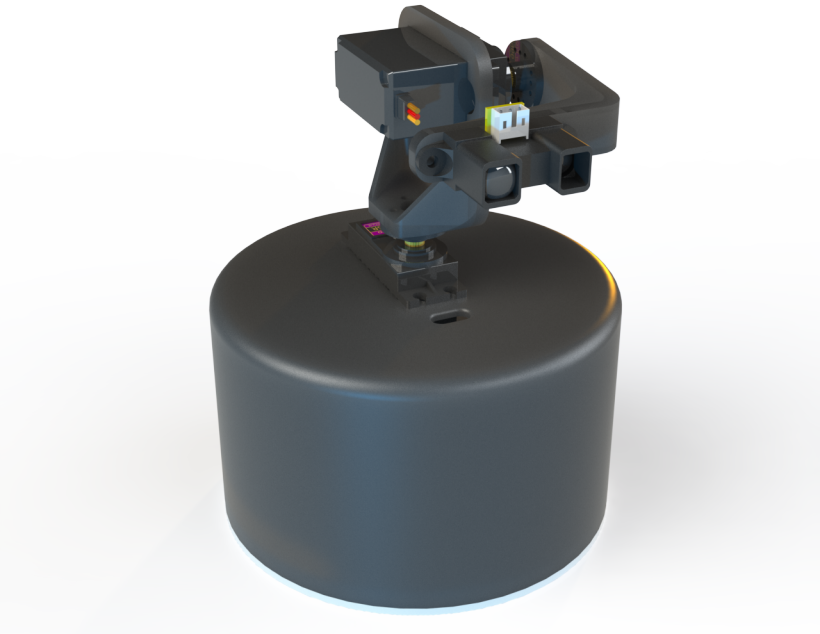
# DIY 3D Scanner

Rohith Tatineni, Trevor Zou, and Van Myers

October 1, 2021



## 1 Testing the sensor

To test the sensor, we attached the sensor to Analog input 9 and used the provided analogue input tutorial sketch, which mapped the voltage readings from 0 to 255 and changed the blink frequency of an LED based on that reading. Instead of using the LED, we printed the mapped values to the serial monitor. We could then get voltage readings from the sensor when placed at various distances away from the wall, and use the data to form a calibration curve.

## 2 Error Plot and Calibration Function

We used Matlab's curve fitting tool to fit a linear and exponential curve to the data. The spec sheet for the sensor had what looked like an inverse exponential curve. We had an

only slightly greater $R^2$ value for the linear fit (.9978) than the exponential fit (.9891), but when we looked at the error on two testing points, we saw that the linear fit was would actually be a much more accurate fit.

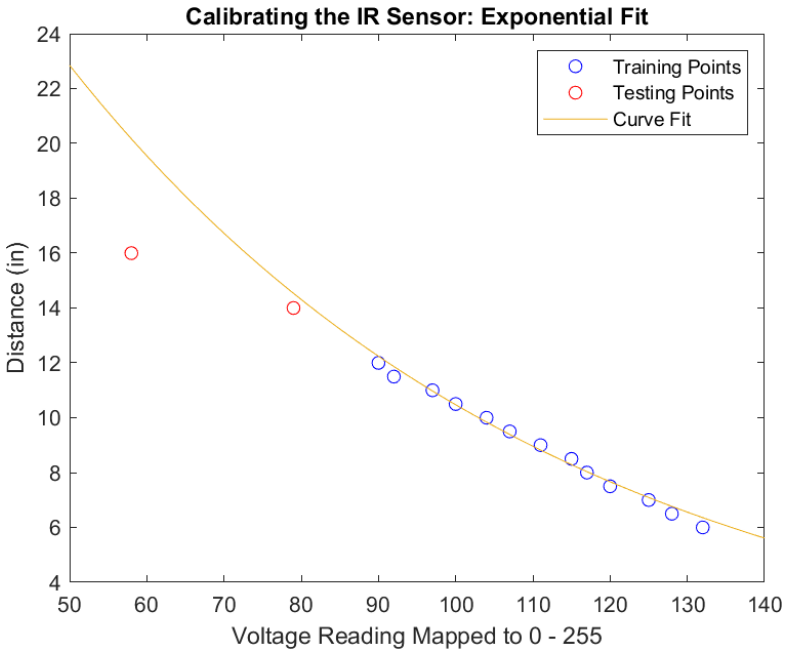



Figure 1: Exponential fit with $r^2 = .9891$

$$Distance = 49.84e^{-.0156V_{mapped}}$$

We figured that for the range of distances we would be using our scanner at, a linear fit would be the most useful. In the end, the equation for our fit curve was:

$$Distance = 24.75 - .1423V_{mapped}$$

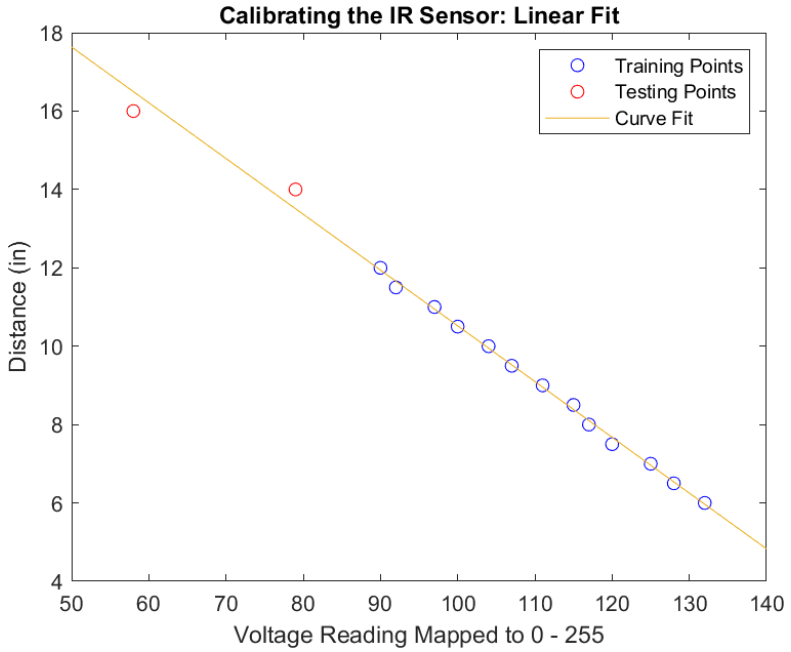



Figure 2: Linear fit with $r^2 = .9978$

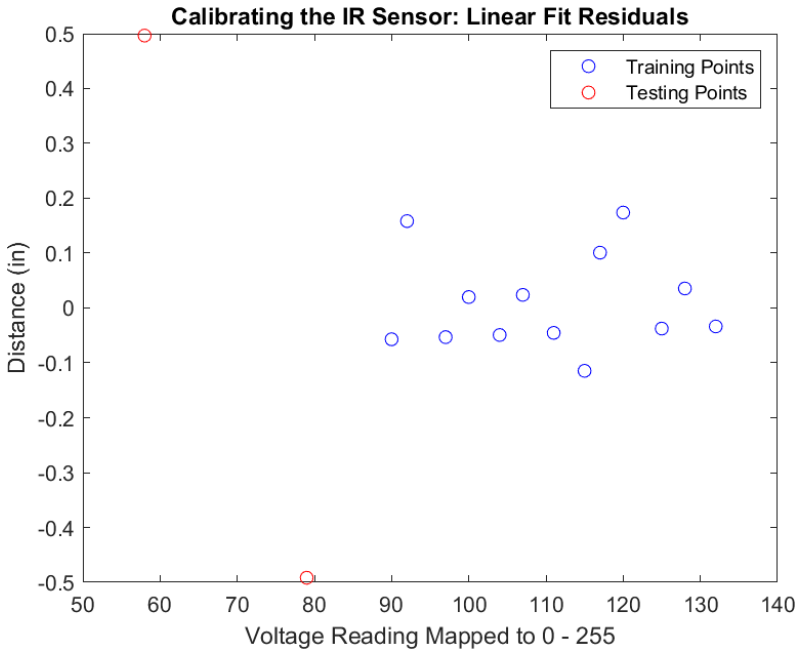



Figure 3: Residual plot of our final linear fit.

Note that this is the distance from the sensor to an object, but not the distance from our plotting origin to the object, because our setup does not have the sensor at the center

of rotation.

# 3 Setup

## 3.1 Electronic Setup

Below is the schematic drawing of the the circuit we used to create the DIY 3D scanner. The circuit is comprised of 2 servos, an IR distance sensor, and an Arduino UNO rev 3.

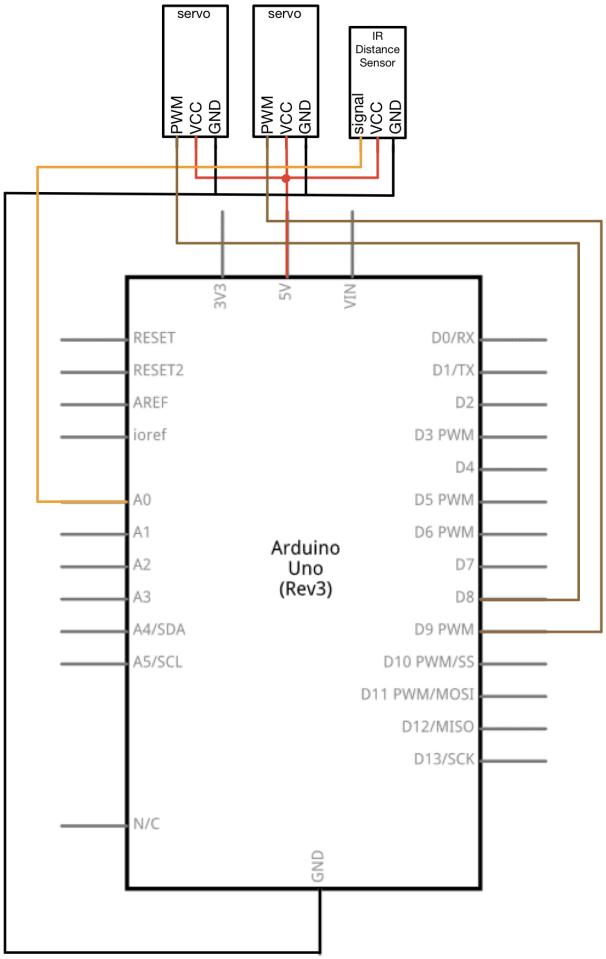



Figure 4: Schematic diagram of the circuit

## 3.2 One-Servo Sweep

We designed our mechanism to be based around 2 servos, so instead of sweeping one servo from a top down view, we swept our pan mechanism 40 degrees from center on

each side while keeping our tile mechanism at a constant 25 degrees. We lined the sensor up so that it would sweep across the skinny part of the T.

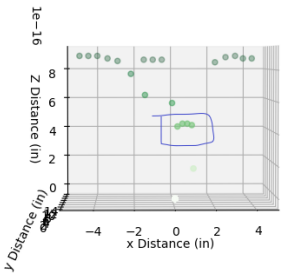



Figure 5: One Servo Sweep front view

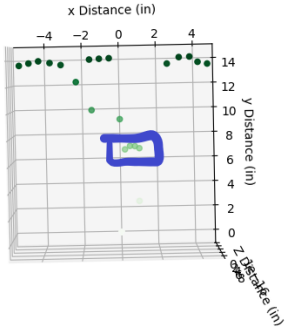



Figure 6: One Servo Sweep Top View

This logically makes sense– Points are for the most part co planar with the YZ plane, which makes sense. The X ranges from -.5 to 1 in. We measured by the stem of the T to be about the same. We also got rid of a lot of noise so they wouldn't plot.

## 4 Scanning

To do our full scan, we put cardboard cutout of our T about 10 inches away from our leveled sensor, and swept our pan and tilt angles 20 degrees and 40 degrees respectively from center on each side, starting from the top right and ending at the bottom left. We moved in increments of 2 degrees in the tilt and 1 degree in the pans. At each stop, the Arduino sent the scaled voltage reading and both angles to the serial port, and it was received using our python script. The python script converted the voltage reading to a distance using our calibration fit, and then converted that distance, along with the two angles, from spherical to Cartesian coordinates. During this step we added a constant to the radius because the sensor sticks out past the center of rotation. This means that our transformed coordinates have a common origin and can be plotted.

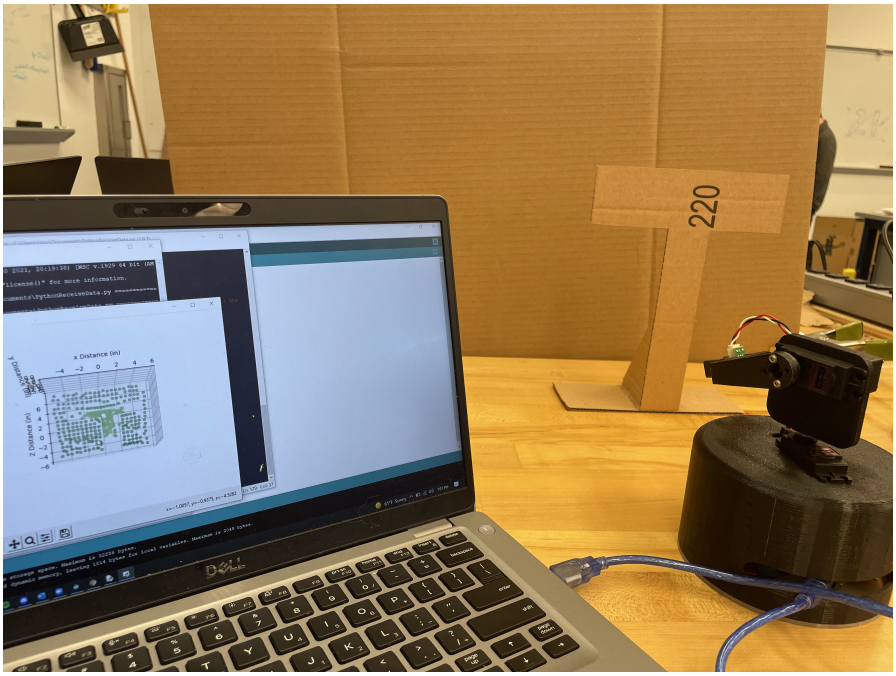



Figure 7: Our scanning and live plotting setup.

We then plotted those in 3d space using MatPlotLib. In order to reduce noise, we didn't plot any points greater than 20 inches away from our plotting origin. We also noticed that the scanner distance reading frequently spiked even when held at a constant distance, so at each stop we also took 5 readings and used the minimum reading of the batch in an effort to minimize their effect. It helped a little but there was still quite a bit of noise when transitioning from the wall to our T.We also had a plotting cutoff at 800 points in order to avoid cluttering the window with points on the table. We also tried to shade with a gradient in proportion to each point's y coordinate in order to make the background points darker than the T points.

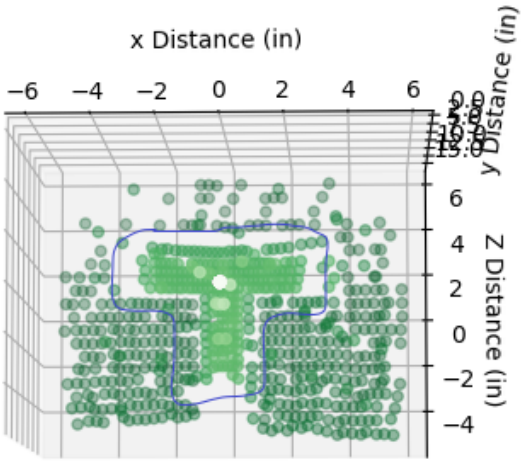



Figure 8: 3D Scan front view

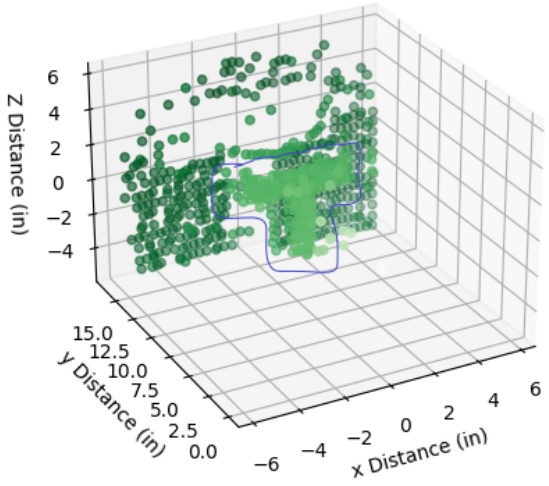



Figure 9: 3D scan Isometric View

We can see that our scan did a pretty good job. Our T was about 8 inches tall and 6 inches across, so the proportions and scaling seems pretty good.

# 5 Reflection

The hardest part was probably just figuring out the spherical to cartesian coordinate transformations, for a while the points were plotted facing the sky. Our team structure was very effective for this mini-project. Rohith did a great job with the mechanical design and had a working platform for us to test the scanning programs early in the project. Trevor and Van were able to branch out and pick up additional programming skills. Van had never used Python before. They learned some basic numpy and matplotlib. Our process reinforced Daniela Faas's advice last year. She said that having a strong mechanical engineer for the core of your team makes PIE projects ten times easier because the rest of the team doesn't have to depend on writing simulations for everything and hoping for it to work in practice at the last minute. Although we did write a simple program to feed dummy data to our plotting program, having a completed hardware system made validation simple and represenatitive of final performance.

# 6 Source Code

## 6.1 Arduino

```
#include <Servo.h>

Servo myservo1;  // create servo object to control a servo
Servo myservo2;
int pos1 = 0;    // variable to store the servo position
int pos2 = 0;
uint16_t LOOP_INTERVAL = 300;
int analogInPin = A0;    // select the input pin for the distance sensor
int sensorValue = 0;  // variable to store the value coming from the sensor

void setup() {
  Serial.begin(9600);
  myservo1.attach(9);
  myservo2.attach(8);
}
void loop() {
     uint16_t x, y, z, a, b, res, dist;
  // put your main code here, to run repeatedly:
  for ( int i = 50; i <= 90; i=i+2) {
         myservo2.write(i);
         delay(LOOP_INTERVAL);
   // statement block
  for ( int j = 90; j <= 130; j=j+1) {
     myservo1.write(j);
      x = map(analogRead(analogInPin), 0, 1023, 0, 255);;
      y = map(analogRead(analogInPin), 0, 1023, 0, 255);;
      z = map(analogRead(analogInPin), 0, 1023, 0, 255);;
      a = map(analogRead(analogInPin), 0, 1023, 0, 255);;
      b = map(analogRead(analogInPin), 0, 1023, 0, 255);;
      // take the minimum of three scans to remove outliers.
      res = min(min(min(min(x, y), z), a), b);
      Serial.print(res);    Serial.print(",");
      // adjust angles to match refrence frame
      Serial.print((i-70)*-1);    Serial.print(",");
      Serial.println(-1*(j-110));
      delay(LOOP_INTERVAL);
   }
}
exit(0);
}
```

## 6.2 Python

```
import serial
import matplotlib.pyplot as plt
import numpy as np
from mpl_toolkits import mplot3d
from matplotlib.animation import FuncAnimation
import collections
```

```
# maximum points to plot
pointnumb = 800
counter = 0

def pol2cart(r, theta, phi):
    # convert angles and distance to cartesian coordinates
    return [
         (tiltlen + r) * np.sin(phi) * np.cos(theta),
         (tiltlen + r) * np.sin(phi) * np.sin(theta),
         (tiltlen + r) * np.cos(phi)
         ]

def motor2xy(theta, phi):
    # convert sensor angles to theta and phi in radians
    rtheta = np.radians((-1*theta)+90)
    rphi = np.pi/2 - np.radians(phi)
    return(rtheta, rphi)



def plotapoint(distance,theta,phi, counter):
    # take adjusted motor angles, rotate, and convert to radians.
    rtheta, rphi = motor2xy(theta, phi)
    # turn to cartesian coordinates from spherical.
    data = pol2cart(distance, rtheta, rphi)
    # record data to plot all at once (it is very slow if you plot live)
    xcors[counter-1] = data[0]
    ycors[counter-1] = data[1]
    zcors[counter-1] = data[2]
    return [xcors, ycors, zcors]

#create our coordinate vectors
xcors = np.zeros(pointnumb)
ycors = np.zeros(pointnumb)
zcors = np.zeros(pointnumb)

#scanner offset from origin
tiltlen = 1 #inches

arduinoComPort = "COM8"

baudRate = 9600

serialPort = serial.Serial(arduinoComPort, baudRate, timeout=1)

#
# main loop to read data from the Arduino, then display it
fig = plt.figure()
ax = fig.add_subplot(projection='3d')

ax.set_xlabel('x Distance (in)')
ax.set_ylabel('y Distance (in)')
ax.set_zlabel('Z Distance (in)')
```

```
62
64 while counter < pointnumb:
     counter += 1
66
     #
68   # ask for a line of data from the serial port, the ".decode()" converts
       the
     # data from an "array of bytes", to a string
70   #
     lineOfData = serialPort.readline().decode()
72
     #
74   # check if data was received
76
     if len(lineOfData) > 0:
78     #
       # data was received, convert it into 4 integers
80     #
       a, b, c = (str(x) for x in lineOfData.split(','))
82
       # calibration
84     distance = 24.75 - .1423 * float(a)
       theta = float(c)
86     phi = float(b)
       if distance + tiltlen < 17:
88         plotapoint(distance, theta, phi, counter)
90 # final plot
   ax.scatter3D(xcors, ycors, zcors, c=ycors, cmap='Greens');
92 plt.show()
```

### 6.3 Video

Here is the link to the video of scanner in action.

Long link: `https://drive.google.com/file/d/17ICyEFU9ICWOLI9Ov7ARLDkxNSrB-_V5/view?usp=sharing`