Mini-Project 2: DIY 3D Scanner

Section 4
Dongim Lee, Liam Bennicoff, Sreylen Thor

1. Description of the Process Used to Test the Sensor

We began by connecting the GP2Y0A02YK0F IR distance sensor to the Arduino Uno. Our primary objective was to ensure that the sensor could measure distances accurately over a set range (20 cm to 150 cm). We used the Arduino IDE's Serial Monitor to read the raw analog voltage from the sensor. The test involved moving a flat object (the alphabet block "E") at known distances in front of the sensor and recording the sensor's voltage readings.

To verify the readings, we used the "AnalogInOutSerial" example in the Arduino IDE to print sensor values to the Serial Monitor, adjusting the distance to check how the sensor reacted. This initial test helped us confirm that the sensor was working within its range.

2. Calibration Plot: Analog Voltage vs. Actual Distance

The calibration process involved recording analog voltage readings for known distances between 20 cm and 150 cm, at intervals of 10 cm. We then plotted the voltage readings against the actual distances to create a calibration curve.

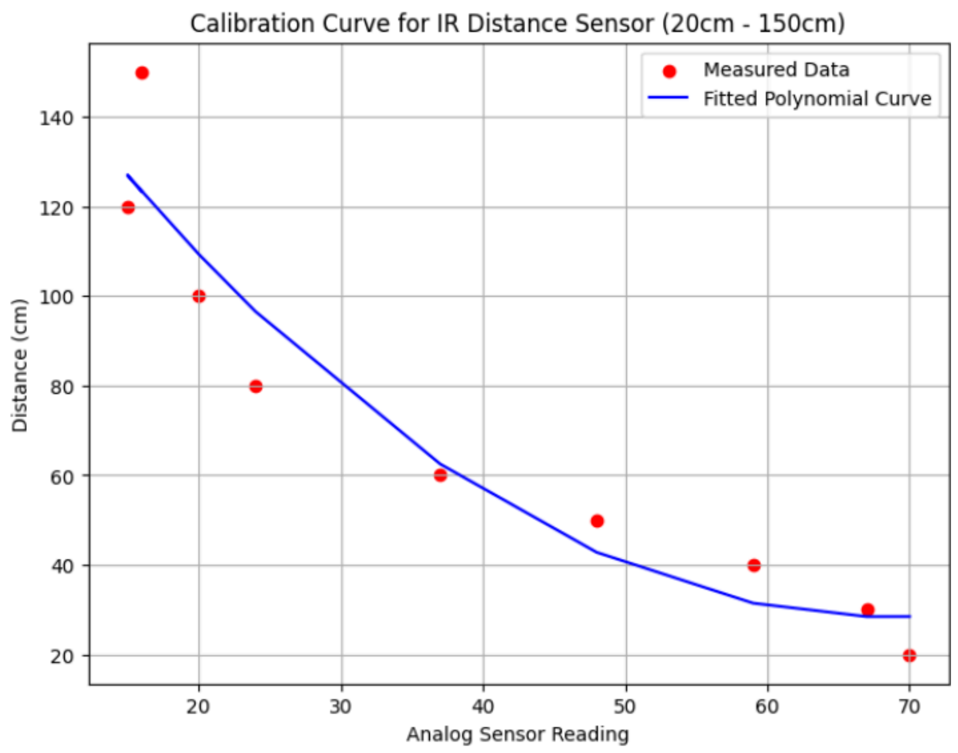

Figure 1: Calibration Curve - Analog Voltage vs. Distance

3. Error Plot: Predicted Distance vs. Actual Distance

We measured several new distances (25 cm, 45 cm, 65 cm, 85 cm, and 110 cm) and recorded the corresponding analog readings. Using the polynomial calibration function, we then predicted the distances based on these new analog readings.

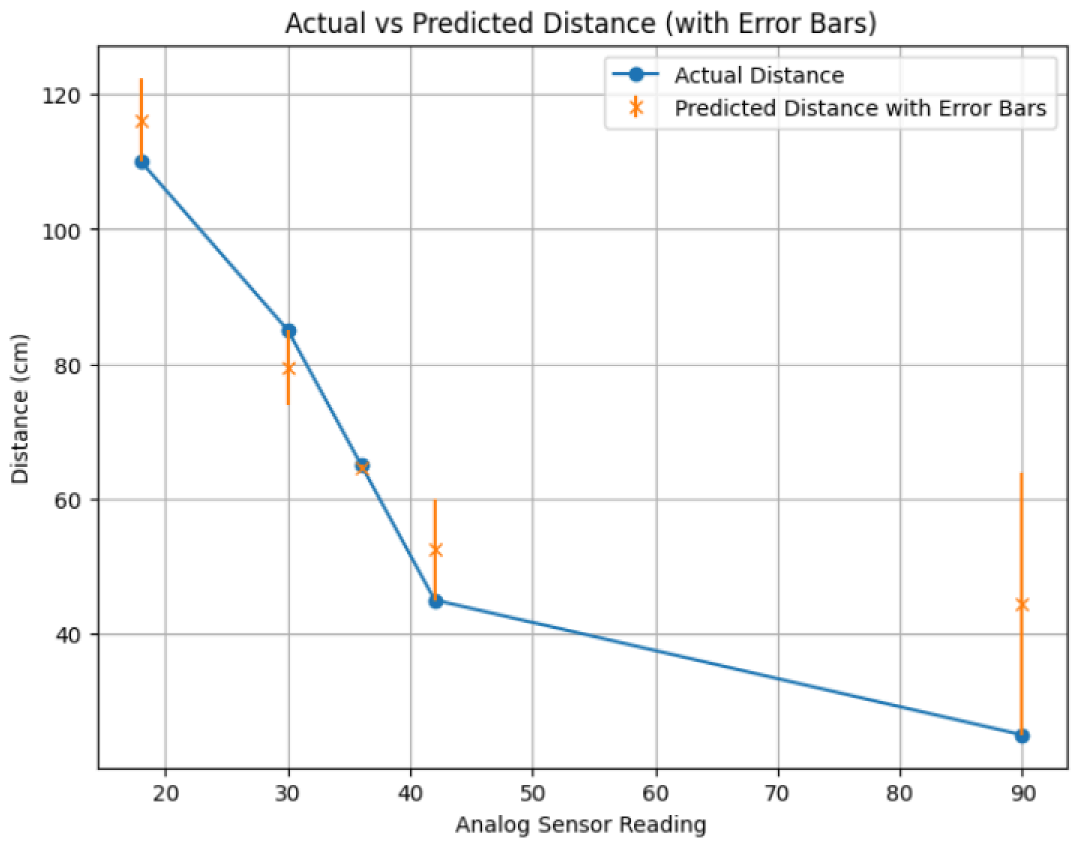

Figure 2: Error Plot - Predicted Distance vs. Actual Distance

4. Explanation of the Calibration Function

The calibration function is a quadratic polynomial that was fitted to the data points:

$$Distance = 0.0345 \times (Analog\ Reading)^2 - 4.7260 \times (Analog\ Reading) + 190.0926$$

This function represents the relationship between the analog sensor reading measured by the IR sensor and the distance. For analog readings between approximately 15 and 70, which correspond to distances from 20 cm to 150 cm, the polynomial model fits well and provides accurate distance predictions.

5. Image of the Setup for 1 Servo Scan and Horizontal Data

We connected the pan servo to pin 10 and mounted the distance sensor on it. This setup allowed us to scan the alphabet block along a horizontal plane.

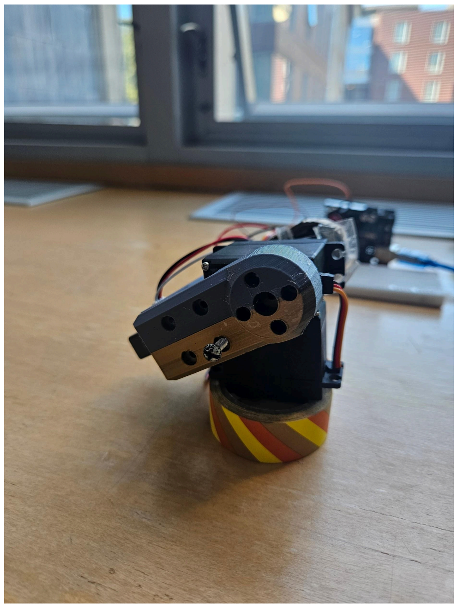

Figure 3: Single Servo Setup forVertical Scan

We then collected data for a horizontal scan of the letter "E" and plotted the results.

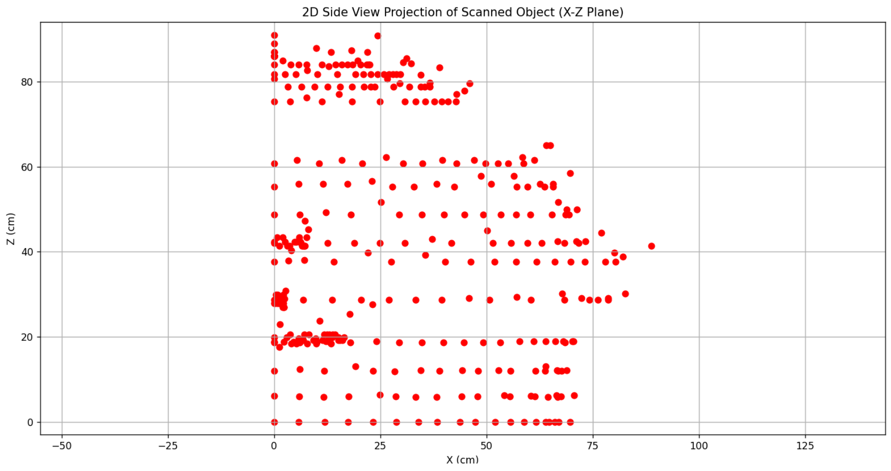

Figure 4: Visualization of the Vertical Scan Data

6. Image of the Setup for 2 Servo Apparatus and 3D Data Visualization

For the full 3D scan, we introduced the tilt servo and connected it to pin 9. The pan and tilt servos worked together to scan the alphabet block at various angles.

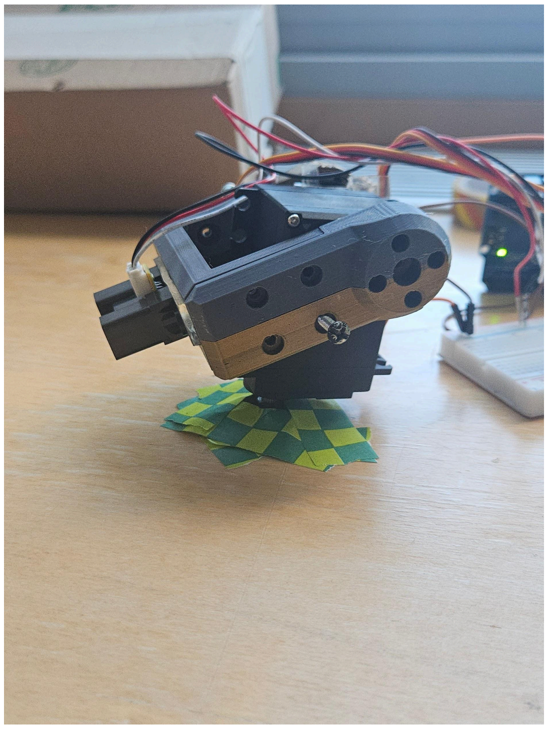
Figure 5: Two-Servo Setup for 3D Scan

We then visualized the 3D data:

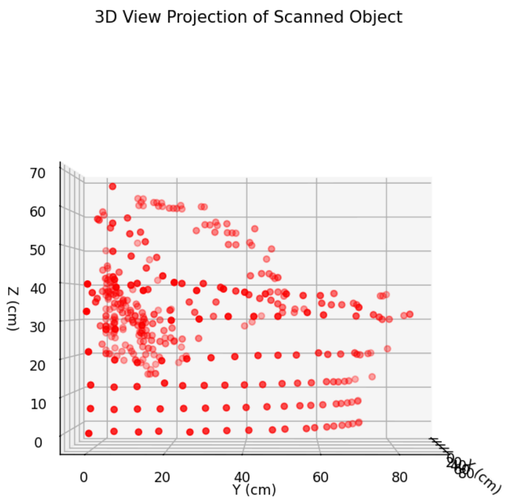

Figure 6: 3D Data Visualization of the Scanned Object

This plot shows a dense set of points representing the side view of the letter "E", scanned from multiple angles. One possible reason why the top of the letter E is curved is because our cardboard letter was not stable.

7. Circuit Diagram and Explanation

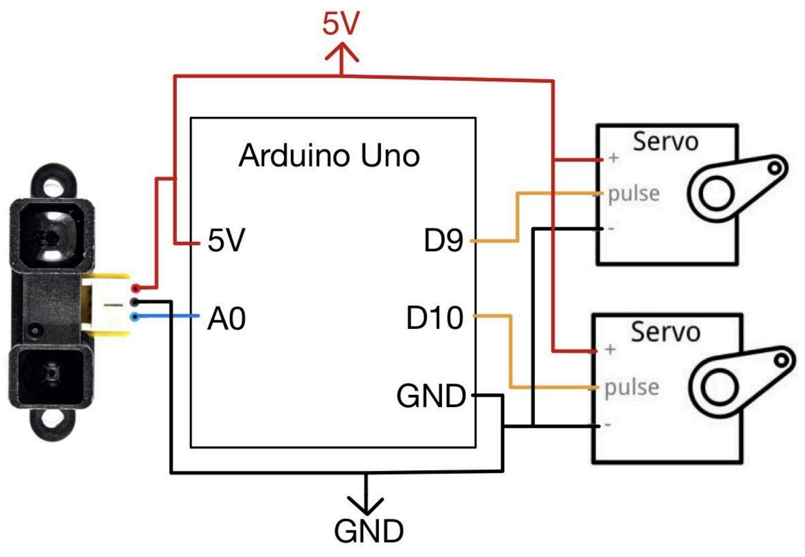

Figure 7: Circuit Diagram with Two Servos, Distance Sensor, and the Arduino

Connections:

1. Pan Servo connected to digital pin 10.
2. Tilt Servo connected to digital pin 9.
3. IR Distance Sensor connected to analog pin A0.

This setup allows the Arduino to control the two servo motors and read distance data from the IR sensor, with all components sharing a common power and ground connection.

8. Explanation of and Reflection on the Design

Our software used a combination of the Arduino IDE and Python for data collection and visualization. We controlled both servos using Arduino's Servo library and mapped the analog sensor readings to distance values. The serial output from the Arduino was processed using a Python script that filtered out points and visualized in 2D.

The mechanical setup involved mounting the IR sensor onto the servos. This setup worked well for our needs, though some improvements could be made, such as securing the sensor to prevent movement during scanning, which occasionally caused minor inaccuracies in the distance readings.

Improvements could be made in two areas: servo stability and the calibration function. During fast movements, the servos sometimes caused vibrations, which affected the accuracy of the readings. A sturdier mount or slowing down the servo movements would help reduce these vibrations and improve accuracy. Additionally, while the basic linear mapping for the calibration function worked reasonably well, using a more precise model, such as a polynomial function, could better capture the sensor's non-linear behavior and enhance distance accuracy.

9. Source Code

Arduino Source Code: Servo Control and Data Collection

```
#include <Servo.h>

Servo servoPan;
Servo servoTilt;

const int panServoPin = 10;  // Pan servo
const int tiltServoPin = 9;  // Tilt servo
const int sensorPin = A0;    // Distance sensor

float sensorMinDist = 20;    // Minimum measurable distance in cm
float sensorMaxDist = 150;   // Maximum measurable distance in cm

int panAngle = 0; // Initialize angles
int tiltAngle = 0;

bool running = true;  // Variable to control scan execution

void setup() {
  servoPan.attach(panServoPin);
  servoTilt.attach(tiltServoPin);

  Serial.begin(9600);
}

void loop() {
  if (running) {
    // Loop through tilt angles (0 to 90 degrees in 5-degree steps)
    for (tiltAngle = 0; tiltAngle <= 90; tiltAngle += 5) {
      servoTilt.write(tiltAngle);  // Set tilt servo angle
      delay(300);  // Allow servo to move to the correct position

      // Loop through pan angles (0 to 90 degrees in 5-degree steps)
      for (panAngle = 0; panAngle <= 90; panAngle += 5) {
        servoPan.write(panAngle);  // Set pan servo angle
        delay(300);  // Allow servo to move to the correct position

        int sensorValue = analogRead(sensorPin); // Read value from the sensor

        // Map the analog sensor value to a distance (20 to 150 cm)
        float distance = map(sensorValue, 0, 1023, sensorMinDist, sensorMaxDist);
        // Send the pan angle, tilt angle, and distance to the serial monitor
        Serial.print(panAngle);
        Serial.print(",");
        Serial.print(tiltAngle);
        Serial.print(",");
        Serial.println(distance);
      }
    }
    running = false;  // Stop the loop after one full scan
  }
}
```

This Arduino code controls two servos (pan and tilt) to scan the front of an upright object. For each combination of pan and tilt angles, it reads the distance sensor data and sends the results to the serial monitor as a CSV line (`pan_angle, tilt_angle, distance`). The angles are increased in 5-degree steps to ensure a dense set of points for accurate plotting.

Python Source Code: Data Processing and Visualization

```
import pandas as pd
import matplotlib.pyplot as plt
import math

df = pd.read_csv('scan_data.csv', header=None,
                    names=['PanAngle', 'TiltAngle', 'Distance'])

# Convert angles from degrees to radians for trigonometric calculations
df['PanAngleRad'] = df['PanAngle'].apply(math.radians)
df['TiltAngleRad'] = df['TiltAngle'].apply(math.radians)

# Define a distance threshold to filter out irrelevant points
min_distance = 20  # Minimum distance in cm (to remove noise)
max_distance = 150  # Maximum distance in cm (to remove background)

# Filter the DataFrame based on the valid distance range
df_filtered = df[(df['Distance'] >= min_distance) & (df['Distance'] <= max_distance)]

# Compute Cartesian coordinates for 3D space (X, Y, Z) from the spherical data (pan,
tilt, distance)
df_filtered['X'] = df_filtered['Distance'] *
df_filtered['TiltAngleRad'].apply(math.cos) *
df_filtered['PanAngleRad'].apply(math.cos)
df_filtered['Y'] = df_filtered['Distance'] *
df_filtered['TiltAngleRad'].apply(math.cos) *
df_filtered['PanAngleRad'].apply(math.sin)
df_filtered['Z'] = df_filtered['Distance'] *
df_filtered['TiltAngleRad'].apply(math.sin)
```

```
# 2D Side View Projection (X-Z Plane) for the scanned object
plt.figure(figsize=(8, 8))
plt.scatter(df_filtered['X'], df_filtered['Z'], c='r', marker='o')
plt.xlabel('X (cm)')
plt.ylabel('Z (cm)')
plt.title('2D Side View Projection of Scanned Object (X-Z Plane)')
plt.grid(True)
plt.axis('equal')
plt.show()
```

This Python script reads the scanned data from a CSV file (with columns `pan_angle`, `tilt_angle`, `distance`) generated by the Arduino. It filters the data based on the distance threshold to eliminate background noise. It calculates Cartesian coordinates from the spherical coordinates (pan and tilt angles, and distance) and visualizes the result using a 2D side view (X-Z projection).