Mini-Project 2: DIY 3D Scanner

Olivia Jo Bradley, Regan Mah

September 2021

1 Introduction

For Mini-Project 2, our team was tasked with creating a DIY 3D scanner using Servo motors and the Sharp, infrared distance sensor. The goal was to use the motors as a pan/tilt mechanism that could be controlled through computer programming, storing data from the Sharp sensor and developing the data into a 3D visualization. This paper will focus on our process as we attempted to create our own DIY 3D scanner.

2 Bill of Materials

- Arduino
- USB cable
- Sharp GP2Y0A02YK0F IR distance sensor
- 3-pin Molex connector with red, black, and white wire tails
- 2 Hobbyking HK15138 servo motors
- Breadboard
- 3D printed pan-tilt structure base, pan platform, and tilt platform.
- Multiple 1/4-40 bolts and nuts

3 Testing Process

To start this project, we first chose to test each component individually. We connected just the IR sensor, and collected data on various distances and voltage data to calibrate our sensor. We then disconnected the IR sensor, and connected one of the servo motors. After testing that we could control each servo one at a time, we moved to testing both at once. Most of this was done during our first class session. During our second class session we set up both servos and the IR sensor, as well as the pan-tilt assembly. Once the full assembly was created, we were able to focus on troubleshooting our data collection.

4 Mechanical Structure

We decided 3D print the parts to make the pan/tilt structure. The bottom base holds the system steady, and encloses the bottom pan servo. On the shaft of this bottom servo, an L shaped bracket is attached. This bracket has holes to allow for another servo to be mounted, the tilt servo. Another l bracket is attached to the shaft of the tilt servo. This bracket has mounting holes for the IR sensor to be mounded laying horizontally. The L shaped brackets are lightweight and easy to manufacture. This was very helpful, as we were able to have the 3d printed assembly created after the second class period. We were able to spend more time collecting and computing data with our saved time.

Figure 1 displays the full 3D printed setup.

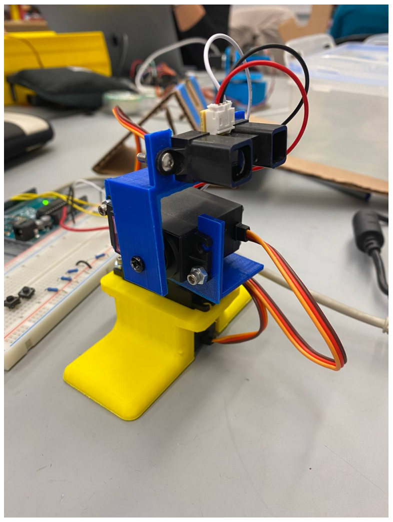


Figure 1: The mechanical structure

5 Breadboard and Wiring

5V from the Arduino Uno was used to power the servos and sharp, with the servos connected to digital pins 9 and 10 to receive commands and the sharp connected to the analog 0 pin for collecting data. All components were connected to ground which was also provided by the Arduino Uno. Figure 2 is the schematic for the Arduino, sharp, and servos.

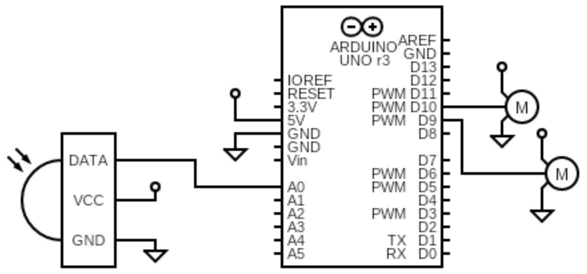


Figure 2: Breadboard Schematic

And figure 3 shows the completed breadboard setup for our DIY 3D Scanner.

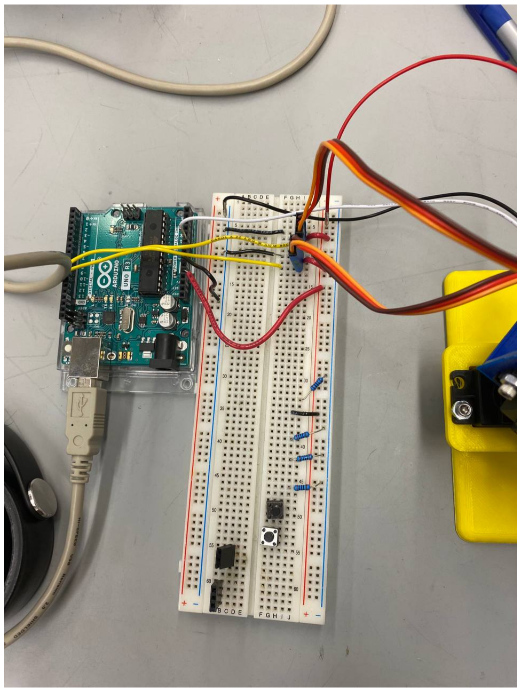


Figure 3: Picture of the Final Breadboard Setup

6 Code

Our team decided to use MATLAB to code the servos and sensor.

The code first initialized the Arduino board to the MatLab, setting up the servos, sensor, and their ports as well as initialized the rotors to the 0 degrees position. This is to make sure that everything is working and the motors are responding to commands. The code then takes inputs of how many degrees the pan and tilt move as well as the number of tests to run, so that the user can specify how many tests to run and how many iterations to complete of testing. After taking inputs, the code moves to the main central loop.

The main central loop contains the ACT function that rotates the pan and tilt motors as well as takes data from the motors and Sharp sensor. The loop is run using a while loop where the controlFlag iterates by one each time the ACT function is completed until controlFlag is equal to the amount of tests specified (this is to make sure that the program does not run forever).

We utilized for loops to first move the panServo to a specified degree, then iterated through the tilt angles on that pan degree. Then it would move onto the next pan degree and repeat the tilt process. This would run until all the pan and tilt angles were covered. In MATLAB, the Servo angle is defined from 0 to 1, with 0 representing 0 degrees and 1 representing 180 degrees. This feature meant that our code utilized a linspace array that would divide the 0-1 range evenly to send commands to the rotors. While the servos moved, the Sharp would take data at each pan/tilt angle combination, and we saved the servos positions and the sensor data in one matrix.

The letter that we used to test our setup was an L. Figure 4 depicts the L setup.

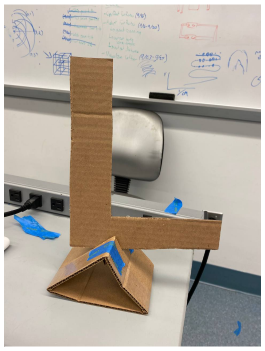


Figure 4: The cardboard L used to test the 3D scanner

First we tried our setup using only a tilt servo run. This was done while the servo was attached to the whole pan-tilt assembly, but only the tilt servo and IR sensor was connected to the board. After taking this data and confirming that the rotors worked, we moved onto the full 3D scan.

We took one run of data using 100 points in both pan and tilt, then proceeded to processing the data for visualization.

7 Data Processing and IR Calibration

First step in data processing was finding the Calibration curve. Our team ran data collection on the Sharp sensor, stopping a piece of cardboard at various known distances and then collecting the voltage read at that distance. We then used MATLAB's cftool to fit an exponential line to the the voltage vs. distance and create an equation to convert our voltage readings to distance readings. This equation allowed us to convert all the voltage points taken to distance. Figure 5 shows the plot of the calibration data and the calibration curve.

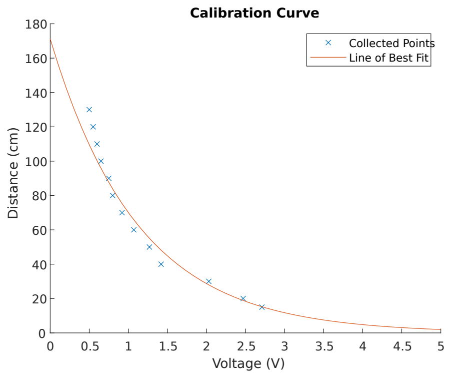


Figure 5: Calibration Data Curve

We then created an error graph to show the difference between our curve fit and measured data. We created a residual graph that took the difference between the measured distanced and the fit distance (measured-fit). This plot is depicted in figure 6.

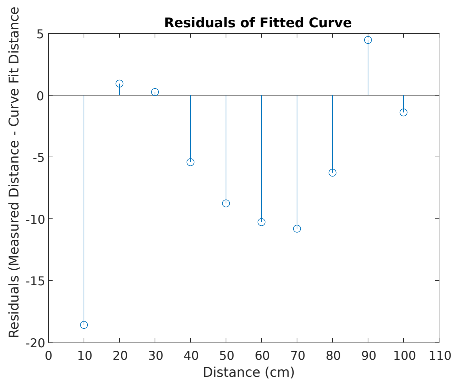


Figure 6: Caption

After converting the voltage readings to distance readings using our calibration data, we then plotted the data of our test trial and full trial.

Figure 7 shows the data from our one tilt trial. The first curve on the far left of the plot is the sensor seeing the floor while the second higher curve depicts the lower portion of the L.

Using the Heatmap function in MATLAB. To plot the heatmap, we used the readings from the panServo, tiltServo, and sharp test without changing or manipulating the values. The plot has the pan angles on the x, tilt on the y. Farther areas appeared lighter, while shorter areas appeared darker. Figure 8 shows the full heatmap of the 3D scan.

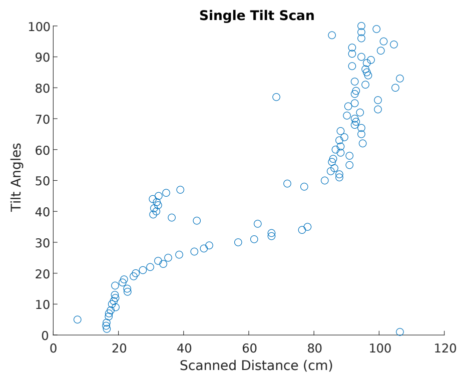


Figure 7: Visualization of a single tilt scan

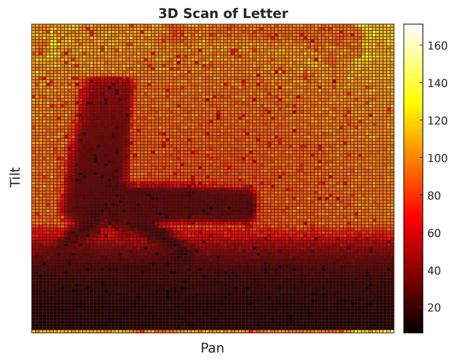


Figure 8: Heatmap Data and 3d Scan

8 Conclusion and Reflection

Overall the 3D scanner worked brilliantly. We collected data that was processed to produce an image that clearly identified the L from its surroundings. There were a couple of interesting things to note during the project.

One of the first mishaps was our pan servo rotor. The rotor often would fail midway through a run, or just not work at all. This seemed to be due to the connection of the rotor to the breadboard. We had tried taping the motor connections to another set of pins, but it still seemed to have issues. It still worked for our test, but it was not as consistent as we had hoped it would be. We also had to shorten the tilt range since our design meant that the tilt could not tilt back the full 180 degrees. Overall it didn't harm the data collection–our image was still high-quality.

A later mishap was our data collection. At one point, our pan and tilt data became swapped, which messed up our data computation. This was later resolved, but more complete code comments and a slower ramp up to confirm we were correctly collecting data would help (after believing that we collected good data with 20 pan angles and 20 tilt angles, we jumped up to 100 points).

One thing we would change if we did this again would be to change our pan-tilt mount. The way we chose to build it meant that the axis of rotation was below the sensor, and each time the sensor tilted, the distance was slightly changed. This is slightly noticeable in our 3d plot, where the sides of the L are at a slight angle, rather than a straight 90 degrees. If we did this again, we would try and spend a little more time designing our pan-tilt assembly.

9 Full MATLAB Code

All the MATLAB code is attached after this page.

The first file miniproject2code.mlx is the main testing file. This file contains the code to setup the Arduino, take user inputs for testing, and then run a main loop that commands the pan and tilt to move while taking data from the Sharp. The data is then saved in a matrix that can be processed in DataVisualization.mlx.

The second file is DataVisualization.mlx, which loads in data and processes it for our calibration curve, single tilt scan, the 3D scan, and error plot.

Mini-Project 2 Code

```
% Clean up to set up for new run
clc
clear
```

Setup control system + Servo setup

```
% Create arduino and arduino position servo ref-objects
    [robotArduino, tiltServo, panServo, blinkLED, rawRangeIn] = SETUPARDUINO('COM9');
% Turn on board LED on and off to signal program has started
    Blink(robotArduino,blinkLED,5);
    disp('Warning! Data Collection Active!')
```

```
Warning! Data Collection Active!
```

```
    disp('Warning! Pan and Tilt Servos Active!')
```

```
Warning! Pan and Tilt Servos Active!
```

```

% Configure test loop to collect n data points
    nDegrees = input(['Enter the number of stops during robot movement, followed by enter: ']);
    nTests = input(["Enter the number of loops to complete for testing, followed by enter: "]);

    clc;        % clear command window

    r = rateControl(0.25);      % create a 0.25 hz loop rate
    reset(r);
```

Run Robot Control Loop (code that runs over and over)

```
controlFlag = 1;                        % create a loop control

% scale desired angle to 0-1 for writeposition Arduino command
% 0 = 0 deg on servo, 1 = max angle (180 deg for servos)
pan = linspace(0, 0.75, nDegrees);
tilt = linspace(0, 1, nDegrees);

while(controlFlag  < nTests + 1)    % loop till ntests data captured
    % run the act function to collect data
    sensorData = ACTRCServoArduino(panServo, tiltServo, tilt, pan, robotArduino, rawRangeIn);
    Blink(robotArduino, blinkLED, 1); % Check the loop is running
    waitfor(r);                             % wait for loop cycle to complete
    controlFlag = controlFlag + 1;          % increment loop
end
```

Robot Functions (store local functions here)

Clean shut down

finally, with most embedded robot controllers, its good practice to put all actuators into a safe position and then release all control objects and shut down all communication paths. This keeps systems from jamming when you want to run again.

```
% Stop program and clean up the connection to Arduino
% when no longer needed
writePosition(panServo, 0.5);          % always end servo at 0.5
writePosition(tiltServo, 0.5);          % always end servo at 0.5
clc
disp('Arduino program has ended')
```

```
Arduino program has ended
```

```
clear robotArduino
beep
```

```
function [robotArduino, panServo, tiltServo, blinkLED, rawRangeIn] = SETUPARDUINO(COMPORT)
% SETUPARDUINO creates and configures an arduino to be a simple robot
% controller. It requires which COM port your Arduino is attached to as an
% input and returns an Arduino object called robotArduino

% create a glocal arduino object so that it can be used in functions

robotArduino = arduino(COMPORT, 'Uno', 'Libraries', 'Servo');

% configure pin 13 as digital-out LED
blinkLED = 'D13';
configurePin(robotArduino, blinkLED, 'DigitalOutput');

% create a servo object driving PWM pin 9
% Min pulse duration: 1120 microseconds
% center: 1520 microseconds
% MaxPulseDuration: 1920 microseconds
panServo = servo(robotArduino, 'D9', 'MinPulseDuration', 10*10^-6,...
    'MaxPulseDuration', 1925*10^-6);
tiltServo = servo(robotArduino, 'D10', 'MinPulseDuration', 10*10^-6,...
    'MaxPulseDuration', 1925*10^-6);

% configure A0 pin as an analog input
    rawRangeIn = 'A0';
    configurePin(robotArduino,rawRangeIn,'AnalogInput');

% RC mode expects to start up with the joystick centered
% in MATLAB function servo position is 0-1 so 0.5 (1520) is centered

writePosition(panServo, 0); % always start servo-command at 0
writePosition(tiltServo, 0); % always start servo-command at 0

pause(5.0); % wait for arduino to send stable PWM
pause(2.0);

end
```

```
function [] = Blink(a, LED, n)
% Blink toggles Arduino a LED on and off to indicate program running
% input n is number of blinks
% no output is returned

    for bIndex = 1:n
        writeDigitalPin(a, LED, 1);
        pause(0.2);
        writeDigitalPin(a, LED, 0);
        pause(0.2);
    end
end
```

Act (store Act related local functions here)

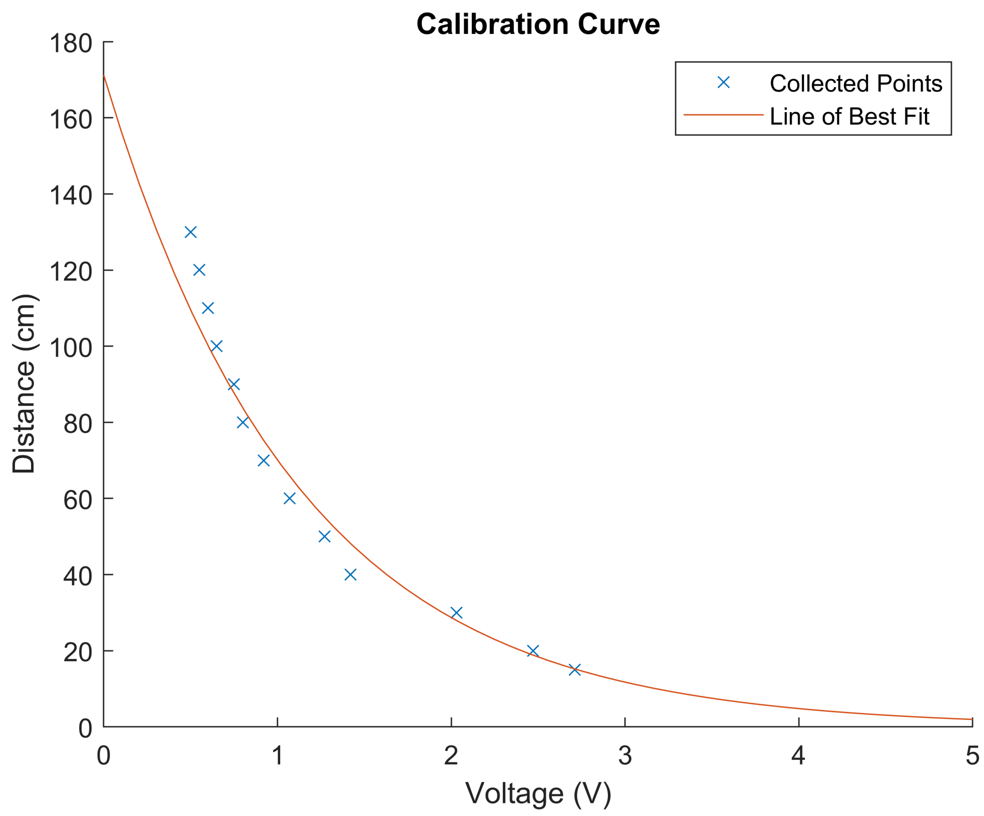
```
function sensorData = ACTRCServoArduino(panServo, tiltServo, tilt, pan, robotArduino, rawRangeI
% Code to move the pan and tilt servos as well as take in the data
% from the sharp and the positions of the pan and tilt servos
% Note that in setup we switched around the pan/tilt names

sensorData = [0;0;0;]; % Set up the blank matrix for data collection
savedPanAngles = []; % Set up pan angle collection
savedTiltAngles = []; % Set up tilt angle collection

for i = pan
    % write the pan angle based on the pan linspace
    panAngle = i;
    writePosition(tiltServo, panAngle); % Move pan servo
    pause(1.0); % Wait some time
    for j = tilt
        % write the tilt angle based on the tilt linspace
        tiltAngle = j;
        writePosition(panServo, tiltAngle); % Move tilt servo
        pause(1.0); % Wait some time
        % Save the data within the sensorData
        sensorData =[sensorData(1,:), readVoltage(robotArduino, rawRangeIn)
                     sensorData(2,:), tiltAngle
                     sensorData(3,:), panAngle];
    end
end

end
```

Data Processing and Visualization

```
clear all
load('fullScan.mat');
```

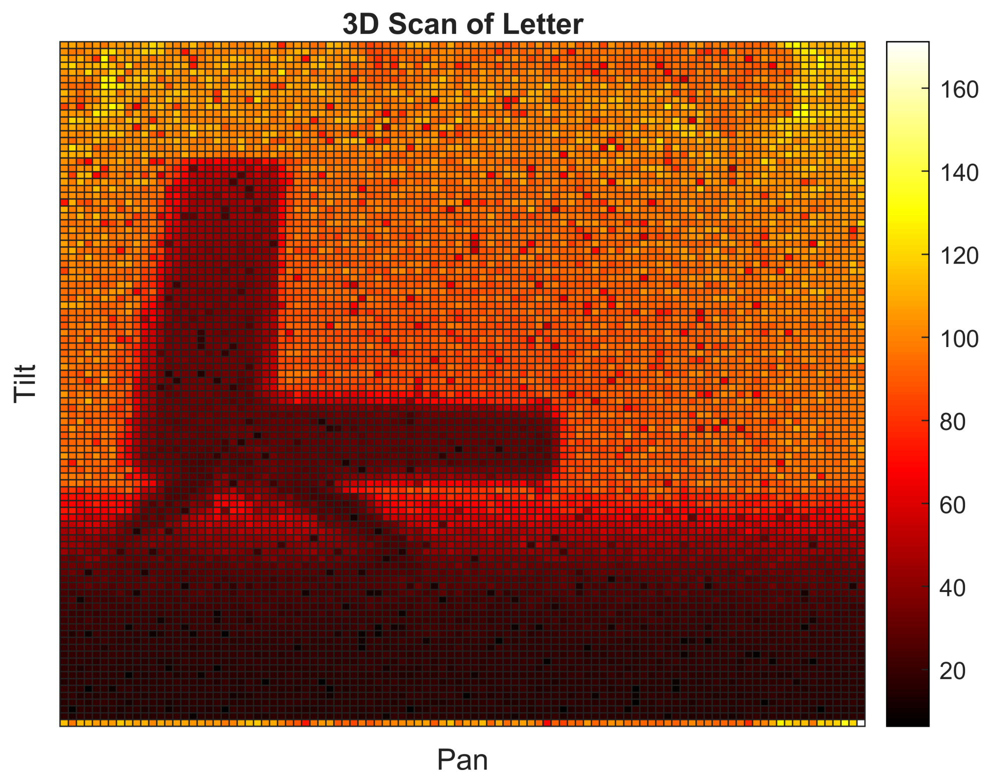
Calibrate Distance to Voltage

```
%collected IR calibration data
calibrateData = [15, 2.71;
    20, 2.47;
    30, 2.03;
    40, 1.42;
    50, 1.27;
    60, 1.07;
    70, 0.92;
    80, 0.8;
    90, 0.75;
    100, 0.65
    110, 0.6
    120, 0.55
    130, 0.5];
calibrateDataDistance = calibrateData(:, 1);
calibrateDataVoltage = calibrateData(:, 2);
%cftool         %use this tool to find the line of best fit
a = 171.1799;
b = -0.8929;
x = linspace(0, 5, 50);
calCurve = a*exp(b*x);

% Plot Calibration Information
figure(1)
hold on
plot(calibrateDataVoltage, calibrateDataDistance, "x")
plot(x, a*exp(b*x))
hold off
title("Calibration Curve")
ylabel("Distance (cm)")
xlabel("Voltage (V)")
legend("Collected Points", "Line of Best Fit")
```



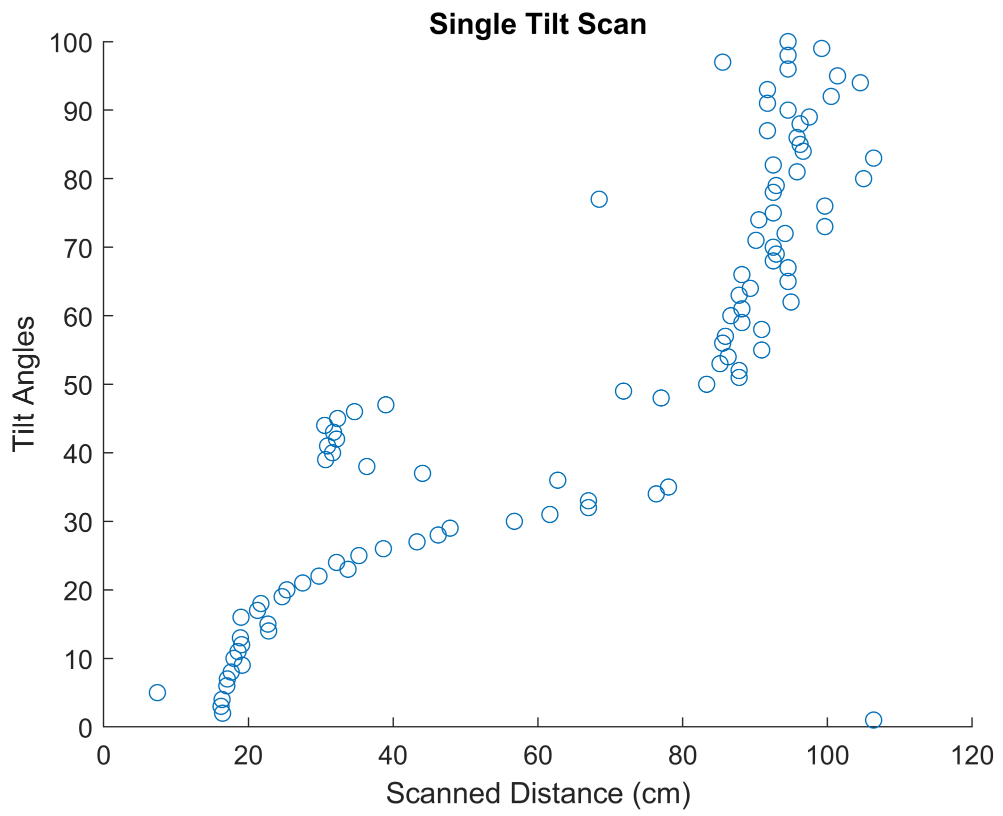
Creating the heatmap of the scan

```
Az = sensorData(3, :); % Azimuth - pan angle
El = sensorData(2, :); % Elevation - tilt angle
Rd = sensorData(1, :); % voltage data

pan = fliplr(Az); % Flip because our data collects upside down
tilt = fliplr(El); % Flip because our data collects upside down
dist = a*exp(b*Rd); % Convert Voltage to Distance using line of best fit above


refitArray = [dist; tilt; pan]'; % transpose array to make it vertical
dataTable = array2table(refitArray, 'VariableNames',{'Distance','Tilt','Pan'});

% Create heatmap
SCAN = heatmap(dataTable,"Pan","Tilt",'ColorVariable', "Distance", "Title","3D Scan of Letter",
% Get rid of tick labels
SCAN.XDisplayLabels = nan(size(SCAN.XDisplayData));
SCAN.YDisplayLabels = nan(size(SCAN.YDisplayData));
```



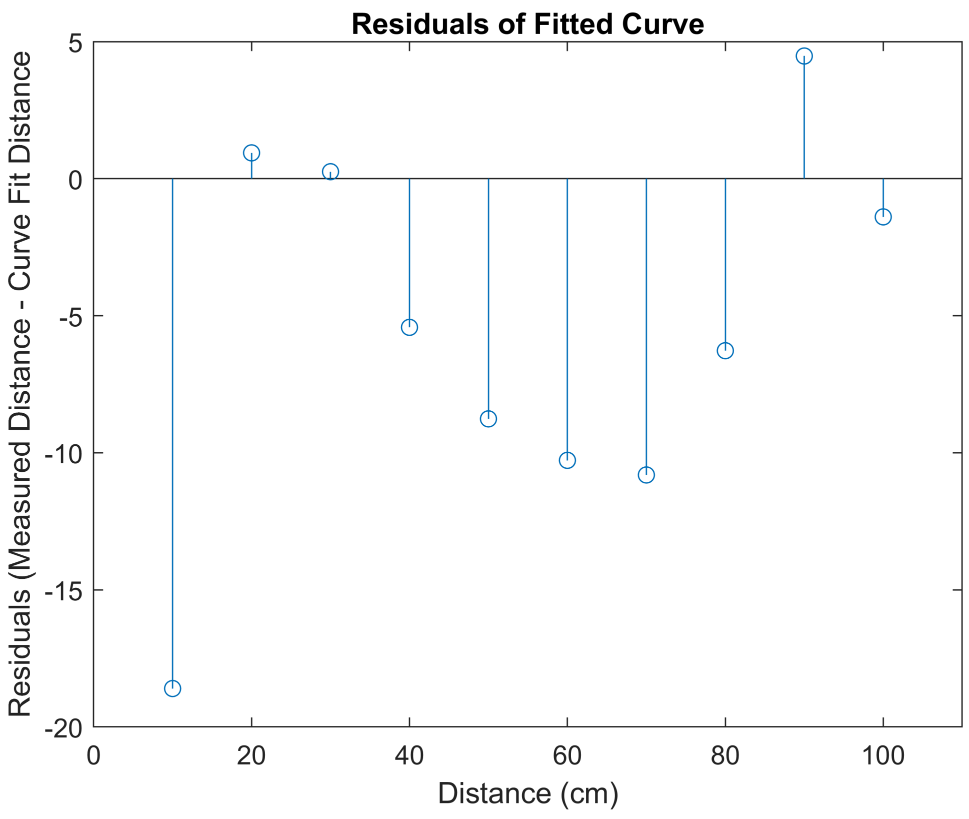
Display single scan

```
load("SingleAxisScan.mat")

scatter(singleScan(2, :), 1:length(singleScan))
xlabel("Scanned Distance (cm)")
ylabel("Tilt Angles")
title("Single Tilt Scan")
```



Display Error Plot

```
load("ErrorPlotData.mat")

a = 171.1799;
b = -0.8929;
positionData(:,3) = a*exp(b*positionData(:,2));


stem(positionData(:,1), positionData(:,1)-positionData(:,3))
xlim([0 110])
xlabel("Distance (cm)")
ylabel("Residuals (Measured Distance - Curve Fit Distance")
title("Residuals of Fitted Curve")
```


Residuals of Fitted Curve
Residuals (Measured Distance - Curve Fit Distance
Distance (cm)
5
0
-5
-10
-15
-20
0
20
40
60
80
100