Bhargavi Deshpande, Tane Koh, Sam Wisnoski (Group 4)
Principles of Integrated Engineering Section 2

Mini Project 2: DIY 3D Scanner

I. Introduction:

In this project, our team constructed a rudimentary 3D scanner. We used an infrared distance sensor and several servo motors to take distance readings at an assortment of different angles and then used those readings to recreate a 3D scan.

We utilized two Tiankongrc MG996R servo motors, one infrared distance sensor, and an Arduino R4 microcontroller. We used the Arduino to control the servo motors and to read the data given from the distance sensor.

We started the project by creating a basic circuit that could read the data given by the distance sensor and do basic controls over the servo motors. Then, we manually calibrated the distance sensor by doing readings at fixed distances and mapping the voltage readings, and then creating a function to project the relationship between the two. Next, we designed mounts to connect the servos and sensor, and wrote code to control the servo motors and translate the data given by the sensor. Finally, we built a MATLAB function to create a visualization of the readings.

II. Testing, Calibration, and Error:

First, we connected our sensor to the appropriate pins on the Arduino and verified that it worked using the "AnalogInput" example from the Arduino development environment. We used basic arduino commands to read the sensor and print an output voltage for each loop, as seen in figure 2.1.

```
void loop() {    #LOOP OVER read_sensor()
read_sensor()
}

void read_sensor(){
current_voltage = analogRead(SEN);  #READ THE SENSOR DATA
Serial.println(current_voltage);    #PRINT THE SENSOR DATA
}
```

Figure 2.1: Code for reading and displaying distance sensor results

Next, we used a ruler to mark out intervals of five cm between 20 cm and 150 cm, which is the range the distance sensor functions accurately. We placed the sensor on a makeshift stand to ensure it would not sense the ground and held a flat board at each

interval of 5 cm. At each interval, we recorded the distance to the board and the output voltage from the distance sensor to the analog port.

After we collected all the points, we used google sheets to create figure X.X, a graph of Voltage vs. Distance. Taking the line of best fit of this graph, we can approximate a distance given an output voltage. After testing different lines of best fit, we chose to use a power series line of best fit with the equation:

$$\text{Distance} = 16410(\text{Output Voltage})^{-1.06}$$

as found in figure 2.2. The R^2 value for this graph is 0.995, indicating the line of best fit fits the data well.

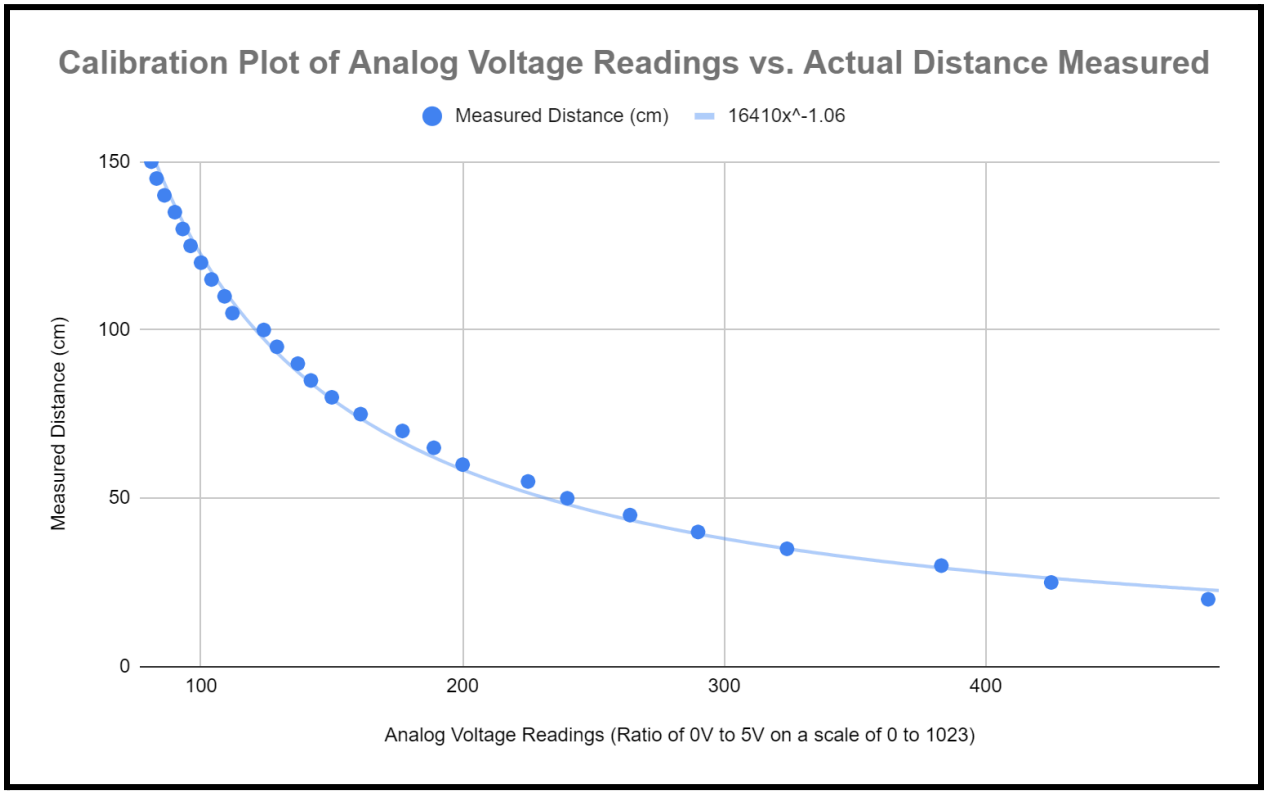

Figure 2.2: Calibration Plot of Analog Voltage Readings vs. Actual Distance Measured

Our calibration function is a mathematical power series function depicting the relationship between the voltage output of a sensor and the distance between the sensor and the object it is hitting. In this case, the input value is the voltage output of the sensor, and the output value of the function is the distance. A negative exponent value indicates that there is an inverse relationship - the higher the voltage, the lower the distance sensed. This function is preferable to a linear equation or a more simple function since the relationship between the variables is more complex and doesn't follow a linear pattern.

We then tested our distance function by taking distance sensor measurements at known locations and using our equation to translate the output to measured distance. An error plot showing predicted distance and actual distance for distances not included in our original calibration routine is shown in figure 2.3. Four of the five points we tested aligned nearly exactly

with our calibration curve, while a single point was a few centimeters off, confirming our chosen function mimics distance accurately.

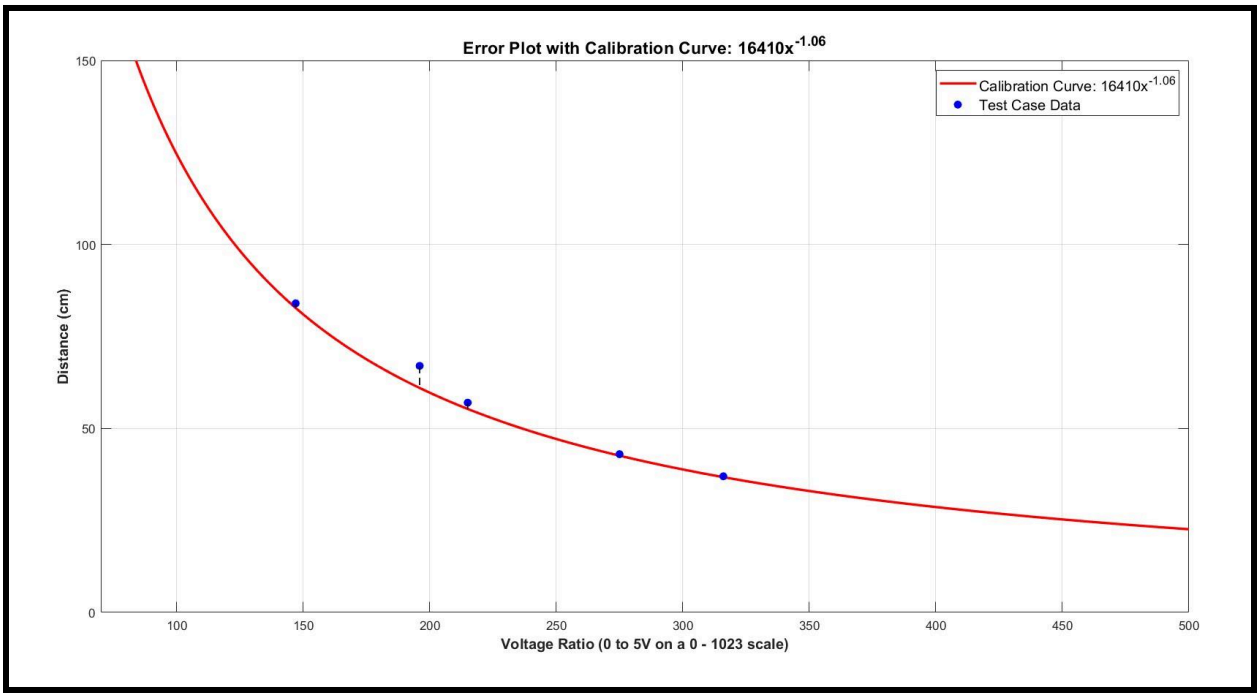

Figure 2.3: Error plot of our predicted distance curve and actual distances of points not used in calibration

III. Design Process

Mechanical Design

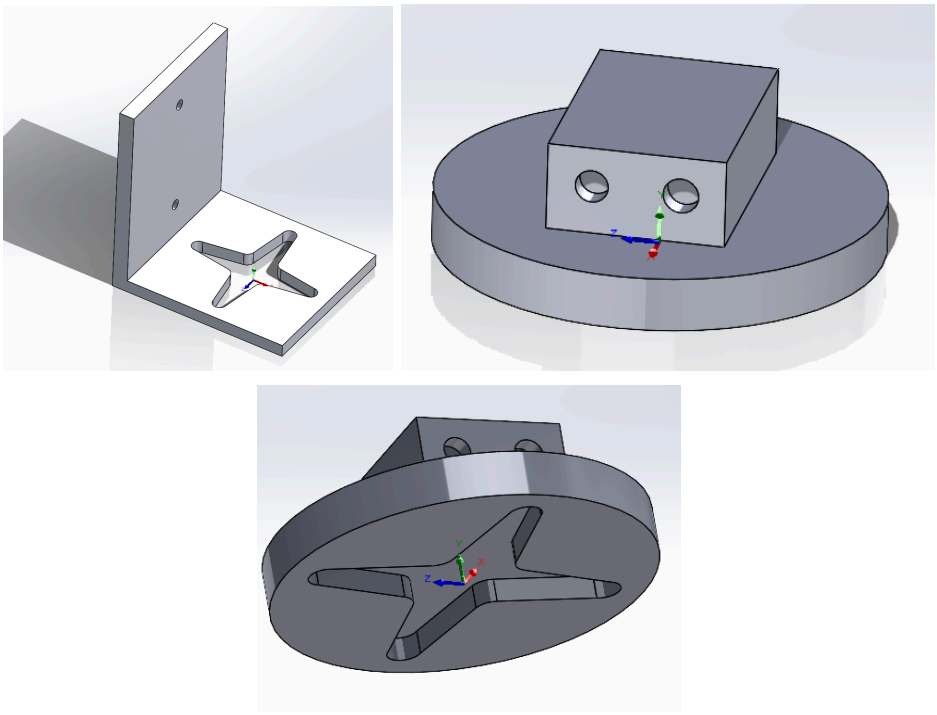

Figure 3.1: Computer-aided Design of pan and tilt mechanism attached to two servos

The mechanical design utilized two servo motors to create pan and tilt features for our infrared sensor. A 3D printed base mount was attached to the shaft of a servo that was placed pointing upwards on the table. The top of the mount had extrusion with heat inserts to secure a second servo on top of the first one. While the first servo was placed with its shaft pointing upwards, the second servo sat vertically with its shaft facing to the left. Once the additional servo was placed onto the base mount, another L-shaped mount attached to the shaft of the second servo. The infrared sensor was mounted to the outside of the L-shaped mount. Thus, as the second servo shaft spun, the infrared sensor would tilt up and down. As the first servo spun, the mounted infrared sensor, which included the second servo, would pan across the front face.

There are a couple aspects of our mechanical design that we could have improved. The tolerances for the 3-D printed mounts were not extremely precise. While the measurements allowed us to accomplish the task, a tighter fit between the servos and the mounts would have helped increase stability in our structure. Additionally, we should have designed the mount responsible for titling the distance sensor to rotate around the axis at the center of the pan mount. Our current design causes the distance sensor to tilt a varying distance from our base sensor and center. This made the math for the angles and the MATLAB code for plotting the distance sensor data more complicated.

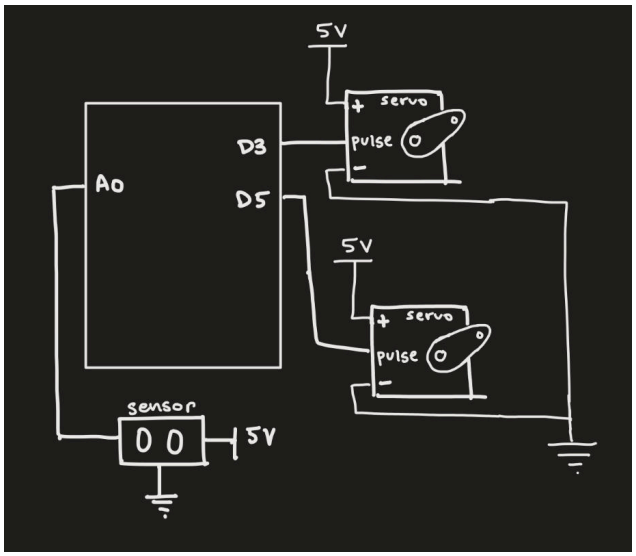

Figure 4.1: Circuit Diagram
Distance sensor and servo motors connected to Arduino UNO. Servo motors are connected through the digital pins D3 and D5, and the distance sensor is connected through the analog pin A0.

Servo motors
We connected the motors to digital pins on the arduino. Signals from the arduino will direct the motors to turn for a certain angle.

Distance sensor
We connected the distance sensor to an analog pin on the arduino. The sensor will return a voltage to the analog pin that varies depending on the distance sensed.

Visualization

The Arduino can return three values for us: the distance (determined by our calibration function and the values returned from the distance sensor), the angle of rotation of the first servo, and the angle of rotation of the second servo. We oriented the servos on different axes, so they operate on our self-assigned x- and y-axes, and thus the angles returned are the horizontal and vertical angle orientation of the distance sensor. Given these data points, we can determine the actual 3-dimensional coordinates of the points that we are returning from the sensor. First, we normalized all of the angles, since the given angles were in the range of 0 to 100 and 10 to 70. We subtracted them by 50 and 40, respectively, to center them around 0, and then took the absolute value in order to get proper coordinates. However, we retained a counter of whether the original angle was negative or positive in order to reflect the coordinates as positive or negative on the x-y plane. Then, to determine the z-distance of each point, we multiplied the distance given by the cosine of both angles. To determine x-distance, we multiplied the distance by the cosine of the horizontal angle and the sine of the vertical angle, and multiplied the distance by the sine of the horizontal angle and the cosine of the vertical angle to find the y-distance. Then, given the x, y, and z-coordinates of each reading, we plotted that.

$$z = distance * cos(vertangle) * cos(horizangle)$$
$$x = distance * cos(vertangle) * sin(horizangle)$$
$$y = distance * sin(vertangle) * cos(horizangle)$$

IV. Results:

1.) An image of your setup for the 1 servo scan of your letter and the data depicting the top view

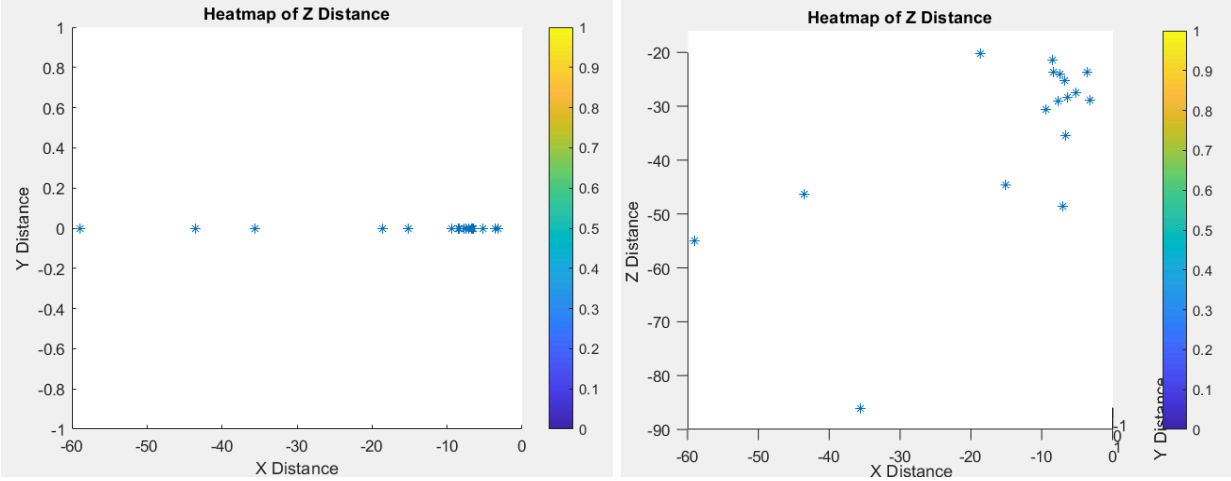

Figure 5.1: Two graphs demonstrating our one-servo scan. X distance vs. Y distance is on the left, X distance vs. Z distance is on the right.

Video of scan: https://youtube.com/shorts/Z_PRv1qmyf8

Video of scan results:
https://drive.google.com/file/d/1daY8EKDE-Z6QIWlLK3txqwsANbifUyzi/view?usp=sharing

2.) An image of your setup for the 2 servo apparatus and visualization of the 3D data resulting from your scan (a plot, an image, a video, etc)

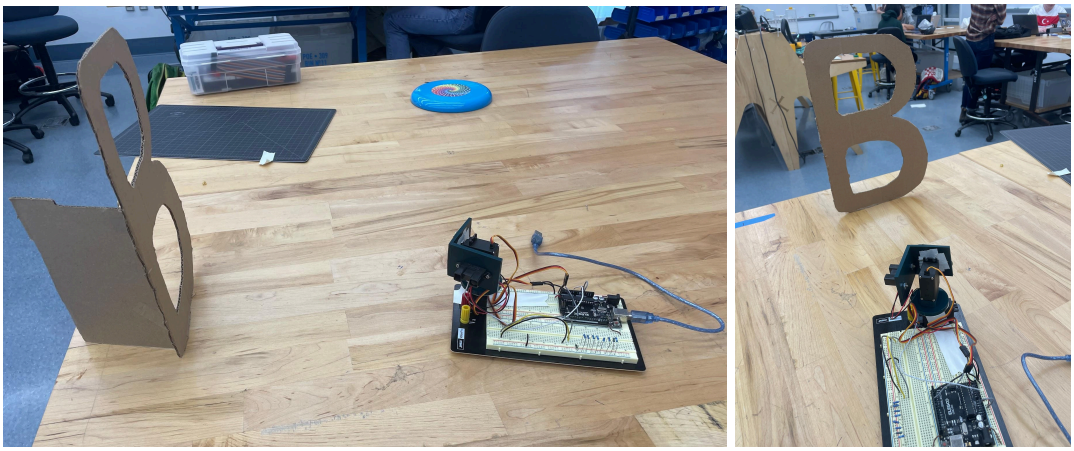

Figure 5.2: Image of Setup for both all scans

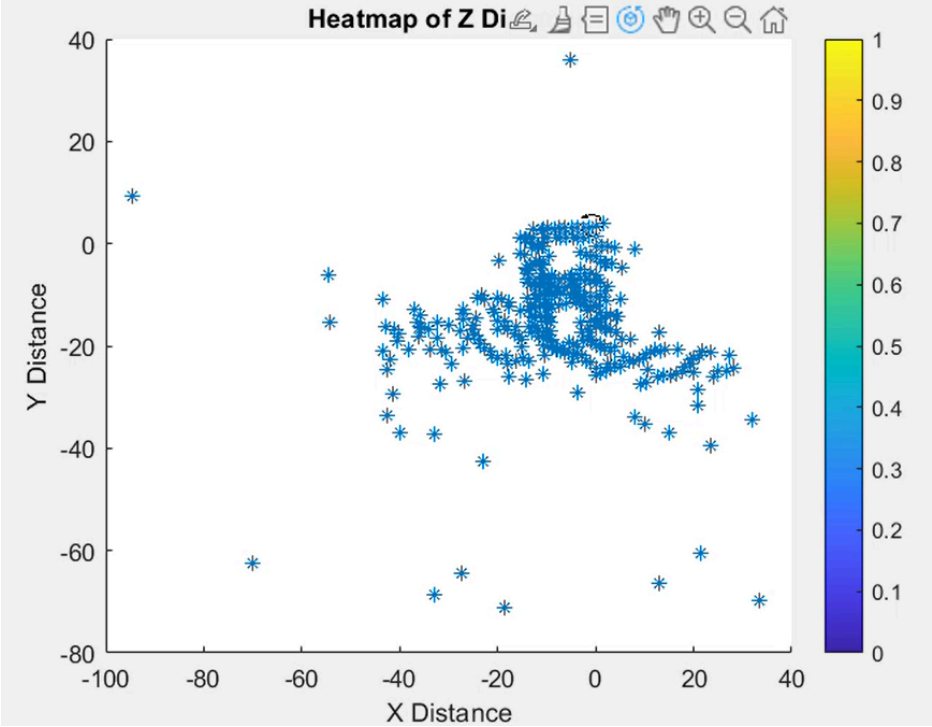

Figure 5.3: A full 3D scan using the set-up in figure 4.2

Video of scan: [me when i scan (youtube.com)]

Video of scan results:
https://drive.google.com/file/d/1KMLkyTk_6UsoFsKn7wdgPU3L7px1fapv/view?usp=sharing

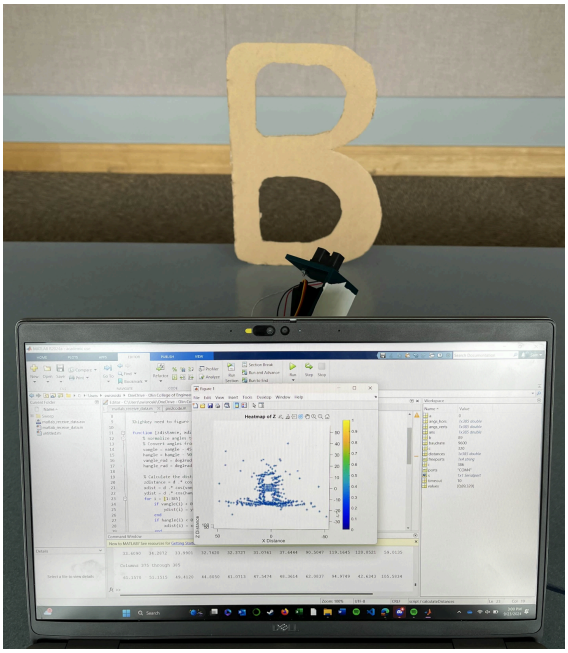

Figure 5.4: An example scan compared to the letter.

V. Reflection

1.) An explanation of and reflection on your design (including software, electrical and mechanical parts). What worked well and what could be improved?

Overall, our project seemed to work very well. The electrical design was simple and functioned exactly as desired and anticipated. The mechanical components were also designed very elegantly and simply, and we worked past some errors on tolerancing by using adhesives to attach components. For the software component, we integrated a lot of code from the previous project, but also worked to integrate the servo motors into the code and then run the code from MATLAB. There were some problems with the servo motor operations and visualizations that we ran into, but we debugged those errors.

Our team worked very effectively on this project. We were able to use class times productively, and delegate tasks so every team member was able to contribute to the project while not being overly burdened. We were all satisfied with the quality of work contributed.

If we could continue to develop this project, we would like to alter the mechanical design so that the distance sensor has even less movement and just rotation, or add to the code so it reflects the movement of the sensor. Additionally, if we could integrate multiple sensors or attach it to a rail, we could get a much higher quality scan.

IV. Appendix

Arduino Code

```
#include <Servo.h>

const uint8_t SEN = 0;
Servo myservo3;  // create Servo object to control a servo
Servo myservo5;
// twelve Servo objects can be created on most boards

bool sweep_bool = true;
bool sweep_direction = true;
int ang_hor = 0;    // variable to store the servo ang_horition
int ang_vert = 0;    // variable to store the servo ang_horition
double current_voltage; // variable to store sensor reading
double current_distance; // variable to store sensor conversion

void setup() {
  myservo3.attach(3);  // attaches the servo on pin 9 to the Servo object
```

```
  myservo5.attach(5);  // attaches the servo on pin 9 to the Servo object
  current_voltage = analogRead(SEN); // setting current voltage to the sensor reading
  Serial.begin(9600); //
}

void loop() {

while (sweep_bool){  // don't run after one sweep has been completed
  sweep(); // call the sweep function
}

}

void sweep(){
  for (ang_vert = 10; ang_vert <= 70; ang_vert +=3){ // parses vertically from 10
degrees to 70 degrees
    myservo5.write(ang_vert); // tells servo to go to vertical angle position
    delay(100);
    if (sweep_direction){  // determines if the servo is sweeping left to right or
right to left
      for (ang_hor = 0; ang_hor <= 100; ang_hor += 2) { // goes from horizontally 0
degrees to 100 degrees in steps of 1 degree
      myservo3.write(ang_hor); // tell servo to go to horizontal angle position
      delay(100); // waits 10 ms for the servo to reach the ang_horition
      read_sensor(); // reads the distance and angle and prints them by calling
read_sensor
      }
      sweep_direction = false;  // switches from left to right
    }
    else{
      for (ang_hor = 100; ang_hor >= 0; ang_hor -= 2) { // goes from 0 degrees to 100
degrees
      // in steps of 1 degree
      myservo3.write(ang_hor);  // tell servo to go to ang_horition in variable
'ang_hor'
      delay(100); // waits 10 ms for the servo to reach the ang_horition
      read_sensor(); // reads the distance and angle and prints them by calling
read_sensor
      }
      sweep_direction = true; // switches from right to left
    }

  }
  sweep_bool = false; // tells our while loop to not run the program again
}

void sweep_onedim(){ // does a single sweep in the horizontal direction
    ang_vert = 30;  // sets a constant verticle angle
    myservo5.write(ang_vert);
```

```
    for (ang_hor = 0; ang_hor <= 100; ang_hor += 2) { // goes from horizontally 0
degrees to 100 degrees
      // in steps of 1 degree
      myservo3.write(ang_hor); // tell servo to go to horizontal angle position
      delay(100); // waits 10 ms for the servo to reach the ang_horition
      read_sensor(); // reads the distance and angle and prints them by calling
read_sensor
    }
  sweep_bool = false;  // tells our while loop to not run the program again
}

void read_sensor(){
  current_voltage = analogRead(SEN);  // reads current voltage
  current_distance = int(16410*pow(current_voltage,-1.06));  // converts to distance
  Serial.print(ang_hor);    Serial.print(",");  // sends horizontal angle to MATLAB
  Serial.print(ang_vert);    Serial.print(","); // sends verticle angle to MATLAB
  Serial.println(current_distance);  // sends distance to MATLAB
}
```

Matlab Code

```
% Clear previous serial connection object
clear s
% Display available serial ports on the system
freeports = serialportlist("available");
% Choose which port to use for Arduino (can be hardcoded)
ports = "COM4"; % Can also use freeports(2) for dynamic selection
baudrate = 9600; % Set the baud rate for serial communication
% Establish a connection to the specified serial port
s = serialport(ports, baudrate);
% Initialize a timeout counter in case MATLAB cannot connect to the Arduino
timeout = 1;
% Initialize empty arrays to store horizontal angles, vertical angles, and distances
angs_hors = [];
angs_verts = [];
distances = [];
i = 1; % Initialize index for storing values
% Main loop to read data from the Arduino, process, and display it
while timeout < 10 % Retry up to 10 times if no data is received
   % Check if data is available to read from the Arduino
   while s.NumBytesAvailable > 0
       % Reset timeout counter since data was received
       timeout = 0;
       % Read data from the serial port and convert it to an array of integers
       values = eval(strcat('[', readline(s), ']'));
       % Assign individual values to variables a, b, and c
       a = values(1); % Horizontal angle
```

```
        b = values(2); % Vertical angle
        c = values(3); % Distance
        % Print the results for each data set
        fprintf('a,b,c = %d,%d,%d\n', a, b, c);
        % Only store values where distance is within a specific range
        if c < 150 && c > 20
            angs_hors(i) = a; % Store horizontal angle
            angs_verts(i) = b; % Store vertical angle
            distances(i) = c;  % Store distance
            i = i + 1; % Increment index for next set of values
        end
    end
    % Pause for 0.5 seconds before next iteration
    pause(0.5);

    % Increment timeout counter if no data is received
    timeout = timeout + 1;
end
% Call the function to calculate distances and plot results
calculateDistances(distances, angs_verts, angs_hors)
% Function to calculate 3D distances from angles and plot a heatmap
function [zdistance, xdist, ydist] = calculateDistances(d, vangle, hangle)
    % Normalize vertical and horizontal angles (calibrate if needed)
    vangle = vangle - 30;
    hangle = hangle - 50;
    % Convert angles from degrees to radians
    vangle_rad = deg2rad(abs(vangle));
    hangle_rad = deg2rad(abs(hangle));
    % Calculate the distances using trigonometric functions
    zdistance = d .* cos(vangle_rad) .* cos(hangle_rad); % Z (depth) distance
    ydist = d .* cos(vangle_rad) .* sin(hangle_rad);     % Y (lateral) distance
    xdist = d .* cos(hangle_rad) .* sin(vangle_rad);     % X (vertical) distance
    % Adjust signs of distances based on the direction of the angles
    for i = [1:length(d)]
        if vangle(i) < 0
            ydist(i) = ydist(i) * -1;
        end
        if hangle(i) < 0
            xdist(i) = xdist(i) * -1;
        end
    end
    % Plot a 3D scatter plot of the distances
    figure;
    plot3(-ydist, xdist, -zdistance, "*"); % Scatter plot with stars as markers
    colorbar; % Display a color bar to show magnitude of values
    title('Heatmap of Z Distance');
    xlabel('X Distance');
    ylabel('Y Distance');
```

```
    zlabel('Z Distance');

    % Set the view to look top-down, giving a 2D heatmap-like plot
    view(2);
end
```