'d'* with respect to the starting sensor location, we set constants for the default start angle of each servo (`HORZ_SWEEP_START`, `VERT_SWEEP_START`) as well as a set sweep range (`HORZ_SWEEP_RANGE`, `VERT_SWEEP_RANGE`). We then loop set up two nested `for` loops such that for each angle (when sweeping around the vertical axis), the sensor loops through all the angles of the opposite axis sweep and prints the value at each angle swept (7.3, Lines 29-35). A `delay` is added to ensure that the servos have enough time to sweep before using `analogRead` and collecting the sensor value.

The script for the one-axis sweep is similar, but instead of having two nested `for` loops, there is only one `for` loop that sweeps around the vertical axis, holding the other servo at a constant angle to position the sensor roughly perpendicular to the front face of the letter *'d'*.

5.2 Python

To visualize the one servo scan, we started by loading the information that we had stored in a csv into a numpy array (7.2, `plotSingleSweep` Line 19). Then we converted the Arduino `analogRead` values to a distance measurement using the formula we got from calibration (7.2, `voltageToDist`) . Finally, we converted the polar coordinates that we had to Cartesian ones (7.2, `polartoCoordinate`) and plotted them.

To visualize the two servo scan, we followed a very similar approach, except for the fact that we had to shift some of the angles used in the calculations in order to rotate our point cloud so that it could be visualized easily on one plane. After shifting the angles (7.4, `plotHeatMap` Lines 60-62) and converting `analogRead` values to distances (7.4, Line 66), we converted to Cartesian coordinates (7.4, `convertToXYZ`) and plotted a heatmap.

6 Reflection

In this lab we learned how to control servos, collect data from Arduino, and parse and visualize data in a separate program. One of the most difficult aspects of this lab was getting a 3-D visualization that actually resembled our foam letter. By stepping back and first plotting the data without doing any conversions to Cartesian coordinates first, we were able to confirm that we were seeing the data that we expected to see. After doing that we had to do a lot of experimentation with visualization techniques, and finally got our letter to appear by rotating our point cloud and using a heat map to visualize it.

We also learned about building mechanical structures. After messing around a lot with sheet metal and screws, we learned that hot glue and wood can be equally effective and a lot faster. In the future, we'll approach projects by prototyping and moving quickly first before using materials like metal, particularly if the mechanical part of the project is simple and is only really there to enable other things (in this case the sensor scanning).

7 Source Code

7.1 Arduino One-Axis Sweep Code

```
#include <Servo.h>

const int HORZ_SWEEP_START = 80;
const int HORZ_SWEEP_RANGE = 60;
const int SETTLE_TIME = 200;

Servo horz;
Servo vert;

void setup()
{
  horz.attach(10);
  vert.attach(9);
  Serial.begin(9600);
  runScan();
}

void runScan()
{
  // runScan collects the angle and sensor values from 80 to 140 degrees
  vert.write(60);
  for(int j = HORZ_SWEEP_START; j < HORZ_SWEEP_START + HORZ_SWEEP_RANGE; j++){
    horz.write(j);
    delay(SETTLE_TIME);
    int sensorValue = analogRead(0);
    Serial.print(j);
    Serial.print(",");
    Serial.print(sensorValue);
    Serial.print(",");
    Serial.print("\n");
  }
}

void loop()
{
  /**
   leaving this here because it won't compile otherwise
   */
 }
```

7.2 2D Scan Visualization Code

```
import numpy as np
import matplotlib.pyplot as plt

def analogReadToDist(a):
    return voltageToDist(analogReadToVoltage(a))

def voltageToDist(v):
    return (np.log(v) - np.log(4.2223)) / -0.024 # calibration generated equation

def analogReadToVoltage(a):
    return a * 1.0 / 1023 * 5

def polarToCoordinate(a, d):
    x = d * np.cos(a*np.pi/180)
    y = d * np.sin(a*np.pi/180)
    return (x,y)
```

```
def plotSingleSweep():
    data = np.loadtxt("sweepdata.csv", delimiter=',')
    sensorVals = data[:,1]
    distances = np.vectorize(analogReadToDist)(sensorVals)
    angles = data[:,0] - 80 # subtracting 80 to remap so min angle=0 instead
    x,y = polarToCoordinate(angles, distances)
    plt.plot(x,y)
    plt.xlabel('X (cm)')
    plt.ylabel('Y (cm)')
    plt.title('3D Scanner Single Axis Visualization')
    plt.show()

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

7.3 Arduino Two-Axes Sweep Code

```
#include <Servo.h>

const int HORZ_SWEEP_START = 80;
const int HORZ_SWEEP_RANGE = 60;
const int VERT_SWEEP_START = 40;
const int VERT_SWEEP_RANGE = 60;
const int SETTLE_TIME = 200;

Servo horz;
Servo vert;

void setup()
{
  horz.attach(10);
  vert.attach(9);
  Serial.begin(9600);
  runScan();
}

void runScan()
{
  // runScan collects the two sets of angle and sensor values from 40 to 100 degrees vertically a
  for(int i = VERT_SWEEP_START; i < VERT_SWEEP_START + VERT_SWEEP_RANGE; i++){
    for(int j = HORZ_SWEEP_START; j < HORZ_SWEEP_START + HORZ_SWEEP_RANGE; j++){
      vert.write(i);
      horz.write(j);
      delay(SETTLE_TIME);
      int sensorValue = analogRead(0);
      Serial.print(i);
      Serial.print(",");
      Serial.print(j);
      Serial.print(",");
      Serial.print(sensorValue);
      Serial.print(",");
      Serial.print("\n");
    }
  }
}

void loop()
{
  /**
   leaving this here because it won't compile otherwise
   */
```

```
}
```

7.4 3D Scan Visualization Code

```
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
import matplotlib.pyplot as plt
import numpy as np

def analogReadToDist(a):
    return voltageToDist(analogReadToVoltage(a))

def voltageToDist(v):
    if(v == 0):
        # v cannot be 0 (the equation below won't work - so setting it to a
        # large value that won't affect the visualization)
        return 200
    return (np.log(v) - np.log(4.2223)) / -0.024

def analogReadToVoltage(a):
    return a * 1.0 / 1023 * 5

def getArrayFromFile(path):
    return np.loadtxt(path, delimiter=',')

def convertToXYZ(v,h,r):
    x = r*np.cos(v*np.pi/180)*np.cos(h*np.pi/180)
    y = r*np.cos(v*np.pi/180)*np.sin(h*np.pi/180)
    z = r*np.sin(v*np.pi/180)
    return (x,y,z)

def plotScannerData(x,y,z):
    hf = plt.figure()
    ha = hf.add_subplot(111, projection='3d')
    ha.plot_wireframe(x,y,z, cmap=cm.coolwarm, cstride=5)
    ha.set_xlabel('x (cm)')
    ha.set_ylabel('y (cm)')
    ha.set_zlabel('z (cm)')
    ha.set_xlim3d(0, 50)
    ha.set_ylim3d(0, 50)
    ha.set_zlim3d(0, 100)
    plt.show()

def plotHeatMap(x,y,z):
    hf = plt.figure()
    z_min, z_max = -np.abs(z).max(), np.abs(z).max()
    plt.pcolor(x, y, z, cmap='bone', vmin=0, vmax=z_max - 120)
    plt.title('3D Letter Heatmap')
    plt.xlabel("Horizontal Distance From Sensor (cm)")
    plt.ylabel("Vertical Distance From Sensor (cm)")

    # Set the limits of the plot to the limits of the data
    plt.axis([x.min(), 20, y.min(), 40])
    plt.colorbar()
    plt.show()

if __name__ == '__main__':
    data = getArrayFromFile('data_09222015.csv')

    # Take the vertical and horizontal angles (from the data) and map it to the
    #  first quadrant (so that the minimum angle is 0).
    # Add a constant the the V and H to account for discrepancies in plane alignments
```

```
60      V = np.reshape(data[:,0] - min(data[:,0]) - 20, (60, -1))
        H = np.reshape(data[:,1] - min(data[:,1]) + 50, (60, -1))
62      analogReadVals = np.reshape(data[:,2], (60, -1))

64      # Calculate the distance read (the radius) and convert that and the angles
        #  to Cartesian
66      R = np.vectorize(analogReadToDist)(analogReadVals)
        X, Y, Z = convertToXYZ(V,H,R)
68
        # Plot heatmap of front face (2D shape with color to represent depth)
70      plotHeatMap(X,Z,Y)
```