Final Report: Spatial Mapping Using Time-of-Flight

Microprocessor Systems 2DX3

Daniel Kim
Instructor: Dr Yaser M. Haddara, Dr. Thomas Doyle, Dr. Shahrukh Athar
Due Date: April 9th 2025

Device Overview

Features

- 360° LIDAR Scanning: The device captures distance measurements in a full 360° sweep within a vertical plane. It uses a VL53L1X time-of-flight (ToF) sensor mounted on a stepper motor and is ideal for creating indoor spatial maps
- High-performance microcontroller:
    - Model: MSP432E401Y featuring an ARM Cortex-M4F processor
    - Bus Speed: Configurable up to 120 MHz; commonly set to 22 MHz for this design
    - Operating Voltage: Approximately 5 V input (via micro USB), regulated to 3.3 V where required
    - Memory: 1 MB of flash and 256 KB of SRAM
    - Approximate Cost: $70-$100 CAD for the development board and components
- Stepper Motor Mechanism: A 28BYJ-48 stepper motor, driven by a ULN2003 driver, provides accurate incremental rotation for distance measurements. Typical operating voltage is 5 V, with each full rotation covering 360° in precise steps
- Time-of-Flight Sensor (VL53L1X): Operates at 2.6–5.5 V (3.3V for this system), supporting up to 4 m distance measurements. Communicates with the microcontroller over I2C, typically configured at 100 kbps (can support up to 400 kbps)
- Serial Communication Specs:
    - UART: Data transmitted to a PC at 115200bps
    - I2C: Used for communication between microcontroller and sensor at 100kbps
- Programming and Visualization
    - Firmware: Written in C/C++ for the MSP432E401Y, including drivers for the GPIO, I2C, UART, and stepper control
    - Receiver: A Python script using PySerial and Open3D to capture incoming distance data and generate a 3D visualization

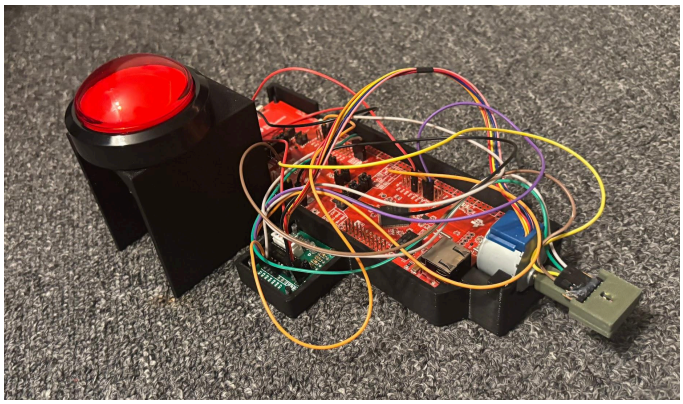

Figure 1. Physical LiDAR device

General Description

This device is a compact and cost-effective LIDAR system designed for detailed indoor spatial mapping. It combines a high-performance MSP432E401Y microcontroller, a stepper motor, and a VL53L1X time-of-flight sensor. Internally, the sensor measures the round-trip time of infrared pulses, converts it to a digital distance reading, and transmits it to the microcontroller over I2C. The microcontroller then processes and packages these distance measurements and sends them to a PC via a UART link at 115200 bps. By capturing measurements across a full 360° vertical sweep and manually displacing the sensor along the x-axis for additional slices, the device builds a dataset in an xyz file. A Python or MATLAB visualization script on the PC converts these readings from polar to Cartesian coordinates and renders a 3D point cloud, enabling an intuitive view of indoor environments. Onboard LEDs and push-buttons offer user interaction, providing real-time status indicators and scan control.

Block Diagram

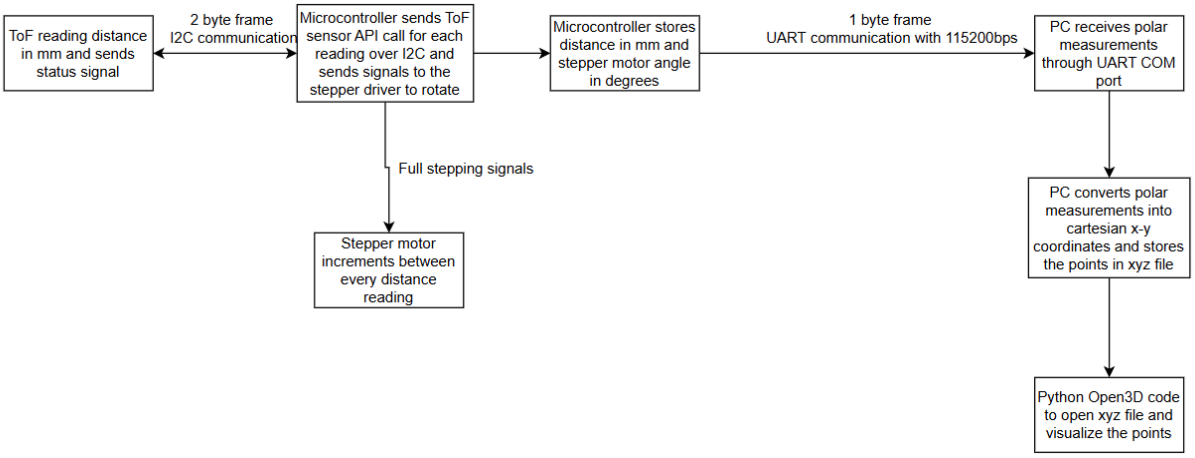

Figure 2. Data flow graph

Device Characteristics Table

Component	**Power Requirement**
MSP432E401Y Microcontroller	5V
VL53L1X ToF Sensor	3.3V
ULN2003 Driver	5V

Table 1. Power requirements

Pin	**Purpose**
PB2	I2C clock (SCL) for communication with ToF
PB3	I2C data (SDA) for communication with ToF
PG0	Sensor XSHUT control for ToF (active-low)
PH0-PH3	Stepper motor control outputs to driver
PM0	Push button input for initiating measurements (active-low)
PF4	Status LED (on when system is in measuring process)
PN0	UART transmission status LED
PN1	Measurement status LED
PJ1	Push button input for stop condition (active low)

Table 2. Microcontroller pinouts

Parameter	Specifications
I2C frame format	Begins with a start condition, followed by a 7-bit device address and a read/write bit, then an acknowledgement, 1 byte of data, a final ACK/NACK, and a stop condition
I2C clock frequency	Set at 100 kbps (although the VL53L1X sensor can operate at speeds up to 400 kbps)
UART frame format	Begins with 1 start bit, 1 byte of data, and 1 stop bit, no parity
UART baud rate	115200bps
Microcontroller bus speed	22MHz

Table 3. Communication and timing specifications

Detailed Description

Distance Measurement

The distance measurement process starts with an initialization phase, where all necessary microcontroller ports and peripherals are configured. This includes:

- Port B for I2C communication with the ToF sensor
- Port G for the sensor's reset line
- Port H for stepper motor control
- Port M, N, F, and J for push button inputs and onboard LED status lights.

Once the system clock is set using PLL and SysTick, the microcontroller verifies the sensor's identity through an API call and continuously polls its boot state until the sensor is ready.

When the user presses the designated push button connected to PM0, the main distance measurement loop is activated. In the loop, the code waits until the sensor indicates that a new distance reading is available, retrieving it through an API call. This function converts the time-of-flight of the 940nm infrared pulse into a digital distance value in millimeters. This conversion is based on the formula:

$$Distance = (speed\ of\ light(mm/s) * time\ delay(s))/2$$

For example, a $6.67*10^{-9}$s delay corresponds to

$$Distance = (3.00 * 10^{11} * 6.67 * 10^{-9})/2$$

$$Distance = 1000mm$$

Which is based on the sensor measuring the time delay between the emission of the infrared pulse (which travels at the speed of light) and its reflection, divided by two because it travels twice the total distance. Internally, the VL53L1X converts this time-of-flight data to a digital distance measurement in millimeters via its built-in ADC, and the microcontroller retrieves this value over I2C at an approximate speed of 100 kbps.

This measurement is signaled by the flashing of an LED on PN1, and then sent to the PC over UART for further processing. After the data acquisition, the motor is rotated 22.5° (using full step), leading to a total of 16 measurements across a full 360° scan. After completing the rotation sequence for one scan, the motor reverses 360° to return the sensor to its starting position, preventing wires from getting tangled, and sends a marker via UART to indicate the end of that scan slice. The user then manually displaces the device along the x-axis by a predetermined increment, so that successive scan layers can be combined to create a 3D map.

Visualization

The visualization process is implemented using Python, primarily leveraging the PySerial library to receive data from the microcontroller and Open3D to render a 3D point cloud. After setting the appropriate COM port (for example, COM4 at 115200 bps) and opening the serial connection, the script waits for the user to press Enter, then receives the UART data from the microcontroller.

As each line of data arrives, the script decodes it into a string and attempts to parse it as a floating-point distance value. An internal angle counter (j) keeps track of the current rotational step. By multiplying j with a fixed angle increment (in radians), the script calculates two-dimensional positions as follows:

x-coordinate: calculated as the cosine of the current angle increment multiplied by distance

$$xcoord = cos(angle) * distance$$

y-coordinate: calculated as the sine of the angle increment multiplied by the distance

$$ycoord = sin(angle) * distance$$

For example, if the sensor measures a distance of 1000 mm at an angle of 45°, convert the angle to radians (45° = 0.7854 radians). Using the polar-to-Cartesian formulas:

x = 1000 * cos(0.7854) ≈ 1000 * 0.7071 ≈ 707 mm

y = 1000 * sin(0.7854) ≈ 1000 * 0.7071 ≈ 707 mm

Meanwhile, a depth variable (i) represents the third axis (e.g., manual displacement). Whenever the script detects the marker 'i' from the microcontroller, it concludes one scan slice, increments i by a configured factor (z_distance_factor), and resets j to zero for the next slice. If the marker 'q' is received, it indicates that the scanning process is finished. Each valid (x, y, i) coordinate is saved to an .xyz file.

Once data collection is complete, the script reads the .xyz file into Open3D. Depending on user settings, either a simple point cloud or a line set is generated. In the latter case, points from each circular scan are connected to form rings, and corresponding points between adjacent slices (separated by increments of i) are also linked. Finally, the script calls Open3D's draw_geometries function to display the resulting 3D model.

Application Note, Instructions, and Expected Output

The device is designed to provide an affordable, portable solution for creating 3D maps of indoor spaces. It leverages a VL53L1X time-of-flight (ToF) sensor mounted on a stepper motor that rotates in fixed increments, while a Texas Instruments MSP432E401Y microcontroller manages sensor data collection and communication. By sequentially capturing distance measurements across a 360° sweep, and incorporating manual displacement along the x-axis between scans, the system constructs a detailed spatial map. This device is ideal for applications in indoor navigation, building layout analysis, and educational projects that require cost-effective spatial data acquisition.

Usage Instructions

1. Programming and Initialization
    - Firmware Flashing: Load the microcontroller code onto the MSP432E401Y using your preferred development environment (e.g. Keil uVision). Ensure the code compiles without errors, and push the reset button after loading.

Figure 3. Location of the reset button on the microcontroller

2. Starting a Scan
    - Press the large push button to initiate the measurement loop.
    - After a valid distance is acquired by the ToF sensor, an onboard LED flashes briefly as a visual confirmation (PN0 and PN1).

Figure 4. Location of onboard LEDs on microcontroller

- After the data is sent to the PC, the stepper motor rotates the sensor by an increment to loop until it covers the entire 360° span, then returns to the home position.

3. Multiple Scan Acquisition
    - A third LED, PF4, is active while the rotational scanning process is ongoing. The next scan is ready when this LED is deactivated.
    - In the Python script, the x and y are defined to be the vertical plane (y being up and down, x being left and right). The z-axis is defined to be the axis between each scan, so moving the system forward between each scan moves it in the defined z-axis.
    - After one complete rotational scan, manually reposition the sensor assembly by a known increment in the z-axis. This increment can be adjusted by the user in the python script to match the physical displacement, a variable called z_distance_factor (in mm).
    - Repeat the scanning process until the desired number of cross-sectional slices is obtained.
4. Data Logging
    - In the Open3D visualization, the user can choose to display the lines between the slice points and between the slices. This is set in the Python code using a boolean called show_lines (True for showing the lines, False for the point cloud).
    - After finishing the scanning process, press the onboard push button to end the scanning process and signal to the Python script to start the visualization process.

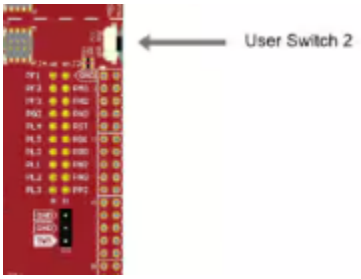

Figure 5. Location of onboard push button on microcontroller

- After pressing the button, this user should see an Open3D window open representing the space scanned.
- As an example of an expected output, this hallway was scanned using a 40cm z_distance factor and show_lines = True:

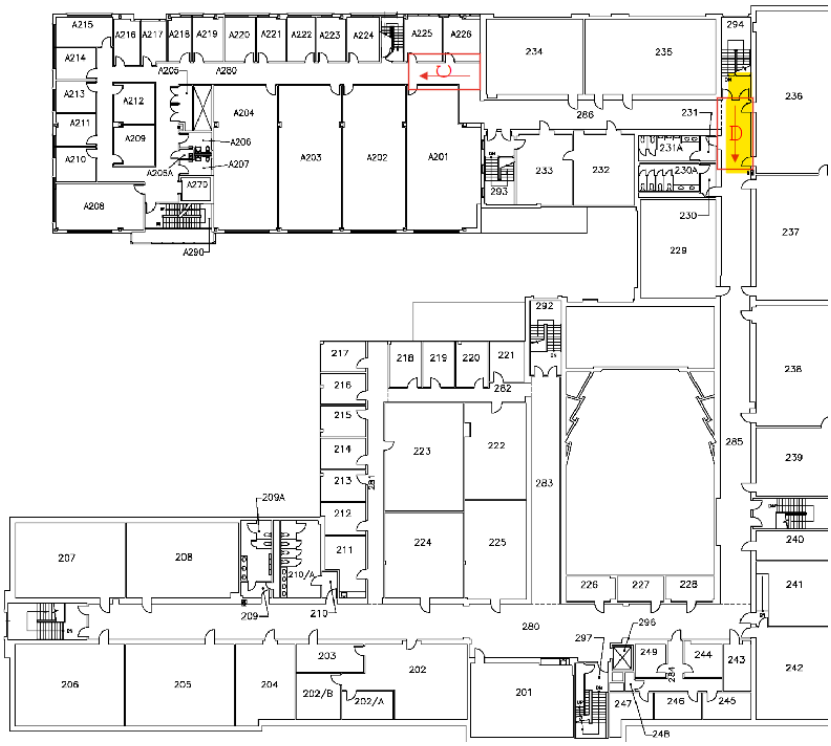

Figure 6. Location of hallway in ITB second floor

Figure 7. Hallway scanned by the system

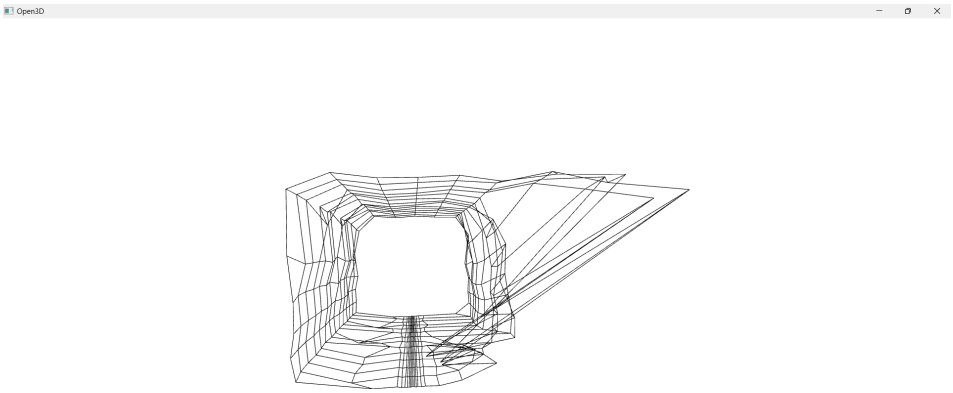

Figure 8. Front view of Open3D projection

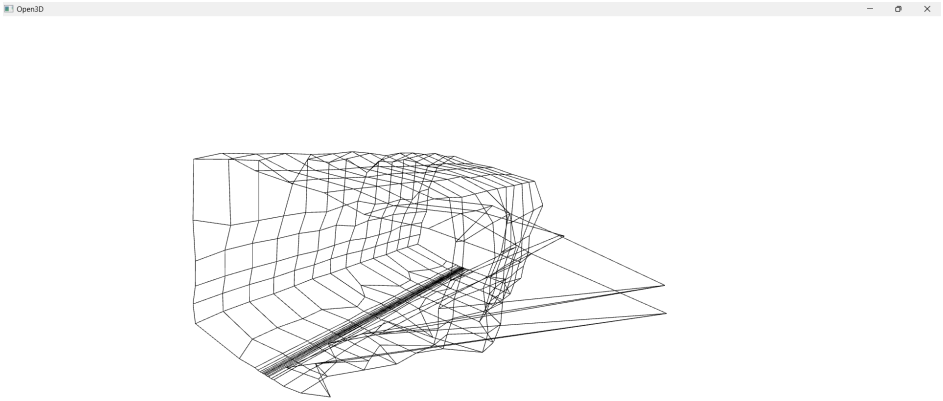

Figure 9. First isometric view of Open3D projection

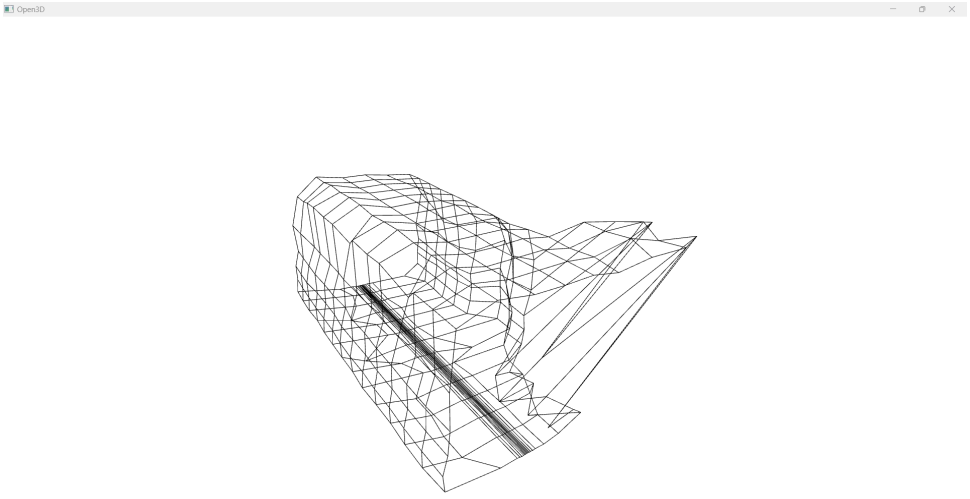

Figure 10. Second isometric view of Open3D projection

Limitations

1. Floating-point and trigonometric precision
    - The microcontroller's single-precision floating-point unit (FPU) can introduce rounding errors during trigonometric calculations
    - In tests comparing the Cartesian coordinates against known reference points, errors of approximately 0.5%-1% were observed, about a 1mm error at a 2000mm distance
2. Quantization Error (ToF sensor)
    - The VL53L1X sensor outputs measurements in 16-bits, and a calculation would lead to a quantization error of $4000mm/2^{16} = 0.06mm$
    - However, the sensor only outputs measurements with a resolution of 1mm, leading to the practical quantization error of 1mm
3. Serial Communication
    - Tests using Windows Device Manager and logging scripts indicated that increasing the UART baud rate beyond 115200bps led to a data loss
    - 115200bps was chosen as a reliable maximum for communication
    - Although the ToF sensor supports up to 400kbps over I2C, practical tests showed that configuring I2C at 100kbps resulted in more stable data acquisition with little to no error codes
4. System bottleneck
    - The overall scan speed is limited by the stepper motor's requirement for a minimum delay (~2ms per phase) to prevent missed steps and avoid overheating, as well as by the sensor timing budget for each distance measurement
    - This limitation was verified experimental by gradually reducing the step delay and observing that delays shorter than 2ms resulted in the motor stalling and overheating

Circuit Schematic

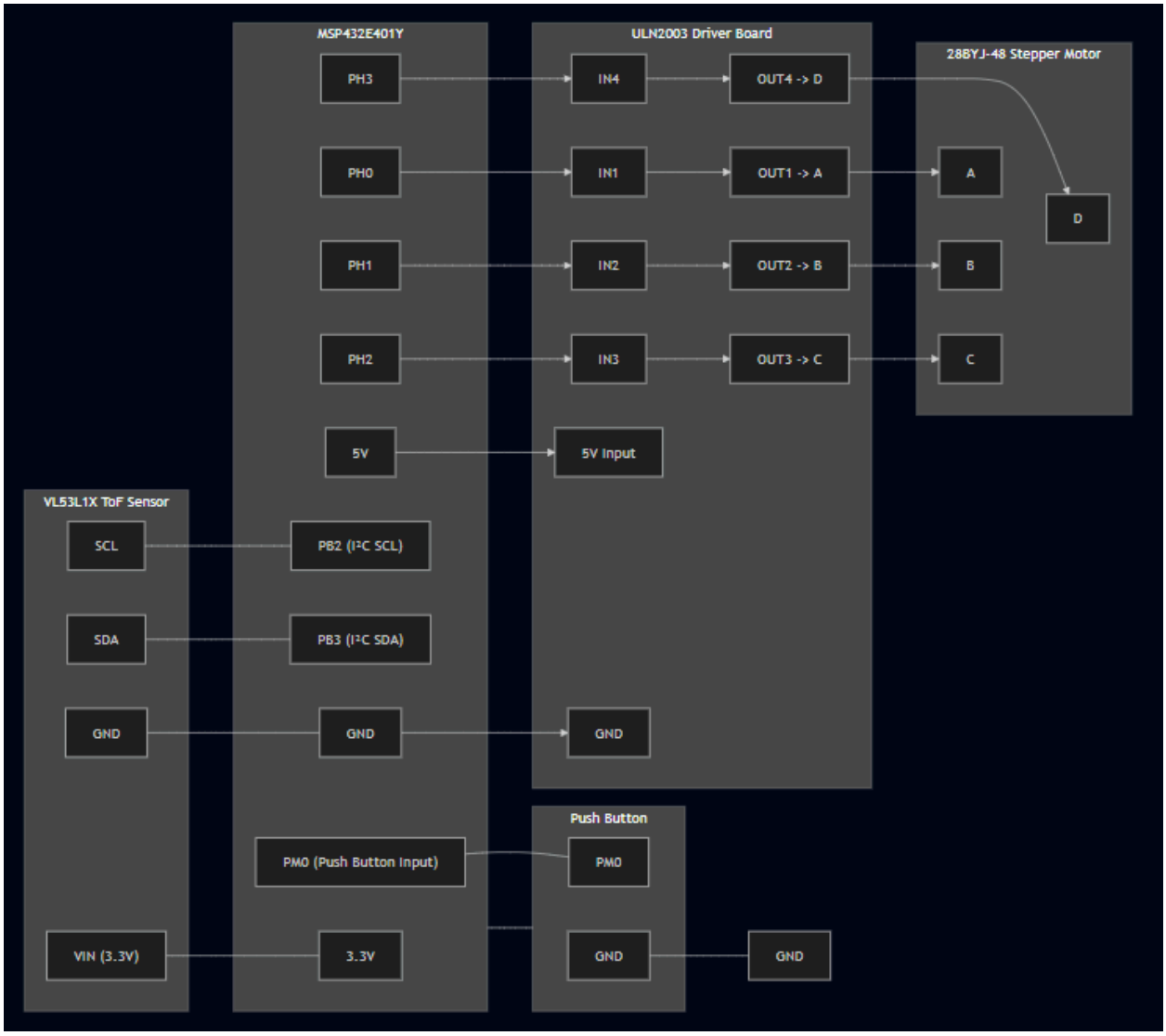

Figure 11. Circuit schematic of system

Programming Logic Flowchart

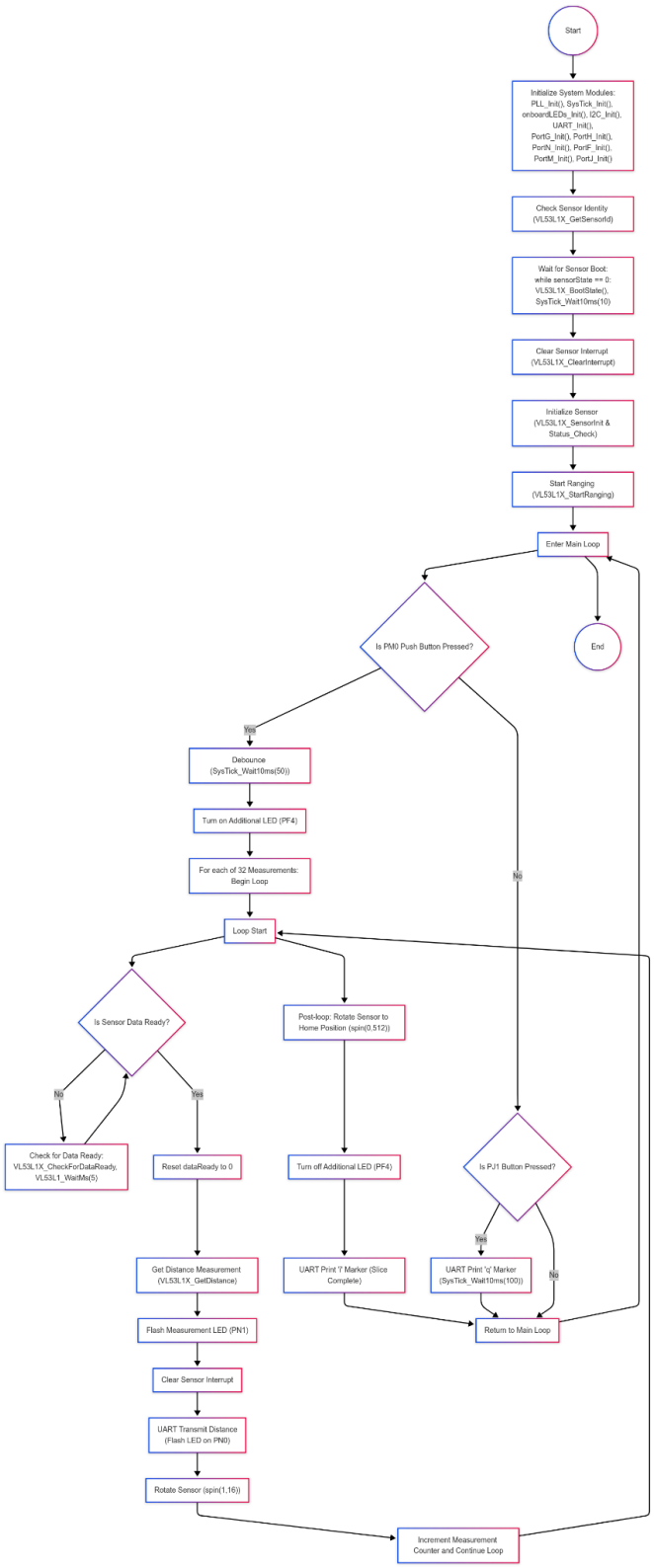

Figure 12. Flowchart for microcontroller

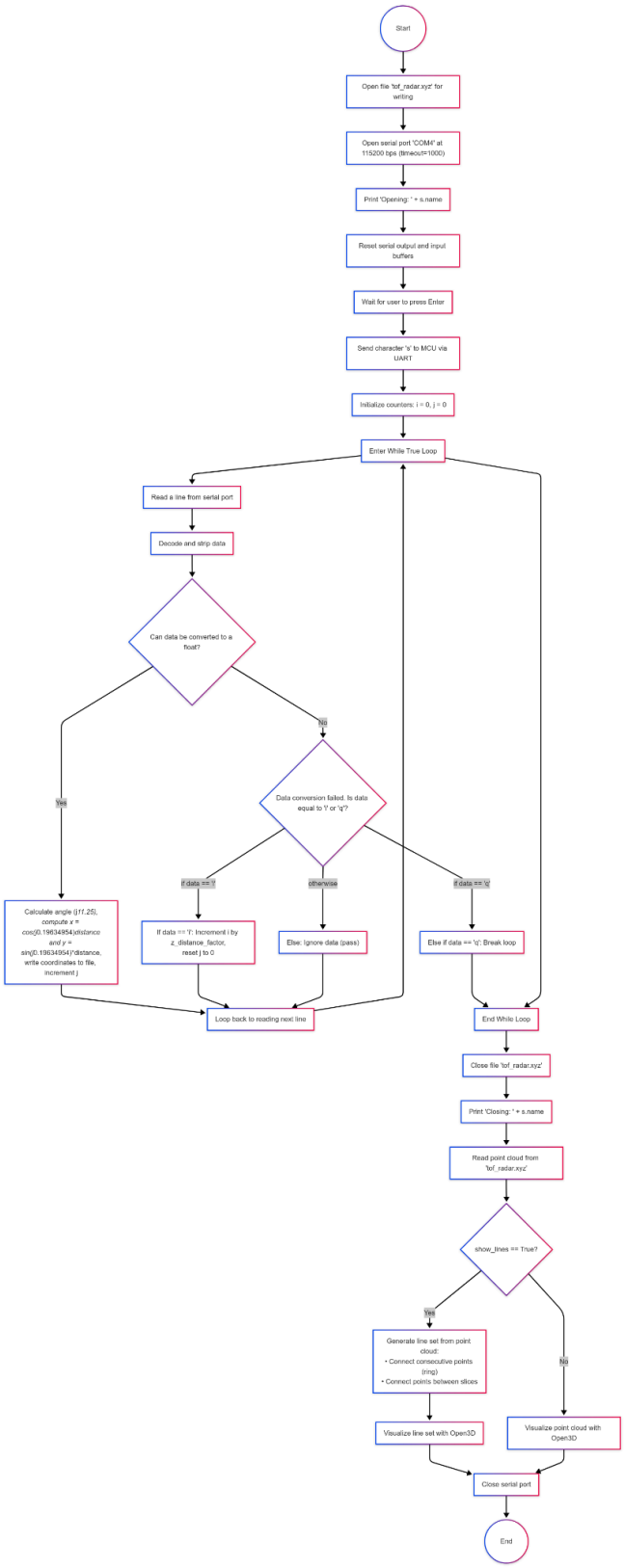

Figure 13. Flowchart for PC Python script

Sources

[1] Rajguru Electronics (I) Pvt. Ltd., “Stepper Motor 5V 4-Phase 5-Wire & ULN2003 Driver Board for Arduino,” Rajguru Electronics, 2019. [Online]. Available: https://www.rajguruelectronics.com/Product/1467/28BYJ-48%20-%205V%20Stepper%20Motor.pdf. [Accessed: Apr. 07, 2025].

[2] Pololu, “VL53L1X Time-of-Flight Sensor Datasheet,” Pololu. Available: https://www.pololu.com/file/0J1506/vl53l1x.pdf, Accessed: Apr. 07, 2025.