COMPENG 2DX3

Final Report: Embedded 3D Spatial Mapping System

Joey McIntyre – mcintj35 – 400520473

April 8th, 2025

Dr. Shahrukh Athar

Dr. Thomas Doyle

Dr. Yaser M. Haddara

Table of Contents

Device Overview

Features

Texas Instruments MSP432E401Y Microcontroller [1]:

- 120-MHz Arm Cortex-M4F Processor Core with Floating Point Unit (FPU)
- 1024KB of Flash Memory with 4 Bank Configuration
- 256KB of SRAM with Single-Cycle Access
- 2-GB/s Memory Bandwidth at default Bus Speed of 120MHz.
- 22MHz Assigned Bus Speed (400520473)
- 6KB EEPROM
- Eight UARTs
- Ten I^2C modules with 4 Transmission Speeds
- Arm Primecell 32-Channel configurable uDMA Controller
- 15 Physical GPIO Blocks
- Two 12-Bit SAR-Based ADC Modules – Each Supports up to 2Msps
- Three Independent Analog Comparator Controllers
- 16 Digital Comparators
- -40 – 105 Degree C Extended Operating Temperature

ST VL53L1X Time-of-Flight Sensor [2]:

- Up to 4m Distance Measurement
- Up to 50Hz Ranging Frequency
- 27 Degree Full Field-of-View
- 2.6V-3.5V Operating Voltage
- -20 – 85 Degrees C Operating Temperature

28BYJ-48 – 5V Unipolar Stepper Motor [3]:

- 5V Rated Voltage
- 4 Phases
- 512 Steps For 360 Degree Rotation
- 100Hz Frequency

Communication:

- I^2C bus (100kHz) for Sensor Interface
- UART Serial Link (115200 Baud Rate) for Data Transmission to PC

User Interface:

- One On-Board Pushbutton (PJ1) for an Interrupt Triggered Start
- Three On-Board LEDs (D1, D2, D3) as Status Indicators (User Specific - 400520473)
    - D1 (PN1) for Measurement Status
    - D2 (PN0) for UART Tx
    - D3 (PF4) for Additional Status – Motor Movement

PC Software:

- Keil uVision 5 for Development of Embedded System

- Python 3.8+ (Import NumPy and Open3D) for Data Processing and 3D Visualization

Operating Voltage:

- 5V DC via USB (LaunchPad Supply)
- 3.3V Logic Provided by On-Board Regulators

Estimated Cost:

- $254.99 – COMPENG 2DX3 Kit from McMaster Campus Store

General Description

This embedded system is a 3D spatial mapping device that captured distance measurements from its surrounding environment and communicates a 3D model of its surroundings to a PC. This system consists of a VL53L1X Time-of-Flight (ToF) sensor which emits an invisible infrared laser to measure the time of flight to determine the distance to objects that reflect the IR photon up to 4m away. This sensor is mounted on a stepper motor using a small 3D printed piece so it can rotate 360 degrees in a vertical plane to take 32 distance measurements evenly spaced around of the full circle in that plane. The microcontroller facilitates this process by interfacing with the ToF sensor via I^2C to initiate distance measurements and read distance data. It also controls the stepper motor via GPIO outputs to step the sensor through each desired measurement angle. The press of pushbutton Pj1 triggers an interrupt that starts one full scanning sequence. During this process, the on-board LEDs flash to provide live feedback. LED D1 blinks to indicate a distance measurement, LED D2 blinks to communicate the start of the UART communication, and LED D3 blinks with each motor step so the user can confirm the device is functioning as intended while running.

Distance measurements that are collected for each angle are stored in the microcontroller memory temporarily, before being transmitted to the users PC with a UART serial connection. The UART uses a baud rate of 115200, which is a standard high speed for reliable communication with a PC. The microcontroller acquires and processes the sensor data, and the PC preforms the processing and visualization aspects. This is done by running a Python program which receives the distance measurements and the angle position via a serial port, converts the measurements into cartesian coordinates using trigonometric transformations, and integrates the multiple data sets that come from moving the device along the non-automated axis to build a full 3D point cloud of its environment. Using the Open3D software, the point cloud is rendered in a graphical window so the user can visualize the mapped space and compare it to the expected results.

Overall, this device functions as a low-cost 3D lidar scanner. It acquires a vertical slice of distance data in the Y-Z plane, and is manually moved in consistent increments along the X-axis to map an area. This system relies on seamless integration and communication between multiple components to achieve the desired result, which requires a deep understanding of the 2DX3 course curriculum to achieve. The physical system I implemented can be observed in Figure 1 below.

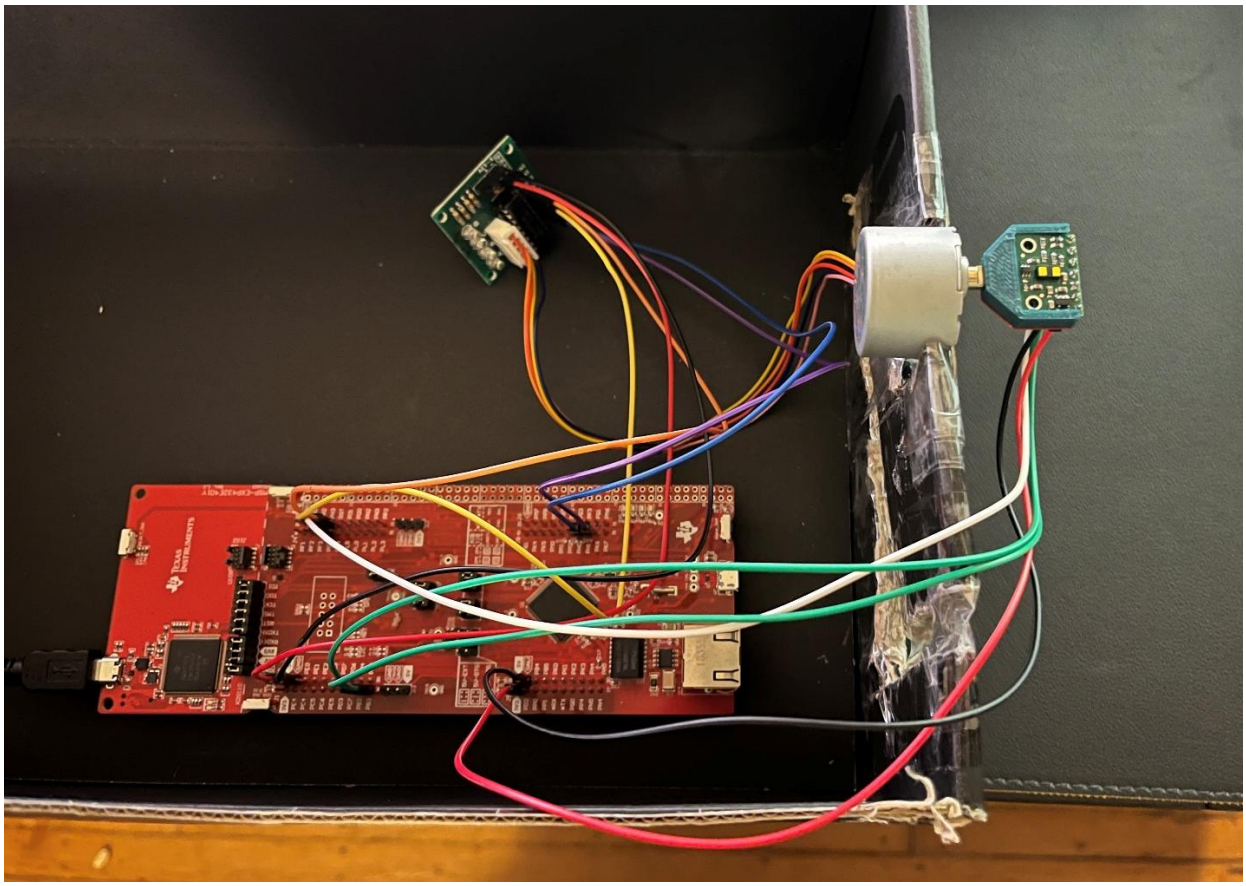

Figure 1: Photo of 3D Spatial Mapping System - Final Physical Implementation

Block Diagram (Data Flow Graph)

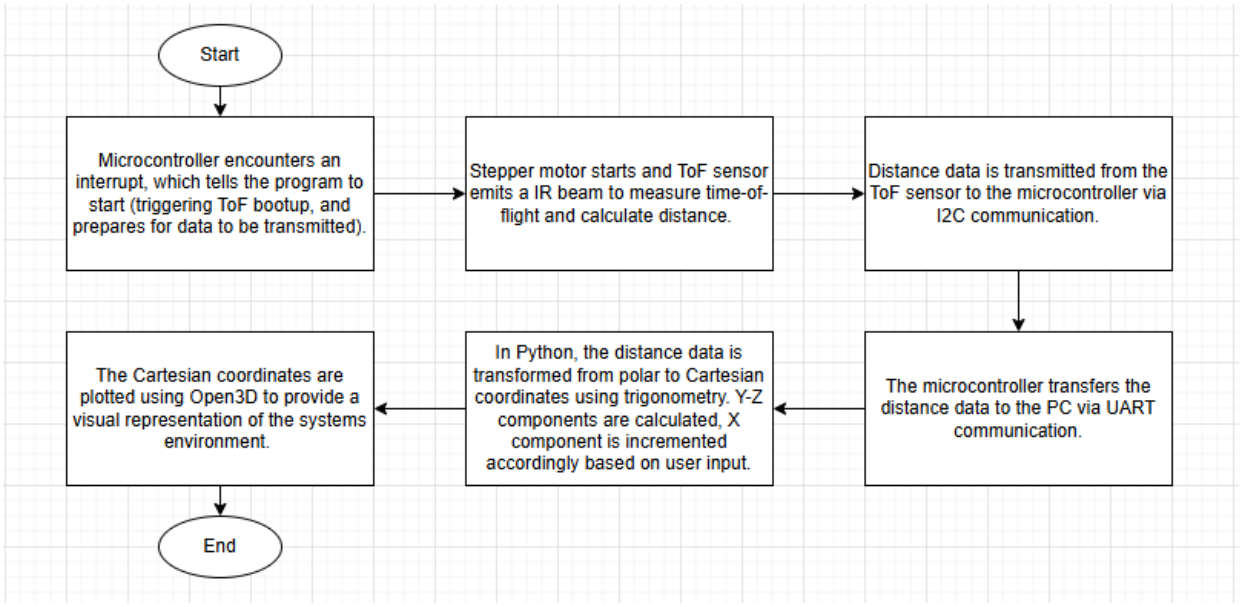

Figure 2: Block Diagram of the Embedded Spatial Mapping System

Device Characteristics Table

Table 1 below summarizes the important technical characteristics and connections of the device. The focus for this table was to summarize the users technical "need to know" characteristics to build it themselves.

Table 1: Key Device Characteristics and Pin Assignments

Specifications	Values	
Bus Speed (MCU Clock)	22MHz (Configured via PLL)	
Operating Voltage	3.3V Logic (MCU & sensor), 5V motor	
UART Port / Baud Rate	COM4 (115200 pbs, 8-N-1)	
I^2C Bus Speed	100kHz	
Python Version	3.8+	
Measurement Status LED	PN1 (LED D1)	
UART Transmission Status LED	PN0 (LED D2)	
Motor Activity LED	PF4 (LED D3)	
Pushbutton Pin (Interrupt)	PJ1 (active-low interrupt trigger)	
VL52L1X ToF Sensor Connections		
Microcontroller Pin	Sensor Pin	Function
PB2	SCL	I^2C Clock
PB3	SDA	I^2C Data
PG0	XSHUT	Sensor shutdown (active-low)
3.3V	VIN	Sensor power
GND	GND	Common ground
28BYJ-48 Stepper Motor Connections		
Microcontroller Pin	Sensor Pin	Function
PH0	IN1	Motor coil control
PH1	IN2	Motor coil control
PH2	IN3	Motor coil control
PH3	IN4	Motor coil control
5V	+	Motor supply voltage
GND	-	Common ground

Detailed Description

Distance Measurement

ToF Sensor Operation

The distance measurement in this project is done by the VL53L1X Time-of-Flight sensor. It uses an infrared laser to emit pulses and measure the time it takes for the light to return to the sensor after reflecting off an object. It initially computes the distance based on the time-of-flight principle:

$$Measured\ Distance = \frac{Photon\ travel\ time * Speed\ of\ light}{2}$$

This formula, as discussed in the studios, achieves a direct distance reading by multiplying time-of-flight by the speed of light, and dividing by two to account for the round trip. The sensor has different distance modes, for the sake of our system its configured in long-distance mode to allow for measurements up to approximately 4m in range (obtained in millimeters so around 4000). The microcontroller communicates with the VL53L1X via the I^2C bus (using port B pins PB2 and PB3). Upon initialization, the micro preforms a sensor set up routine by resetting and booting the sensor and calling its configuration functions to load the required parameters.

Once the initialization process is complete, the sensor is ready to take distance measurements. This process begins with the micro issuing a start ranging command, them waits for the sensor to indicate the data is ready using a simple polling loop. When the data is ready, the micro retrieves the distance in millimetres by calling the get distance function which reads the sensors internal distance register over I^2C before clearing the sensors interrupt flag and preparing for the next measurement.

During a scan, the system provides visual confirmation when each measurement is taken using LED D1. This measurement LED is flashed every time a distance reading is captured from the sensor to give the user real-time indications that the device is collecting distance data at each scan location. If the sensor fails to return data (if an object is out of range or not detected), the system logs an invalid reading and continues scanning. This is to account for when the scan location has a perpendicular hallway which goes too far for the sensor's measurement capabilities, ensuring the sensor doesn't get stuck in this location. Because of these features, the VL53L1X sensor can deliver reliable distance data at each position necessary.

Scanning Mechanism and Data Acquisition

To map a 2D vertical slice of the environment, the system uses a stepper motor to rotate the ToF sensor through precise increments. The 28BYJ-48 stepper motor is connected to the microcontroller through a ULN2003 driver which allows the micro to energize the motor coils in the desired sequence. For this project, the motor takes 32 11.25-degree steps to scan the surrounding environment evenly and accurately. The motor is mounted so that its axis of rotation is horizontal, meaning when it rotates the full 360 degrees, the sensor can sweep the vertical (Y-Z) plane.

The microcontroller controls the stepper motor by outputting a sequence of signals on PH0-3. Each step of the motor corresponds to a small change in angle of the sensor. As previously mentioned, 11.25-degree increments were used for the measurement routing to collect accurate data in a realistic amount of time. A loop is used to rotate the motor in this fixed increment, and then it is held still (as recommended during project deliverable 1) to collect the distance measurement at that angle. The cycle goes: step → settle → measure → record → repeat, until a full 360-degree sweep is complete (32 measurements). In addition to observing the motor move, and additional statues LED (D3) is toggled to indicate the motor moving to a new position.

After one vertical scan, all the distance measurements are stored in an array in the microcontroller's memory, along with the angle they were measured at. The micro has plenty or RAM to store this data locally, which ensures that the timing of the scan is controlled by the micro and is not subject to communication delays. After the complete sweep, the micro prepares

the data for transmission to the PC. For the sake of this project, I implemented data to be transferred in a simple ASCII comma-separated format, where each measurement has two numbers (the angle in degrees and the distance in millimeters). I chose this data format because it is easy to debug issues with the data using the terminal which helps for debugging. The UART TX line (PA1) sends the data at 115200 bps, which is the maximum standard baud rate supported on most systems, providing a good balance between speed and stability based on the data sizes in this project. When UART transmission begins, LED D2 flashes to give visual confirmation that the data is being communicated. The entire set of data from one scan easily transmits in under a second at this baud rate. This is the end of the data acquisition process, so the PC has received a complete set of angle and distance measurements representing the vertical plane of the systems current location.

Integrating Multiple Scans (X-Axis Displacement)

We have discussed the distance measurement process for a 2D plane, but to build a full 3D map, this device needs to be physically displaced along the X-axis (perpendicular to the scanning plane), so the system can take vertical scans at different X positions. In this project, this was done by moving the entire device manually in fixed, known distances between scans. If I were to redo this project on my own time, I would explore ways to automate the movements in the X direction to make these measurements more precise and the entire system more autonomous. But, for the sake of this project, I measured 22 sets of data in 300mm increments (measured and marked with a ruler) to scan my entire assigned location. The distance between scans and number of scans are inputted into the Python program so that it correctly positions each slice of data in the combined coordinate system.

The Python script assigns the appropriate X coordinate to each set of data to accommodate for these multiple scan locations. For instance, my scans were all 300mm apart, so an X value of +300mm is added to each new dataset, increasing by 300mm each time. This essentially stacks the vertical slices side-by-side. An important part of the measurement process is ensuring the device is oriented the same for each scan when it’s moved along the X axis, so that the data from all scans have the same Y-Z reference frame. This was done by taping down the device, so it remained level and using the floor tiles to ensure the device is facing the same direction after each movement.

The result of integrating multiple scans is a set of points in a 3D space. The resolution along the X direction is determined by the distance between measurements and is limited by precision and accuracy. In my example, I had one slice every 300mm, which made measurement a tedious process as physically moving the device by such small increments took a significant amount of time, but it was worth it because the result was an impressively accurate reconstruction of the scanning environment.

Visualization

Data Processing and Converting Coordinates

The first step of the visualization process is converting the data into Cartesian coordinates so it can be plotted. Once the data set is transmitted to the PC via UART, a Python program processes these measurements. This process involves converting the polar coordinates into Cartesian (x, y, z) coordinates. The coordinate system is defined as follows: the origin is the devices starting position, the Z-axis is the vertical component of the scanned plane, the Y-axis is the horizontal component of the scanned plane, and the X-axis is perpendicular to this scanned vertical plane. With this layout, each vertical scan occurs in a plane where X is constant, and as the sensor rotates in the Y-Z plane, the angle θ changes. θ is defined as zero when the sensor is pointed straight up (its starting position). Using Pythons math library, I can calculate the points in the YZ plane using basic trigonometry with the following formulas (theta is the scan angle, distance is the measured distance):

$$Y\ Coordinate\ (depth) = distance * \cos(\theta)$$

$$Z\ Coordinate\ (height) = distance * \sin(\theta)$$

These equations assume θ = 0 at the vertical position. We can also keep track of the x component of the scan through which depth scan the value is associated with, and the users inputted distance between each slice. With these formulas, we can get every distance in Cartesian (x, y, z) coordinates. For example, say the user wanted to take 3 depth scans that are 300mm apart. If the 3rd measurement from the 3rd depth scan was found to be 1500mm, we can convert to Cartesian coordinates:

$$x = 300mm\ (depth\ scan\ \# - 1) = 300mm\ (3 - 1) = 600mm$$

$$y = 1500 * \cos(3 * 11.25°) = 1500 * \cos(33.75°) \cong 1247.20mm$$

$$z = 1500 * \sin(3 * 11.25°) = 1500 * \sin(33.75°) = 833.36mm$$

The Python program uses the NumPy library to preform these calculations on our arrays of data. The angles are converted to radians for use with NumPy's trig functions, the vectorized sin and cos operations are applied to the array obtaining the Y and Z coordinates. The output for this step is a set of (x, y, z) coordinates in cartesian form.

3D Point Cloud Rendering

For visualization on the reconstructed scan location, we utilize the Open3D library in Python to create a 3D point cloud of the collected data. After calculating the coordinates of each point from all the scans, the Python program compiles them into a single list of 3D points (each in Cartesian form). Open3D is then used to generate a point cloud from this list. This is a set of coloured points in 3D space, which can be observed from any angle in an interactive window. Each point appears at its corresponding location in 3D space, coming together to form recreate the location of the scan. In the Open3D window, the user can rotate, pan and zoom the view to observe the results from different angles to ensure the systems accuracy.

To improve this visual confirmation, the program connects adjacent points in each vertical scan with line segments and connects points of similar angle or relative position between the different slices. This effectively creates a mesh grid of lines to help the user see continuous surfaces, and where certain objects are in the scan. For example, the part of the scan where there is a flat wall

will appear as such, and locations where this differs such as doorways, can be easily visualized because they will extrude differently. Open3D allows adding LineSet geometries for this purpose.

This visualization process is very fast as it generates and displays the point cloud in a fraction of a second after receiving the data. The real time processing is handled easily by the PC to avoid slowing down the microcontroller with floating-point math. The tasks being distributed between the embedded device (data capture) and the PC (data processing and visualization) effectively uses the strengths of each component and promotes efficiency.

Application Example, Instructions, and Expected Output

Application Example

This embedded 3D spatial mapping system is perfect for mapping and representing a indoor location such as hallways or small rooms. The current systems accuracy and scale is limited by the motor and ToF sensor, which is why it can only map small rooms; however, this can still have numerous important applications. This system can represent the dimensions of a room to a relatively high degree of accuracy, which could be useful in construction or décor applications, when trying to determine a room layout. This is particularly useful for rooms with unique dimensions or designs that may be difficult to measure conventionally. The systems accuracy can also be very easily improved by altering the code to measure at a higher resolution, and by displacing the system along the x-axis in smaller increments. Adding more distance measurements will significantly increase the time it takes to map an area, but in scenarios where accuracy is of the essence, this is a good option to have.

Using this project as reference and using the knowledge we've gained in the 2DX3 course, this system could be scaled up significantly by using a more powerful motor and lidar sensor to greatly increase the systems possible applications. With these improvements, the system could have many important applications in the real world. This could include improving autonomous cars by detecting obstacles in real time, improving construction sites, urban development, and even have environmental benefits from mapping forests or tracking natural disasters. This shows how the system already has applications in the real world, and with a little fine tuning (and a higher budget), its possibilities are endless.

User Instructions for Setup and Operation

This device is designed to be a standalone scanner connected o a PC for visualization. Below is a step-by-step guide for setting up and using the embedded 3D spatial mapping system.

1. Hardware Setup: Assemble the system by mounting the VL53L1X ToF sensor on the stepper motor so it faces upwards. Follow the instructions shown in Table 1. Ensure the sensor is connected to the microcontrollers I^2C pins (PB2, PB3) and powered (3.3V and GND). Connect the 28BYJ-48 stepper motors ULN2003 driver board to the

microcontroller, wiring the driver inputs to Port H, and connecting it to 5V and GND. Verify that PJ1 is easily accessible, and note that LEDs D1, D2, and D3 are visible to indicate status. Finally, connect the microcontroller to the PC using a USB cable.

2. Software Setup: Ensure that on the PC, Python is installed with the required libraries (PySerial, NumPy and Open3D) and Keil uVision 5 is installed.
3. Open the Keil code and download it onto the microcontroller (translate → build → load), ensuring no errors pop up.
4. Open the python code in IDLE (or PowerShell) and change the UART port so it corresponds with PC being used (Device Manager → Ports). Was COM4 on my PC but will differ for different devices.
5. Ensure system is in correct starting location with the desired orientation.
6. Run the python code and await instructions in the shell.
7. When prompted, press Enter to start the program if you are ready.
8. When prompted, enter the number of depth scans you would like to take (> 1).
9. When prompted, enter the distance between each depth scan (in mm).
10. Press the reset button on the microcontroller when prompted to do so. This should initialize the sensor. Confirmation will be given in the shell if successful.
11. When prompted, press the on-board button PJ1. This triggers an interrupt to begin the scanning process. Make sure you are out of the sensors way when pressing the button.
12. The motor will begin to rotate, and the sensor will be making distance measurements. Observe the indicator LEDs. LED D1 should flash 32 times in the full 360° scan, once for each 11.25° step. LED D3 will also be flashing to indicate motor movements. LED D2 should flash once to indicate UART data transmission.
13. Once 32 distance measurements are taken, wait for the sensor to return to its home position.
14. Move the entire system along the x-axis by the amount you specified earlier (300mm for my test).
15. When prompted, press Enter again in the Python IDLE shell.
16. Press PJ1 (on-board interrupt button) to run another scan.
17. Repeat steps 11 through 16 for the number of times you entered previously (the number of depth measurements).
18. When the last scan is complete, press Enter one final time. This will prompt a message indicating that the mapping is complete and will appear on screen in a pop-up window.
19. Open the Open3D window to see the point cloud of the scanned location. This is an interactive window.
20. When done observing the point cloud, close out of this tab by pressing X in the top right corner of the Open3D window.
21. The 3D map of the scan location with line connections will automatically open on the screen. This is also an interactive window.
22. Complete.

By following these instructions, and user can set up this device and operate it to take a 3D scan of their environment. The system was designed to be very user friendly for demonstration purposes.

Expected Output

The expected output can be observed below. My assigned scan location was location D (400520473). This was on the second floor of the Information Technology Building and is shown from 2 different angles in Figure 3 and Figure 5. Figures 4 and 6 show the Open3D output from this scan location. This consisted of 22 depth scans 300mm apart, each with 32 distance measurements. These figures confirm the system is functioning as intended as the mapped output matches the shape of the scan location perfectly. Smaller details such as the washroom entrance and water fountain (which are both indented into the wall) can even be observed.

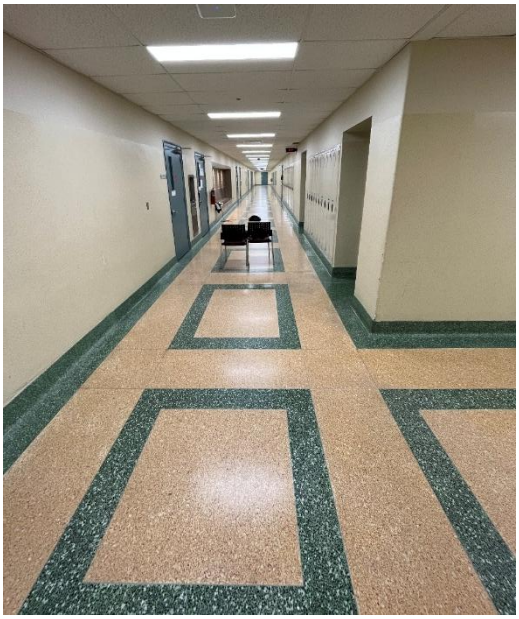

Figure 3: Scan Location - Angle 1

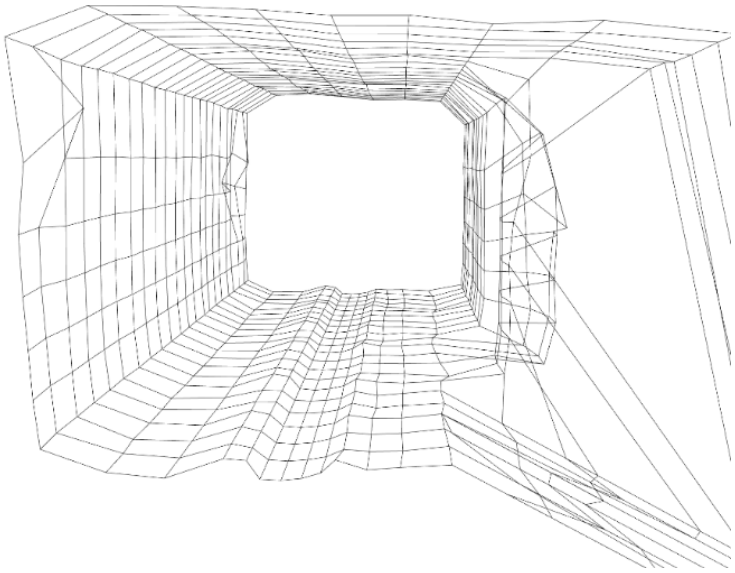

Figure 4: 3D Mapping of Scan Location - Angle 1

Figure 5: Scan Location - Angle 2

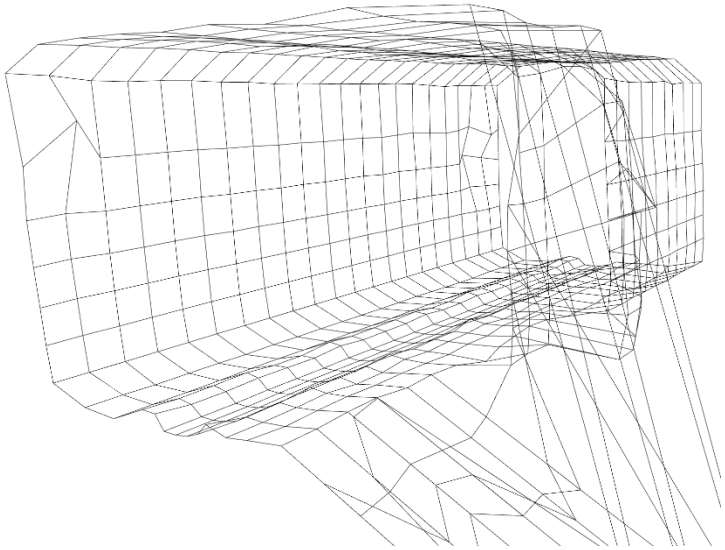

Figure 6: 3D Mapping of Scan Location - Angle 2

In addition to the reference images and 3D mapped output, I have included the point cloud from Open3D as seen in Figure 7. Each point shown in this cloud represent one distance measurement. This gives a good visual representation of the amount of data collected and plotted by the system, while also demonstrating the importance of the connecting lines between points to better visualize the mapped area.

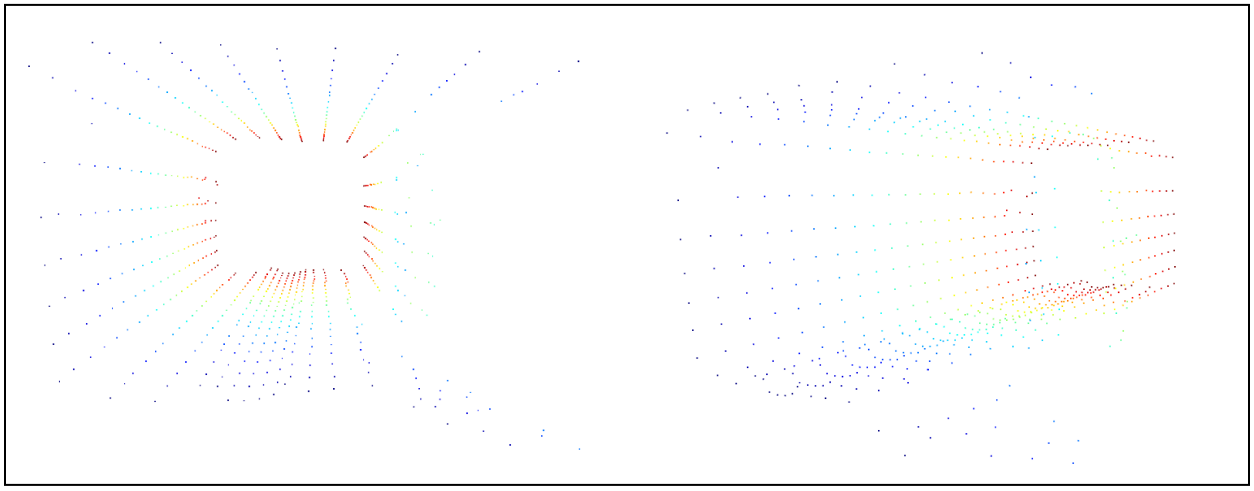

Figure 7: Point Clouds of the Mapped Area - Angle 1 (Left) and Angle 2 (Right)

It is essential for the user to understand the coordinate system used for the 3D data in order to fully understand the point cloud model and the 3D model. The device uses a Cartesian coordinate system defined as follows:

- X-axis: Horizontal axis that the device is manually moved along for multiple slices. This is perpendicular to the Y-Z scanned plane. X=0 corresponds to the position of the first scan, and each additional scan at a new location has a constant X coordinate equal to the displacement from the origin (which is 300mm each slice in my example). This is the length component of the hallway.
- Y-axis: Horizontal axis perpendicular to the X-axis. This is part of the vertical plane which is scanned and measured by the ToF sensor. This is the width component of the hallway.
- Z-axis: Vertical component of the vertical plane scanned by the ToF sensor. This is the height component of the hallway.

I chose this coordinate system as it matched the terminology used in the 2DX3 course. With this setup, the YZ plane is scanned by the sensor on the stepper motor. The X coordinate is constant for each scan and is manually moved down the hallway at a defined distance each time. These multiple scans are combined, creating the 3D model of the hallway.

Limitations

The limitations of this system are tied to hardware and software limitations, such as:

1. Floating-Point and Trigonometric Limitations

The MSP432E401Y microcontroller includes a single-precision floating-point unit (FPU), which allows it to handle and preform mathematical operations with 32-bit floating points. This allows for basic mathematical operations involving non-integer values, but it presents a limitation in numerical precision. Rounding occurs and only about 7 significant digits are retained. This becomes significant when dealing with trig functions which often produce irrational results. Since these functions return approximate floating-point values, rounding errors can add up in the embedded system. Additionally, trig functions are computationally expensive on the microcontroller and may lead to bottlenecks if they are done and used in real time. To avoid this issue, trigonometric computations to convert coordinate systems were done using the PC for more efficient calculations with better efficiency.

2. Maximum Quantization Error for ToF Module

The VL53L1X Time-of-Flight sensor provides distance measurements with a resolution of 1mm as defined in its datasheet. The maximum quantization error is the largest amount of error that can be caused because of the quantization in the conversion process from analog to digital. The VL53L1X sensor has a maximum distance measurement of 4m (4000mm), and the microcontroller has a 12-bit analog to digital converter. Using these values, we can calculate the maximum quantization error to be:

$$Q = \frac{4000mm}{2^{12}} \cong 0.977mm$$

3. Maximum Standard Serial Communication Rate with PC and Verification

The maximum standard UART communication rate that is reliable supported by most PCs is 115200 bits per second (pbs). This is recognized as the highest standard baud rate without requiring custom drivers or settings and is very compatible with USB to UART interfaces. This value can be verified for this system specifically. Communication between the microcontroller and PC was configured at 1115200 pbs, using UART0. This was verified by checking the Windows Device Manager, where the COM port settings for the XDS110 Class Application / User UART showed the configured baud rate.

4. Communication Method and Speed Between Microcontroller and ToF Sensor

Communication between the MSP432E401Y microcontroller and the VL53L1X ToF sensor was implemented using the Inter-Integrated Circuit protocol, more commonly known as I^2C. The microcontroller was configured as the I^2C leader, and the sensor as the follower at address 0x29. Communication between these two devices was at a rate of 100kHz, which is the default I^2C mode.

5. Primary System Speed Limitations and Testing Methodologies

The primary limitations that contribute to the speed of the overall system were the VL53L1X ToF sensors timing budget and the stepper motors mechanical step delay. The sensor has a internal timing budget of 33ms in the long-distance mode, which means each measurement takes

at least that long to complete. Reading the sensor data any faster than this would result in invalid or inaccurate data. Additionally, it was found in studio that the 28BYJ-48 stepper motor requires a minimum pulse interval of 175us per coil excitation to work properly and reliably. This limits the speed the motor can rotate during scanning, slowing down the system as a whole. These limitations were tested by incrementally reducing the motor delay and the sensor polling interval until the system began to miss steps or measure distance inaccurately. These tests confirmed the system cannot go any faster than the ToF sensor range cycle, and the motor has a maximum speed it can rotate before stalling out. Therefore, both of these limitations combined contribute to the minimum time required per scan point in order to collect accurate data.

Circuit Schematic

Figure 8 shows the complete circuit schematic for the 3D spatial mapping system. It includes all hardware connections between the MSP432E401Y microcontroller, the VL53L1X Time-of-Flight sensor, and the 28BYJ-48 stepper motors ULN2003 driver.

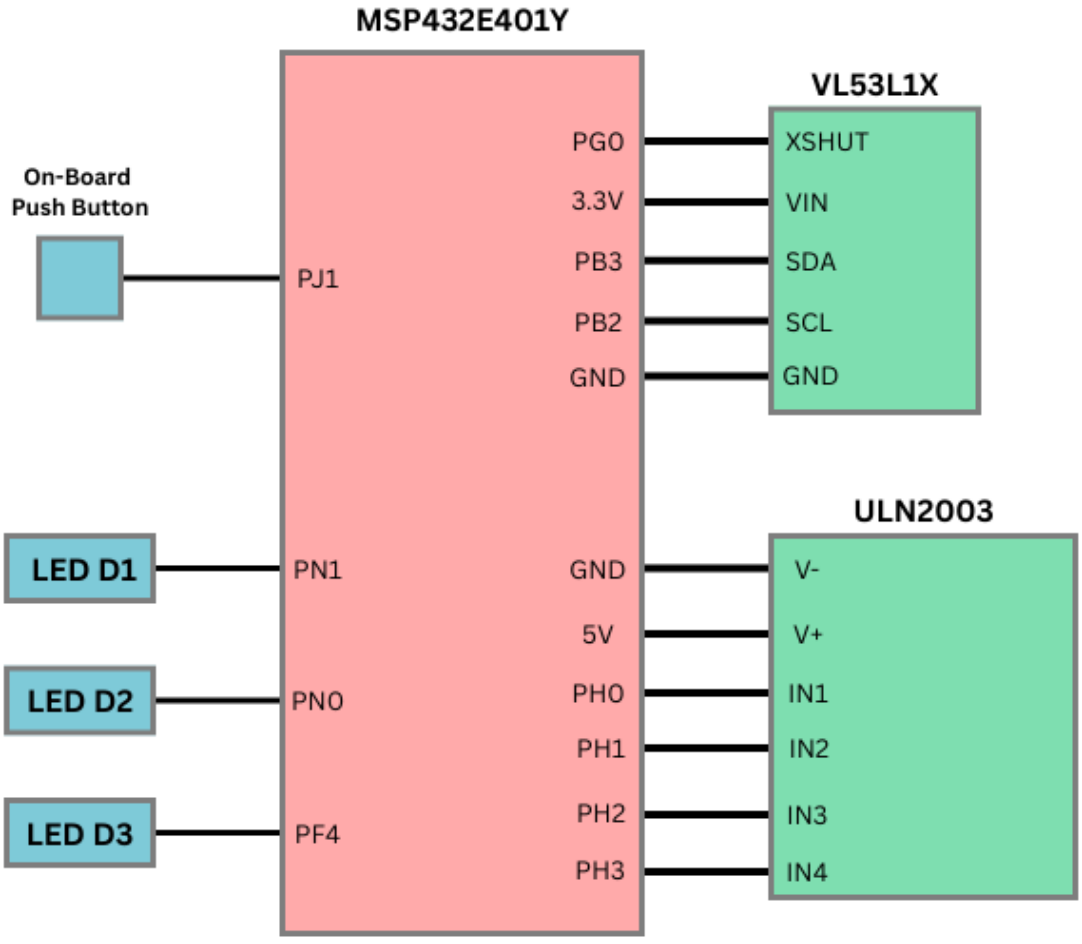

Figure 8: Circuit Schematic for 3D Spatial Mapping System

Programming Logic Flowchart

Figure 9 shows the programming logic flow chart. This includes both the embedded firmware on the microcontroller and the PC software.

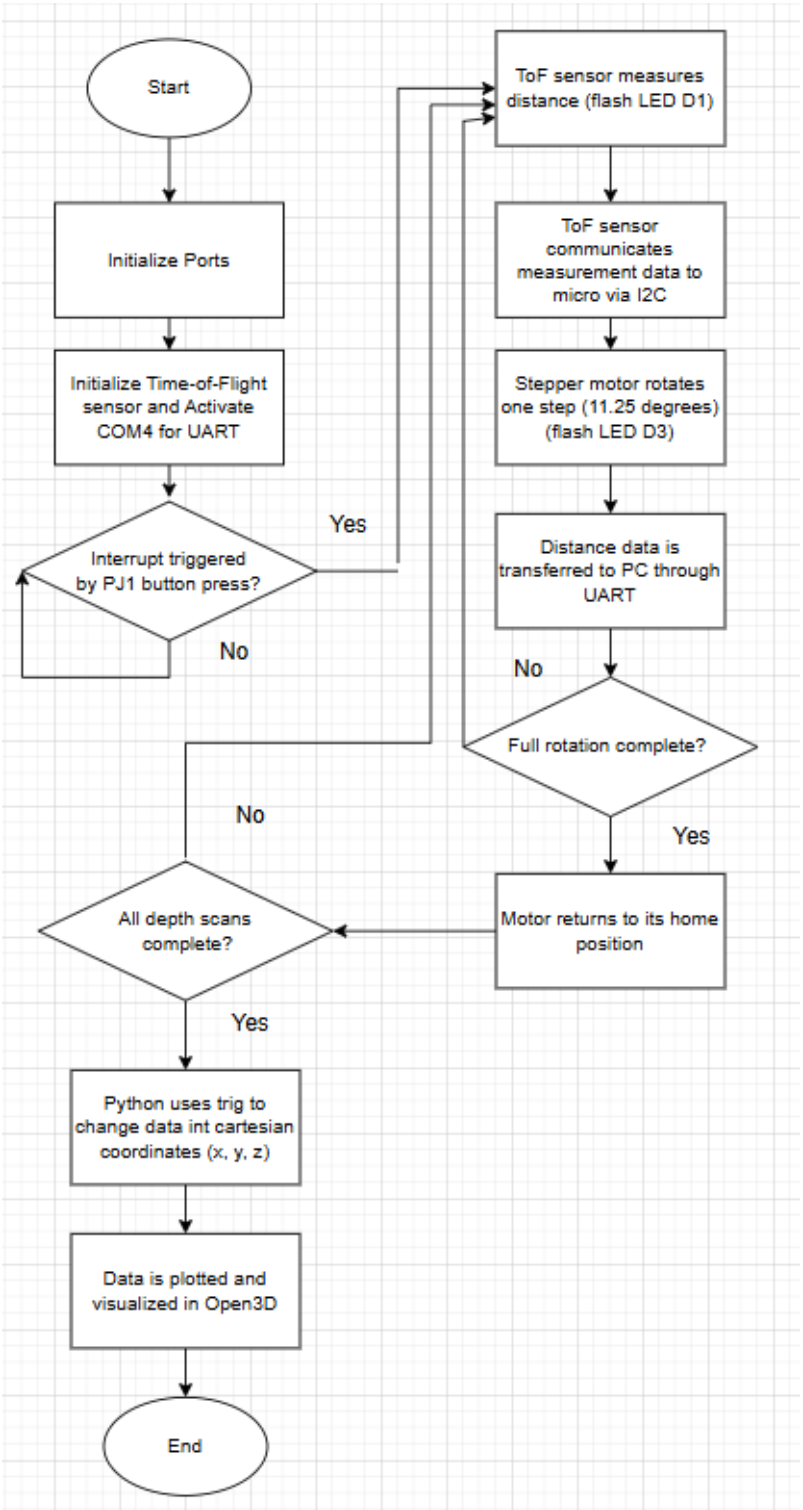

Figure 9: Programming Logic Flowchart

References

[1] “MSP432E401Y SimpleLink™ Ethernet Microcontroller.” https://www.ti.com/lit/ds/symlink/msp432e401y.pdf?ts=1744121287243&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FMSP432E401Y%253Futm_source%253Dgoogle%2526utm_medium%253Dcpc%2526utm_campaign%253Depd-null-null-gpn_en_rsa-cpc-pf-google-ww_en_cons%2526utm_content%253Dmsp432e401y%2526ds_k%253D%257B_dssearchterm%257D%2526DCM%253Dyes%2526gad_source%253D1%2526gclid%253DCjwKCAjwktO_BhBrEiwAV70jXvm7UHo1GBWtSUGXDWi-EIpTZqFPN102G6I_8Yxy4t3GRYM06CKl1hoCwqUQAvD_BwE%2526gclsrc%253Daw.ds

[2] “This is information on a product in full production. VL53L1X A new generation, long distance ranging Time-of-Flight sensor based on ST’s FlightSense™ technology Datasheet - production data Features,” 2022. Available: https://www.st.com/resource/en/datasheet/vl53l1x.pdf

[3] “28BYJ-48 -5V Stepper Motor.” Accessed: Apr. 09, 2025. [Online]. Available: https://www.mouser.com/datasheet/2/758/stepd-01-data-sheet-1143075.pdf?srsltid=AfmBOorwip-31rHivylJftzK6sKqlLDV9mOawS94UaAIELHeGiT4EsWX