Principles of Engineering Lab 2

Filippos Lymperopoulos, Jay Woo

October 5, 2014

Abstract

The goal of Lab 2 was to get even more comfortable with the Arduino development environment and to be able to design a system that employs sensors and actuators. In order to create our system, the infrared 3d scanner and the two servos were positioned in such a way that we could scan an area in the x and y axes. After attaining the raw data from the Arduino, we then had to convert all the voltages into distances using a calibration curve that we fitted using recorded data. Using this calibration curve, we were then able to create 3d plots of our IR scans.

1 System Overview

In order to make our system, we used a typical IR sensor and two Hobbyking servos to control the mechanical system. The two servos move the sensor in two axes - one turns along the horizontal plane, while the other turns along the vertical plane. All power and ground rails are connected to the Arduino's 5V and GND pins. The sensor is connected to one of the Arduino's analog inputs, while the servos are each connected to one of the digital PWM pins. The following code snippet shows how we setup our pins (the rest of the code is in the appendix):

```
void setup()
{
  // Sensor pin
  pinMode(sensorPin, INPUT);

  // Servo pins
  servoHorizontal.attach(servoPinX);
  servoVertical.attach(servoPinY);

  // Opens serial port, sets baud rate to 9600
  Serial.begin(9600);
}
```

Within the loop function of our main Arduino program, we programmed the horizontal servo to sweep 1 degree to the left, every time the vertical servo sweeps 40 degrees. At each servo angle, the serial monitor displays the sensor reading, and this data is then passed to a Python script.

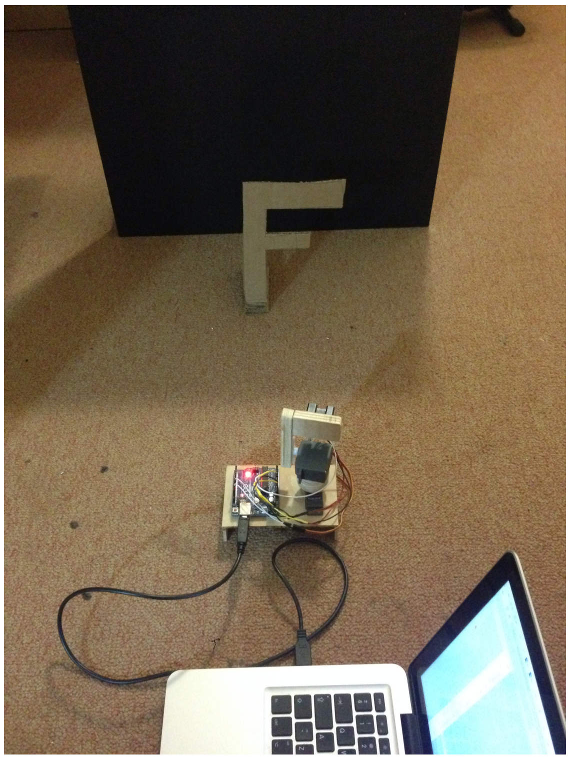

Figure 1: Lab Setup

2 Sensor Calibration

The IR sensor data sheet provided us with several graphs indicating the relation between the recorded voltage over the distance from the sensor to an object. By collecting our own data, we were able to verify this relationship and fit a curve to the data points.

2.1 Distance vs Voltage

We attached the IR sensor to the Arduino to observe the analog voltage from the serial monitor. Using a tape measure, we placed the sensor at the 0 cm mark and had it face a cardboard box. We moved the box in increments of 5 cm, up until 150 cm. After plotting the distances vs the voltages, we attained the following graph and fitted a two-term exponential curve to the data, using MATLAB's curve-fitting tool.

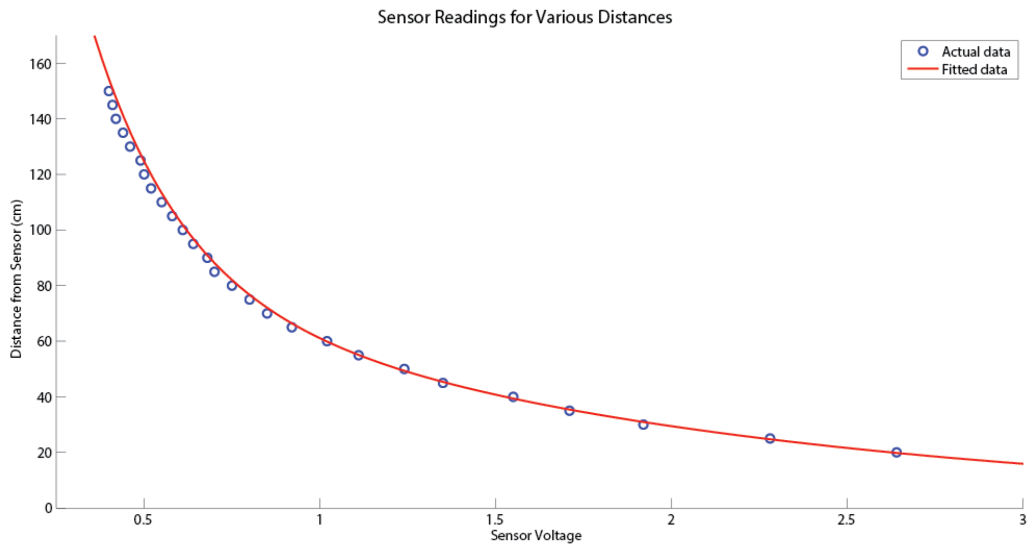

Figure 2: Voltage reading from sensor versus the distance from the box

The relation between the distance (x) to the voltage ($f(x)$) is:

$$f(x) = ae^{bx} + ce^{dx} \tag{1}$$

where $a = 372.7$, $b = -3.962$, $c = 99.51$, and $d = -0.6113$. Using this function, we could then calculate the expected distances from the voltage readings we attained from our calibration process.

2.2 Predicted Distance vs Actual Distance

The following graph shows the relationship between the actual distance from the sensor and the expected distance, given by our fitted curve:

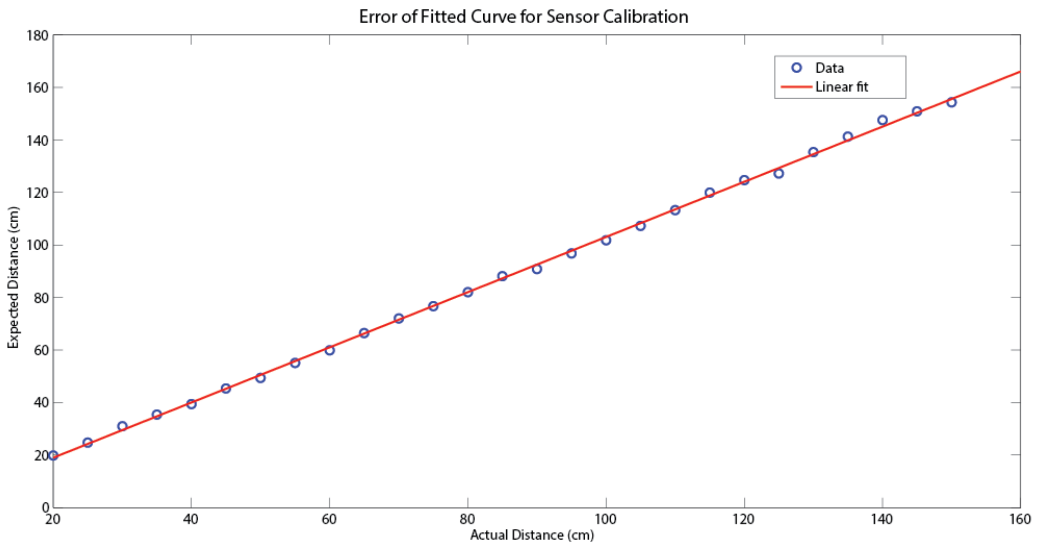

Figure 3: Our error plot

The linear fit had an R^2 value of 0.9993, so our calibration function seems pretty accurate.

3 Mechanical Design

The following image shows our mechanical design:

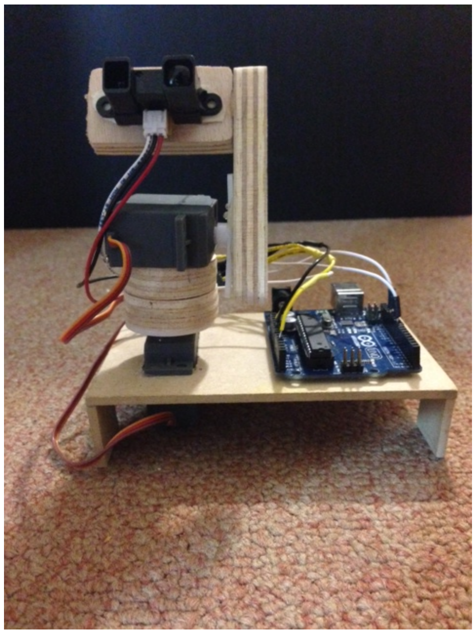

Figure 4: Our creation

The servo on the bottom pans the sensor left and right, while the servo above tilts the sensor up and down.

4 Data Processing

In our Python script, we used pyserial to access the Arduino serial monitor, numpy to facilitate matrix math, matplotlib to plot data, and csv to save the data to a csv file, which can then be passed to MATLAB. Once the Python script has collected all the data from the serial monitor, it then converts each voltage to a distance, using our calibration function.

```
a = 372.2
b = -3.962
c = 99.51
d = -0.6113

for i in data:
       expectedDistance.append(a * math.exp(b * i) + c * math.exp(d * i))
```

We passed all the expectedDistance values to a csv file, which we opened in MATLAB. Afterwards, we made a color scatter plot and a 3d plot of the data points. We thought about converting the spherical coordinates to Cartesian coordinates. However, since the system operated at such short angles, we simply took advantage of small angle approximations - in the end, the data did not look too distorted.

5 3D Plot

The following images show the results from our 3d scanner. We tried scanning our faces, and these were our results:

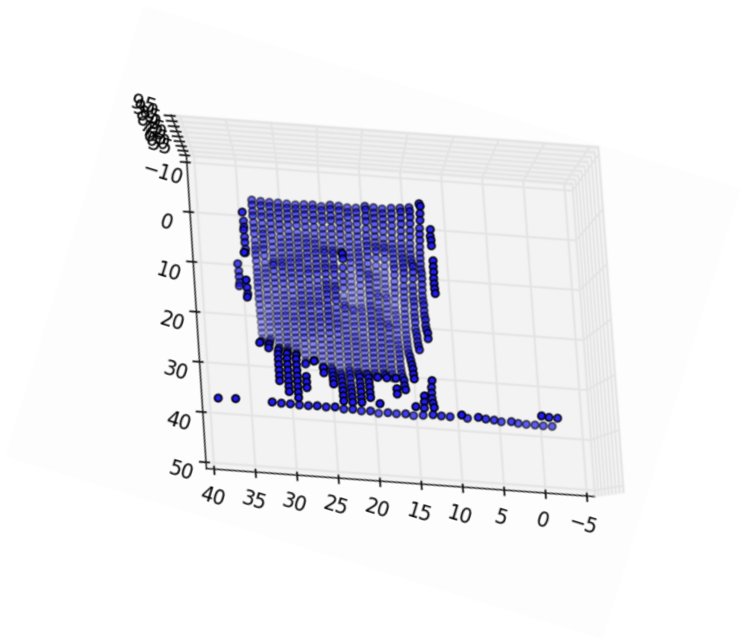

Figure 5: Filippos's face 3D Scan

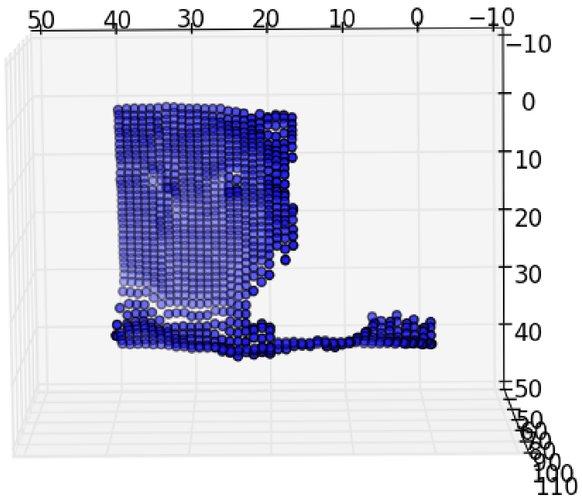

Figure 6: Jay's face 3D Scan

Then, we scanned the letter 'F' and attained the following plots:

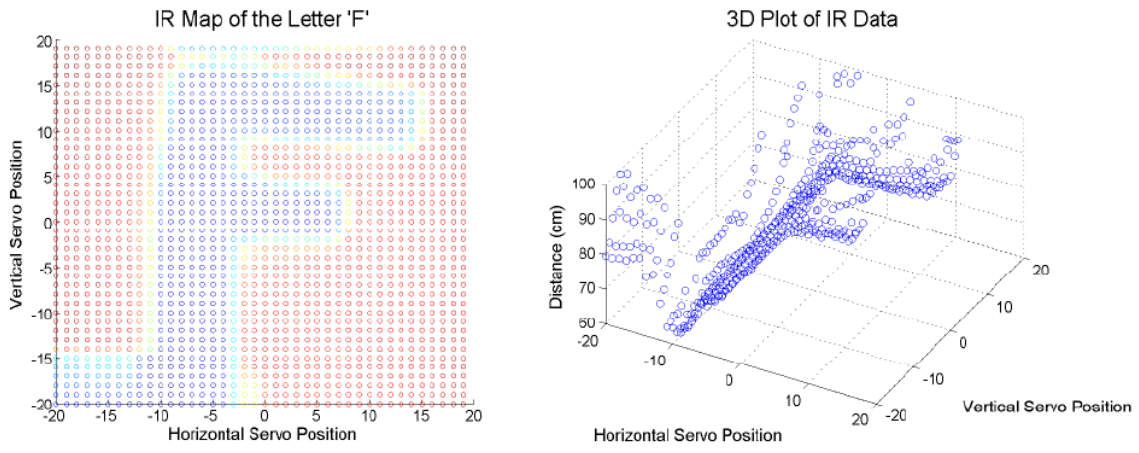

Figure 7: "F" Initial 3D Scan

6 Conclusion

This lab taught us a lot about how to integrate a mechanical component to an electrical system. We see how this could potentially help us for the final project, since we now have a better sense of how to deal with actuators and sensors. Furthermore, we saw how important it was for us to calibrate sensor readings before plotting data - this gave us much more accurate plots.

7 Appendix I - Arduino Code

```
#include <Servo.h>

Servo servoHorizontal;
Servo servoVertical;

const int sensorPin = A1;
const int servoPinX = 13;
const int servoPinY = 9;

// Stores the time to implement delays
long time = 0;

// Time interval between each sensor reading
int sensingInterval = 120;

int counter;
float data_sum;
int servoAngleX;
int servoAngleY;

void setup()
{
  pinMode(sensorPin, INPUT);

  servoHorizontal.attach(servoPinX);
  servoVertical.attach(servoPinY);

  Serial.begin(9600);
}

// Initializes the loop() method.
void loop() {

  // Initializes the servo positions
  servoHorizontal.write(50);
  servoVertical.write(70);

  delay(2000);

  // Keeps track of servo angles to print to the serial
  servoAngleX = -20;
  servoAngleY = -20;

  // Sweeps through the data
  for (int currentXPos = 50; currentXPos < 90; currentXPos++){
    servoHorizontal.write(currentXPos);
    servoVertical.write(70); // Resets the vertical servo

    // Declaration of delay and time variable.
    delay(200);
    time = millis();
    // Vertical sweep
    for (int currentYPos = 70; currentYPos < 110; currentYPos++){
      servoVertical.write(currentYPos);

      // Prints the servo angles to the serial
      Serial.println(servoAngleX);
      Serial.println(servoAngleY);
      servoAngleY++;

      // Averages the data collected over the sensingInterval
      counter = 1;
      data_sum = 0.0;

      while (millis() - time < sensingInterval) {
        counter++;
        data_sum += analogRead(sensorPin) * 5.0/1023.0;
      }

      time = millis();

      // Prints the average value of the data collected
      Serial.println(data_sum / counter);
    }

    servoAngleX++;
    servoAngleY = -20;
  }

  // Halts everything when the scan is done
  while (1){}
}
```

8 Appendix II - Python Code

```
# Importing modules of serial, numpy, matplotlib, math and csv

import serial
import numpy as np
import matplotlib.pyplot as plt
import time
from mpl_toolkits.mplot3d import Axes3D
import math
import csv

# Dimensions of the data matrix
x_size = 40
y_size = 40
totalData_size = x_size * y_size

data = []
expectedDistance = []
servoAngles = []

x_scatter = []
y_scatter = []
z_scatter = []

x_final = []
y_final = []
z_final = []

# Connects Python to the serial port
ser = serial.Serial('/dev/ttyACM2',9600)

# Loops through all the data points - 1st and 2nd line = servo angles (degrees), 3rd
    line = voltage reading
for x in range(totalData_size):
       servoAngles.append(( float(ser.readline()), float(ser.readline()) ))
       data.append(float(ser.readline()))

       ### print data[x], servoAngles[x]

# Converts the voltages to distances using the calibration function
a = 372.2
b = -3.962
c = 99.51
d = -0.6113

for i in data:
       expectedDistance.append(a * math.exp(b * i) + c * math.exp(d * i))

# Meshgrid of x and y array
x_scatter = range(totalData_size)
x_scatter = [n/x_size for n in x_scatter]
y_scatter = range(y_size) * x_size
```

```
# Saves the data to a csv file
csv_file = open("data.csv", "wb")
csv_writer = csv.writer(csv_file, delimiter='\t', quotechar='"', quoting=csv.QUOTE_ALL)

for i in range(totalData_size):
       csv_writer.writerow([x_scatter[i], y_scatter[i], expectedDistance[i]])

# Filters out data points that are more than 100 cm away
for i in range(totalData_size):
       if expectedDistance[i] < 100:
              x_final.append(x_scatter[i])
              y_final.append(y_scatter[i])
              z_final.append(expectedDistance[i])

# Scatter plot of the final data.
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(x_final, y_final, z_final)
plt.show()
```