Lab 2 - A DIY 3D Scanner

Noah Rivkin, Seungin Lyu

September 2017

1 Introduction

In this lab we build a 3D scanner. We use two servo motors, a infrared distance sensor, and an Arduino micro-controller. We then scan a large cut-out letter, and use python script that receives the serial output to visualize the scanned results.

2 Procedures

We began by designing the scanner. We planned to have a close to stationary position for the scanner. We used one servo to rotate horizontally, and the other rotate vertically with an attached IR distance sensor, respectively representing θ, ϕ, r in the spherical coordinate system. We chose to design the scanner with the spherical coordinate system because using r, θ, ϕ makes visualization intuitive when both the servos illustrate rotational motion.

After we were done with building the basic mechanical prototype of the 3D scanner, we moved onto calibrating the IR distance sensor. By calibration we mean the process of mapping the analog output(ranging from 0 to 1023, with output voltage ranging from 0 to 2.7V according to the datasheet) to an actual distance. We found in the datasheet that the sensor is capable of measuring distance from a minimum of 20cm to a maximum of 150cm.

In order to calibrate the distance sensor we aimed the sensor directly at the wall. We started at a distance of 1 cm, and incremented the distance by 1 cm. For the first 10 cm the analog output was unreliable, but between 10 and 15 cm it started to stabilize. By 15 cm the analog output was 560, and had a slope of -10. We confirmed that the near linear behavior continued to at least 45 cm. Using this information, we designed a calibrated mapping function that converts the analog input to a radius value in cm.

$$\mathrm{r} = \frac{15 + (560 - analog_{out})}{10} \tag{1}$$

We designed the function bearing in mind that the shortest distance that could accurately be measured was 15cm, where the analog output was 560. We also took account of the slope of -10. Also please note that the datasheet presents a more of exponential-decay graph for the calibration. However, we stick to the linear calibration model because it is pretty accurate for the range from 15cm to 50cm, which is a range sufficient for the scope of this project. As shown in Figure 2, we confirmed that our calibration function is approximately accurate by extrapolating the data and comparing it with the actual measured values that weren't used in the initial calibration process.

We then attached the distance sensor to one servo motor that rotates from top to bottom within 50 degrees from its initial position as shown in Figure 3. Simply, we only manipulate the ϕ components of our system. We scanned the letter 'L' (which is one of our initials) and plotted the top-down view as shown in Figure 4. We scanned the right-end of the letter. Notice the cluster of points that represent the scanned letter and where the sensor could not scan anything due to the prevalent void space of the letter 'L'.

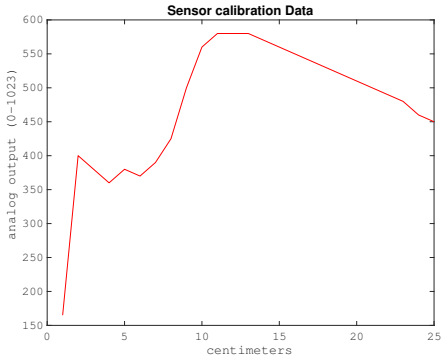

Figure 1: The data we gathered for calibration purposes. The sensor is unreliable at distances under 15cm.

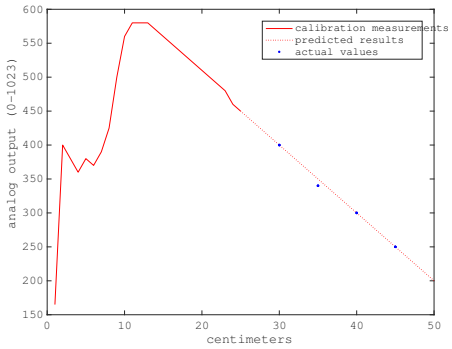

Figure 2: Our extrapolation from the original calibration data and the actual data found.

Figure 3: Image of our setup for the 1-servo scan of the letter

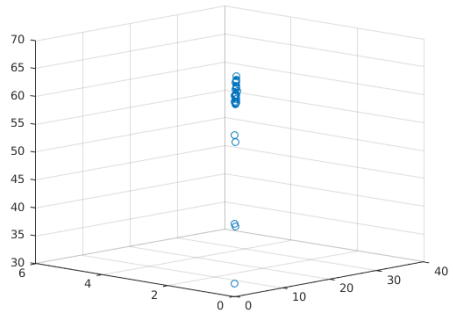

Figure 4: A Top-down view of the 1-servo scan of the letter

We then attached the distance sensor to the complete system with two servo motors that each rotates within 50 degrees from its initial position as shown in Figure 5. We scanned the letter 'L' with the setup shown in Figure 6. Then we rendered the full 3D scatter plot as shown in Figure 7 using a python script. We used python's serialpy module to read serial input (r, θ, ϕ) from the Arduino, then implemented our own mapping function that converts from spherical coordinate to cartesian coordinate with numpy module, and lastly generated the plot using matplotlib module. We included the detailed code implementation in our appendix. We admit that Figure 7 was not scanned with the best possible resolution but it still provides a good sense of how the letter 'L' is shaped.

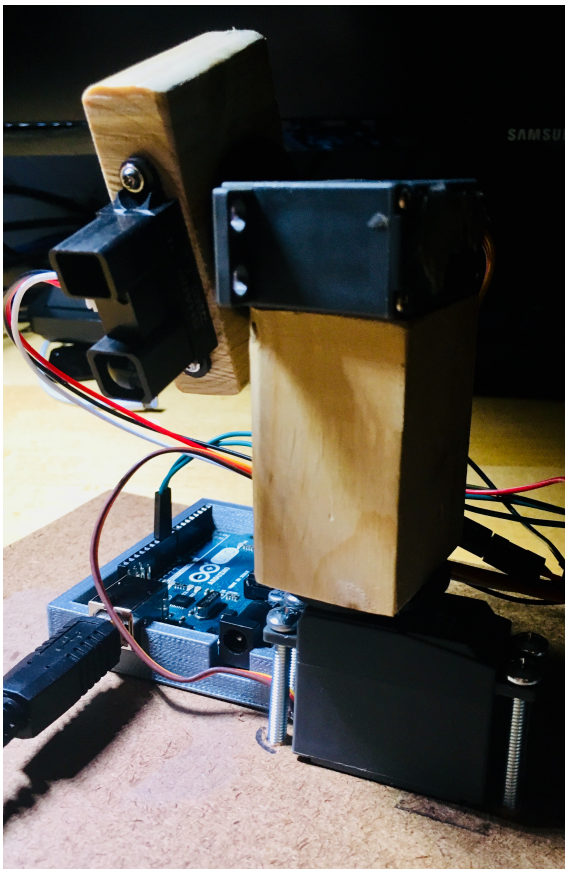

Figure 5: Image of out setup for the 2-servo scan of the letter

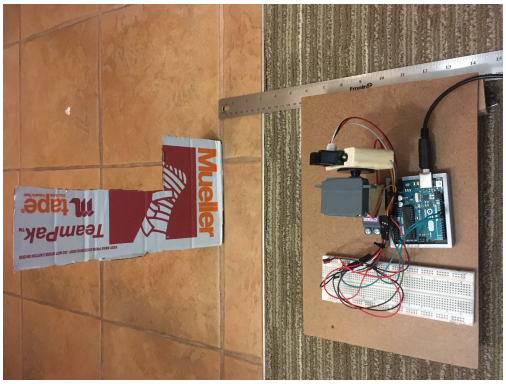

Figure 6: Setup for 3D scatter plot of the 2-servo scan of the letter

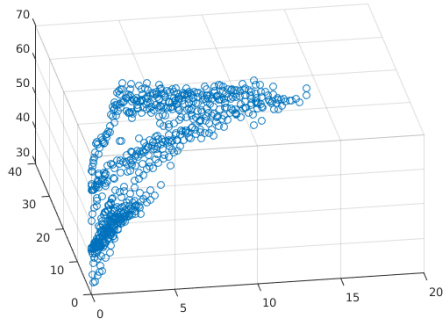

Figure 7: Our extrapolation from the original calibration data and the actual data found.

3 Reflection

- Seungin Lyu : I really enjoyed building the 3D scanner. I learned that the goal of making physical prototypes isn't necessarily related to jump right into 3D printing or fancy cadding. Without any 3D printing or laser cutting, we were able to produce a sturdy, stable, and functional scanner with only two cut wood blocks and some hot glue (in addition to the servos and the sensor). We even cut out our test letter from a cardboard! I believe our approach was rather simple but got the job done. However, I'm aware that such light-weighted approach might not be the best practice for 'big projects' when manufacture precision might be the key to success. I really enjoyed the software part of this lab as well, except for the serial output integration that resulted in numerous encounters with tedious bugs. In general, I wish I had produced a better final visualization of the scanned result, but I'm proud that the scanner is actually working!

- Noah Rivkin : This lab was an interesting mix of hardware and software. I had the opportunity to work on both the mechanical and the electrical sides of the project. I think that it is important to understand how software interacts with physical systems, especially as IoT devices become more prominent. It was very rewarding to see it all come together, and produce a scan. I greatly enjoyed this lab.

4 Appendix

1. Sharp GP2Y0A02YK0F IR distance sensor :`https://media.digikey.com/pdf/Data%20Sheets/Sharp%20PDFs/GP2Y0A02YK0F.pdf`

2. Hobbyking HK15138 servo motors:https://hobbyking.com/en_us/hobbykingtm-hk15138-standard-analog-servo-

3. One-servo mode:

```
#include <Servo.h>

const byte POT = A0;

const byte SERVO_SPEED = 50;  //minimum number of milliseconds per degree)

const byte PHI_INCREMENT = 1;
const byte PHI_MAX = 50;
const byte PHI_MIN = 0;

byte curr_phi = 50;


boolean flag = 0;
int pot_value;
String result = "";


Servo servo_theta;
Servo servo_phi;  // create servo object to control a servo

void setup() {
  servo_phi.attach(10);  // attaches the servo on pin 10 to the servo object
  servo_phi.writeMicroseconds(700);
  Serial.begin(9600);
}


void loop() {
  static unsigned long servo_time;
  // check time since last servo position update
  if ((millis() - servo_time) >= SERVO_SPEED) {
    servo_time = millis();
    if (curr_phi == PHI_MAX || curr_phi == PHI_MIN) {
      flag = (flag + 1) % 2; // Everytime the vertical servo swipes from top to bottom,
          flag is toggled.
    }
    curr_phi = curr_phi + (flag * (-2 * PHI_INCREMENT)) + PHI_INCREMENT;
    servo_phi.write(curr_phi);
    pot_value = analogRead(POT);
    result = String(pot_value) + "," + String(curr_phi);
    Serial.println(result);
  }
}
```

4. Final Arduino Code :

```
#include <Servo.h>

const byte POT = A0;

const byte SERVO_SPEED = 50;  //minimum number of milliseconds per degree)
const byte THETA_MAX = 50;
const byte THETA_MIN = 0;
const byte THETA_INCREMENT = 1;

const byte PHI_INCREMENT = 1;
const byte PHI_MAX = 50;
const byte PHI_MIN = 0;


byte curr_theta = 50;
```

```
byte curr_phi = 50;


boolean flag = 0;
int pot_value;
String result = "";


Servo servo_theta;
Servo servo_phi;  // create servo object to control a servo

void setup() {
  servo_theta.attach(9); // attaches the servo on pin 9 to the servo object
  servo_phi.attach(10);  // attaches the servo on pin 10 to the servo object
  servo_theta.writeMicroseconds(700);
  servo_phi.writeMicroseconds(700);
  Serial.begin(9600);
}


void loop() {
  static unsigned long servo_time;
  // check time since last servo position update
  if ((millis() - servo_time) >= SERVO_SPEED) {
    servo_time = millis();
    if (curr_phi == PHI_MAX || curr_phi == PHI_MIN) {
      flag = (flag + 1) % 2; // Everytime the vertical servo swipes from top to bottom,
           flag is toggled.
      curr_theta = (curr_theta + THETA_INCREMENT) % THETA_MAX;
      servo_theta.write(curr_theta);
    }
    curr_phi = curr_phi + (flag * (-2 * PHI_INCREMENT)) + PHI_INCREMENT;
    servo_phi.write(curr_phi);
    pot_value = analogRead(POT);
    result = String(pot_value) + "," + String(curr_theta) + "," + String(curr_phi);
    Serial.println(result);
  }
}
```

5. Python Serial Receiver Code :

```
import serial
import numpy as np
import matplotlib.pyplot as plt
import csv
from mpl_toolkits.mplot3d import axes3d


def convert_to_cartesian(r, theta, phi):
    """
    Converts a tuple of values in spherical coordinates to cartesian coordinates,
    Returns an iterator of x, y, z in order.
    """
    theta = theta * (np.pi / 180)
    phi = phi * (np.pi / 180)
    r = 15 + (560 - r) / 10  # calibrated for distances over 15 cm
    x = r * np.cos(theta) * np.sin(phi)
    y = r * np.sin(theta) * np.sin(phi)
    z = r * np.cos(phi)
    return iter([x, y, z])


def parse_line(line):
    """
    Parses the serial monitor output and returns tuple of three values (r, theta, phi)
    """

    line = line.decode('utf8').strip("b\'\n\r")
```

```
    coordinates = []
    for coordinate in line.split(','):
        coordinates.append(int(coordinate))
    coordinates = (coordinates[0], coordinates[1], coordinates[2])
    return coordinates


ser = serial.Serial('/dev/ttyACM0', baudrate=9600, timeout=20)
print("connected to: " + ser.portstr)
points = []

while True:
    line = ser.readline()
    coordinates = parse_line(line)
    points.append(coordinates)
    if coordinates[1] == 49 and coordinates[2] == 0: // break when 50 degree rotation
        is over
        break

ser.close()


fig = plt.figure()
ax = fig.add_subplot(1, 1, 1, projection='3d')
result = []

for r, theta, phi in points:
        data = convert_to_cartesian(r, theta, phi)
        x, y, z = data
        result.append((x, y, z))
        ax.scatter(x, y, z, c='r', marker='o')

// saves results into csv files
with open('sphericals.csv', 'w') as f:
    writer = csv.writer(f, delimiter=';', lineterminator='\n')
    writer.writerows(points)

with open('result.csv', 'w') as f:
    writer = csv.writer(f, delimiter=';', lineterminator='\n')
    writer.writerows(result)

// generates the 3d scatter plot
ax.set_xlabel('X (cm)')
ax.set_ylabel('Y (cm)')
ax.set_zlabel('Z (cm)')
plt.show()
```