PIE Mini Project 2:
3D Pan/Tilt Scanner with Infrared Distance Sensor

Anna Luiza Reis Braun, Trinity Haisch, Kuhu Jayaswal

September 26, 2025

1 Introduction

This mini-project focused on designing and building a rudimentary 3D scanner using a pan/tilt mechanism driven by two hobby servo motors and an infrared distance sensor, all controlled by an Arduino. The scanner was tasked with capturing spatial data from an object of known geometry and reconstructing its 3D profile through software-based visualization. The system integrated mechanical design, electrical wiring, and sensor calibration and data was transmitted via serial communication and processed using Python and MATLAB to generate both 2D and 3D visualizations. This project helped reinforce key skills in working with sensors and actuators, and introduced the challenge of turning physical measurements into digital models.

2 Project Breakdown

2.1 Mechanical Design

The pan/tilt mechanism was inspired by a telescope mount, which naturally provides two-axis movement. The objective of the design was to enable the infrared distance sensor to rotate only in the horizontal (pan) and vertical (tilt) directions, ensuring controlled scanning of the target geometry.

Two Vigor VS-2 hobby servos were used to drive the motion. The base servo provides horizontal rotation, while the second servo, mounted orthogonally on an arm, controls vertical tilt. This compact configuration allowed the sensor to sweep across the target plane without introducing unnecessary complexity.

All custom components were modeled in CAD and fabricated with a 3D printer. Particular attention was given to tolerance design: parts were dimensioned to achieve reliable press fits where needed, such as the servo horns and arm connector, while still allowing for disassembly. To improve smoothness in the tilt axis, a needle bearing assembly was added between the arm connector and its resting plate, reducing friction and preventing binding during motion.

The sensor was mounted to the tilt arm using 4-40 bolts and nuts, while the servo horns were fixed with M2.6 $\times$ 8 mm self-tapping screws. Additional structural fasteners, such as hex-head serrated screws and other screws were used to attach another support to the sensor arm and maintain the box closed, respectively.

The bottom box was designed to house both the Arduino Uno and a breadboard and also to provide wider support for the mechanism. This addition kept the electronics secure, improved wire management, and ensured that the mechanism did not tip over. The design met the project disassembly requirement relying solely on screws and friction fits.

The key constraints addressed in the design included the following:

- Stability: A wide base and vertical supports reduced the risk of tipping.
- Disassembly: The design avoided adhesives or duct tape, ensuring that all parts could be reused.
- Printability: The parts were iteratively adjusted in CAD to accommodate 3D printer tolerances.

2.2 Electrical Design

Hardware Used:

- Arduino Uno R4
- Servo Motors (pan and tilt)
- IR Infrared Distance Sensor
- Breadboard & Wires

A compact breadboard was used to connect the components. The pan and tilt servo motors were wired to digital pins 3 and 6 on the Arduino Uno R4, respectively. The infrared distance sensor was connected to analog pin A0 to capture analog voltage readings corresponding to object distance. Power and ground connections were routed through the breadboard across one row.

We tested the infrared distance sensor by connecting it to analog pin A0 on the Arduino and using the "AnalogInput" example sketch to monitor voltage readings. We placed objects at known distances and recorded the sensor outputs to build a calibration dataset. This allowed us to verify the sensor's responsiveness and establish a reliable relationship between voltage and physical distance.

2.3 Software

The software workflow involved three main components: Arduino control code, serial data transmission, and visualization scripts in Python and MATLAB. The Arduino collected analog distance readings and servo angles, then sent the data over serial to a Python script running on the laptop. Python handled data parsing and organization, storing each sweep as a nested list corresponding to servo positions and sensor values. These structured datasets were then imported into MATLAB, where we generated both top-down and 3D visualizations of the scanned object. This multi-platform approach allowed us to separate hardware control from data processing and visualization, making the system easier to debug and extend.

Python handled data parsing and organization, storing each sweep as a nested list corresponding to servo positions and sensor values. The scanning pattern followed a zigzag trajectory: each horizontal sweep was performed at a fixed tilt angle, starting from left to right, then incrementing the tilt slightly and sweeping right to left, and so on. This alternating pattern preserved spatial continuity and simplified grid construction for later visualization. It also ensured faster scanning by eliminating the need to reset the pan servo to its starting position after each sweep, allowing continuous motion and reducing mechanical delay.

2.4 Sensor Calibration

To calibrate the infrared distance sensor, we collected seven measurements by placing objects at known distances and recording the corresponding analog voltage readings from the sensor. These readings were mapped to physical distances using MATLAB's polyfit function to generate a second-degree polynomial calibration equation. The resulting equation was the following where ADC is the analog-digital converter:

$$\text{Distance} = 8.0649 \times 10^{-4} \cdot \text{ADC}^2 - 1.0429 \cdot \text{ADC} + 344.78 \tag{1}$$

This equation was used to transform raw sensor data into estimated distance values throughout the scan. Using the calibrated dataset, we generated both a surface plot and a heatmap to visualize the scanned shape. To evaluate the accuracy of the calibration, we also created error plots comparing the predicted distances to the actual measurements used during calibration. These plots are included below.

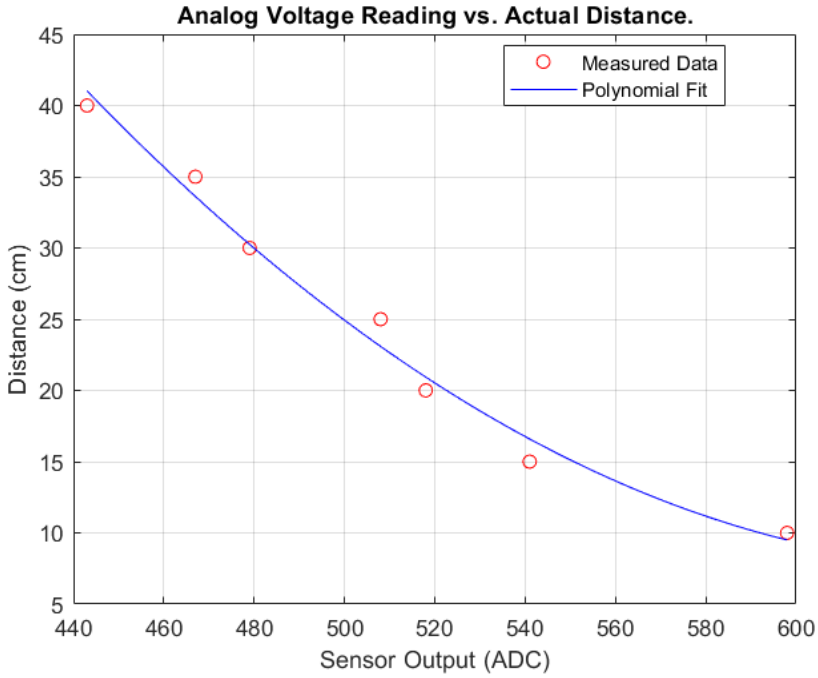

Figure 1: Sensor calibration curve showing the relationship between analog voltage readings (ADC values) and actual measured distances. Red circles represent the measured data points, and the blue curve shows the second-degree polynomial fit used to generate the calibration equation.

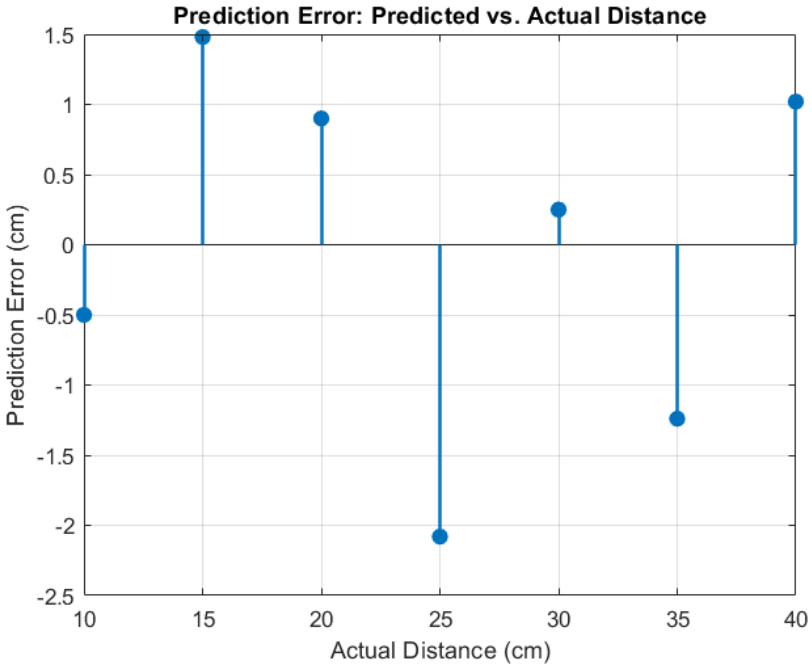

Figure 2: Prediction error plot showing the difference between predicted and actual distances for test measurements. Each blue dot represents the error at a specific actual distance, with vertical lines indicating the magnitude and direction of the error. This visualization helps assess the accuracy of the sensor calibration and highlights any systematic deviations.

3 Implementation

3.1 Circuit Diagram

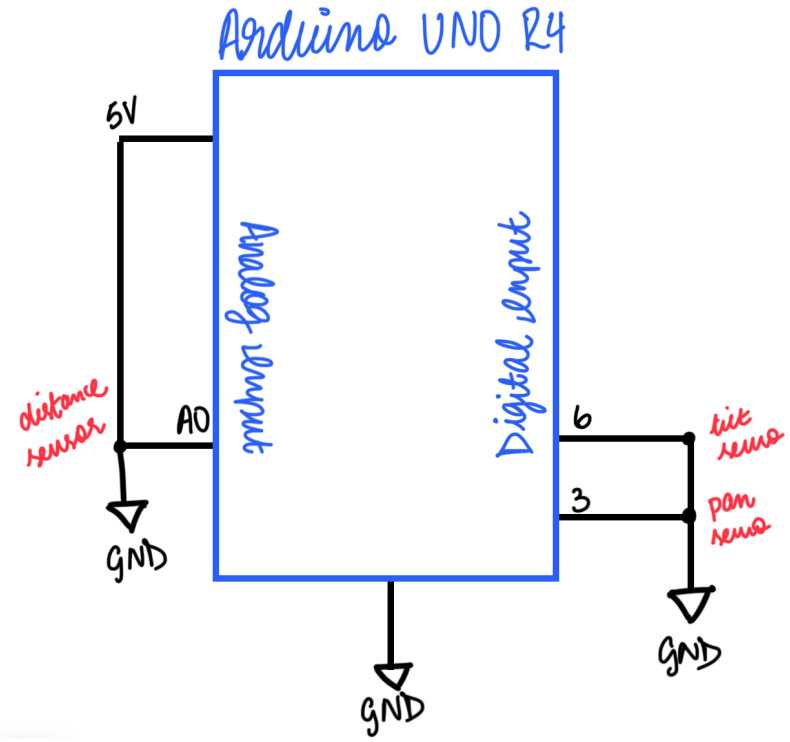

Figure 3: Circuit diagram of Arduino, servos, and sensor.

3.2 Arduino IDE Code

Listing 1: Arduino Sweep Code for Pan/Tilt 3D Scanner

```
#include <Servo.h>

// Create Servo objects for pan and tilt
Servo servo;
Servo servo2;

// Sensor and LED setup
int sensorPin = A0;       // IR sensor analog input
int ledPin = 13;          // Onboard LED (optional)
int sensorValue = 0;      // Stores sensor reading
int outputValue = 0;      // Placeholder (unused)

// Servo position variables
int pos = 0;
int pos2 = 180;
int sensing = 1;          // Flag (unused)

void setup() {
  servo.attach(6);        // Pan servo on digital pin 6
  servo2.attach(3);       // Tilt servo on digital pin 3
  Serial.begin(9600);     // Start serial communication
  pinMode(ledPin, OUTPUT);
}

void loop() {
  sensorValue = analogRead(sensorPin);  // Initial sensor read

```

```
  // Sweep tilt servo from 180 to 90 degrees
  for (pos2 = 180; pos2 >= 90; pos2 -= 2) {

    // Sweep pan servo from 0 to 90 degrees
    for (pos = 0; pos <= 90; pos += 2) {
      servo.write(pos);           // Update pan angle
      servo2.write(pos2);         // Update tilt angle
      sensorValue = analogRead(sensorPin);  // Read sensor
      Serial.print(servo.read()); Serial.print(" ");
      Serial.print(servo2.read()); Serial.print(" ");
      Serial.println(sensorValue);          // Send data
      delay(200);                           // Wait
    }

    pos2 -= 5;  // Additional decrement for tilt

    // Sweep pan servo back from 90 to 0 degrees
    for (pos = 90; pos >= 0; pos -= 2) {
      servo.write(pos);
      servo2.write(pos2);
      sensorValue = analogRead(sensorPin);
      Serial.print(servo.read()); Serial.print(" ");
      Serial.print(servo2.read()); Serial.print(" ");
      Serial.println(sensorValue);
      delay(200);
    }
  }
}
```

Listing 1 contains the Arduino code used to control the pan and tilt servos and collect distance measurements from the infrared sensor. The `setup()` function initializes the servos and serial communication. In the `loop()`, the tilt servo sweeps from 180° to 90°, and for each tilt angle, the pan servo sweeps from 0° to 90° and back. At each position, the sensor reads an analog voltage corresponding to distance, and the Arduino sends the pan angle, tilt angle, and sensor value over serial. This structured sweep enables a full scan of the object's surface, with synchronized mechanical movement and data collection.

3.3 Python Data Parsing Code

Listing 2: Python Script for Serial Data Parsing and Grid Construction

```
import serial

# Set up serial communication with Arduino
arduinoComPort = "COM4"
baudRate = 9600
serialPort = serial.Serial(arduinoComPort, baudRate, timeout=1)

# Initialize data structures
grid = []
row = []
invert_row = 0

# Continuously read data from Arduino
while True:
    lineStr = serialPort.readline().decode()
    print(lineStr)

    if lineStr:
        split_data = lineStr.split(" ")
        servo1 = int(split_data[0])
        servo2 = int(split_data[1])
        sensor_str = split_data[2].split("\r\n")
        sensor_data = int(sensor_str[0])
        line_data = int(sensor_data)

        # Organize data into rows based on servo1 position
        if servo1 == invert_row:
```

```
            if len(grid) % 2 == 0:
                grid.append(row)
            else:
                row.reverse()
                grid.append(row)
            row = []
        else:
            row.extend([line_data])

        invert_row = servo1
        print(grid)
```

Listing 2 shows the Python script used to receive and organize data from the Arduino via serial communication. The script reads each line of incoming data, splits it into servo angles and sensor readings, and stores the values in a nested list structure. The `grid` variable accumulates rows of distance data, reversing alternate rows to preserve spatial continuity during the zigzag sweep. This organized dataset is later used for visualization in MATLAB, enabling reconstruction of the scanned shape in both top-down and 3D views.

3.4 MATLAB Visualization Code

Listing 3: MATLAB Code Visualization Post Sensor Calibration

```
% raw sensor data
star_data = [
    474, 428, 429, 435, 432, 439, 438, 637, 544, 545, 459, 474, 474, 473, 506, 438,
        488, 430, 438, 434, 457, 514, 455, 457, 464, 467, 480, 522, 632, 564, 483, 420,
         421, 475, 420, 441, 390, 372, 406, 415, 419, 411, 408, 209, 42;
    ... % truncated for brevity
];

% calibration equation
Distance = -(8.0649e-04 * star_data.^2 - 1.0429e+00 * star_data + 344.78);

% 3D surface plot
figure;
surf(Distance);
xlabel('Column'); ylabel('Row'); zlabel('Distance');
title('3D Distance Surface Plot');
colorbar;
colormap(flipud(parula));
shading interp;
view([39 75]);

% 2D top view (heatmap)
figure;
figure('Units','normalized','Position',[0.2 0.2 0.6 0.6]);
h = heatmap(flipud(Distance));
h.XDisplayLabels = string(1:size(Distance,2));
h.YDisplayLabels = string(1:size(Distance,1));
h.Colormap = flipud(parula);
h.Title = 'Distance Heatmap (Top View)';
h.XLabel = 'Column';
h.YLabel = 'Row';
```

Listing 3 shows the MATLAB script used to transform raw sensor data into physical distance measurements and generate visualizations of the scanned shape. The calibration equation, derived from polynomial curve fitting, is applied to each analog reading to compute estimated distances. These values are then plotted in two formats: a 3D surface plot and a 2D heatmap. The 3D plot provides a spatial representation of the scanned object, highlighting depth and contour, while the top-down heatmap offers a clearer view of surface geometry and symmetry. Including both perspectives allows for a more complete interpretation of the scan results, which are further analyzed in the Results section.

3.5 CAD Model / Mechanical Assembly

The mechanical system was developed through an iterative CAD design process. The initial models were based in measurements from the provided CAD files for the VS-2 servos and the sensor. These baseline files ensured correct mounting geometry, but all critical dimensions were also verified with calipers. This dual approach prevented errors from small mismatches between the nominal CAD data and the actual hardware. Each CAD iteration refined the servo mounts, sensor bracket, and base enclosure, with updates focusing on alignment, printability, and ease of disassembly.

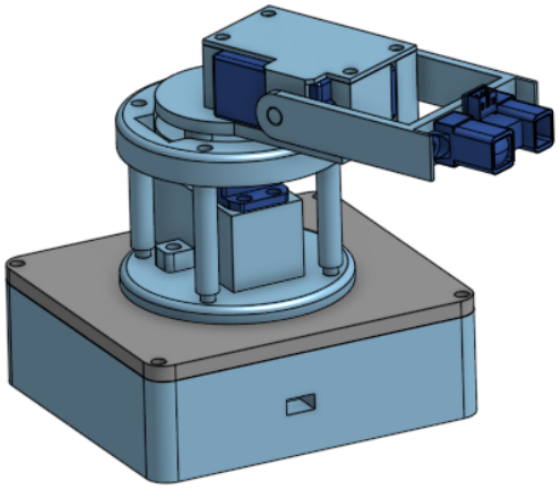

Figure 4: Final CAD model

The base enclosure was a key part of the design. It was modeled as a single unit with dedicated recesses for the Arduino Uno and a small breadboard. This decision ensured that the electronics stayed fixed during operation and provided an organized space for routing wires. By designing the enclosure to "lock" the electronics in place, cable strain was reduced, connectors stayed more secure, and the prototype remained cleaner overall.

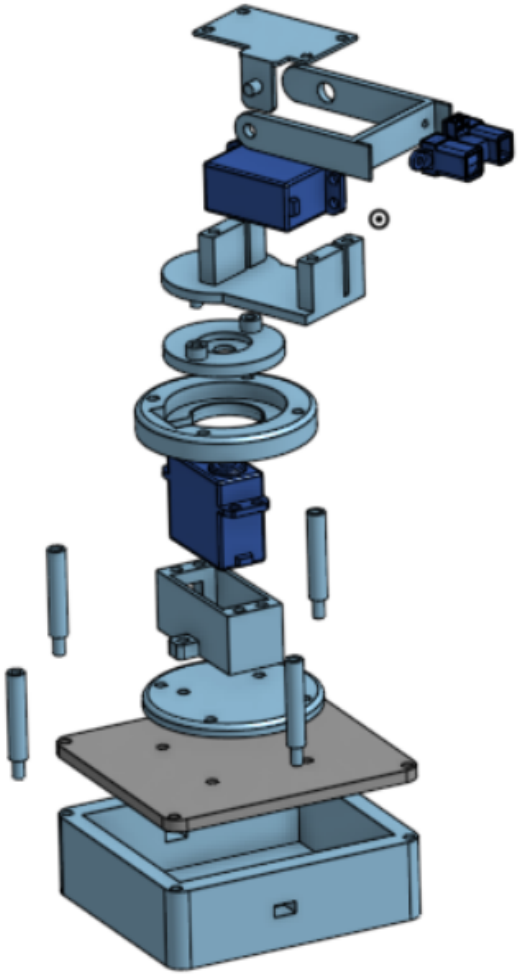

Figure 5: Exploded CAD view

The servo arrangement was designed to produce stable and predictable two-axis motion. The horizontal (pan) servo was fixed rigidly to the base to provide a stable reference frame. On top of this, the vertical (tilt) servo was mounted orthogonally on the rotating plate driven by the pan axis. This hierarchy allowed the tilt axis to move with the pan axis, creating a compact two-degree-of-freedom mechanism. The distance sensor was fixed to the tilt arm using 4-40 bolts and nuts for reliability. Bellow the horizontal rotating plate, a needle bearing was introduced to minimize friction. This addition was critical for ensuring smooth motion during repeated scanning cycles, preventing abrasions that might have occurred with the 3d printed parts contact.

Throughout the CAD process, tolerancing for 3D printing was explicitly considered. Since FDM printers typically produce dimensional variation of ± 0.1 mm in Z and up to ± 0.3 mm in X/Y, adjustments were built into the CAD model to ensure consistent fits. Specifically:

- Friction-fit slots (e.g., for servo horns or vertical servo mount) were modeled with +0.10–0.15 mm clearance relative to the measured component dimensions. This consistently produced snug fits after printing without adhesives.
- Clearance holes for bolts were oversized by $\tilde{0}.25$ mm relative to nominal size to account for hole shrinkage, ensuring screws and nuts could pass through cleanly without post-processing.

Overall, the combination of CAD iteration, measurement-based design, and print-aware tolerancing produced a structure that was both functional and practical. The mechanism housed the electronics

securely, provided smooth and repeatable pan/tilt motion, and can be disassembled without damage, as seen in the following images:

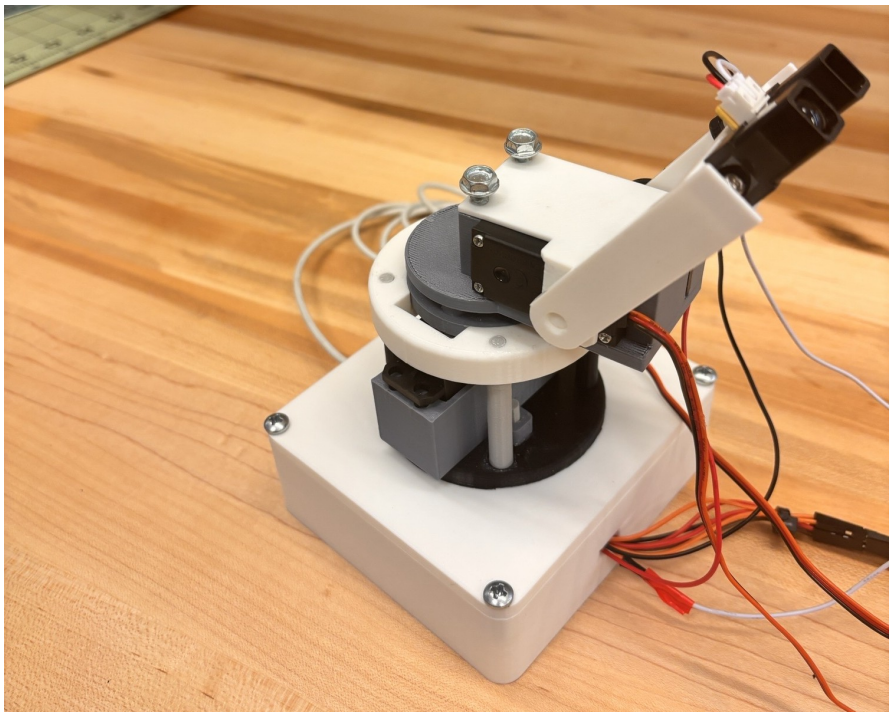

Figure 6: Final Assembly

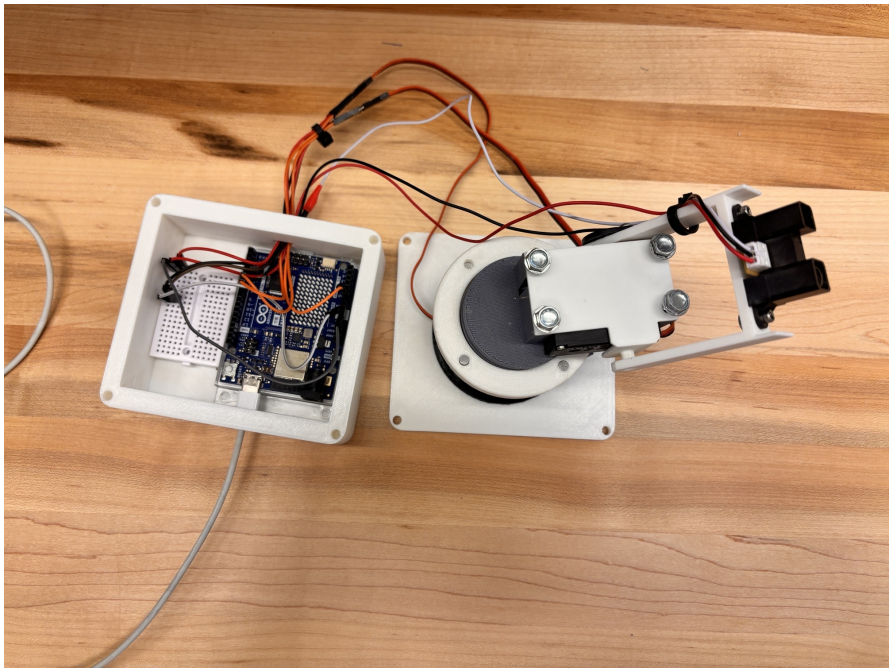

Figure 7: Mid-assembly photo showing the eletronics inside the box

4 Results

4.1 Single Servo Scan

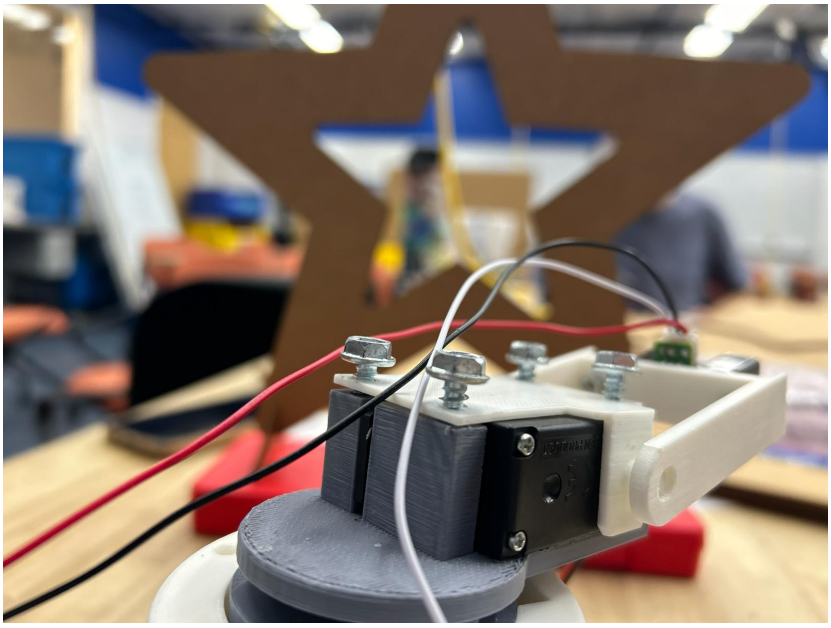

Figure 8: Single-Servo Horizontal Sweep of Target Shape

The data collected from this horizontal sweep was stored as a single row within a larger 2D array structure, representing one pass across the shape. Each value corresponds to an analog reading from the IR sensor at a specific servo angle. An example row from this scan is shown below:

```
star_data = [
  [474, 428, 429, 435, 432, 439, 438, 637, 544, 545, 459, 474, 474, 473, 506, 438,
488, 430, 438, 434, 457, 514, 455, 457, 464, 467, 480, 522, 632, 564, 483, 420, 421,
475, 420, 441, 390, 372, 406, 415, 419, 411, 408, 209, 42
]
```

This row reflects the sensor's response to surface geometry as the servo rotated across the horizontal axis. In the full scan, each row like this was appended to a grid to form a complete 2D dataset for visualization.

4.2 3D Surface Plot of Single Sweep

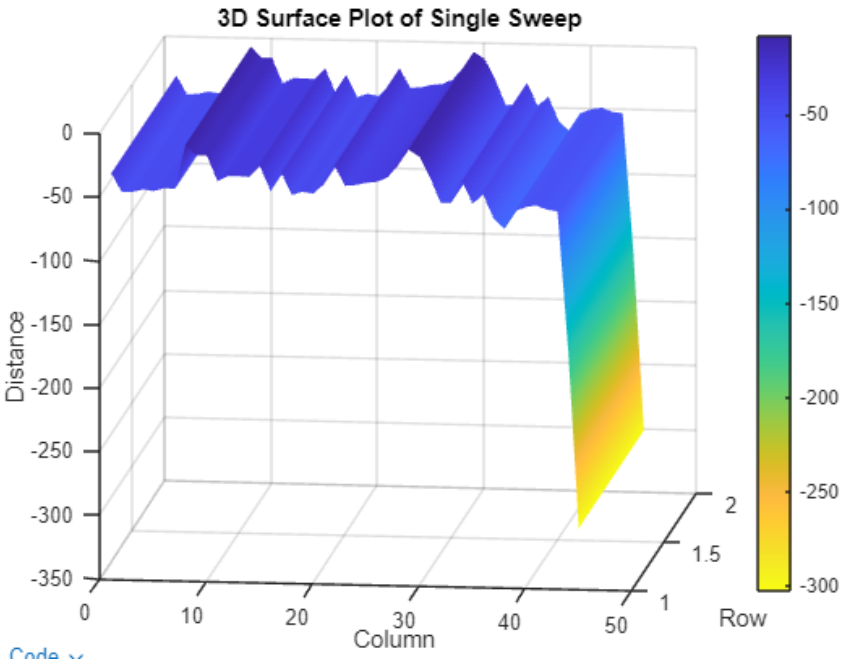

Figure 9: 3D Surface Plot of Distance Readings from a Single Horizontal Sweep

This 3D surface plot visualizes the calibrated distance data collected from a single horizontal sweep using one servo. Although the dataset consists of only one row, the surface plot helps illustrate how distance varies across the sweep. A sharp drop-off near the end of the sweep indicates a significant change in surface geometry, and the bottom two corners of the star-shaped object begin to emerge in the profile.

4.3 Color-Mapped Bar Plot of Single Sweep

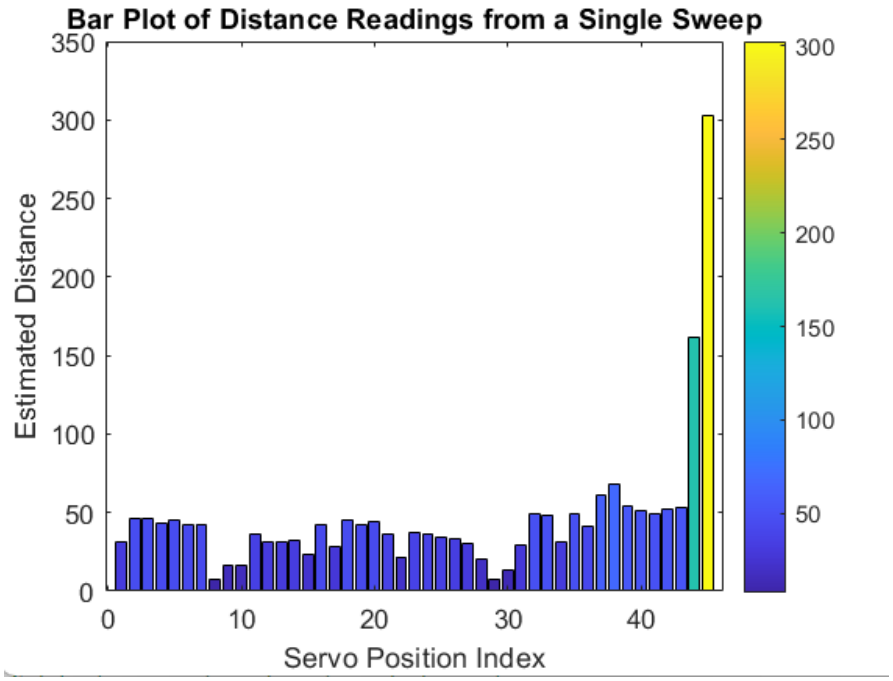

Figure 10: Bar Plot of Distance Readings from a Single Sweep

This bar plot displays the calibrated distance readings from a single horizontal sweep, with each bar representing a measurement at a specific servo position. The bars are color-mapped using a gradient to reflect the magnitude of the distance values, enhancing interpretability. Taller bars toward the end of the sweep suggest the presence of objects or features at greater distances. This visualization provides a clear and intuitive view of the bottom two corners of the star as well.

4.4 Two Servo Scan

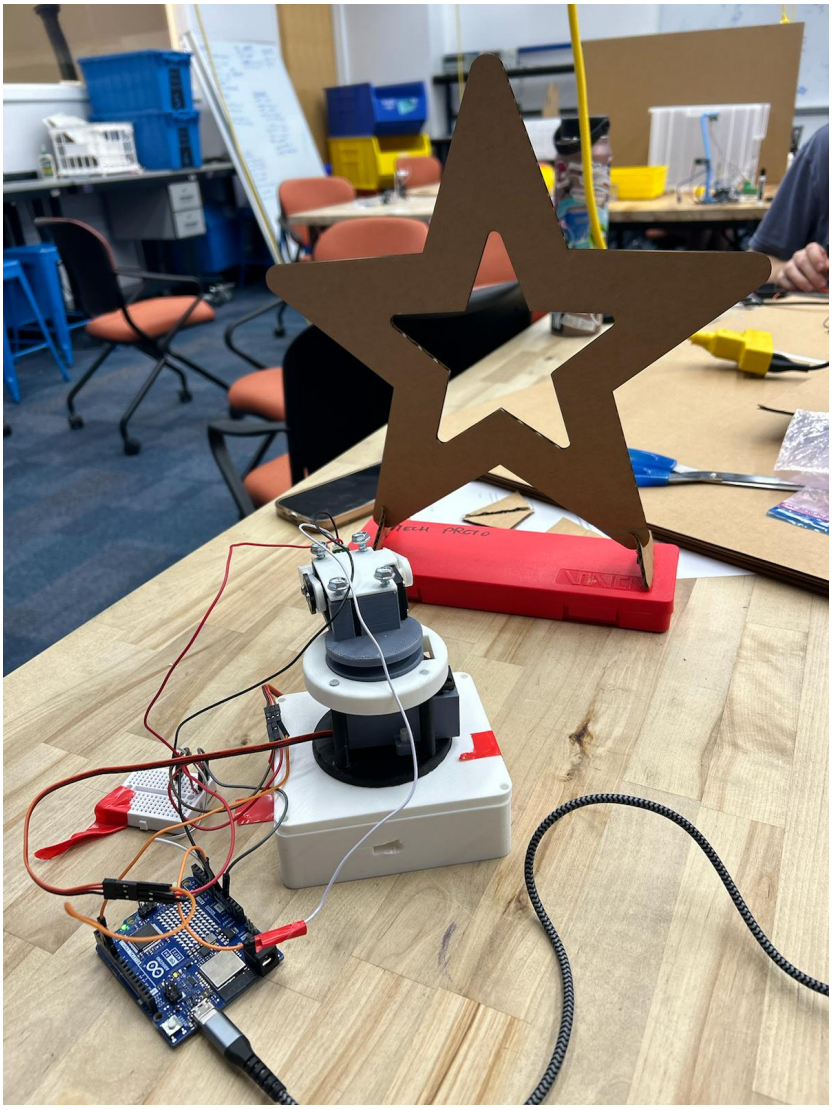

Figure 11: Dual-Servo Pan/Tilt Scan of Target Shape

The upgraded scanning system used two servos—one for horizontal (pan) and one for vertical (tilt) motion—allowing the IR sensor to sweep across a full 2D grid. At each (pan, tilt) coordinate, the sensor recorded an analog voltage corresponding to distance. These readings were stored row-by-row in a 2D array, forming a complete spatial dataset.

An excerpt from the collected data is shown below. Each row represents a horizontal sweep at a fixed tilt angle, and together they form the full scan grid:

```
star_data = [
474, 428, 429, 435, 432, 439, 438, 637, 544, 545, 459, 474, 474, 473, 506, 438, 488,
430, 438, 434, 457, 514, 455, 457, 464, 467, 480, 522, 632, 564, 483, 420, 421, 475,
420, 441, 390, 372, 406, 415, 419, 411, 408, 209, 42;
352, 348, 352, 359, 359, 333, 589, 591, 664, 601, 589, 504, 356, 380, 361, 342, 321,
341, 344, 350, 387, 348, 420, 562, 613, 604, 594, 606, 543, 160, 218, 214, 142, 7,
7, 13, 10, 7, 7, 12, 19, 7, 7, 8, 7;
...
]
```

This matrix captures the full spatial profile of the scanned object. In later steps, this data was calibrated and visualized using MATLAB to produce both 3D surface plots and top-down heatmaps which are shown below.

4.5 3D Surface Plot of Scanned Shape

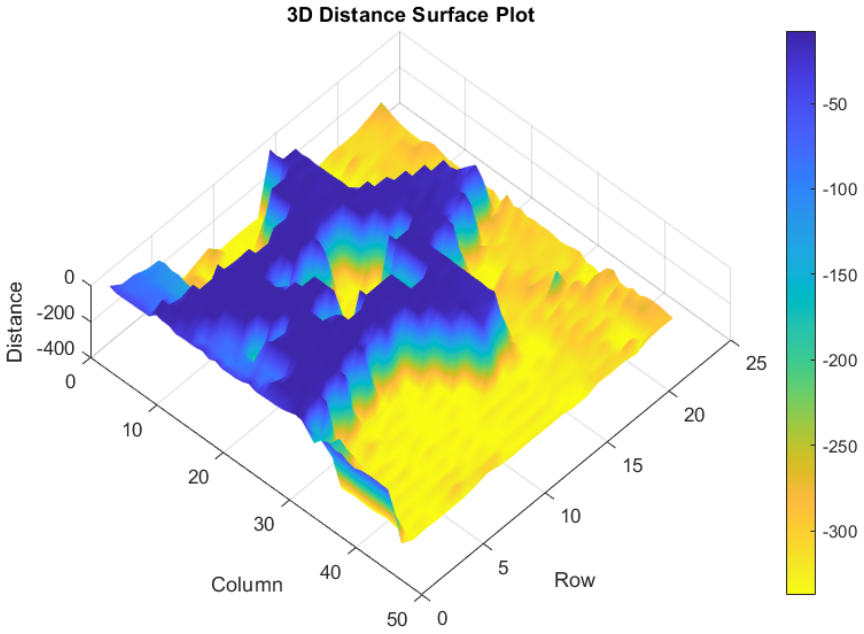

Figure 12: 3D Distance Surface Plot Generated from Dual-Servo Scan

This 3D surface plot visualizes the calibrated distance data collected from the pan/tilt scanner. The peaks and valleys in the surface clearly reveal the contour of the scanned shape, allowing us to map its geometry in physical space. The color gradient enhances interpretability, with cooler tones representing closer surfaces and warmer tones indicating greater distances.

4.6 2D Heatmap of Scanned Shape

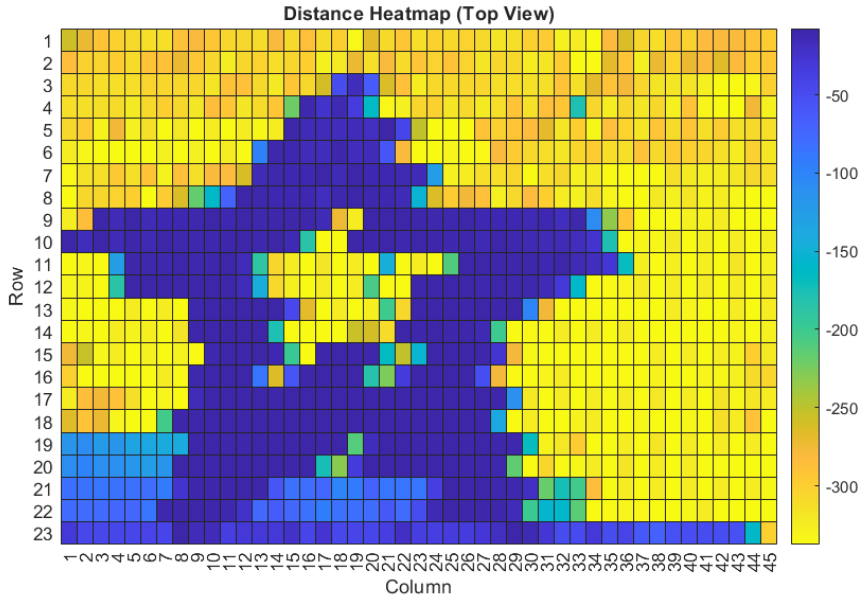

Figure 13: Distance Heatmap (Top View) Generated from Dual-Servo Scan

This heatmap provides a top-down view of the scanned shape, with each cell representing a calibrated distance measurement at a specific pan/tilt coordinate. The star-shaped contour is clearly visible in the center, where darker colors indicate closer surfaces. Surrounding regions appear in warmer tones, signifying greater distances. This visualization complements the 3D surface plot by offering a clearer perspective on surface geometry and symmetry.

5 Reflection

5.1 Software Workflow

One of the key strengths of our software design was the emphasis on scan efficiency. We implemented a zigzag scanning pattern, alternating the pan direction with each tilt increment. This approach significantly reduced mechanical delay by avoiding unnecessary servo resets, allowing for continuous motion and faster data acquisition.

Initially, we experimented with binary thresholding, assigning a value of 1 to detected cardboard and 0 to background based on whether the analog reading crossed a threshold of 99. While this method was computationally simple, the resulting visualizations lacked clarity and failed to capture the shape of the object with sufficient resolution. To address this, we transitioned to recording exact distance measurements and visualizing them using heatmaps. This provided a much richer representation of the scanned geometry.

We also found that our early scans lacked sufficient spatial detail to reconstruct the object's shape. To improve resolution, we increased the number of data points per sweep by reducing the angular step size—effectively scanning with smaller "pixels." This adjustment made the star-shaped contour more distinguishable and enhanced the fidelity of both the heatmap and surface plot outputs.

5.2 Sensor Calibration

Out of curiosity, we plotted the 3D surface using raw analog readings before applying any calibration. Surprisingly, the uncalibrated plot looked nearly identical to the calibrated version. This suggested that the relationship between analog voltage and physical distance was approximately linear. Our polynomial fit confirmed this: the second-degree coefficient was extremely small, indicating minimal curvature in the model.

Although the visual difference between calibrated and uncalibrated data was negligible in this case, calibration remains a crucial step in sensor workflows. It ensures that the data reflects real-world units, enabling meaningful comparisons, error analysis, and integration with other systems. In more complex geometries or with non-linear sensors, calibration can dramatically affect accuracy and interpretability. By validating our calibration curve and observing its minimal impact, we confirmed both the reliability of our sensor and the appropriateness of our modeling approach.

5.3 Mechanically

The design worked well, with the servo arrangement providing stable two-axis motion and the needle bearing minimizing friction for smoother scanning cycles. The base enclosure locked the Arduino and breadboard securely in place. And, because of tolerancing for 3D printing, assembly was smooth.

5.4 Challenges and Future Improvements

Despite the overall success of our system, we encountered several challenges. Servo jitter occasionally disrupted smooth motion, and sensor readings were subject to noise—especially at longer distances. Mounting stability also affected repeatability, as slight shifts in the scanner's position could alter the scan output, also there was a lack of slots through the model for better wire management, which left wires outside the base box unmanaged .

For future iterations, we plan to implement data filtering techniques to smooth out noise and improve measurement consistency. Increasing scan resolution further could help capture finer geometric features. Mechanically, we would focus on integrating a sturdier design that covers the servos for aesthetic purposes, hides all wires, and manages them more effectively. These improvements would collectively strengthen the system's robustness and expand its potential applications.

6 References

- Arduino Forum. *Servo Sweep Example*. Retrieved from https://forum.arduino.cc/t/servo-sweep/604735
- Sharp GP2Y0A02YK0F IR Distance Sensor — Datasheet and CAD files. Provided by course materials.
- Vigor VS-2 Servo — Datasheet and CAD files. Provided by course materials.