DIY 3D Scanner

Jade Campbell, Michaela Fox

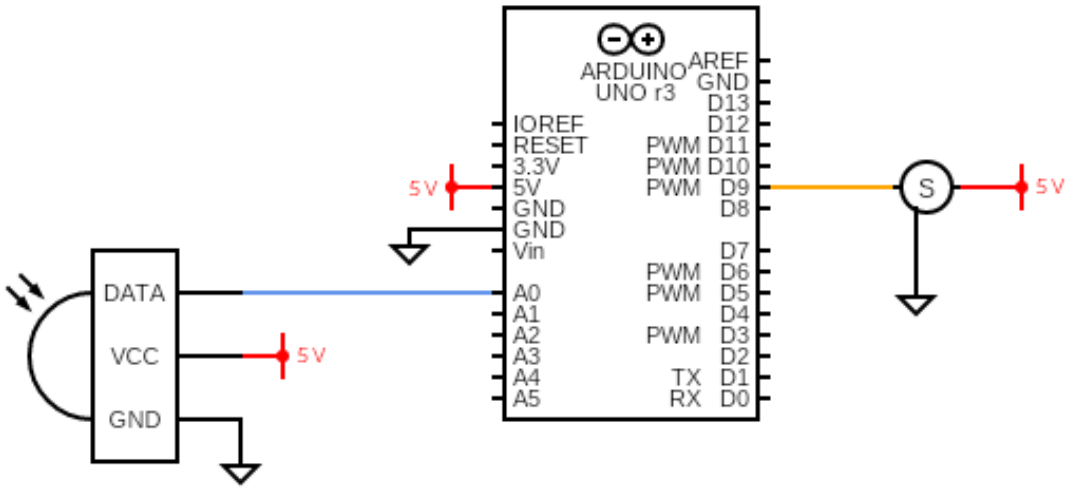
Introduction

In this mini-project, we developed a DIY 3D scanner using a pan/tilt mechanism powered by Arduino to scan an object with a well-defined geometry and visualize the output. We employed an infrared distance sensor that converts object distance into an analog voltage signal, processed by the Arduino, along with two hobby servo motors for scanning. A calibration curve was created and used in tandem with MATLAB to generate a heatmap of a letter, visualizing the scanned data.

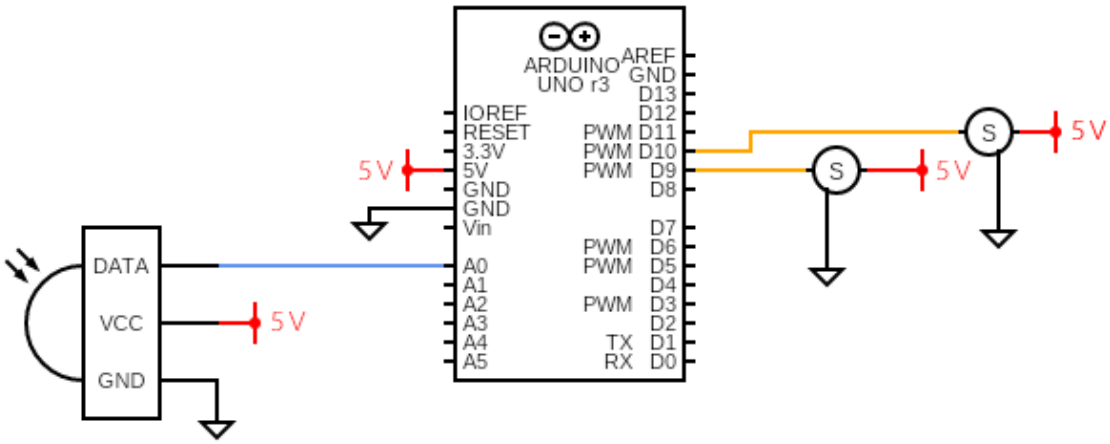
Circuit Diagram

We designed our circuit using an Arduino Uno R4 WiFi in addition to a Sharp GP2Y0A02YK0F IR distance sensor and 2 MG996R Digital Servos. Our circuit diagrams are below:

Single Servo Scanner:



Double Servo Scanner:



For both circuits we connected our distance scanner to an analog pin to allow analog voltage to be read from the sensor.

Servo Testing

We tested our servos using the sweep example sketch to ensure that the full range of motion was available. The code is shown below:

```
#include <Servo.h>
Servo myservo1;
Servo myservo2;
// create servo object to control a servo
// twelve servo objects can be created on most boards
int pos = 0;
// variable to store the servo position
void setup()
{
    myservo.attach(9);
    myservo.attach(10);
    // attaches the servos to PWM pins 9 and 10
}
void loop()
{
```

```
    for (pos = 0;
    pos <= 180;
    pos += 1)
    {
        // goes from 0 degrees to 180 degrees
        // in steps of 1 degree
        myservo1.write(pos);
        myservo2.write(pos);
        // tell servo to go to position in variable 'pos'
        delay(15);
        // waits 15ms for the servo to reach the position
    }
    for (pos = 180;
    pos >= 0;
    pos -= 1)
    {
        // goes from 180 degrees to 0 degrees
        myservo1.write(pos);
        myservo2.write(pos);
        // tell servo to go to position in variable 'pos'
        delay(15);
        // waits 15ms for the servo to reach the position
    }
}
```

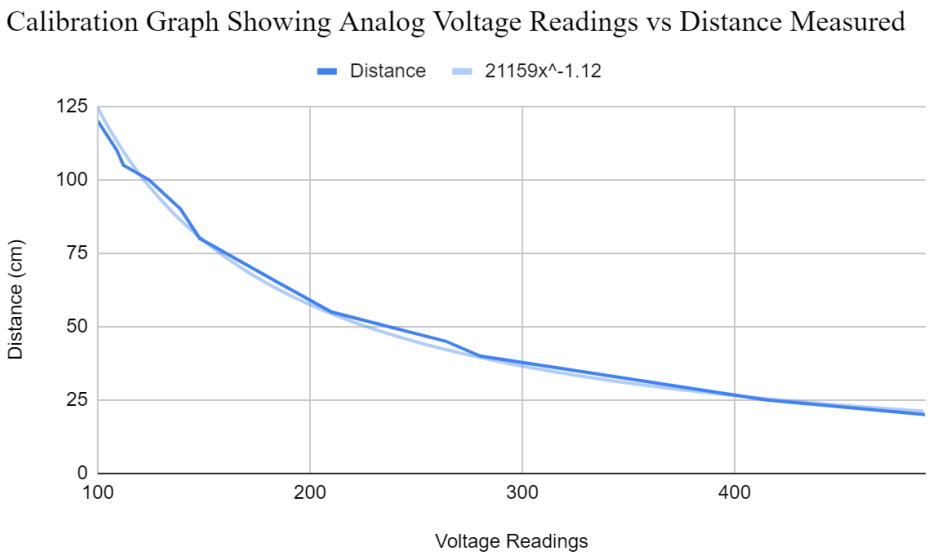
Sensor Calibration

To calibrate our sensor, we originally planned to use code that converts the voltage sent back by the sensor to distance. The code used to do this is below:

```
/*
Reading Voltage from Analog Pin
By Michaela Fox and Jade Campbell
Reads an analog input on pin 0, converts it to voltage, and prints the result
to the Serial Monitor.
*/
// the setup routine runs once when you press reset:
void setup()
```

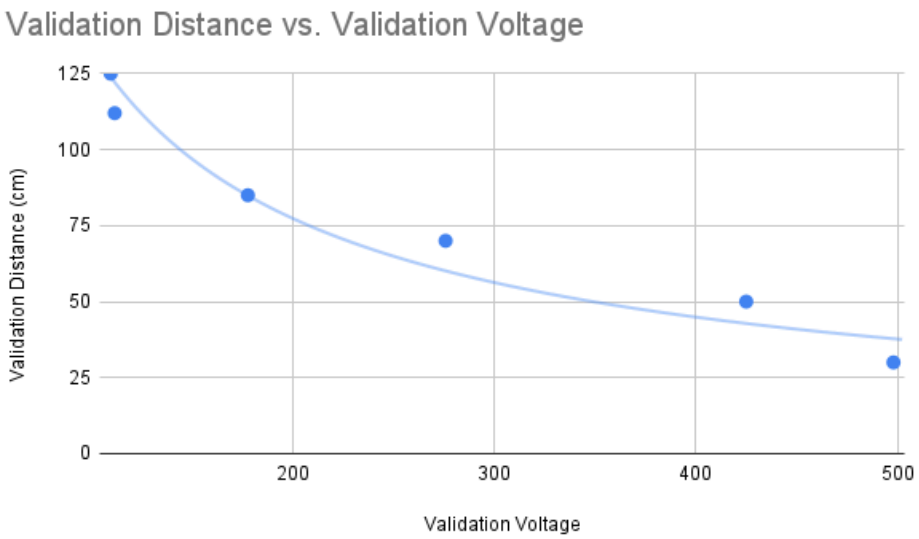
```
{
    // initialize serial communication at 9600 bits per second:
    Serial.begin(9600);
}
// the loop routine runs over and over again forever:
void loop()
{
    // read the input of the distance sensor:
    int sensorValue = analogRead(A0);
    // Convert the analog reading (which goes from 0 - 1023) to a voltage (0 -
5V):
    float voltage = sensorValue * (5.0 / 1023.0);
    // print out the value you read:
    Serial.println(voltage);
}
```

We then took this data and made a plot in MATLAB allowing us to get a calibration function. Our function showed an inverse relationship, meaning that as voltage increased, distance decreased. This lined up with what we expected using the distance sensor data sheet as a reference. We originally had an issue where our distances used for calibration were too small, leading us to have to redo our calibration curve with new distance values.



Calibration curve of the DIY 3D scanner project illustrating the inverse relationship between analog voltage readings and distance measured by the IR sensor.

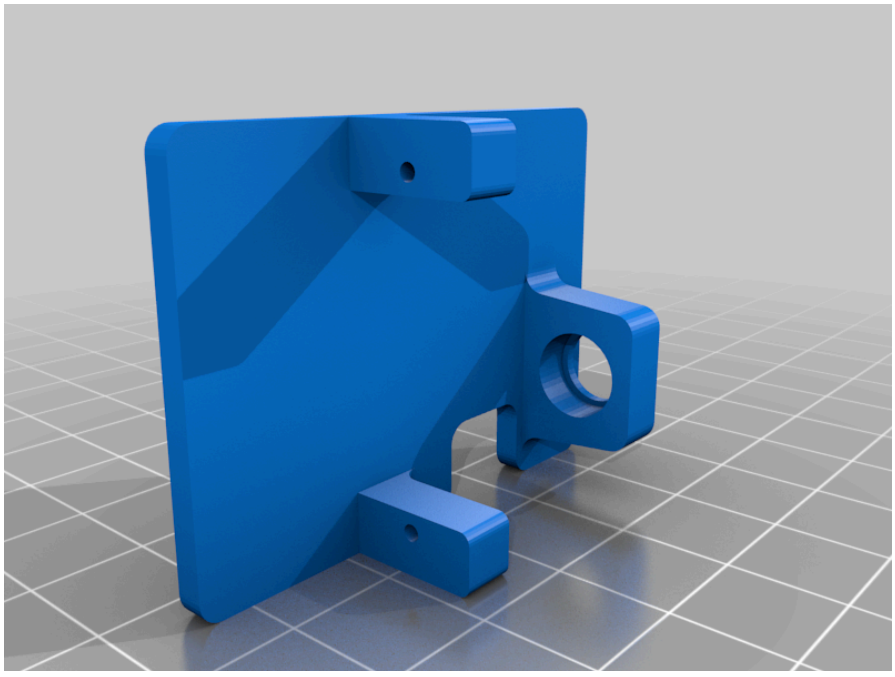
Our calibration was mostly accurate, as can be seen below:



The graph above shows distances that were not used as part of our original calibration. The shape of the trendline is the same, showing that our calibration curve was accurate.

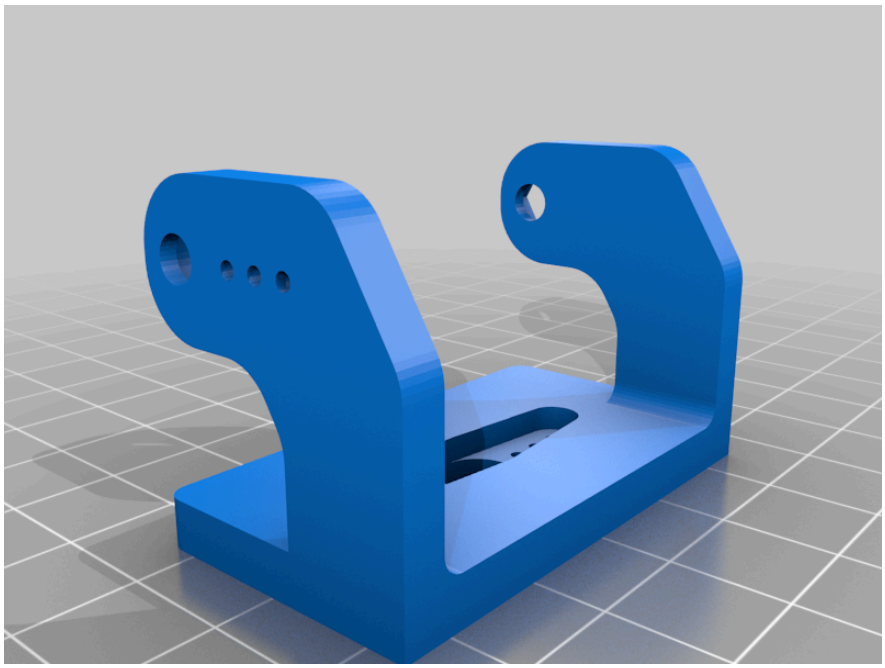
Pan Tilt Design

For our design we created 2 parts. The top part had a space to connect the servo and a flat area to connect our distance sensor. The other side had a hole for a screw to be connected to allow the top part to spin.



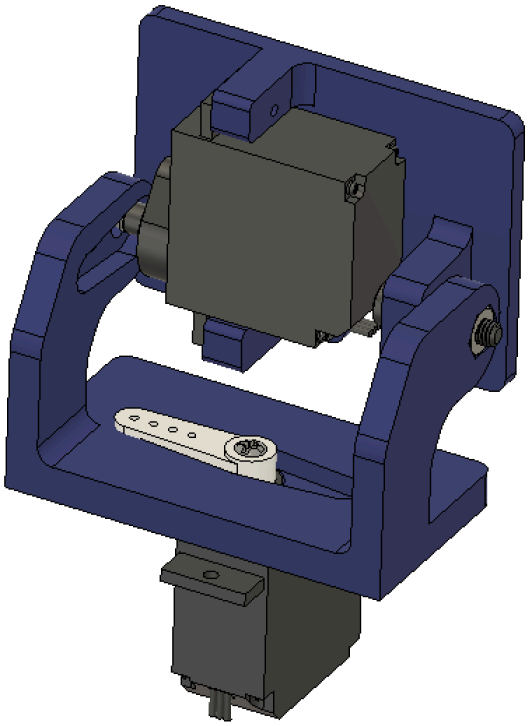
The servo was held in between the top and bottom rectangle pieces. The distance sensor was screwed into the other side of the flat piece.

The base connected to the top part through the screw, with an area for the second servo at the bottom to allow it to be attached.

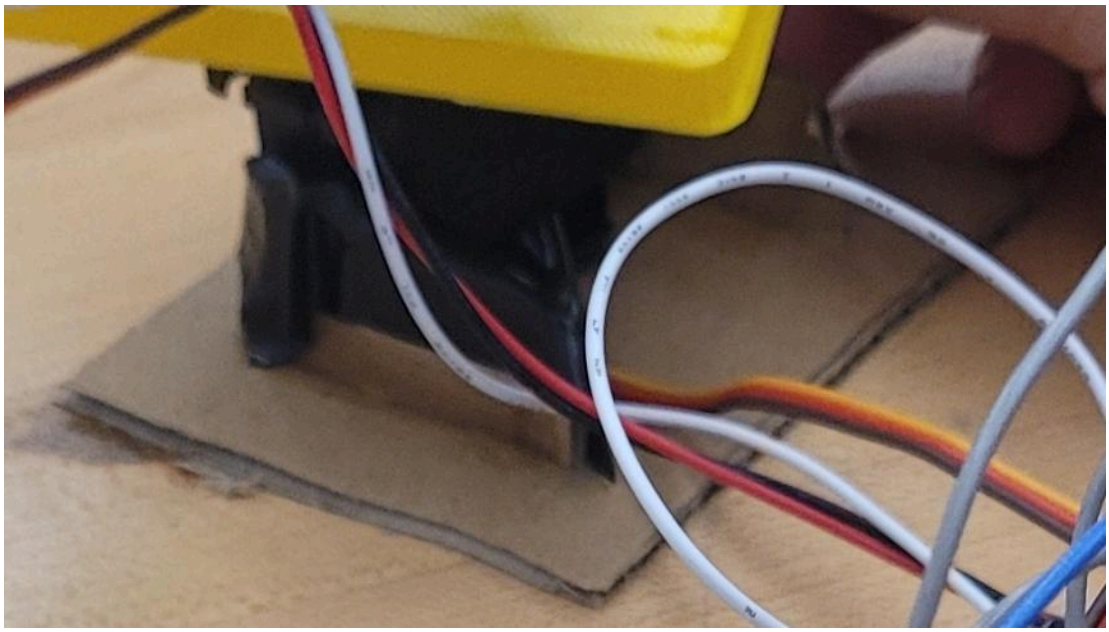


This base piece included both areas where the rotating parts of the servo were attached.

When the two pieces were connected it looked like this:



We created a cardboard base to hold the bottom servo which can be seen in this picture:

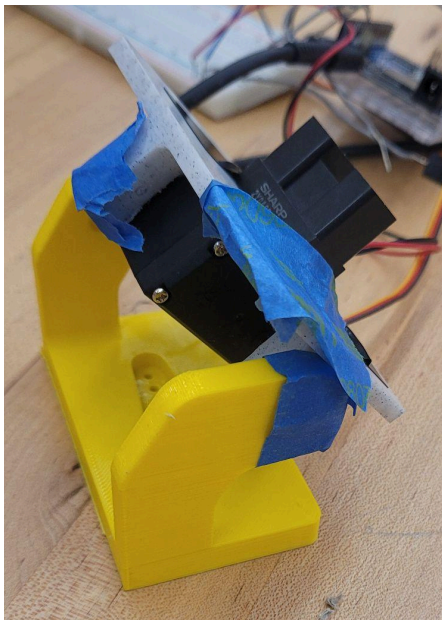


We had an issue where our pan tilt was too top heavy which we remedied by reinforcing our base with tape and wood glue. We also had an issue where we could not find a screw big enough to attach the top to the base. Instead of 3D printing another piece and wasting filament we decided

to cut a straw so that it would fit in the hole and reinforce it with tape to ensure it would not fall out. This allowed our top piece to retain its ability to rotate freely.

One Servo Scan Results

We designed our pan tilt to be easily used for 1 or 2 servos. A picture of our single servo setup is below:



We also created a cardboard letter to scan with our sensor. This letter was used for both the 1 servo and 2 servo scans:

Our J had to be backwards because we originally cut it out of a W.B Mason box, with the W.B Mason logo visible from the front side. The different colors of the logo were leading to inaccurate scans as the scanner picked up each color differently. We decided to remedy this by just turning our letter around so that there was a plain surface for the distance scanner.

We created code that sweeps the distance sensor across our letter, from an angle of 45 to 150 degrees and back:

```
/*
Sweeping an Area with a Distance Sensor and Reading Voltage (Single Servo Version)
By: Michaela Fox and Jade Campbell
*/
#include <Servo.h>
Servo myservo1;
// create Servo object to control a servo
int potpin = A0;
// analog pin used to connect the distance sensor
int val;
// variable to read the value from the analog pin
int sensorValue = 0;
// variable to read the distance
float voltage = 0;
// variable to hold the analog to voltage value
```

```
int pos = 0;
// variable to store the servo position
void setup()
{
    myservo1.attach(9);
    Serial.begin(9600);
    // start the serial communication
}
void loop()
{
    // Sweep from 45 to 150 degrees
    for (pos = 45;
    pos <= 150;
    pos += 1)
    {
        myservo1.write(pos);
        delay(15);
        // wait 15 ms for the servos to reach the position
        // Read the sensor data (voltage)
        sensorValue = analogRead(potpin);
        voltage = sensorValue * (5.0 / 1023.0);
        // Send servo position and voltage to MATLAB
        Serial.print(pos);
        Serial.print(",");
        Serial.println(voltage);
    }
    // Sweep from 150 to 45 degrees
    for (pos = 150;
    pos >= 45;
    pos -= 1)
    {
        myservo1.write(pos);
        delay(15);
        // wait 15 ms for the servos to reach the position
        // Read the sensor data (voltage)
        sensorValue = analogRead(potpin);
```

```
        voltage = sensorValue * (5.0 / 1023.0);
        // Send servo position and voltage to MATLAB
        Serial.print(pos);
        Serial.print(",");
        Serial.println(voltage);
    }
}
```

We then used this code to create position and voltage vectors in MATLAB:

```
clear s  % Clear any previous serial connection stored in 's'
ports = "COM6";  % Specify the COM port to connect to (replace with your own port)
baudrate = 9600;  % Set the baud rate for serial communication

s = serialport(ports, baudrate);  % Initialize a serial port connection with the specified COM port and baud rate
timeout = 0;  % Initialize a timeout counter
i = 0;  % Initialize an index for storing data

% Loop until the timeout counter reaches 5
while timeout < 5
    % Check if there is data available in the serial buffer
    while s.NumBytesAvailable > 0
        timeout = 0;  % Reset the timeout counter when data is received
        values = eval(strcat('[', readline(s), ']'));  % Read a line of data from the serial port and evaluate it as a numeric array

        % Store the data into respective variables
        pos = values(1);  % First value is the servo position
        voltage = values(2);  % Second value is the voltage from the distance sensor

        % Display the servo position and voltage in the command window
        fprintf('Servo Position: %d, Voltage: %.2f V\n', pos, voltage);

        % Store the position and voltage in arrays for later use
```

```
        position(i) = pos;
        volts(i) = voltage;
        i = i + 1;  % Increment the index for the next data point
    end

    pause(0.5);  % Pause for 0.5 seconds before checking for more data
    timeout = timeout + 1;  % Increment the timeout counter after each
iteration

end
```

The code reads in data from the serial port connected to the Arduino at a specified baud rate. It extracts and displays the servo position and voltage, storing these values in arrays for later use.

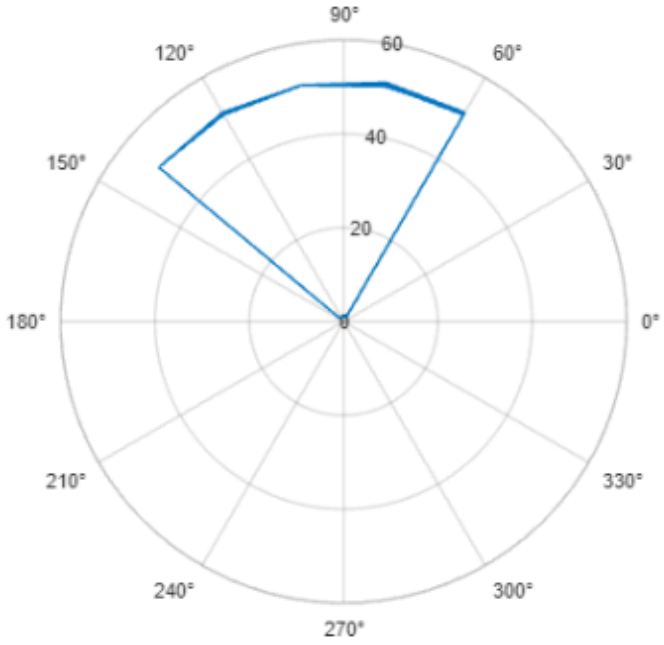
We then converted our voltages to distances using our previously calculated calibration equation:

```
% Calibration: Convert voltage to distance
% Using previously calculated calibration equation
% The equation used here is: distance = 21159 * volts^-1.12
% This assumes a non-linear relationship between voltage and distance

distance = zeros(length(volts), 1);  % Initialize the 'distance' array to store
calculated distances

% Loop through the voltage readings and apply the calibration equation
for i = 1:length(volts)
    distance(i) = 21159 * volts(i)^-1.12;  % Calculate the distance for each
voltage using the calibration function
end

% Combine positions and voltage readings into a single array for reference or
analysis
total_array = [positions(:) volts(:)];

% Convert positions from degrees to radians for use in the polar plot
positions = deg2rad(positions);
```

```
% Create a polar plot of the positions (angles) against the voltage readings
figure;
polarplot(positions, volts);  % Plot the data in polar coordinates
```

The code converts voltage readings from a sensor into distances using our pre-calculated calibration equation. It then plots these distances against corresponding angular positions in a polar plot, where the positions are converted from degrees to radians.

The polar plot showing the location of our letter in a 3D space is located below:

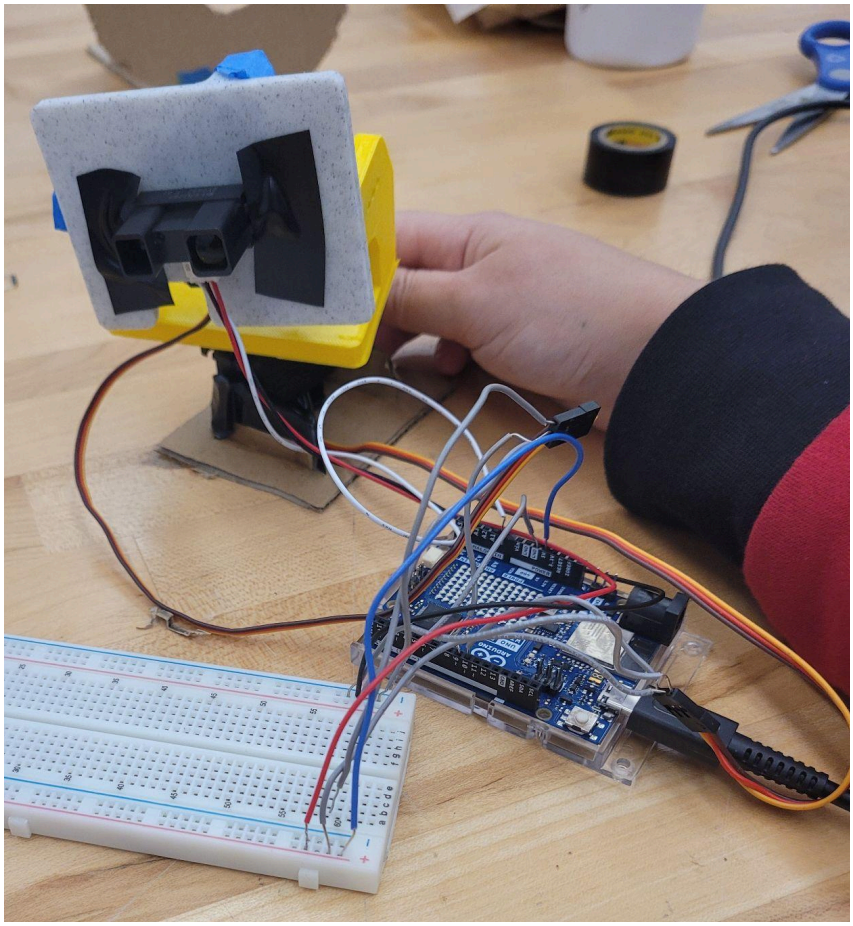


Polar plot visualizing the relationship between the servo motor's angular position and the corresponding voltage readings from the distance sensor.

This plot gives us a better idea of the range of the distance scanner, as well as a better idea of the location of our letter in a 3D space.

Two Servo Scan

Due to the design of our pan tilt, turning our 1 servo scanner into a 2 servo scanner was simple. We attached the second servo to the bottom of the base of our 1 servo scanner, creating a temporary cardboard base to ensure it stood up and did not slide around. Our design can be seen below:



Our Arduino code used to make the servo sweep the letter and transfer the data to Python is below:

```
*
Sweeping an Area with a Distance Sensor and Reading Voltage
By: Michaela Fox and Jade Campbell
*/
#include <Servo.h>
Servo horizontalServo;
// Create Servo object for horizontal movement
Servo verticalServo;
// Create Servo object for vertical movement
int potpin = A0;
// Analog pin used to connect the distance sensor
```

```
int sensorValue = 0;
// Variable to read the distance
float voltage = 0;
// Variable to hold the analog to voltage value
int horizontalPos = 0;
// Variable to store the horizontal servo position
int verticalPos = 0;
// Variable to store the vertical servo position
void setup()
{
    horizontalServo.attach(9);
    // Attach horizontal servo to pin 9
    verticalServo.attach(10);
    // Attach vertical servo to pin 10
    Serial.begin(9600);
    // Start the serial communication
    // Print CSV header
    Serial.println("Horizontal,Vertical,Distance");
}
void loop()
{
    // Sweep horizontal from 0 to 90 degrees
    for (horizontalPos = 0;
    horizontalPos <= 90;
    horizontalPos += 10)
    {
        horizontalServo.write(horizontalPos);
        // Move horizontal servo
        delay(15);
        // Wait for the servo to reach the position
        // Sweep vertical from 0 to 80 degrees
        for (verticalPos = 0;
```

```
    verticalPos <= 80;
    verticalPos += 10)
    {
        verticalServo.write(verticalPos);
        // Move vertical servo
        delay(15);
        // Wait for the servo to reach the position
        // Read the sensor data (voltage)
        sensorValue = analogRead(potpin);
        voltage = sensorValue * (5.0 / 1023.0);
        float distance = 21159 * pow(voltage, -1.12);
        // Calculate distance
        // Print angles and distance to Serial in CSV format
        Serial.print(horizontalPos);
        Serial.print(",");
        Serial.print(verticalPos);
        Serial.print(",");
        Serial.println(distance);
        // New line for the next data entry
    }
    // Return vertical servo to 0 degrees before next horizontal increment
    verticalServo.write(0);
    delay(15);
    // Wait for the servo to reach the position
    // Check if horizontalPos has reached the maximum value
    if (horizontalPos >= 90)
    {
        return;
        // End the loop and stop execution
    }
}
// Optionally add a delay here before repeating the loop
```

```
    delay(1000);
    // Wait for a second before starting over
}
```

This code controls both servos separately, allowing it to sweep an area in both horizontal and vertical directions while reading and recording distance measurements. The collected data, including servo angles and calculated distances, is stored in CSV format for easy use in Python.

We ran into an issue with our original code where it would move both servos in sync and return one position value. We realized our issue when we attempted to graph our results and realized we needed another dimension to achieve proper results.

We also decided to switch from MATLAB to Python, as it provided easier syntax and tools for obtaining 3D data as opposed to MATLAB, which we struggled to properly plot our coordinates in.

The Python code we used to read in our Arduino data is below:

```
import serial
import csv
import time



# Adjust the port and baud rate according to your setup
PORT = 'COM6'  # Replace with your serial port (e.g., 'COM3' for Windows, '/dev/ttyUSB0' for Linux)
BAUD_RATE = 9600
CSV_FILE = 'output.csv'



def main():
    # Create a serial connection
    ser = serial.Serial(PORT, BAUD_RATE, timeout=1)
    time.sleep(2)  # Give some time for the connection to establish
```

```
    # Open a CSV file for writing
    with open(CSV_FILE, mode='w', newline='') as file:
        writer = csv.writer(file)

        # Read and discard the header
        header = ser.readline().decode('utf-8').strip()  # Read header from
serial
        print(f"Header: {header}")  # Print header for verification
        writer.writerow(header.split(','))  # Write header to CSV



        try:
            print("Reading data...")
            while True:
                # Read a line from the serial port
                line = ser.readline().decode('utf-8').strip()

                if line:  # Check if the line is not empty
                    print(f"Received: {line}")  # Print the received line for
verification
                    writer.writerow(line.split(','))  # Write data to CSV
        except KeyboardInterrupt:
            print("Exiting...")  # Gracefully exit on Ctrl+C
        except Exception as e:
            print(f"An error occurred: {e}")
        finally:
            ser.close()  # Close the serial connection


if __name__ == '__main__':
    main()
```

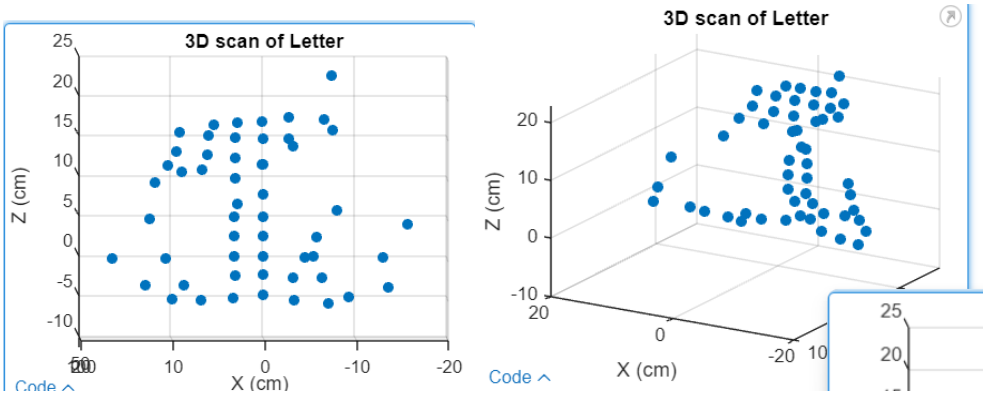
This code establishes a serial connection with an Arduino to read servo position and distance measurement data, logging it in real-time to a CSV file. It handles potential errors and allows for a graceful exit when interrupted by the user.

We then made a 3D graph showing where our letter was in the space using MATLAB:

```
% Load the data (angles and distances)
% Assuming data is stored in a CSV file: [horizontalAngle, verticalAngle, distance]
data = csvread('scan_data.csv');

% Extract angles and distances
horizontalAngles = data(:, 1);  % Horizontal angles (servo1)
verticalAngles = data(:, 2);    % Vertical angles (servo2)
distances = data(:, 3);         % Distance values from IR sensor

% Convert polar coordinates to Cartesian (x, y, z)
x = distances .* sind(horizontalAngles- 45) .* cosd(verticalAngles- 18);  % X-axis
y = distances .* cosd(horizontalAngles - 45) .* cosd(verticalAngles - 18);  % Y-axis
z = distances .* sind(verticalAngles - 18);                              % Z-axis (height)

% Plot the 3D representation
figure;
scatter3(x, y, z, 'filled');
xlabel('X (cm)');
ylabel('Y (cm)');
zlabel('Z (cm)');
title('3D scan of Letter');
grid on;
```

The code reads angle and distance data from our previously created CSV file, converts the polar coordinates to Cartesian coordinates (x, y, z), and plots a 3D representation of our scanned letter based on the distance sensor's readings.

These 3D scans can be seen below:



2D and 3D scans of a letter created using coordinates from a distance scanner.

There seemed to be a lot of noise in our scans. While we do not know the definitive reason, our hypothesis is that it was picking up tools and materials on our table or other tables, or possibly the table itself. Despite this, our backwards J is still very visible in both scans, although it is easier to see it in the 3D plot, as the noise affects the overall shape less.

Reflection

Our testing and wiring of our servos and distance sensor went relatively well. Everything worked and we were able to easily learn and understand the syntax. However, that is the only thing that went well. Everything that could have gone wrong went wrong during this project. We had an issue with our 3D prints that put us behind, one laptop refused to connect the Arduino to MATLAB, and we discovered we had made major mistakes a day before the deadline, in addition to both of us being unusually busy during this period with other work. While some of these major issues were unpreventable, to ensure a smoother project experience in the future, we would most likely try to start earlier and check in with professors and CAs more often to ensure we were on the right track.