Microprocessor System Project 2DX3 Project Report

Instructor: Dr. Haddara, Dr. Doyle, Dr. Athar

Date: April 10, 2024

Luca Namestnik - 400 449 981 - namestnl

Contents

1. Device Overview

1.1 Features

- Texas Instruments MSP432E401Y:
  - Processor – ARM Cortex M4 processor core
  - Bus Speed – 40 MHz
  - Memory – 1024KB flash memory
  - SRAM – 256KB
  - EEPROM – 6KB
  - Operating Voltage – 2.5V to 5V
  - ADC – Two 12-bit SAR based
  - Programming Language – C or Assembly through Keil UVision
  - Communications - Serial - I2C and UART
  - Baud Rate – 115200bps
  - Cost ~ $75

- Time of Flight Sensor (VL53L1X)
  - Max Distance – 4000mm
  - Ranging Frequency – 50Hz
  - Operating Voltage – 2.6V to 3.5V
  - Communication – I2C (400KHz max frequency)
  - Cost ~ $7

- Unipolar Stepper Motor
  - 28BYJ-48
  - Rotation - 512 steps
  - Operating Voltage – 5V
  - Direction – Clockwise or counterclockwise
  - Cost ~ $12

- Other Libraries
  - Python – calculations and data processing
  - Pyserial - UART communication
  - Open3D - visualization

- Status Updates
  - Onboard LEDs PN1 and PF4 for measurement status and ToF status

- Execution
  - ‘Enter’ key starts each plane’s measurements

- Resistors
  - 4 – 10KΩ resistors
  - 2 sets of 2 in parallel for ToF SCL and SDA
  - Pull-up resistors

- System Size
  - 400mm x 260mm x 180mm

1.2 General Description

The embedded spatial measurement system is a compact and cost-effective solution designed for acquiring spatial information in indoor environments. Utilizing a time-of-flight sensor and rotary mechanism, this system offers 360-degree measurement capabilities within a single vertical geometric plane (y-z), while integrating fixed distance samples along the orthogonal axis (x-axis). The acquired spatial data is stored in onboard memory for later retrieval and is then communicated to a separate library and made into a 3-D model.

This system includes the Texas Instruments MSP432E401Y microcontroller that connects to and rotates a stepper motor with a Time of Flight Sensor (VL53L1x) mounted to the motor. The motor stops every 11.25° allowing the ToF to use LIDAR to take a measurement on the y-z plane, meaning the sensor will therefore take 32 measurements for each plane. Each measurement is sent to Python using I2C communication and is stored in an array waiting for all desired planes to be measured. After each planar measurement, the system will wait for you to press the 'enter' key on your computer keyboard to communicate through UART that you have moved the device along the x-axis and are ready to take the next measurement. Once you have measured all the planes in the indoor environment the measurement data stored in Python will calculate the xyz coordinate corresponding to each value and stored on a separate file.

The Python script then takes the data that is stored in a separate file and creates point clouds on Open3D and then connects the corresponding dots to show a high-quality 3-D model of the surrounding environment.

1.3 Block Diagram (Data Flow Graph)

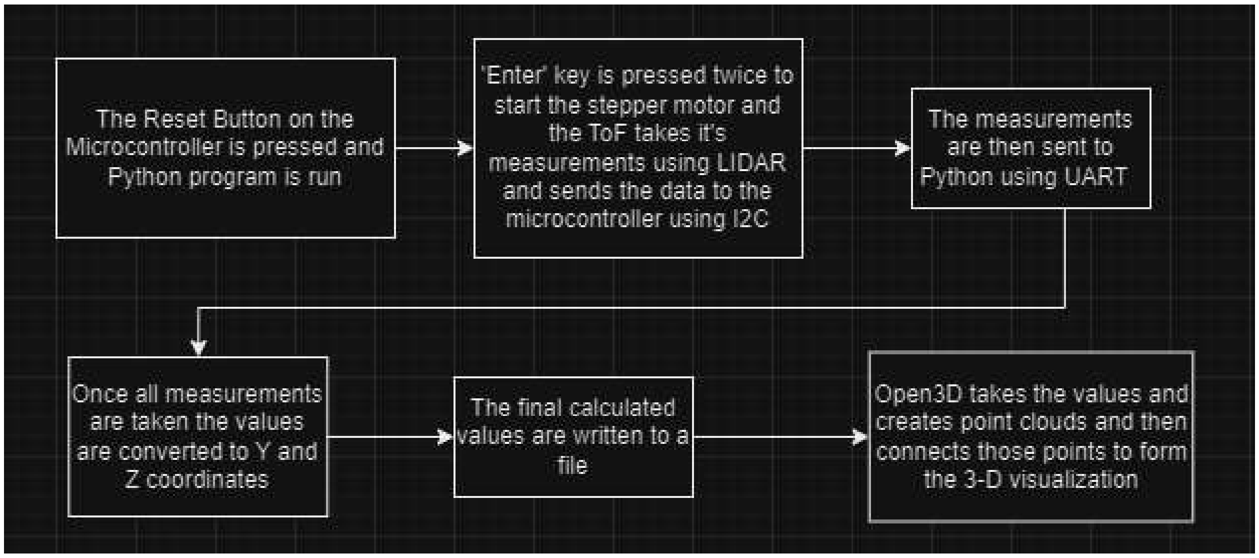

The Reset Button on the Microcontroller is pressed and Python program is run
'Enter' key is pressed twice to start the stepper motor and the ToF takes it's measurements using LIDAR and sends the data to the microcontroller using I2C
The measurements are then sent to Python using UART
Once all measurements are taken the values are converted to Y and Z coordinates
The final calculated values are written to a file
Open3D takes the values and creates point clouds and then connects those points to form the 3-D visualization

Figure 1: Data Flow Graph

2 Device Characteristics Table

Table 1: Device Characteristics

Characteristic	Description
Bus Speed	40 MHz
Baud Rate	115200bps
Stepper Motor Connections	IN1 to IN4 connected with PM0 to PM3 respectively. + and – connected to 5V and GND respectively.
ToF Connections	PB2 to SCL PB3 to SDA Vin and GND connected to 3.3V and GND respectively
Measurement Status	Onboard LED D1 (PN1)
Python	Version 3.6-3.10 pyserial Open3D
Max. Range	4000mm
ToF Distance Modes: max distance in dark conditions: max distance in light conditions	Short: 136 cm: 135 cm Medium: 290 cm: 76 cm Long: 360 cm: 73cm
Stepping Angle (degrees)	11.25
Measurements per plane	32

3 Detailed Description

3.1 Distance Measurement

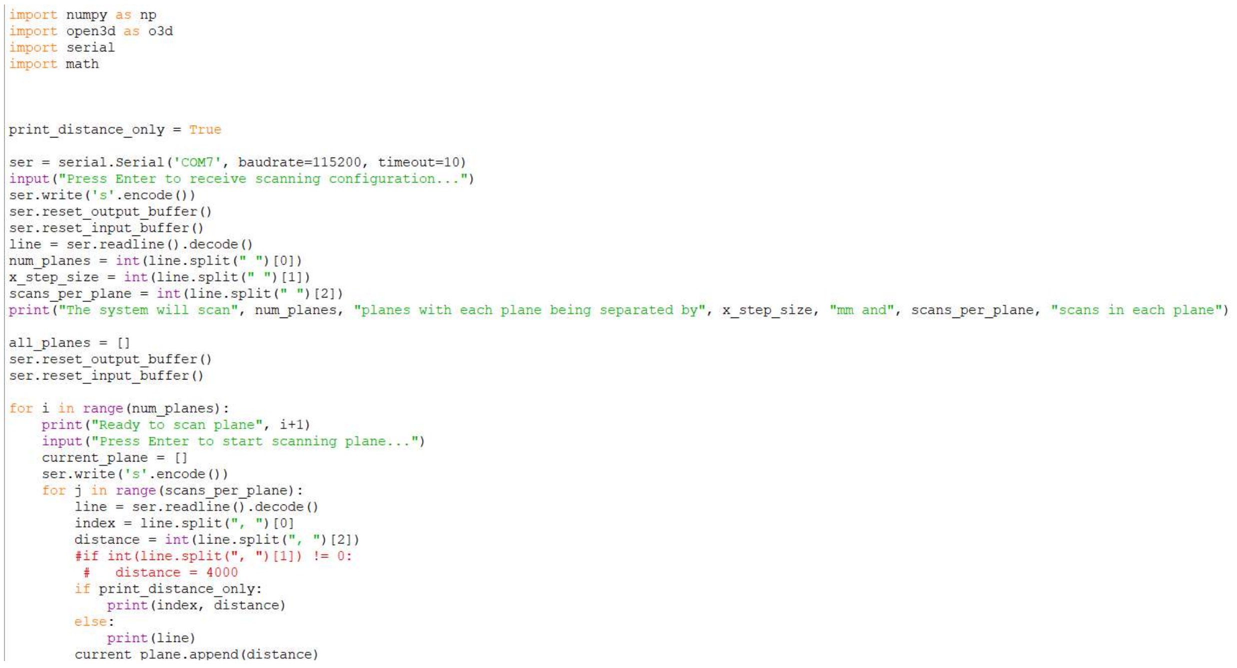
The method the Time of Flight Sensor uses to measure distance from an object is LIDAR. The ToF has two small gold squares, the first is an emitter that sends out a 940 nm laser, and the laser once hitting the object is received back by the sensor (second square). Once received the sensor uses timing data it has collected to calculate the distance. The equation used to calculate the distance in mm is: $D(mm) = \frac{1}{2}(time \ \times speed\ of\ light)$.

This information is sent from the VL53L1X (ToF) to the microcontroller through the I2C communication protocol with the leader/follower setup. The clock signal is given through the SCL connected to PB2 and the data is sent through the SDA connected to PB3. The GetDistance() function is used in the Keil code to get the value distance value in millimeters. This data is transmitted serially to the laptop and Python through its corresponding COM port in our case it is (COM7) using UART.

The measurement mode used is long mode, meaning that in perfectly dark conditions our ToF sensor can measure up to 4000mm. If you are trying to map out a room with very ambient light or if you are outside with lots of sunlight the long mode may only measure up to 730 mm. Be mindful of this when measuring inside with lighting settings you can adjust, always choose the darker setting for a better result with this system.

The stepper motor rotates at an angle of 11.25° meaning we have 32 total measurements per plane based on the equation $\frac{360}{11.25} = 32$. After each measurement the laptop asks you to press the 'enter' key to prompt your next measurement this is UART communication between the laptop and microcontroller. During each measurement, the onboard LED D1 will flash showing that a measurement has been taken.

Once the measurements are done this data now needs to be translated to its proper XYZ coordinates. In this system the Z coordinate is horizontal, and the Y coordinate is vertical. The X coordinate is incremented in each plane by a value of 1000mm meaning 1 meter between each measurement. Furthermore, in the Python script, we calculate the Y and Z values using trigonometry and then write all the XYZ values to a separate file.



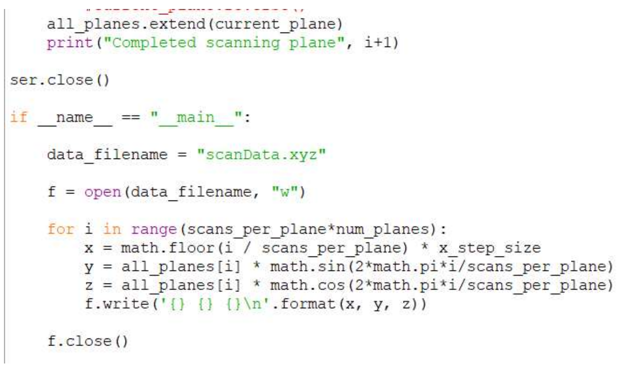
```
import numpy as np
import open3d as o3d
import serial
import math


print_distance_only = True

ser = serial.Serial('COM7', baudrate=115200, timeout=10)
input("Press Enter to receive scanning configuration...")
ser.write('s'.encode())
ser.reset_output_buffer()
ser.reset_input_buffer()
line = ser.readline().decode()
num_planes = int(line.split(" ")[0])
x_step_size = int(line.split(" ")[1])
scans_per_plane = int(line.split(" ")[2])
print("The system will scan", num_planes, "planes with each plane being separated by", x_step_size, "mm and", scans_per_plane, "scans in each plane")

all_planes = []
ser.reset_output_buffer()
ser.reset_input_buffer()

for i in range(num_planes):
    print("Ready to scan plane", i+1)
    input("Press Enter to start scanning plane...")
    current_plane = []
    ser.write('s'.encode())
    for j in range(scans_per_plane):
        line = ser.readline().decode()
        index = line.split(", ")[0]
        distance = int(line.split(", ")[2])
        #if int(line.split(", ")[1]) != 0:
         #   distance = 4000
        if print_distance_only:
            print(index, distance)
        else:
            print(line)
        current_plane.append(distance)
```



```
    all_planes.extend(current_plane)
    print("Completed scanning plane", i+1)

ser.close()

if __name__ == "__main__":

    data_filename = "scanData.xyz"

    f = open(data_filename, "w")

    for i in range(scans_per_plane*num_planes):
        x = math.floor(i / scans_per_plane) * x_step_size
        y = all_planes[i] * math.sin(2*math.pi*i/scans_per_plane)
        z = all_planes[i] * math.cos(2*math.pi*i/scans_per_plane)
        f.write('{} {} {}\n'.format(x, y, z))

    f.close()
```

Figure 2: Python Code Showing Calculations

3.2 Visualization

The visualization for the system is done through Open3D software that is executed through the Python script. Once the XYZ values are calculated and in a separate file, the Python Script takes each measurement and creates point clouds of the room or surroundings. The points are made with 32 per plane since this is the amount taken per X displacement in our case (1000mm). After the point cloud is made the points are connected by lines to the adjacent points, this creates a high-definition 3-D model of the room. You can see the calculations for XYZ in Figure 3 and in Figure 4 an example of what the final 3-D model would look like.

```python
pcd = o3d.io.read_point_cloud(data_filename, format="xyz")

yz_slice_vertex = []
for i in range(0,scans_per_plane*num_planes):
    yz_slice_vertex.append([i])

lines = []
for i in range(0, scans_per_plane*num_planes, scans_per_plane):
    for j in range(scans_per_plane):
        if (j == scans_per_plane - 1):
            lines.append([yz_slice_vertex[i+j], yz_slice_vertex[i]])
        else:
            lines.append([yz_slice_vertex[i+j], yz_slice_vertex[i+j+1]])

for i in range(scans_per_plane):
    for j in range(0, scans_per_plane*(num_planes-1), scans_per_plane):
        lines.append([yz_slice_vertex[i+j], yz_slice_vertex[i+j+scans_per_plane]])

line_set = o3d.geometry.LineSet(points=o3d.utility.Vector3dVector(np.asarray(pcd.points)),lines=o3d.utility.Vector2iVector(lines))
o3d.visualization.draw_geometries([line_set])
```

Figure 3: Python Code for creating visualization

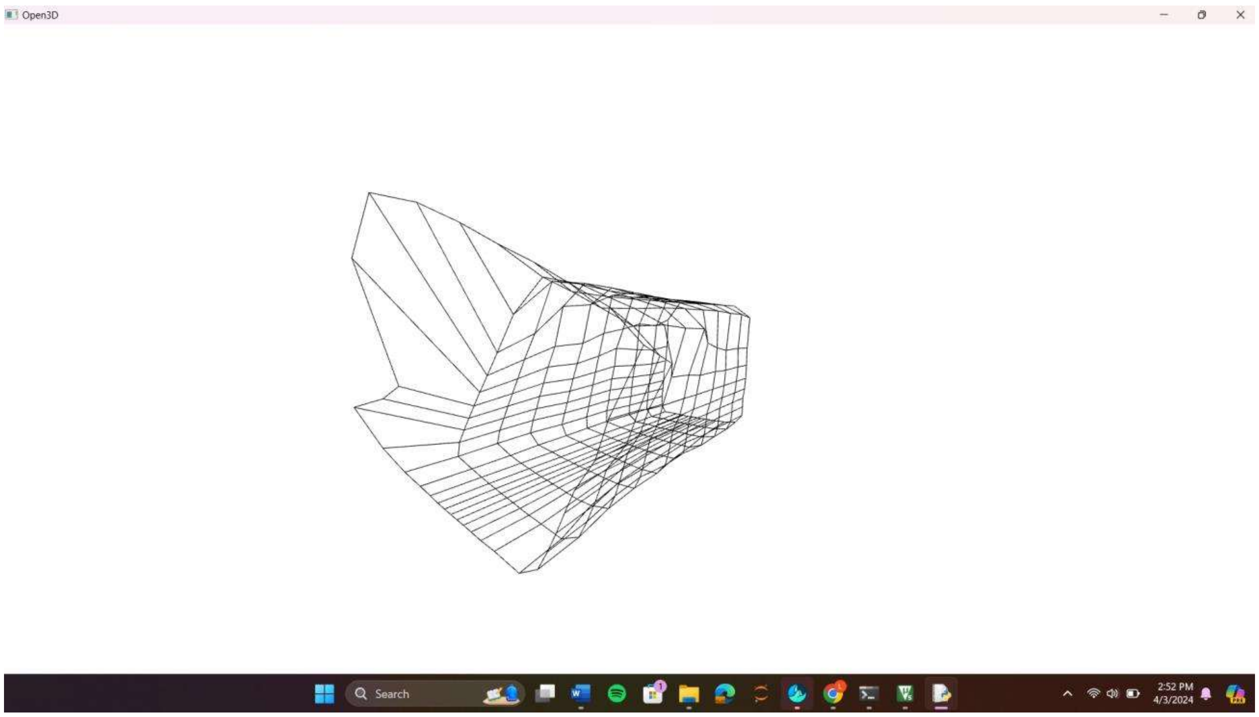


Figure 4: Example Final 3-D model

4 Application Example (Application Note, Instructions, Expected Output)

Application Note:

There are many methods of use for this system, the primary use would be for the mapping of a room as seen in the 'Expected Output' section below.

Other uses for this system would be as follows:

1) Mapping the inside of a pool to figure out how much material is needed.
2) This can be used on loading docks to ensure the truck or vehicle can fit.
3) A car wash similar to loading docks can check if the vehicle is going to fit into the car wash.
4) Create a model of the inside of a house for potential buyers to know the exact dimensions of each room.
5) Create a 3-D model of the connecting point for a prosthetic to ensure precise attachment.

Instructions

Hardware setup:

1) Connect IN1 to IN4 on your stepper motor to PM0 to PM3 of your microcontroller respectively.
2) On your breadboard connect your 10KΩ resistors in two parallel sets tied to the input of 3.3V.
3) The other side of the resistors connect PB2 and PB3 from your microcontroller.
4) Connect those nodes to the ToF sensor's SCL and SDA respectively.
5) Connect the Vin of ToF and GND of ToF to 3.3V and GND of microcontroller.
6) Connect + and – of stepper motor to 5V and GND respectively.

This setup can be seen in Figure 5 below:

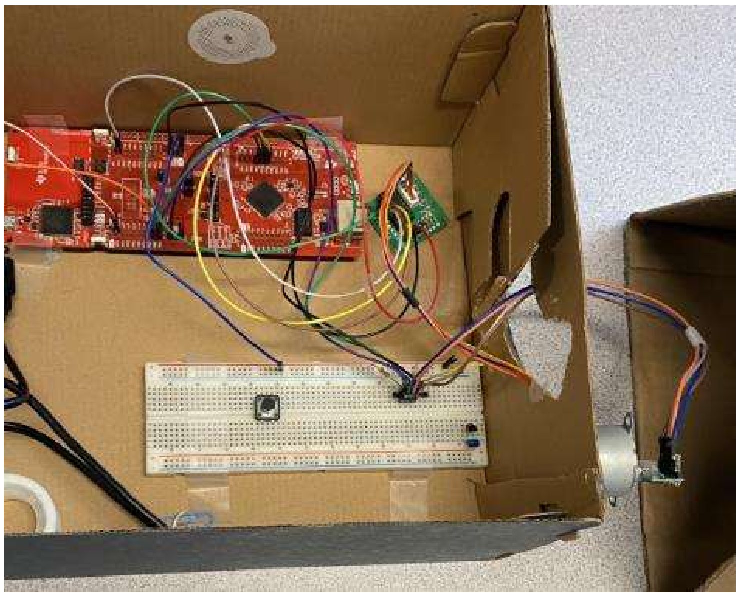


Figure 5: Hardware Setup (Ignore pushbutton and capacitors)

System Execution:

1) Download/Install Python onto your laptop if you don't already have it.

2) Open your command prompt and install pyserial and Open3D.

3) Connect your microcontroller to your PC

4) Ensure you are connected to the correct port, check your port in the device manager, and then change the Python code to the corresponding COM port.

5) For steps 5 and 6 refer to Figure 6.

5) Adjust the number of scans to your desired amount and adjust the number of planes to the desired amount at the top of the Keil program beside the 'SCANS' and 'PLANES' respectively.

6) If you wish to change the displacement value to smaller or larger based on the size of the room you can do so at the top of the Keil program as well beside the 'X_DISP' variable.

7) Reset your microcontroller and run your Python code

8) Press 'enter' twice to start the scan of your first plane.

9) Ensure you are in a room that is not too bright so the sensor can take accurate measurements.

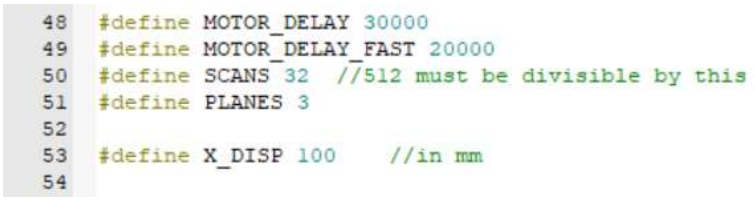
10) The stepper motor will start to rotate and the ToF will take measurements each time LED D1 flashes.

11) Once the scan is completed move the system to your next scan position and press 'enter' to scan the next plane.

12) Repeat this process for each of the planes you wish to scan.

13) Once you have scanned every plane you can open the final scan of the room as Open3D automatically displays it on the screen.

14) Analyze your model if you are not satisfied with the 3-D model you can repeat this whole process.



```
#define MOTOR_DELAY 30000
#define MOTOR_DELAY_FAST 20000
#define SCANS 32  //512 must be divisible by this
#define PLANES 3

#define X_DISP 100    //in mm

```

Figure 6: Keil Code variable that can be changed

Expected Output

An example of where this system can be used in real life is mapping a hallway. Figures X and Y below show the comparison between the physical hallway B in McMaster University's Information Technology Building and the 3-D model created by this embedded system. The scan used 10 planes with a displacement of 1000mm between each scan; the scan was taken at a height of about 750mm above ground level.

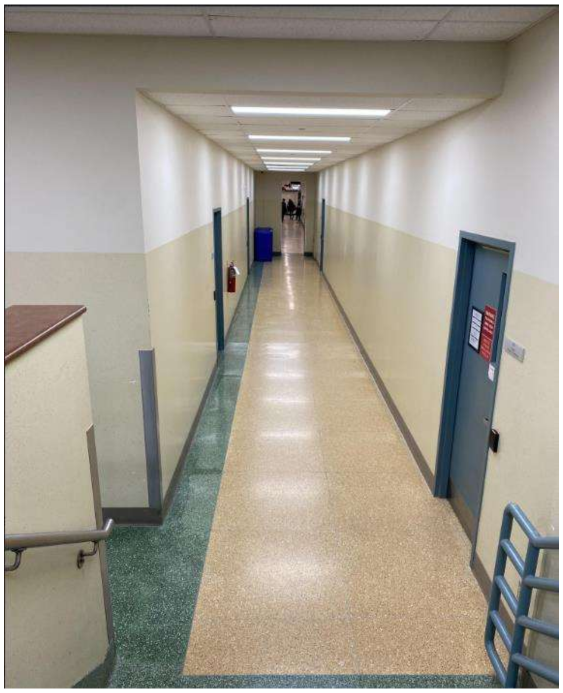


Figure 7: Hallway B

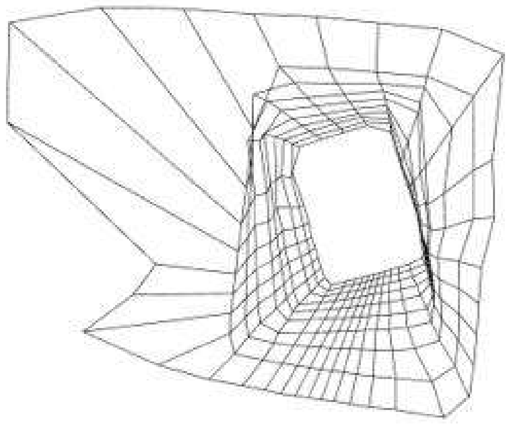


Figure 8: 3-D model of Hallway B

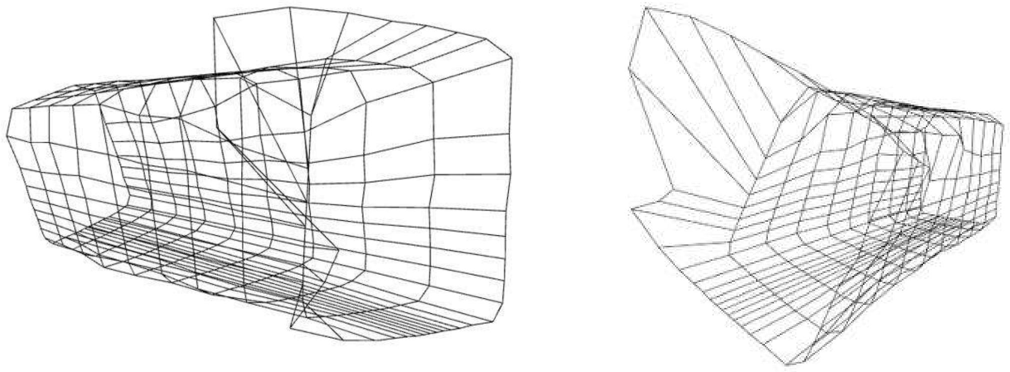


Figure 8: 3-D model side views of hallway B

As you can see in the above Figures the hallway is wider at the front of the scan and then becomes a rectangular hallway. Initially it is tilted because the scan was taken from an unsteady platform but as the displacement increased into the hallway the platform used was steadier therefore given a more level scan. To achieve a scan that is even throughout ensure you use the same platform to take your scan and only change your X-displacement. Overall, the output 3-D model matches the expected output of hallway B.

5 Limitations

1. The first limitation is the maximum distance that the Time-of-Flight sensor can measure. With the VL53L1X ToF, the maximum distance is 4000mm meaning if the sensor attempts to measure further than this threshold there will be errors in the measurement.
2. The limitations of the microcontroller's floating-point capability and trigonometric functions in this design are twofold. Firstly, the microcontroller's Floating-Point Unit (FPU) supports 32-bit single-precision operations, potentially leading to a precision loss for higher precision calculations. Secondly, Python's math library operates on 64-bit double-precision floating-point values, inherently introducing errors. Converting integer distance values from the sensor to floating-point for trigonometric calculations may compound inaccuracies, affecting the overall measurement reliability.
3. The maximum quantization error for each Time-of-Flight (ToF) module is calculated as 0.061 mm per bit. This is determined by dividing the total distance of 4000 mm (in long-range mode) by 2^16 ($\frac{4000}{2^{16}} = 0.0061$), which represents the resolution of the sensor. Therefore, for each bit of data captured by the ToF module, the potential measurement error is approximately 0.061 mm.
4. The maximum standard serial communication rate achievable with the PC is determined by the USB transfer speed of 128 Mbps. However, the microcontroller cannot reliably transmit data at this maximum speed without risking errors on the receiving end. Thus, the implemented serial communication speed is set to the standard baud rate of 115200 bps. This choice was verified by adjusting the baud rate using Realterm to assess the maximum allowable value for transmitting and receiving valid data.
5. The limiting factor of the speed of the measurement system is due to the time it takes to measure each distance. This is due to the ToF having a maximum frequency of 50Hz. But with the timing budget making the measurement take longer this increases the accuracy of the measurement. Furthermore, the stepper motor adds to this time due to the delay between each step.

6 Circuit Schematic

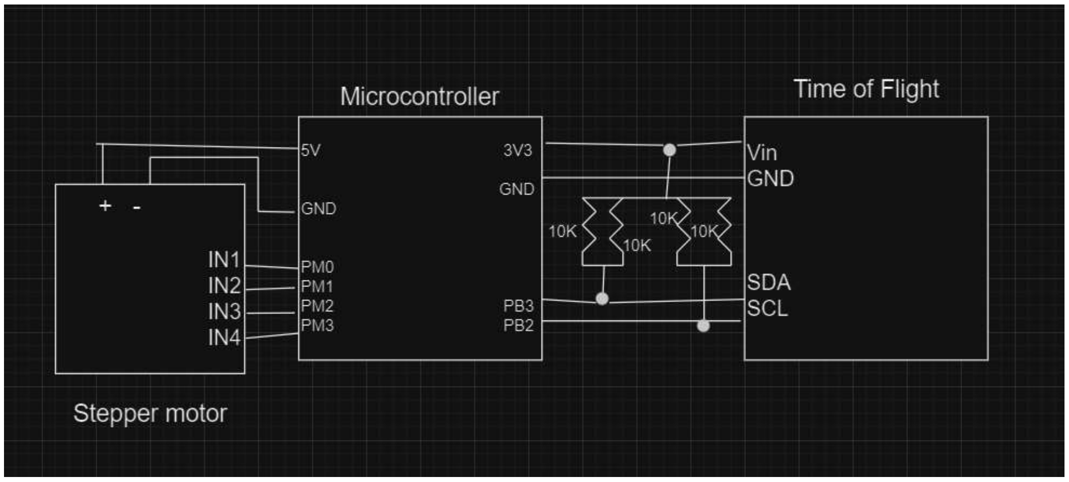


Figure 9: Circuit Schematic

7 Programming Logic Flowchart

Microcontroller (Keil Program)

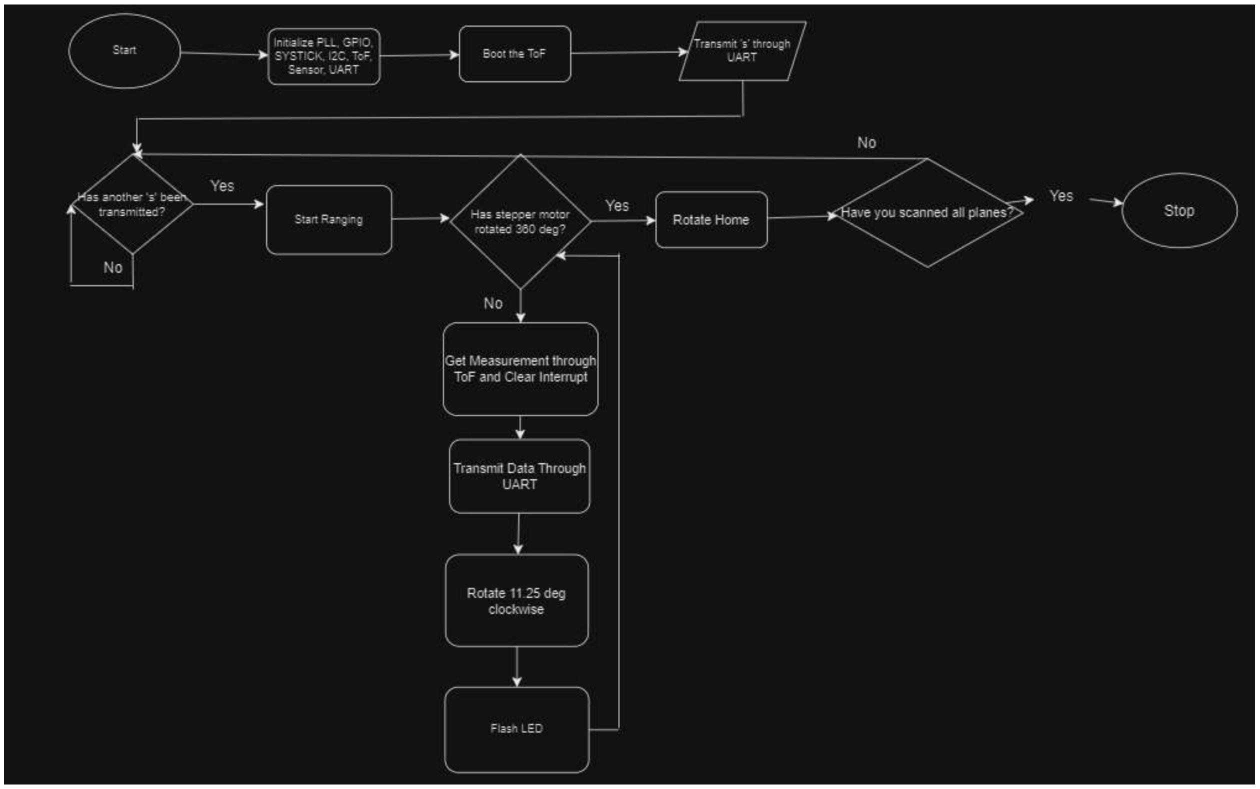
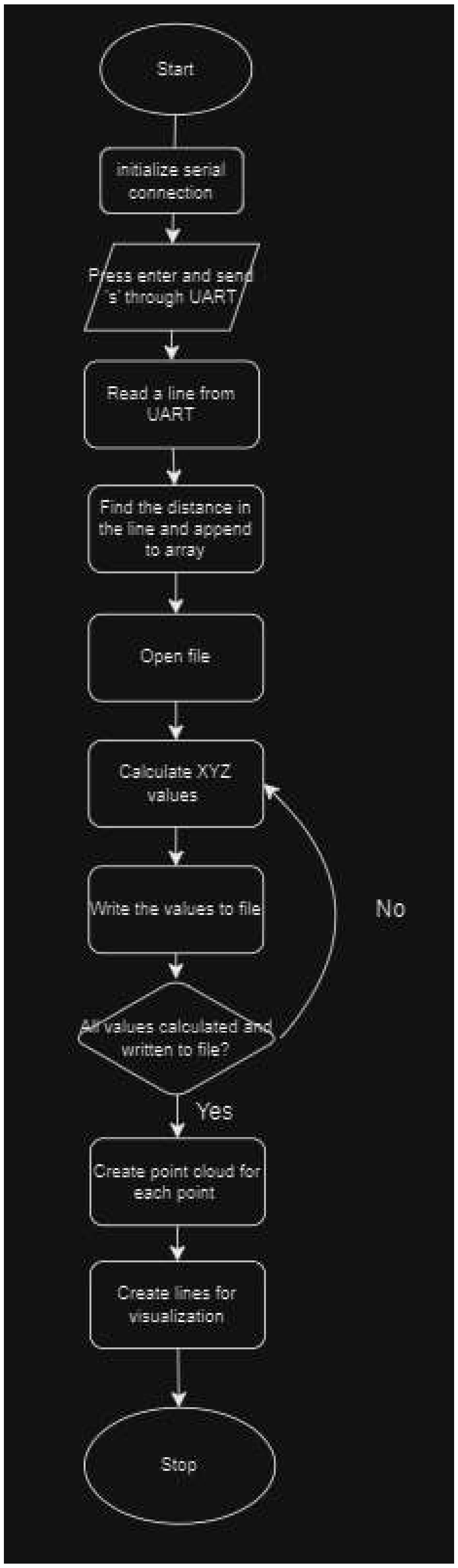

Start
Initialize PLL, GPIO, SYSTICK, I2C, ToF, Sensor, UART
Boot the ToF
Transmit 's' through UART
Has another 's' been transmitted?
Yes
No
Start Ranging
Has stepper motor rotated 360 deg?
Yes
No
Rotate Home
Have you scanned all planes?
Yes
No
Stop
Get Measurement through ToF and Clear Interrupt
Transmit Data Through UART
Rotate 11.25 deg clockwise
Flash LED

Figure 10: Micro Flowchart

Python Program:



Start
initialize serial connection
Press enter and send 's' through UART
Read a line from UART
Find the distance in the line and append to array
Open file
Calculate XYZ values
Write the values to file
No
All values calculated and written to file?
Yes
Create point cloud for each point
Create lines for visualization
Stop

Figure 11: Python Program Flowchart