POE - Lab 2

Ben Kahle and Ry Horsey

September 2014

1 Introduction

In this lab we utilized an IR distance sensor and two servos to create a lidar scanner capable of scanning a 3D environment. The first part of the lab focused on understanding the IR sensor and calibrating its output to produce accurate distance measurements. Once the calibration process was completed and tested, we built a simple scanner with one servo capable of scanning ina horizontal sweep to capture distance data along one axis. We then added the second servo to the scanner in order to scan the full three dimensional space in front of the scanner.

2 The IR Sensor

In order to test the functionality of the IR sensor, we first took a series of measurements at increasing distances to determine the distance to voltage relationship of the sensor. To remove varience between readings, we took 25 samples from the sensor and took the average of the values we recorded. We then took readings at 1 inch intervals from 6 inches through 24 inches. We started at 6 inches (15.18cm), as the sensor is rated to report accurate distances at a minimum distance of 15 cm. While we took the measurements at 1 inch increments, we recorded all of the distances in centimeter values. We read the sensors output voltage into the arduino's analog port which provides digital conversion to an integer within the range 0-1023. We then used these initial measurements to calibrate the output of the sensor as a function of the distance. Finally, we took another series of readings at 1 inch intervals (starting at 6.5 inches) to test the calibrated recorded distance against the actual distance. The calibration and test data are presented below.

2.1 Calibration

To calibrate the IR Sensor we took measurements of the Arduino out port over the serial port at various distances. We decided that we would be taking measurements at a range of about 12 inches. We therefor decided to calibrate from half that distance to twice of it, i.e. from 6 inches to 24 inches. The plot of the collected data can be seen in Figure 1. Using the recorded data, we calculated a first, second, and third order function to approximate the sensor measurement curve. In retrospect, we note that the functions are not ideal representations of the data signaling either an inaccurate fitting algorithm or user error, however we found the third order approximation accurate enough to suit our needs for this lab.

2.2 Implementation

We implemented our calibration code with a first order, second order, and third order approximation of the calibration function. This flexibility allowed us to quickly test the accuracy of the three functions and use the one that works best for a specific distance range. We chose to use the 3rd order function in our scans for this lab because the approximation was the best at the 25 to 60 centimeter range which we were targeting. For a different scan with a different range of distances, a different approximation may provide more accuracy. We implemented a function, distance (line 40), which takes a float, the averaged sensor value, and an integer representing desired approximate order, and returns a distance in centimeters using the approximation function of the specified order. However, while perfectly functional, the software does not mimic the simple elegance of the hardware.

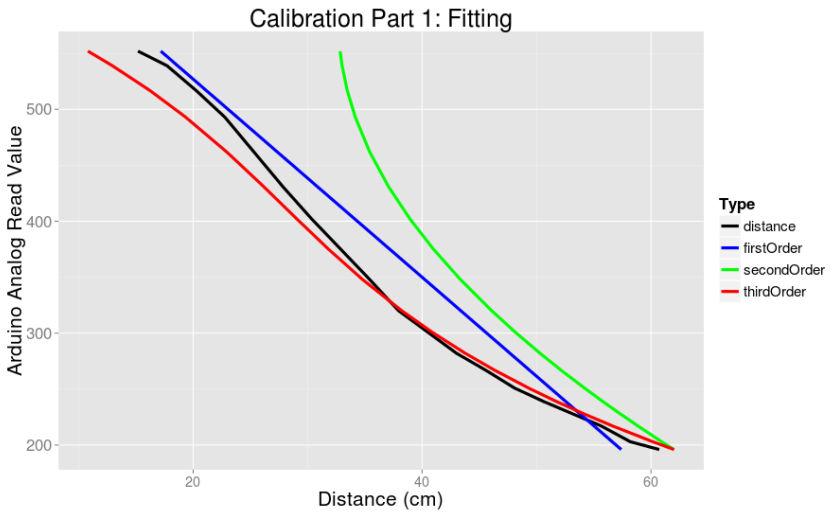

Figure 1: This figure shows the analog signal read by the Arduino from the IR sensor at one inch intervals as well as the three approximation functions.

2.3 Error

To test our calibration function, we followed a similar procedure to that of collecting our sensor data, however we took measurements at half inch increments and recorded the distance output of the calibration function.

Figure 2 shows the error in the first and third degree distance functions using the calibration function to test our interpolations. It is clear that the third order approximation is superior to the first order approximation for the majority of distances above 27 cm, however the linear approximation is far stronger closer to zero, an unsurprising artifact of our chosen interpolation methods. The second degree approximation was considered not worthwhile to implement given its performance as shown in Figure 1.

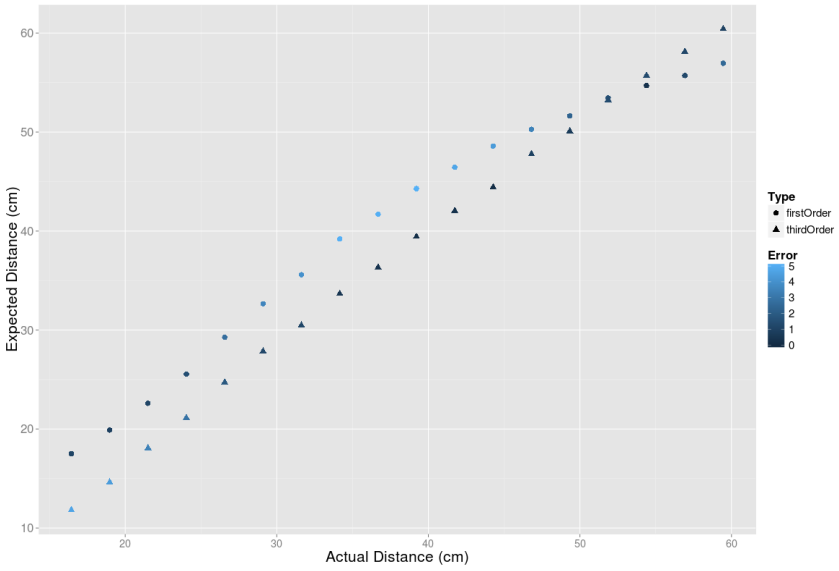

Figure 2: This figure shows the measured and actual distance of our sensor using the calibration function. The circles show the error in the first order approximation, and the triangles the error in the third order approximation.

3 2D scan

We performed a 2D scan of the letter "B" by horizontally sweeping a servo with the IR sensor mounted on it. We took 25 readings from the sensor at each degree of rotation and averaged them to create the final value for the degree. The resulting data plotted in Figure 4 shows the distance from the sensor and the horizontal position.

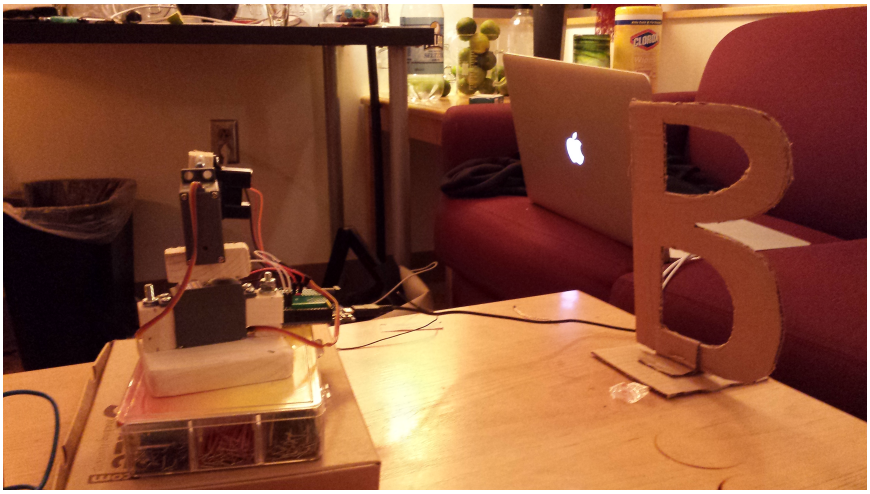

Figure 3: This figure shows the experimental setup used to scan the letter "B" in two dimensions utilizing only one servo.

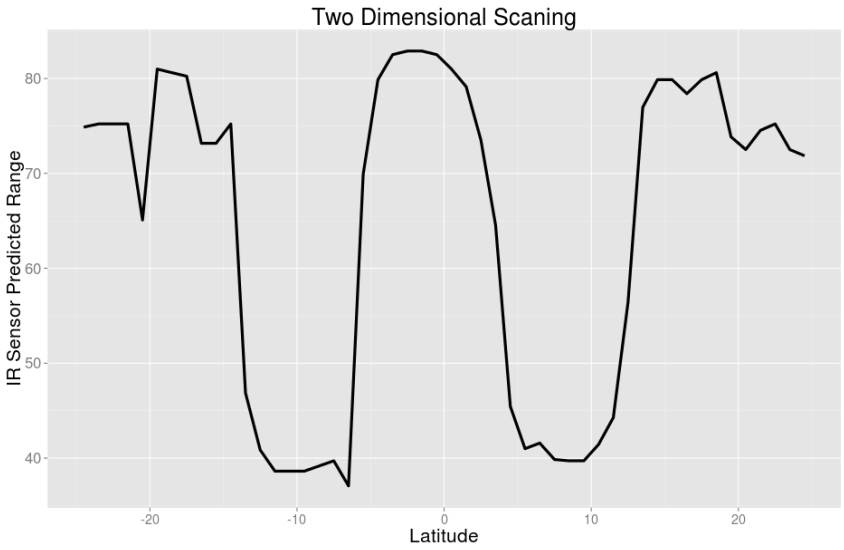

Figure 4: This figure shows the scan taken of the letter "B" in one horizontal sweep by the one servo apparatus. The X-axis is the latitudinal position and the Y-axis is distance from the IR sensor in centimeters.

4 3D scan

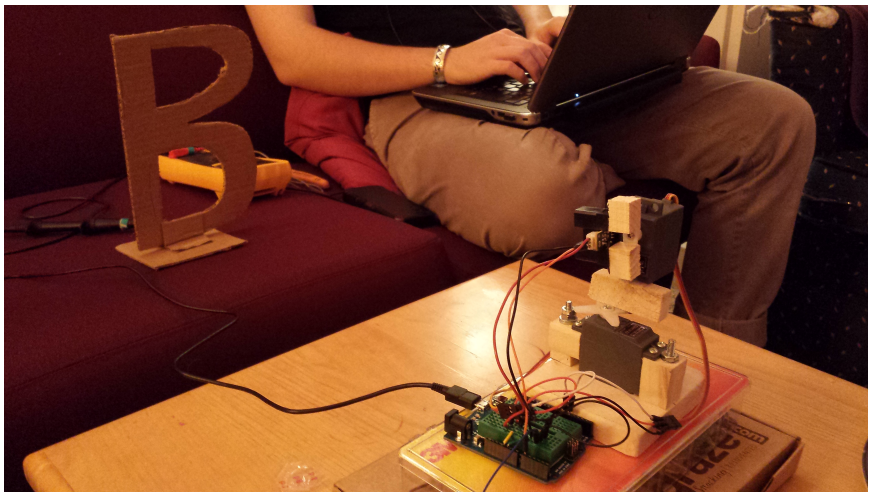

Figure 5: This figure shows the experimental setup used to scan the letter "B" in three dimensions.

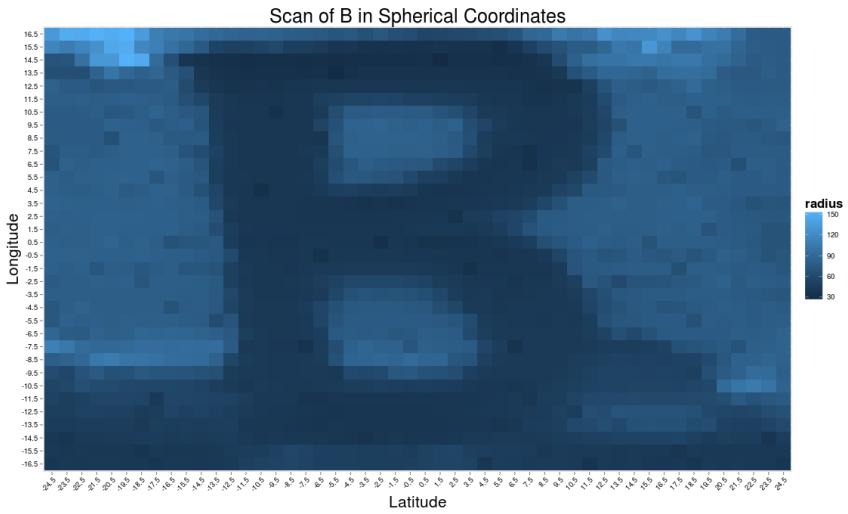

Figure 6: This figure shows the scan taken of the letter "B" by our two servo apparatus. The B is clearly visable in this image, however it is interesting to note the difficulty that the IR sensor has differentiating distances beyond approximately 60 centimeters. This is unsuprising given Figure 2 in the sensor's datasheet, which shows the decreasing diferentiation between subsequent values at advanced distances.

5 Conclusion

A goal we attempted to achieve in this lab was simplicity in design. We wanted to build the scanner with an elegant and robust system of hardware and software. To accomplish this in building the physical components, we designed an orientation of the servos that allowed us to attach them with minimal amounts of material (wood) in easily manufactured shapes while still keeping the axes of rotation of the sensor constant. This design is small, structurally sound, and was quick to manufacture with easily accessible materials. In software, we were not as successful in creating an elegant solution during the scope of this lab. We wrote the "scan" function (line 64), which will sweep any servo from any start point to any end point regardless of direction. This abstraction allowed us to easily sweep horizontally for our 2D scan as well as create a simple raster movement for the 3D scan. However, there is still improvement that could be made to create a software system capable of being more easily utilized in a wider array of use cases.

6 Appendix

6.1 Arduino Code

```
1  #include <Servo.h>
2
3  int sensorPin = A0;    // select the input pin for the potentiometer
4  int sensorValue = 0;  // variable to store the value coming from the sensor
5  const int servoPin1 = 5;
6  const int servoPin2 = 6;
7  const int samples = 25; // Number of samples to average
8  const int approxOrder = 3; // Orer of distance approximation function (1,2,3)
9  int values[samples];
10 Servo servo1;
11 Servo servo2;
12 int servo1Pos = 0;
13 int servo2Pos = 0;
14
15 void setup() {
16   Serial.begin(9600);
17   servo1.attach(servoPin1);
```

```
18    servo2.attach(servoPin2);
19  }
20
21  /*
22  * Calculate the average value of a list of integers
23  */
24  float average(int values[]) {
25    int sum = 0;
26    int length = sizeof(values)/sizeof(int);
27    float averageValue = 0;
28    for (int i = 0; i < length; i++) {
29      sum += values[i];
30    }
31    averageValue = sum/length;
32    return averageValue;
33  }
34
35  /*
36  * Tranform the IR sensor measurement into centimeters
37  * value: the value of the IR sensor
38  * order: the order of the approximation function to use (1,2,3)
39  */
40  float distance(float value, int order) {
41    float dist;
42    switch (order) {
43      case 3:
44        dist = -0.000001211*pow(value,3)+0.0015*pow(value,2)-0.7193*value+154.49;
45        break;
46      case 2:
47        dist = 0.0002*pow(value,2)-0.2315*value+99.687;
48        break;
49      case 1:
50        dist = -0.113*value+79.55;
51        break;
52      default:
53        return 0;
54        break;
55    }
56    return dist;
57  }
58  /*
59  * Do a full scan of one direction, printing servo angles and distance in cm at each degree
60  * type: 0 = horizontal; 1 = vertical
61  * start: start degree for server
62  * end: end degree for server
63  */
64  void scan(int type, int start, int end) {
65    Servo servo;
66    int *servoPos;
67    if (!type) {
68      servo = servo1;
69      servoPos = &servo1Pos;
70    } else {
71      servo = servo2;
72      servoPos = &servo2Pos;
73    }
74    int averageValue;
75    int high;
76    int low;
77    int add = 0;
78    if (start <= end) {
79      add = 1;
80    } else {
81      add = 0;
82    }
83    *servoPos = start;
84    while(1) {
85      if (*servoPos == end) {
```

```
      break;
    }
    servo.write(*servoPos);
    for (int i = 0; i < samples; i++) {
      values[i] = analogRead(sensorPin);
      delay(2);
    }
    averageValue = average(values);
    Serial.print(servo1Pos);
    Serial.print(",");
    Serial.print(servo2Pos);
    Serial.print(",");
    Serial.println(distance(averageValue, approxOrder));
    if (add) {
      (*servoPos)+=1;
    } else {
      (*servoPos)-=1;
    }
  }
  //Give time to reset to start
  servo.write(start);
  delay(300);
}

void loop() {
  int vertStart = 120;
  int vertEnd = 85;
  int hozStart = 180;
  int hozEnd = 130;
  //Start in bottom right corner
  servo2.write(vertStart);
  servo1.write(hozStart);
  //Wait for servos to reach start position
  delay(1000);
  //Print new lines between runs for easy splitting
  for (int k = 0; k < 20; k++) {
    Serial.println("");
  }
  //For each vertical degree, do a horizontal scan.
  for (int i = vertStart; i != vertEnd; i -= 1) {
    servo2Pos = i;
    servo2.write(servo2Pos);
    delay(5);
    scan(0, hozStart, hozEnd);
  }
  delay(10000);
}
```

6.2 Processing and Plotting Code

```
#Postprocessing for PoE Lab 2
require(ggplot2)
require(grid)
require(sphereplot)

#Inport, recenter, and flip scan data
scanData = read.csv("dataIn.csv", header=FALSE)
processingDf = data.frame()
centerX = diff(range(scanData[, 1])) / 2 + min(scanData[, 1])
centerY = diff(range(scanData[, 2])) / 2 + min(scanData[, 2])
processingDf[1:nrow(scanData), "xNorm"] = scanData[, 1] - centerX
processingDf[1:nrow(scanData), "yNorm"] = scanData[, 2] - centerY
processingDf[1:nrow(scanData), "radius"] = scanData[, 3]
processingDf[1:nrow(scanData), "xNorm"] = processingDf[1:nrow(scanData), "xNorm"] * -1
processingDf[1:nrow(scanData), "yNorm"] = processingDf[1:nrow(scanData), "yNorm"] * -1

```

```
#Plot scaned data
ggplot(processingDf, aes(x=factor(xNorm), y=factor(yNorm), fill=radius))+
  geom_tile()+
  labs(x="Latitude", y="Longitude", title="Scan of B in Spherical Coordinates")+
  theme(axis.title.x = element_text(size=rel(1)), axis.title.y=element_text(size=rel(1)),
      title = element_text(size=rel(1.75)), axis.text.x = element_text(size=rel(1),angle=45,
       vjust=1, hjust=1, colour="Black"), axis.text.y = element_text(size=rel(1), colour="
      Black"))

#Calibration Fitting Plot
ggplot(newDf, aes(x=data,y=yValue,color=Type))+
  geom_line(size=1.5)+
  labs(x="Distance (cm)",y="Arduino Analog Read Value",title="Calibration Part 1: Fitting")+
  theme(legend.text=element_text(size=rel(1.25)),axis.title.x = element_text(size=rel(1)),
      axis.title.y = element_text(size=rel(1)), title = element_text(size=rel(1.75)), axis.
      text.x = element_text(size=rel(1.5)), axis.text.y = element_text(size=rel(1.75)))+
  scale_color_manual(values=c("Black", "Blue", "Green", "Red"))

#Calibration Error Plot
errorDf = read.csv("errorIn.csv", header=FALSE)
colnames(errorDf) = c("dist", "est", "Type")
temp_index = nrow(errorDf)
errorDf[(temp_index+1):(temp_index+2), "dist"] = c(min(errorDf[, "dist"]), max(errorDf[, "
    dist"]))
errorDf[(temp_index+1):(temp_index+2), "est"] = c(min(errorDf[, "dist"]), max(errorDf[, "
    dist"]))
errorDf[,"Error"] = abs(errorDf[,"dist"] - errorDf[,"est"])
ggplot(errorDf, aes(x=dist, y=est, colour=Error, shape=Type))+
  geom_point(size=rel(3.5), na.rm=TRUE)+
  labs(x="Actual Distance (cm)", y="Expected Distance (cm)", c)+
  theme(legend.text=element_text(size=rel(1.25)),axis.title.x = element_text(size=rel(1)),
      axis.title.y = element_text(size=rel(1)), title = element_text(size=rel(1.75)), axis.
      text.x = element_text(size=rel(1.5)), axis.text.y = element_text(size=rel(1.75)))

#OneD Scan Plot
oneDimDf = read.csv("oneDimScan.csv")
ggplot(oneDimDf, aes(x=lat,y=radius))+
  geom_line(size=1.5)+
  labs(x="Latitude",y="IR Sensor Predicted Range",title="Two Dimensional Scaning")+
  theme(legend.text=element_text(size=rel(1.25)),axis.title.x = element_text(size=rel(1)),
      axis.title.y = element_text(size=rel(1)), title = element_text(size=rel(1.75)), axis.
      text.x = element_text(size=rel(1.5)), axis.text.y = element_text(size=rel(1.75)))
```