3D Scanner Report

Executive Summary

0 - Table of Contents

1 - Introduction 2

2 - Procedure 3

2.1 Components of Pan Tilt Mechanism 4

2.2 Calibration plot fit to exponential model 6

2.3 Calibration plot fit to polynomial model 6

2.4 Calibration plot fit to power series model 7

2.5 Error plot 8

2.6 Set up for one servo scan 10

2.7 Set up for two servo scan 10

2.8 Visualization from one servo scan 12

2.9 Visualization from two servo scan 13

3 - Circuit Diagram 15

3.1 Circuit diagram schematic 15

4 - Reflection 16

5 - Source Code 18

1. Introduction

The goal of this project was to utilize an Arduino, two servo motors, and an infrared sensor to create a 3D visualization of an object with predefined geometry. In order to easily scan the object, a pan/tilt mechanism was fabricated using two servos and a sensor that is controlled by an Arduino. The data from the sensor was captured using the Arduino and later transferred to Matlab for easy manipulation. In Matlab, the recorded XYZ coordinates were transformed from spherical to cartesian coordinates to create the final 3D visualization plot.

2. Procedure

Pan Tilt Fabrication

The design of the pan tilt mechanism was done in SolidWorks and then 3D printed. A few of its design features include a wider base for stability, screw holes on the base for potential expansion, mounts that result in the sensor always being at the center of axis for easy calibration, easily accessible screw holes for assembly/disassembly, and a mount for counter weight to reduce unwanted vibrations. Our mechanism could be broken down into 3 major components, the base (Fig. 2.1a), the U stand (x-axis rotation)(Fig. 2.1b), and the T stand (y-axis rotation)(Fig. 2.1c) as seen in the figure below:

Figure 2.1a

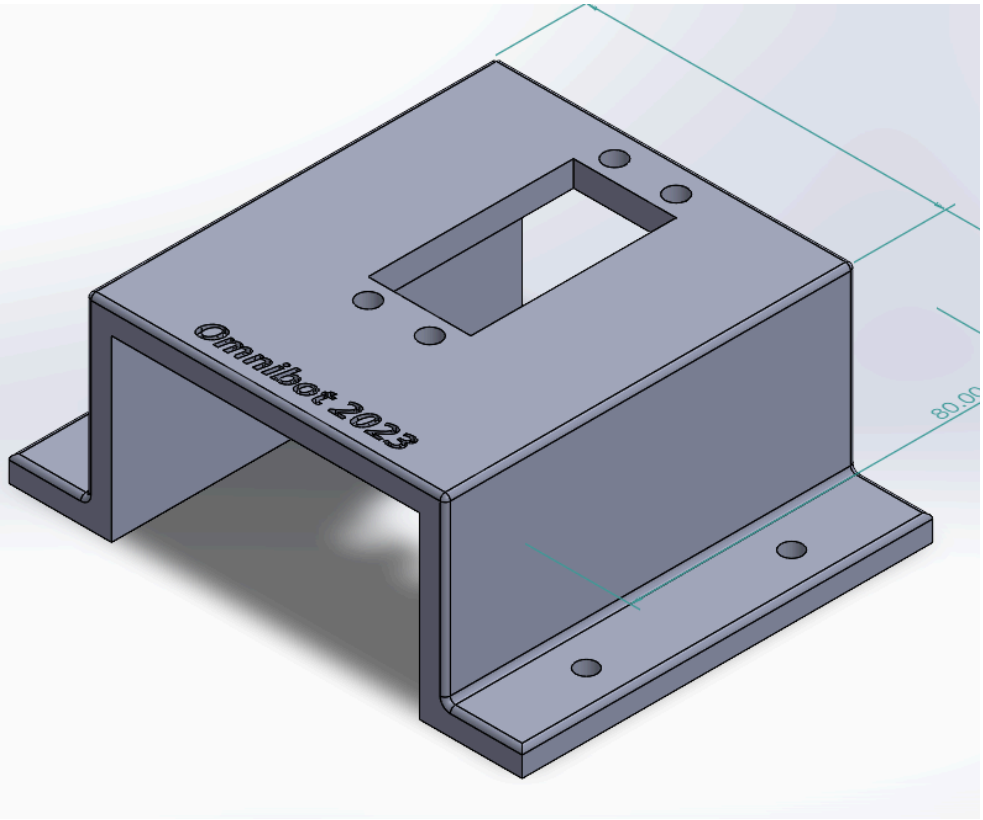


Figure 2.1a CAD of the base of the pan-tilt mechanism.

Figure 2.1b

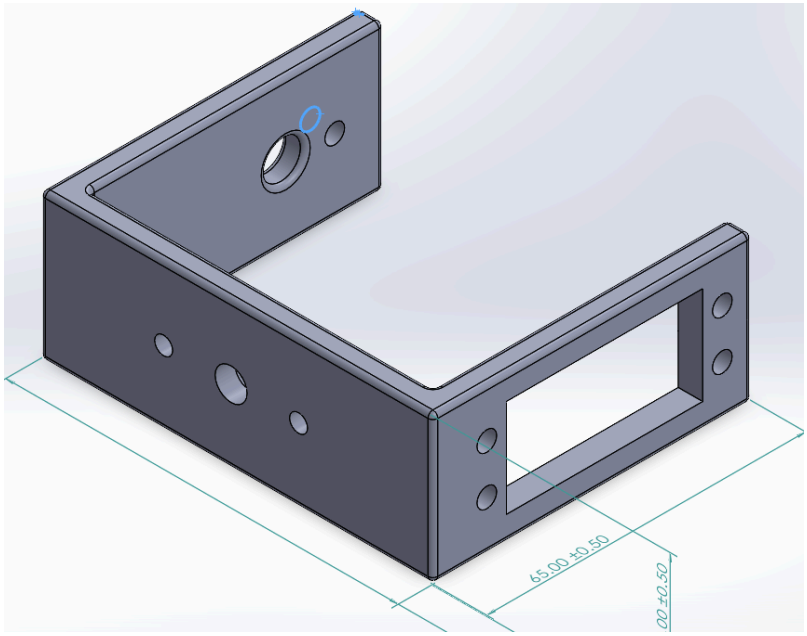


Figure 2.1b CAD of the U stand which is held by the x-axis servo and holds the y-axis servo (Note: on right side of the U stand, the larger hole is used as a pass through so a screwdriver could be inserted to access the screw on the y-axis servo w/out stripping the screw)

Figure 2.1c

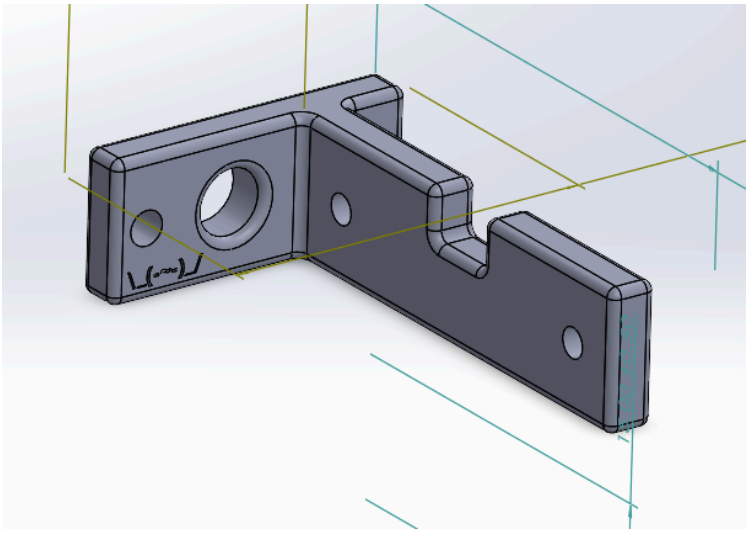


Figure 2.1c CAD of the T stand

The completed build had a weight of approximately 256 grams (including the steel bearing counter weight). Assembled, it stands at around 11.3 cm with it casting a 13.4 cm by 8 cm shadow under.

Software Fabrication

Our first step in the process was to build our circuit to account for two servo motors and the infrared sensor. Then, we established that the sensor was working by placing multiple obstacles at different distances away from the sensor and observing that the serial monitor was displaying varying values. Once we established that the sensor was working and output real time data that measured voltage, we calibrated the sensor using known distances in centimeters and compared that to corresponding output voltages. We did this by taping the sensor down on the ground and placing a large cardboard box at predefined distances of 20, 30, 40, 50, 60, 70, 80, 90, and 100 centimeters. This process ensured that the sensor remained stationary and only the object was moved after the respective voltage was recorded, thereby reducing any variability that might arise from moving the sensor. In addition, we learned from the documentation that values of 10 centimeters and below would cause outliers, therefore we started the calibration from 20 centimeters in increments of 10. After, we fit the data in Google Sheets with multiple line of best fit models to determine the model with the greatest accuracy.

Figure 2.2

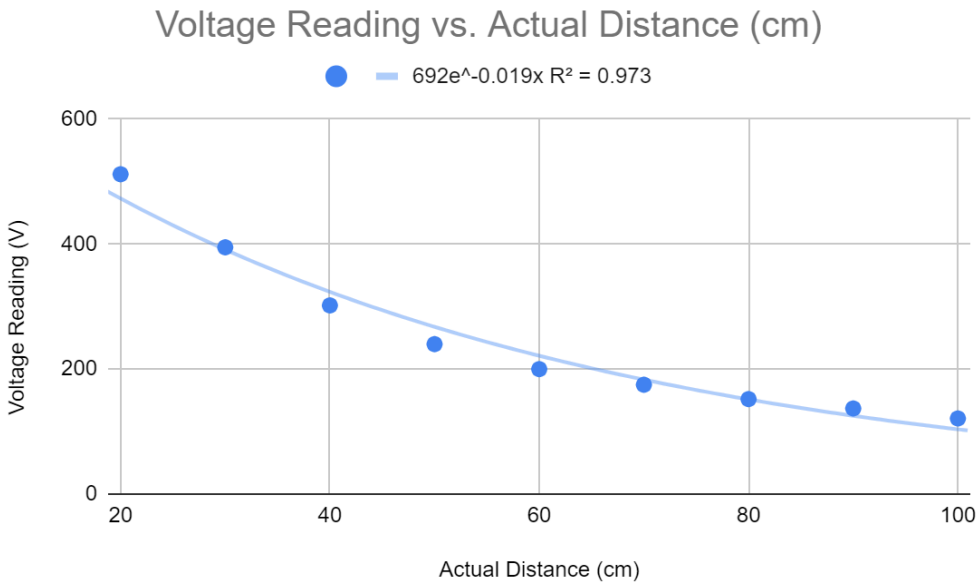


Figure 2.2 Calibration plot fit to an exponential model

Figure 2.3

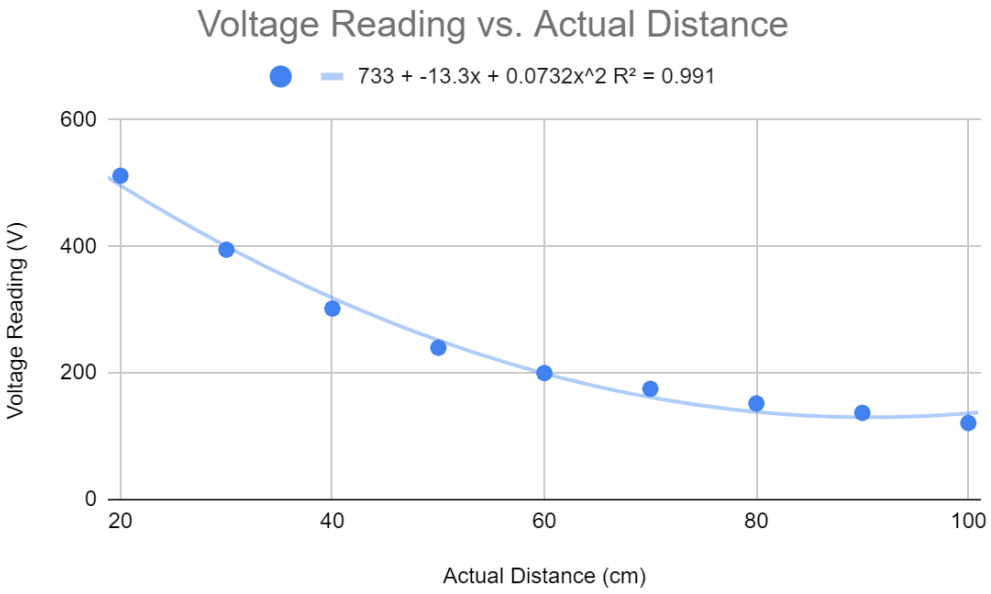


Figure 2.3 Calibration plot fit to a polynomial model

Figure 2.4

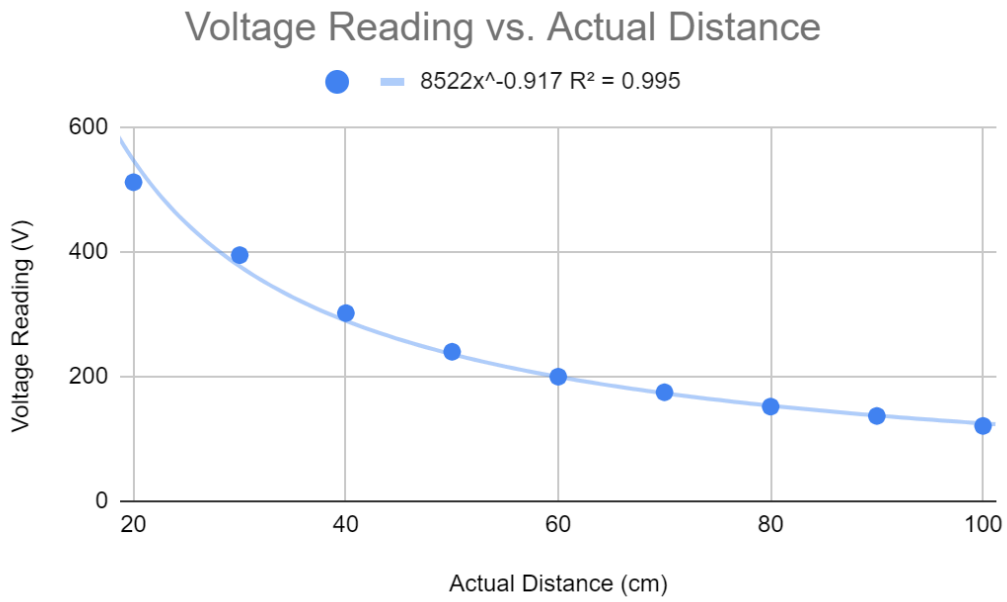


Figure 2.4 Calibration plot fit to a power series model

From these plots, we can determine that the best fit line is from the power series model (figure 2.5) which has the greatest R-squared-value of 0.995 compared to the other models. Therefore, the resulting equation for the relationship between the voltage output (V) from the infrared sensor and the distance measured (cm) is

$$V = a * x^b = 8522 * x^{-0.917} \text{ where x is distance in cm}$$

An error plot for the graph fit to a power series model with distance values varying from the initial calibration test is shown in figure 2.5.

Figure 2.5

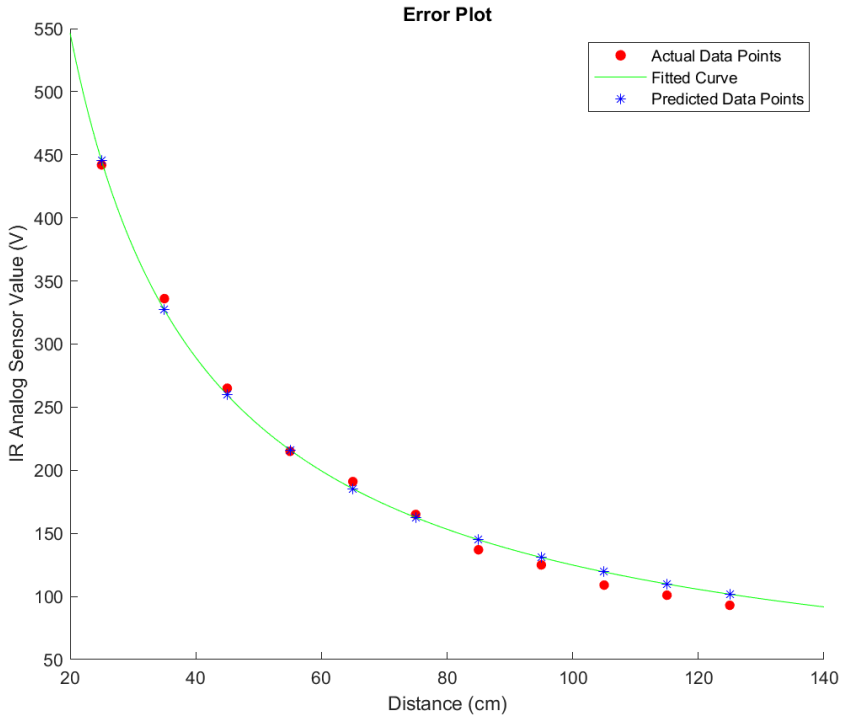


Figure 2.5 Error plot of graph

The voltage values for the error plot were taken at predetermined distances of 25, 35, 45, 55, 65, 75, 85, 95, 105, 115, and 125 centimeters. Based on figure 2.6, we can tell that our calibration curve was relatively accurate because the distance between the actual voltages and the predicted voltages for the respective distances are quite close. In order to confirm this was true, we calculated the percent error, which was about 4%.

The next step was to make the sensor successfully complete one full scan. For the one servo scan, we had the top servo move through a span of 70 degrees. For the two servo scan, we programmed the rotation in such a way that the bottom servo would move a total of 40 degrees in increments of one degree, and at every increment the top servo would cover 70 degrees (traveling from bottom to top) and reset back to the bottom angle;. This was done because it was

pointless for the sensor to collect data both ways (going up and coming back down) since there would only be a repeat of data. The set up for our one and two servo scans are shown below in figures 2.6 and 2.7.

Two Servo Scanning Code:

```
    for (bottomDegree = 0; bottomDegree < degreeIncrement * bottomSteps;
bottomDegree += degreeIncrement) {
      bottomServo.write(bottomDegree); // Move the bottom servo
      for (topDegree = 20; topDegree < degreeIncrement * topSteps;
topDegree += degreeIncrement) {
        topServo.write(topDegree); // Move the top servo
        voltage = analogRead(sensorIR); // Read voltage from IR sensor
        Serial.print(voltage);   // Print voltage to serial monitor
        Serial.print(" ");   // Print a space for formatting
      }
      Serial.println();   // Print a newline character
      topServo.write(20);   // Reset the top servo position
    }
    bottomServo.write(0);   // Reset the bottom servo position
  }
```

Figure 2.6

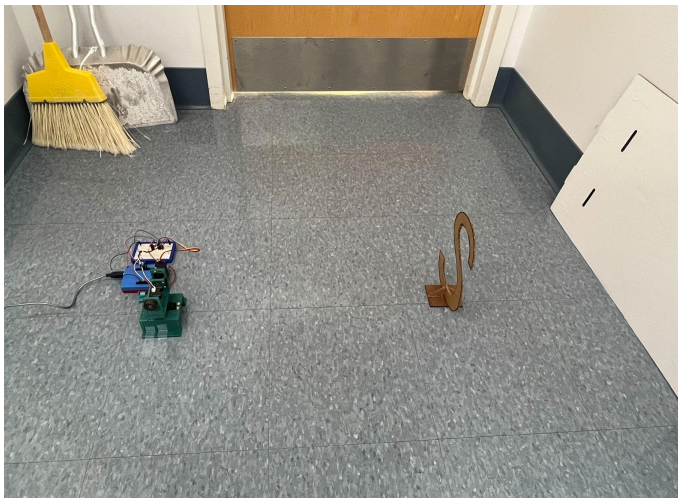


Figure 2.6 Set up for one servo scan

Figure 2.7

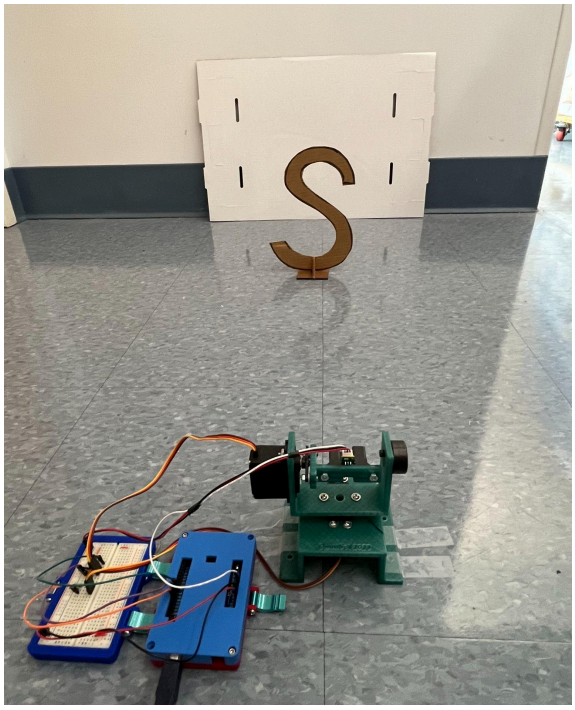


Figure 2.7 Set up for two servo scan

We then copied and pasted the data output from the serial monitor in the Arduino IDE into Excel where we exported the document to a CSV file format. After, we accessed the file through Matlab where the data was input into a matrix format. The specific format we used made a matrix with rows representing the angles of the bottom servo and columns representing the angles of the top servo. In order to have more control over the Arduino code, we incorporated a button that, when pressed, would start the Arduino code.

The values from the collected data matrix were then converted from spherical to cartesian coordinates using the “sph2cart” function in Matlab. Once the conversion was complete, we used “plot3” to plot a visualization for the one servo scan and used “pcshow” to capture the XYZ dimensions and plot the data for the two servo scan. The final visualizations are shown below in figures 2.8 and 2.9 .

```
% Convert spherical coordinates to Cartesian coordinates
[x, y, z] = sph2cart(az, el, T_calib(i));


% Visualize the graphed data in 3D with pcshow
pcshow([graphing_K(:, 1) graphing_K(:, 2) graphing_K(:, 3)])
```

Figure 2.8

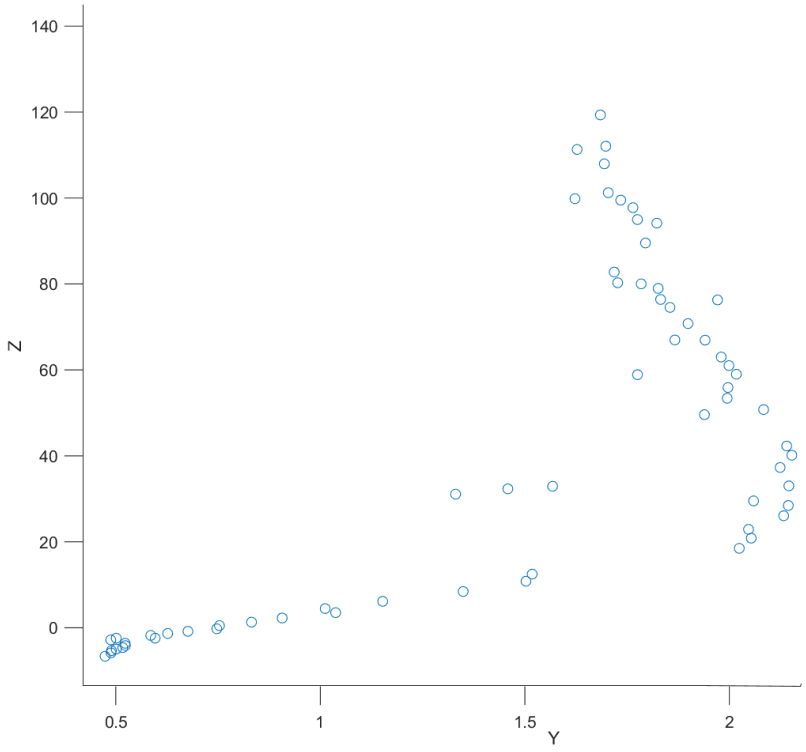


Figure 2.8 Visualization from one servo scan

Figure 2.9a and 2.9b

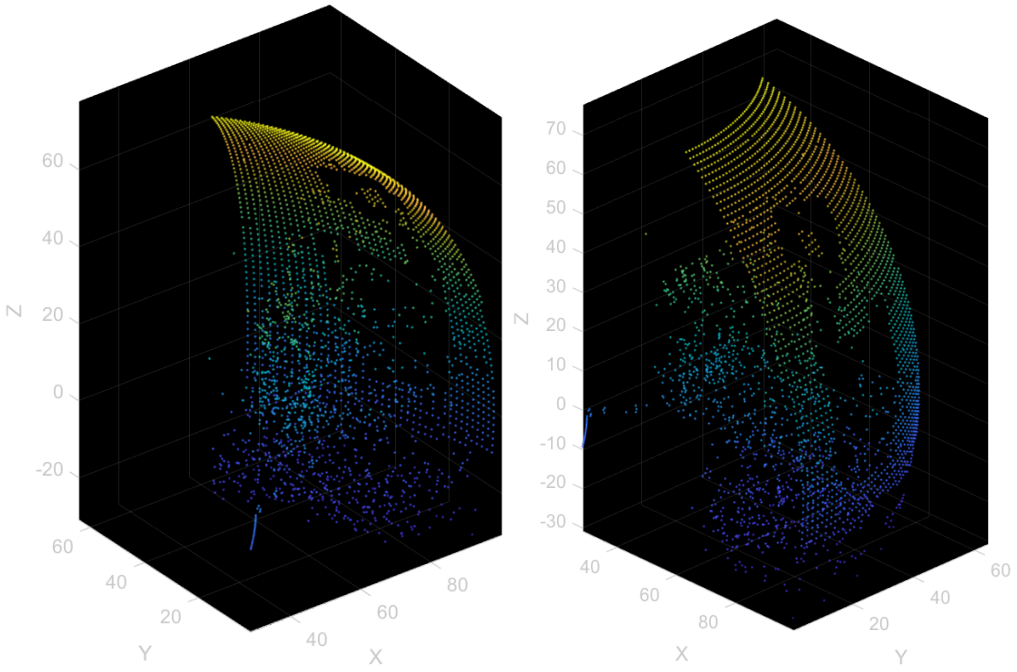


Figure 2.9a and 2.9b Visualizations from the two servo scan

Figure 2.9c

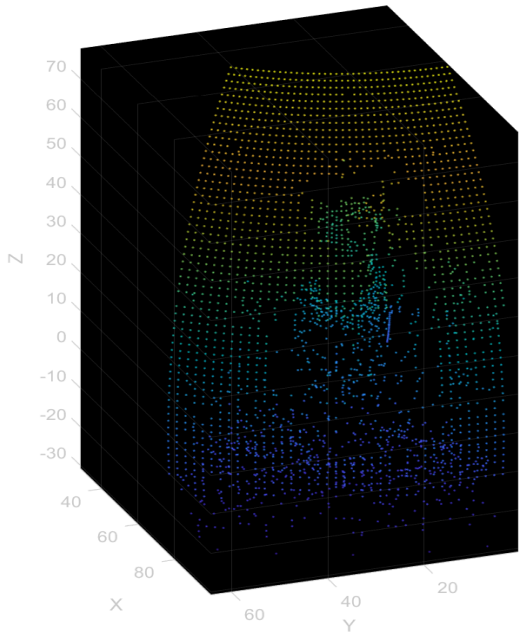


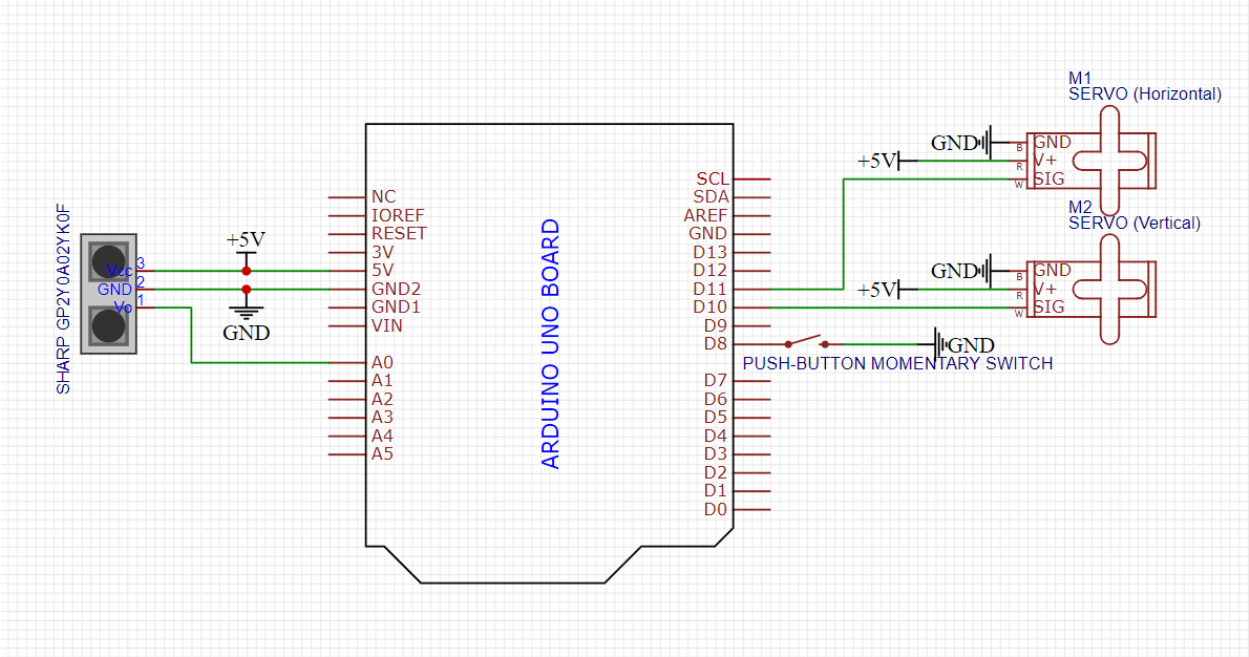
Figure 2.9c Visualization from the two servo scan

From the visuals in figure 2.9, we can determine that the scanner had a hard time detecting objects themselves accurately. The scanner had better accuracy in detecting the absence of the “S” in the background (seen more clearly in figures 2.9a and 2.9b), rather than being able to detect the “S” in the foreground (seen in 2.9c). This can be attributed to the excess noise in the data collected.

3. Circuit Diagram

The circuit diagram for this project contains the Arduino board, an infrared sensor made by Sharp, two normal sized servos, and a push button. The data line of the Sharp infrared sensor is connected to A0 while the servos are connected to the PWM lines of D11 and D10. The push button is connected to pin D8 and is used to initialize the code in a controlled manner. No major calculations were made for the circuit. The circuit diagram is shown below in figure 3.1.

Figure 3.1



4. Reflection

Overall, we think we did a pretty good job. We were successful in detecting and mapping the objects using the Arduino and Matlab. Although, we did face many problems throughout the course of the project including trouble calibrating the sensor, data transferring, and plotting the data.

In specific, we encountered communication issues between the serial port and Matlab where the data matrix was unable to store values or would fill up before the code went through a whole run of scanning. We realized that this was because Matlab was having difficulties keeping up with the speed of the Arudino, and thus could not read the values as it was printing in real time. Another issue that came up was figuring out how to plot the data successfully. We ran into issues with noise in the final graph which is seen in figures 2.9a, 2.9b, and 2.9c. To solve this, we had to filter out readings that were too high or too low and set them to a boundary value. For example, we set distance values greater than 100 centimeters to 100 centimeters and values less than 30 centimeters to 30 centimeters.

Limitations

There were some limitations with our model that we observed through testing. One of the main observations we noticed was that the sensor had a hard time detecting an object if the background was extremely cluttered. Its performance would be best when the object was placed in front of a wall or a uniform background. The hardware method to remedy this would be to install a low-pass/band-pass filter on the receiver-end of the sensor to isolate the potential noise it picks up. Some other hardware methods include shielding the sensor from external electromagnetic interference and proper grounding of the circuit. On the other hand, the software

method to remedy this would be to implement filtering algorithms, find noise reduction software, modulate the transmitted signal and demodulate the received signal, and so on. All of these remedies are time-consuming and require extensive testing.

In addition, the sensor is only able to scan in 2D and would not be able to scan a third dimension. In all our scans, we were only able to see the object's front face and the absence of the object in the background. Therefore, it was not able to scan the third dimension or, in other words, the depth of the object.

5. Source Code

IR Sensor Calibration Code in Arduino

```
#include <SharpIR.h>   // Include SharpIR library for reading the IR
sensor

const int sensorIR = A0;   // Variable to store sensor pin
int sensorValue = 0;   // Variable to store sensor value

void setup() {
  Serial.begin(9600);   // Initialize serial communication at 9600 baud
}

void loop() {
  sensorValue = analogRead(sensorIR);   // Read analog voltage from the IR
sensor
  Serial.println(sensorValue);   // Print the voltage value to the serial
monitor
  delay(1000);   // Delay 1 second
}
```

Calibration Code in MATLAB

```
clf;


% Data measured for initial calibration

cali_dist = [20 30 40 50 60 70 80 90 100]';

cali_meas = [512 395 302 240 200 175 152 137 121]';

% Curve fitting - Linear

[f, ~] = fit(cali_dist,cali_meas,'poly1');

% R-Squared: 0.872


% Curve fitting - Polynomial

[f, ~] = fit(cali_dist,cali_meas,'poly2');
```

```
% R-Squared: 0.991


% Curve fitting - Power (Best Option)
[f, gof] = fit(cali_dist,cali_meas,'power1');
% R-Squared: 0.995


% Set coefficients for the power equation (found in Google Sheets)
% Equation > f(x) = a*x^b
a = 8522;
b = -0.917;


% Graph initial calibration data points along with fitted curve
plot(f,cali_dist,cali_meas);
title('Curve Fit of 3D Scanner (a = 8522, b = -0.917)')
xlabel('Distance (cm)')
ylabel('IR Analog Sensor Value')
legend('Calibration Data Points', 'Fitted Curve')
hold off;
```

Error Plot Code in MATLAB

```
% Data measured to check calibration curve and compute error
error_dist = [25 35 45 55 65 75 85 95 105 115 125];
error_meas = [442 336 265 215 191 165 137 125 109 101 93];


% Graph measured data points along with fitted curve to determine error
X = 20:0.1:140;
Y = a*X.^b;
```

```
s = scatter(error_dist, error_meas, "filled", "red"); hold on;
s.SizeData = 30;
plot (X, Y, "green");

% Plot predicted values for these data points based on calibration curve
Y_points_error = a*error_dist.^b;
scatter(error_dist, Y_points_error, "blue", "*");
percent_error = 0;
error = 0;

title('Error Plot')
xlabel('Distance (cm)')
ylabel('IR Analog Sensor Value (V)')
legend('Actual Data Points', 'Fitted Curve', 'Predicted Data Points')

% Calculate the error for each point in the error data
for i = 1:11
    error = abs(Y_points_error(1,i) -
error_meas(1,i))/Y_points_error(1,i);
    percent_error = percent_error + error;
end

% Calculate the average error and convert to percentage
percent_error = percent_error / 11;
percent_error*100
```

Servo Scanning Code in Arduino

```
// Include Servo and SharpIR libraries for controlling servos and the IR sensor
#include <Servo.h>
#include <SharpIR.h>

// Create two servo objects
Servo bottomServo;
Servo topServo;

// Initialize starting degrees for both servos
int bottomDegree = 0;
int topDegree = 0;

// Set a number of steps(degrees) for both servos
int bottomSteps = 40;
int topSteps = 90;

int degreeIncrement = 1;   // Set a degree increment for servo movement
unsigned int distance_cm;   // Variable to store distance in centimeters
const int buttonPress = 8;   // Set a pin number for the button used as input
const int sensorIR = A0;   // Set a pin for the IR sensor
int voltage = 0;   // Variable to store voltage readings from the IR sensor
SharpIR mySensor(SharpIR::GP2Y0A02YK0F, A0);   // Create an instance of the SharpIR sensor

void setup() {
  Serial.begin(9600);   // Initialize serial communication at 9600 baud
  pinMode(buttonPress, INPUT_PULLUP);   // Configure the button pin as an input with a pull-up resistor

  bottomServo.attach(11);   // Attach the bottom servo to pin 11
  topServo.attach(10);   // Attach the top servo to pin 10

  bottomServo.write(0);   // Set the initial position of the bottom servo to 0 degrees
```

```
  topServo.write(20);   // Set the initial position of the top servo to 20
degrees
}

void loop() {
  int buttonState = digitalRead(buttonPress);   // Read the state of the
button

  if (buttonState == LOW) {   // Check if the button is pressed
    // Start servo motion if button is pressed
    for (bottomDegree = 0; bottomDegree < degreeIncrement * bottomSteps;
bottomDegree += degreeIncrement) {
      bottomServo.write(bottomDegree); // Move the bottom servo
incrementally
      delay(10);

      for (topDegree = 20; topDegree < degreeIncrement * topSteps;
topDegree += degreeIncrement) {
        topServo.write(topDegree); // Move the top servo incrementally
        delay(10);

        voltage = analogRead(sensorIR); // Read analog voltage from the IR
sensor

        Serial.print(voltage);   // Print the voltage value to the serial
monitor
        Serial.print(" ");   // Print a space for formatting
      }
      Serial.println();   // Print a newline character to indicate the end
of a scan
      topServo.write(20);   // Reset the top servo position
    }
    bottomServo.write(0);   // Reset the bottom servo position
  }
}
```

One Servo Scan Visualization:

```
% Read data from 'one_servo_scan.csv'
K = readtable('one_servo_scan.csv');
K_new = table2array(K);

% Perform calibration using a power-law model
K_calib = 8522 * K_new.^-0.917;

% Initialize a matrix for graphing with dimensions [70, 3]
graphing_K = [70, 3];
counter = 1;

% Loop through elevation angles from 1 to 70 degrees
for i = 1:70
    az = 1*(pi/180);
    el = (i-19)*(pi/180);

    % Convert spherical coordinates to Cartesian coordinates
    [x, y, z] = sph2cart(az, el, T_calib(i));

    % Store the Cartesian coordinates in the graphing matrix
    graphing_K(counter, 1) = x;
    graphing_K(counter, 2) = y;
    graphing_K(counter, 3) = z;
    counter = counter + 1;
end
```

```
% Visualize the graphed data in 3D with pcshow
pcshow([graphing_K(:, 1) graphing_K(:, 2) graphing_K(:, 3)])
xlabel('X')
ylabel('Y')
zlabel('Z')
```

Two Servo Scan Visualization

```
clf;


% Read data from 'two_servo_scan.csv'
T = readtable('two_servo_scan.csv');
T_new = table2array(T);


% Define the number of steps for bottom and top servos
bottomSteps = 40;
topSteps = 70;


% Perform calibration using a power-law model
T_calib = 8522 * T_new.^-0.917;


% Initialize a matrix for graphing with dimensions
[bottomSteps*topSteps, 3]
graphing_data = [bottomSteps*topSteps, 3];
count = 1;


% Loop through azimuth + elevation angles (representing angles of both
servos)
for i = 1:bottomSteps
```

```
    for j = 1:topSteps
        az = i*(pi/180);
        el = (j-19)*(pi/180);

        % Convert spherical coordinates to Cartesian coordinates
        [x, y, z] = sph2cart(az, el, T_calib(i,j));

        % Store the Cartesian coordinates in the graphing matrix
        graphing_data(count, 1) = x;
        graphing_data(count, 2) = y;
        graphing_data(count, 3) = z;
        count = count + 1;
    end
end

% Visualize the graphed data in 3D with pcshow
pcshow([graphing_data(:, 1) graphing_data(:, 2) graphing_data(:, 3)]);
xlabel('X');
ylabel('Y');
zlabel('Z');
```