SPATIAL MAPPING USING TIME-OF-FLIGHT

Final Report – 2DX3 Winter 2025

Author: Pranav Kukreja

Student Number: 400511832

Microcontroller: MSP-EXP432E401Y

Assigned Location: C

As a future member of the engineering profession, the student is responsible for performing the required work in an honest manner, without plagiarism and cheating. Submitting this work with my name and student number is a statement and understanding that this work is my own and adheres to the Academic Integrity Policy of McMaster University and the Code of Conduct of the Professional Engineers of Ontario. Submitted by [Pranav Kukreja, kukrejap, 400511832]

1 Device Overview

1.1 Features

- All-In-One LiDAR System
  - Indoor scanning and 3D spatial mapping
  - Stepper-mounted VL53L1X Time-of-Flight sensor
  - Z-axis displacement control for full volumetric scanning
  - Compatible with Open3D for visualization
- Texas Instruments MSP432E401Y Microcontroller
  - Arm Cortex-M4F Processor Core
  - Assigned 20 MHz Bus Speed (based on student ID)
  - 256KB Flash, 64KB SRAM, 2KB EEPROM
  - Input Voltage: 3.3V from USB via XDS110 Debug Interface
  - Cost: $60.00 CAD
  - GPIO Status LEDs: PF4 (Measurement), PF0 (UART Tx), PN1 (Additional Status)
  - C Language programming using Keil IDE
- VL53L1X Time-of-Flight Sensor
  - Up to 400 cm distance range
  - Ranging frequency up to 50 Hz
  - Operates at 2.6V–5.5V with I2C (100 kHz)
  - Cost: $30.00 CAD
- MOT-28BYJ48 Stepper Motor with ULN2003 Driver
  - Input Voltage: 5–12 VDC
  - 512 steps per full rotation
  - Direction alternates to avoid wire tangling
  - Cost: $5.00 CAD
- Visualization
  - Python 3.12 (64-bit), Open3D, NumPy
  - Data exported as a .xyz file and visualized as point cloud and line mesh
- Communication
  - I2C between ToF and MCU
  - UART (115200 bps) between MCU and PC

1.2 General Description

This embedded spatial mapping system is designed to perform 3D reconstructions by combining rotational distance measurements with manual linear displacement. At its core is the VL53L1X time-of-flight (ToF) sensor, mounted on a stepper motor to capture 360-degree scans in the vertical YZ plane. The entire system is managed by the Texas Instruments MSP-EXP432E401Y microcontroller, which coordinates sensor readings, motor control, and UART communication with a host PC.

The VL53L1X determines distance by emitting an infrared light pulse and measuring the time it takes for the light to reflect off a surface and return to the sensor. The round-trip time and the known speed of light allows the sensor to calculate the distance to nearby objects. Internally, the sensor performs signal acquisition by detecting the reflected infrared signal, then processes the timing information using its built-in microcontroller. The resulting distance measurement is digitized within the sensor itself, eliminating the need for external analog-to-digital conversion. These preprocessed digital measurements are accessed through the sensor's built-in API over an I²C interface.

During operation, the user manually advances the system along the Z-axis to simulate depth, creating layered slices of 360-degree scans. At each vertical layer, the stepper

motor takes 32 evenly spaced readings, every 11.25 degrees by stepping 16 times per measurement. These readings are triggered via push button and accompanied by LED indicators: PF4 for scan progress, PF0 for UART transmission, and PN1 for system readiness.

After each reading, the microcontroller sends the data to a PC over UART at 115200 bps. On the PC side, a Python script using the Open3D library parses the incoming serial data, converts it from polar to Cartesian coordinates, and visualizes the result as a 3D point cloud and mesh. Each scan forms one "layer" in a 3D model, with the Z-axis representing scan depth and the X/Y axes determined by the angle and measured distance.

This system provides a low-cost platform for LiDAR-style spatial mapping which is ideal for hands-on exploration of embedded systems, distance sensing, and real-time 3D data visualization.

1.3 Block Diagram (Data flow graph)

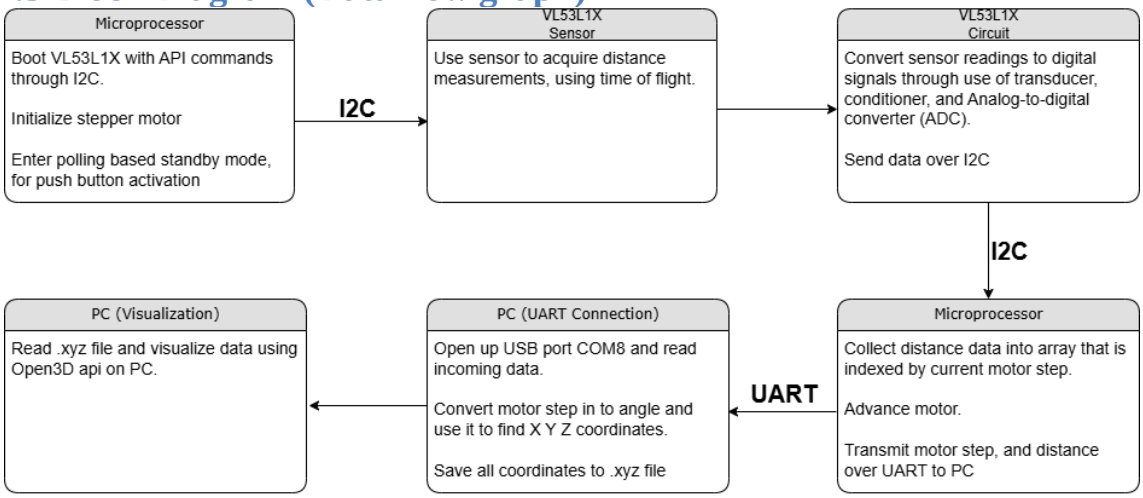


Figure 1: Data flow graph of LiDAR system

2 Device Characteristics Table

MSP432E401Y		ULN2003		VL53L1X	
Feature	Detail	Device pin	Microcontroller pin	Device pin	Micro pin
Bus speed	20 MHz	VDD	5V	VDD	
Serial port	COM8	GND	GND	VIN	3.3V
Baud rate	115200	IN1	PH0	GND	GND
Measurement status	PF4	IN2	PH1	SDA	PB3
UART Tx status	PF0	IN3	PH2	SCL	PB2
Additional status	PN1	IN4	PH3	XSHUT	
Scan trigger	PJ0				

Table 1: Device characteristics table

3 Detailed Description

3.1 Distance and Displacement Measurement

The spatial mapping system developed in this project relies on the VL53L1X Time-of-Flight (ToF) sensor to obtain distance measurements for 3D reconstruction. The sensor emits a series of short 940 nm infrared laser pulses using a vertical-cavity surface-emitting laser (VCSEL). When these pulses reflect off a surface and return to the sensor, the time-of-flight is measured. The sensor computes the distance using the equation:

$$D = \frac{(In\ Flight\ Time)}{2} \times speed\ of\ light$$

This calculation, which divides the total round-trip time by two to account for the path to and from the target, enables non-contact distance measurements up to approximately 4 meters. Notably, the VL53L1X is configured in Long Distance Mode which maximizes the sensor's effective range, with a slight trade-off in its ability to filter ambient noise and measure from lower-reflectivity surfaces.

Upon power-up, the MSP-EXP432E401Y microcontroller initializes the VL53L1X sensor via I²C at 100 kHz using PB2 and PB3 as the clock and data lines. At the same time, it sets up the 28BYJ-48 stepper motor through GPIO pins PH0–PH3. The scanning cycle is triggered manually when the user presses the pushbutton connected to PJ0. A simple polling-based debounce logic is implemented in the code. When the PJ0 input is detected, the system enters a loop to wait until the button is released. This pattern prevents multiple triggers due to mechanical bounce, ensuring that only a single scan cycle is initiated per button press.

```
for (int i = 0; i < motor_index; i++)
{
    dataReady = 0;

    while (dataReady == 0)
    {
        status = VL53L1X_CheckForDataReady(dev, &dataReady);
        VL53L1_WaitMs(dev, 4);
    }
```

Figure 2: Polling loop to process distance measurements

During a scan, the stepper motor rotates the sensor incrementally through a full 360° sweep. The motor divides one full rotation into 512 steps, with one measurement taken every 16 steps yielding an angular resolution of 11.25°. For each step, the microcontroller polls the sensor in a loop using:

This polling loop ensures that the system only proceeds when a valid distance measurement is available. Once ready, the measurement is read, stored in a local buffer (measurements[]), and transmitted via UART in the format “index, distance.” All the

while, visual feedback is provided through onboard LEDs: PF4, PF0 and PN1 which flash with each measurement, toggle for UART transmission and flash for button click respectively.

To prevent cable entanglement, the motor alternates its rotation direction between scans by first rotating clockwise and then counterclockwise for the next scan. This allows for unassisted operation of the rotating sensor, so the user can focus on the manual displacement along the z-axis. Since the microcontroller does not track its own spatial position, the final z-coordinate for each point is computed on the PC during data processing by multiplying the layer index by the displacement value (δ_z). The corresponding x and y coordinates are derived from the sensor's polar readings through trigonometric conversions.

3.2 Visualization

After acquiring distance data on the microcontroller, the system transmits this data in real-time to a connected PC using UART communication. Each reading is sent as a string in the format index, distance, where the index represents the angular position of the reading (from 0 to 31), and the distance is the measured value in millimeters. The UART baud rate is set to 115200 bps, and the data is streamed continuously during each scan cycle. On the PC side, a Python script reads and processes the incoming UART data in real time. The script listens for each line of data and immediately parses the index and distance values.

After completing each scan layer, the microcontroller sends a "COMPLETE" message to the PC over UART. The Python script waits for this confirmation before proceeding to the next layer, ensuring that the data from one rotation is fully captured before beginning the next. This handshake protocol prevents misalignment and data corruption, especially in longer scans or high-noise environments.

The VL53L1X is configured in Long Distance Mode via the provided API, allowing it to capture readings up to 4 meters. While this provides extended range capabilities, it may also increase sensitivity to environmental noise and decrease performance on dark or low-reflectivity surfaces. The selected mode offers a trade-off between range and accuracy and was chosen to maximize the usable scanning volume in this application.

As each line of data is received on the PC, the script parses the index to determine the angle of measurement by multiplying the index by 11.25°, the angular resolution per step. This angle is then converted to radians using the formula:

$$\theta = \text{index} \times 11.25° \times \left(\frac{180}{\pi}\right)$$

Using this angle and the measured distance, the Cartesian coordinates (x, y) are calculated via:

$$x = r \times \cos(\theta)\,, y = r \times \sin(\theta)$$

The z-coordinate is assigned based on the scan layer using:

$$z = layer\ number \times \delta_z$$

where δ_z is a user-defined constant representing the vertical spacing between layers, typically 100 mm. Each point is stored in a growing list of (x, y, z) coordinates.

In addition to displaying a point cloud, the Python script uses Open3D's LineSet feature to construct a mesh-like structure that visually connects points. Within each layer, points are connected sequentially in the angular scan order. Between layers, points from adjacent scans are also linked with an index wraparound mechanism that accounts for the alternating scan direction. This means the first point in one layer connects to the last point in the next, preserving correct geometry across clockwise and counterclockwise sweeps.

After all scan layers have been processed, the complete 3D point cloud is saved to a .xyz file. This file format contains the raw (x, y, z) coordinates in plain text and can be reloaded into Open3D or other 3D visualization tools such as MeshLab. Exporting the data enables future reprocessing, external rendering, or comparative analysis with other scans.

The final model is rendered in an Open3D viewer, which supports interactive panning, zooming, and rotation. The visualization software was tested on a Windows 10 laptop running Python 3.12, NumPy, and Open3D. The modular architecture of the script supports scalability which means more layers, higher resolution scans, or denser point clouds can be added with minimal changes.

4 Application Example

This section outlines the steps required to operate the 3D scanning system from start to finish. The process assumes the user already has all the necessary software tools installed and setup, including the Keil µVision IDE for compiling and flashing the microcontroller code, and Python 3.12 with all required libraries ("pyserial", "open3d", and "numpy") installed on the host PC.

Instructions

Step 1:
1. Connect the MSP-EXP432E401Y development board to your PC via USB.
2.Ensure the VL53L1X ToF sensor is connected to the I²C pins
- Wiring diagrams are provided in device characteristics.
3. Connect the 28BYJ-48 stepper motor to PH0–PH3 via a ULN2003 driver board
- Wiring diagrams are provided in device characteristics.

Step 2:
1. Open the provided project (Final_Project.uvprojx) in Keil µVision.
2. Click "Translate" to compile the project.
3. Click "Build" to generate the hex binary.
4. Click "Download" to flash the program to the MSP432 board.

Once flashed, the microcontroller will wait for the user to press the scan button (PJ0) to begin ranging and transmitting data.

Step 3:
1. Open Device Manager on your PC to check which COM port the MSP432 is connected to:
- Expand the Ports (COM & LPT) section.
- Look for a label such as "XDS110 Class Application/User UART ".
- Note the COM port number (e.g., COM3)

2. Open the file “Visualizer.py” using IDLE.
3. Edit configurations such as “layer_count” or “layer_distance” and update the COM port number.
4. Run the script by selecting Run → Run Module or pressing F5 in IDLE.
5. When prompted by the script:
- Press PJ0 on the microcontroller to begin scanning.
- After each scan, move the device forward along the z-axis while the X/Y points are generated.

6. The script will continue collecting data until all layers are complete.

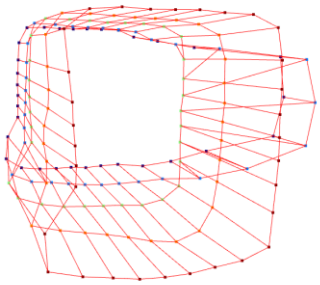
During each scan, the PC receives “index, distance” data via UART, converts it to (x, y, z) points, and updates the visualization using Open3D. Scan completion is synchronized using a "COMPLETE" message from the microcontroller.

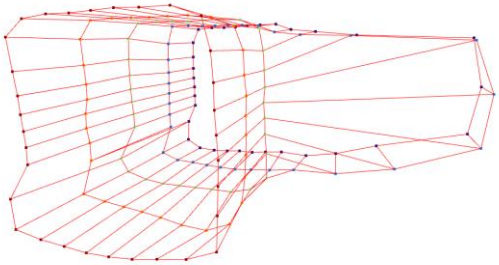
Step 4:
1. After the final scan layer, all 3D points are saved to a file named “scanned_data.xyz”.
2. This file can be reopened in Open3D or third-party tools like MATLAB for further analysis or visualization.

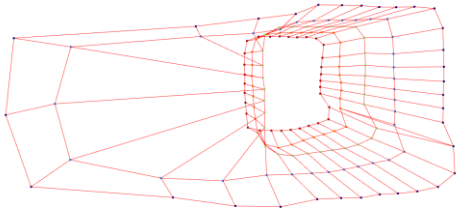
Expected Output

Upon successful operation of the LiDAR scan, the user should observe a series of visual outputs corresponding to each stage of the scanning process. These outputs validate correct behaviours, and the generation of the 3D point cloud. Starting with LED indicators that flash when there is a confirmed scan, UART data transmission, and when the push button is clicked. Furthermore, the IDLE will display “COMPLETE” in the python terminal when the scanning is completed. Finally, an Open3D viewer will display an interactive visualization of the scan that the user can later access with the saved 3D points data file.

Through this visualization, users recreate real objects and places in a virtual scan. The expected output can be compared against my assigned locations and corresponding scans below.







Figures 3,4,5,6: Real Life Hallway vs Virtual Scans

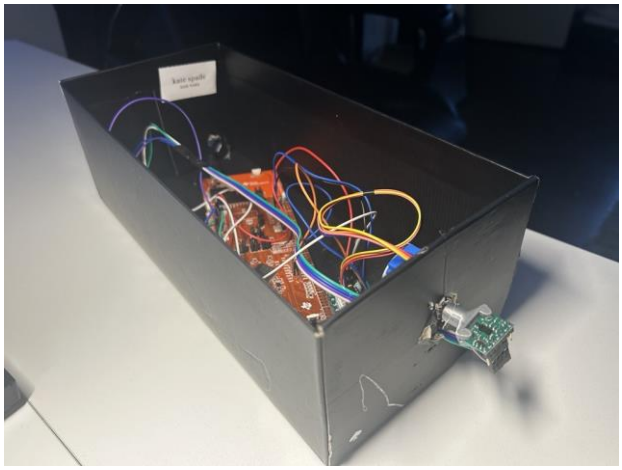


Figure 7: Physical implementation of scanner

Based on the figures above it can be seen that the LiDAR system can capture the overall shape of the hallways, while still maintaining finer details such as the indent of the door. The opening on the left of the real-life example was also scanned till the sensor's maximum of four feet.

5 Limitations

5.1 Floating Point and Trigonometric Processing

The MSP-EXP432E401Y microcontroller includes a hardware Floating Point Unit (FPU) capable of performing single-precision (32-bit) operations such as addition, subtraction, multiplication, and division among others. It also supports conversions between integer and floating-point types which means the system can handle floating-point arithmetic. The trigonometric functions used in this project, including sin() and cos(), are performed on float values using the standard math.h library. Given the presence of the FPU, there are no significant computational delays or bottlenecks introduced by the trigonometric calculations.

5.2 Quantization Error

The VL53L1X Time-of-Flight sensor outputs distance measurements in discrete 1 mm steps, which means its resolution is 1 mm. This resolution is set by the internal signal processing and digital interface of the sensor. The distance values returned by the sensor through the I²C interface are already converted into millimeters using onboard processing and no ADC-based conversion is involved in this process.

Because the sensor cannot distinguish between distances that fall within the same 1 mm interval, the maximum quantization error is equal to ±0.5 mm, which is half of one resolution unit. This represents the worst-case difference between the true analog distance and the digital value returned.

However, for the purpose of this report, the correct maximum quantization error is 1 mm, since this is the smallest possible change the module can detect in its output.

$$Max\ Quantization\ Error = 1\,mm$$

5.3 UART Communication Rate

The maximum standard serial communication rate supported by the PC for the XDS110 UART interface is 128000 bps, as confirmed through Windows Device Manager under the COM port's advanced settings. For this project, the UART was configured to operate at 115200 bps, a widely supported and stable baud rate. This speed provided data transfer throughout each scan layer, with no loss or corruption of UART transmissions. Communication integrity was verified by comparing the number of transmitted UART

lines with the expected count on the PC, and by visually inspecting the formatting of each line for correctness.

5.4 Microcontroller-ToF Communication Speed

Communication between the microcontroller and the ToF module is performed over I²C using a standard 100 kHz clock. The microcontroller uses PB2 and PB3 as SCL and SDA, respectively. This I²C clock rate is compatible with the VL53L1X's requirements and is supported directly by the ST-provided API used to initialize and poll the sensor. While faster communication is possible via Fast Mode, the system retains standard I²C speed for improved signal stability and reduced complexity. The communication time is minimal relative to the sensor's internal ranging delay and does not significantly affect system.

5.5 Primary System Bottleneck

The primary limiting factor in system speed is the ranging delay of the VL53L1X sensor. Although the stepper motor proves to also be a significant bottleneck, it consumes less time per iteration compared to the sensor's ranging cycle. Each ranging event can take 20–60 ms, depending on mode and reflectivity. In testing, attempts to reduce wait times between checks of VL53L1X_CheckForDataReady() resulted in scan failures where the sensor returned no data, halting the system. In contrast, the motor simply skips steps when delay is insufficient, without crashing the program. This confirms that the ranging is the main one of two bottlenecks that the system faces along with the stepper motor.

5.6 Bus Speed Configuration and Timing Adjustments

As part of the system setup, the assigned bus speed was set to 20 MHz, based on the student number. To meet this requirement, the system clock which originates from the internal 480 MHz PLL was divided using the PSYSDIV register. Setting PSYSDIV = 23 results in:

$$Bus\ Speed = \frac{480\ Mhz}{PSYSDIV + 1} = \frac{480}{23 + 1} = 20\ Mhz$$

This configuration ensures that all peripheral operations, including I²C communication and motor control, are synchronized with the designated bus speed.

In addition to clock division, the SysTick timer was recalibrated to generate accurate software delays based on the new system clock. For a 10 ms wait interval, the appropriate delay value was calculated as:

$$SysTick\ Count\ =\ 20\ MHz\ \times 0.01\ s\ =\ 200{,}000$$

This value was implemented in the code using the SysTick_Wait(200000); function. The timing was verified experimentally using an oscilloscope to measure the duration of repeated delay cycles, confirming the reliability of the configured system timing.

6 Circuit Schematic

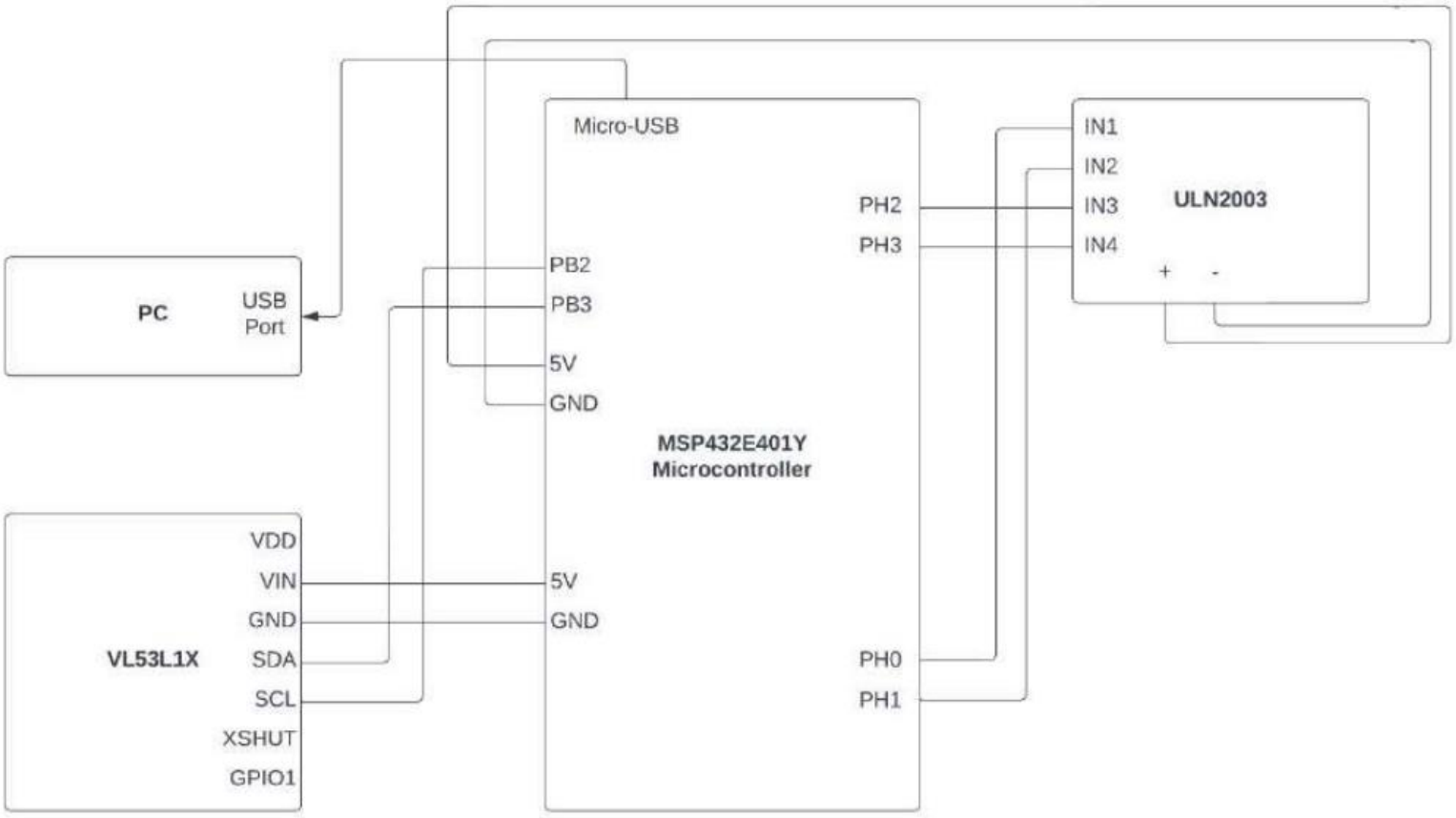


Figure 8: System circuit schematic

7 Programming Logic Flowcharts

C Programming Flowchart

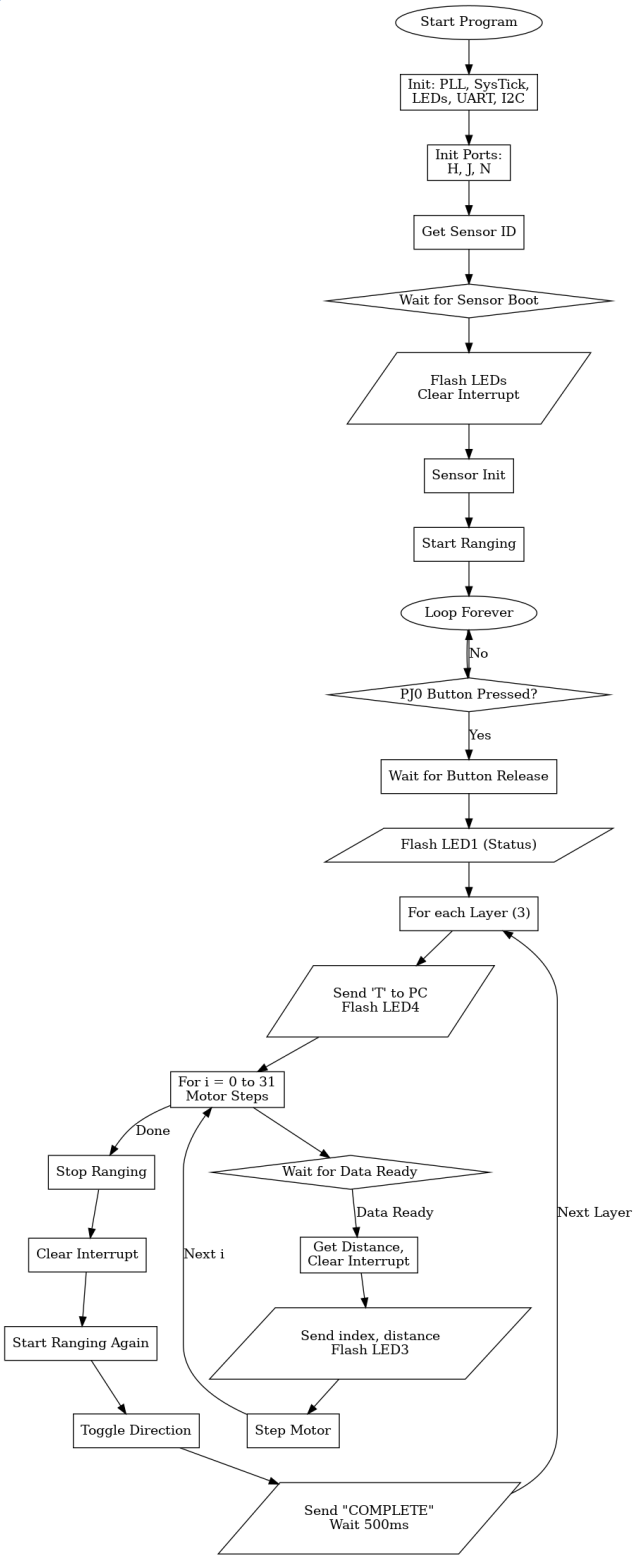


Figure 9: Flowchart for Final_Project.c

Python Programming Flowchart

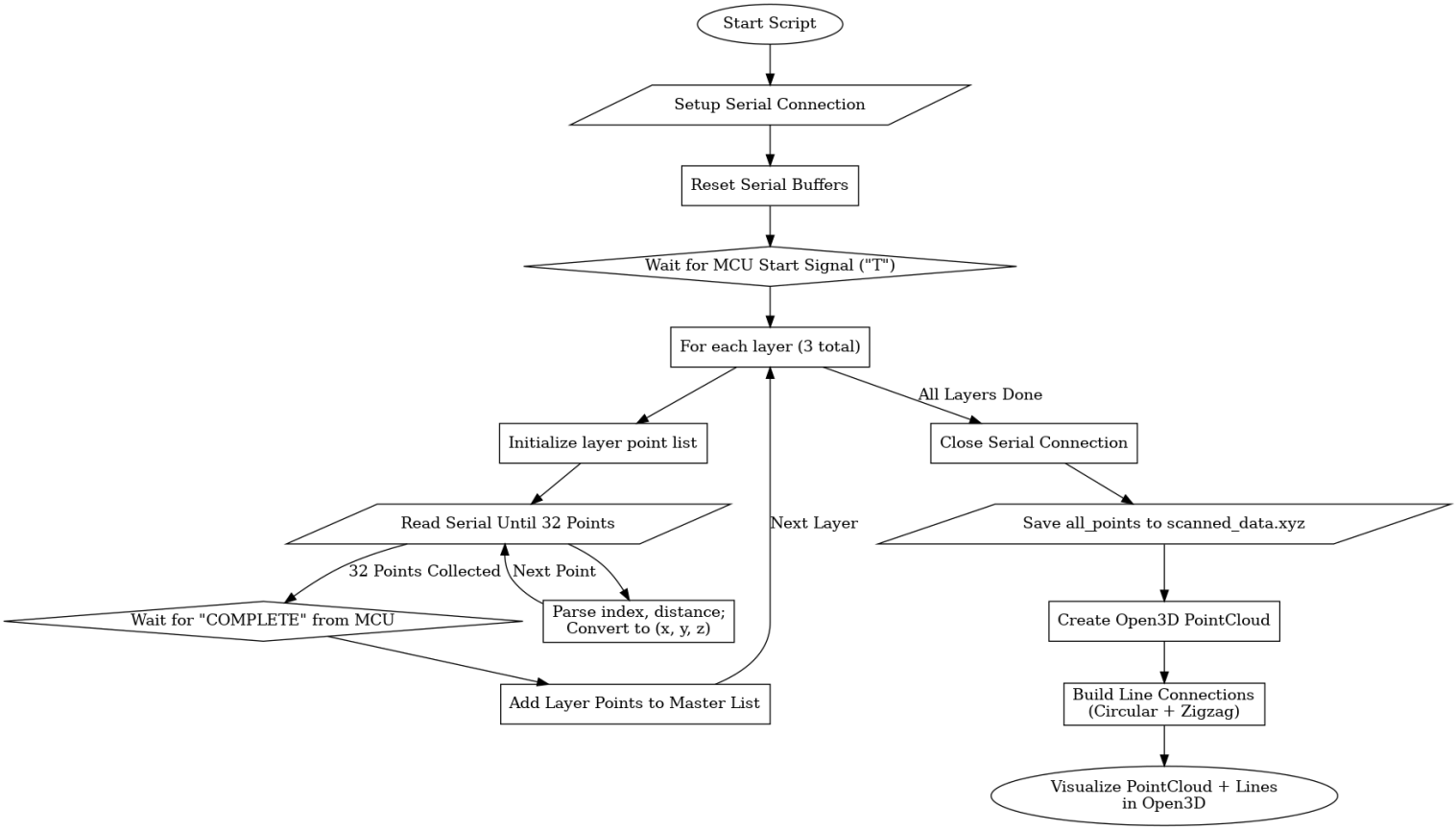


Figure 10: Visualizer.py flowchart

8 References

[1] “MSP-EXP432E401Y Development kit | TI.com.” https://www.ti.com/tool/MSPEXP432E401Y.

[2] “vl53l1x - COMPENG 2DX3: Microprocessor Systems Project.” https://avenue.cllmcmaster.ca/d2l/le/content /512368/viewContent/3979106/View.

[3] “Pololu - VL53L1X Time-of-Flight Distance Sensor Carrier with Voltage Regulator, 400cm Max.” https://www.pololu.com/product/3415/specs.