COMP ENG 2DX3 - Microprocessor Systems Project

Final Project Report

Haseeb Shaikh (shaikh22), 400521659

April 9th, 2025

Device Overview

Features

The system is powered by a Texas Instrument MSP432E401Y Microcontroller programmed using C language code in the Keil uVision5 software platform. This system allows for a 360-degree spatial scanning through the integration of a comprehensive software package and using VL53L1X Time-of-Flight (ToF) sensor that is mounted to a 28BYJ48 stepper motor with a ULN2003 driver board connected to it.

Key features of this microcontroller include:

- 32-bit ARM Cortex M4 processor. 1 MB of Flash Memory and 256 KB of SRAM.
- 12 MHz system assigned bus speed (Max bus speed of 120 MHz)
- 15 ports (A-Q with no port I or port O) and each port has 8 pins
- 4 onboard push-buttons
- 4 built-in User LEDs
- Multiple GPIO pins to interface with the time-of-flight sensor and stepper motor driver board
- Cost: $72.54

The system uses a VL53L1X Time-of-Flight Sensor to take distance measurements in real time, which offers:

- Distance readings in millimeters, up to a maximum of 4000 mm (4 meters) and minimum reading of 4 cm in long distance mode
- 2.6V to 5.5V operating voltage range. Typical ranging frequency of 50 Hz
- Dedicated I^2C data line (SDA) and clock line (SCL) pin connection that returns distance measurement in 16-bit format with units of mm
- Cost: $24.95

To allow full 360-degree coverage within a single plane, the sensor is mounted on a 28BYJ-48 stepper motor. The motor is controlled via a ULN2003 driver board, with the following specs:

- Contains two center-tapped coils to drive an internal gear, which is connected to an outer gear
- Gear ratio of inner rotor to outer motor shaft is 64:1
- Operates in full-step mode with 2048 steps of the internal gear per one full rotation of the output shaft
- Controlled through ULN2003 driver board with 4 LEDs that visually display the different stepping phases of the motor. It is connected by 6 wires (4 for the coils, 1 for power, and 1 for ground)
- ULN2003 driver board requires a 5V power supply
- Cost: $5.72 for both ULN2003 driver board and 28BYJ-48 stepper motor

The system components interact or communicate using the following serial communication protocols:

- I^2C between the VL53L1X ToF sensor and the MSP432E401Y microcontroller

- UART between the MSP432E401Y microcontroller and a PC Communication with a USB connection and is set at a baud rate of 115200 bps (UART 0 configuration is set up through the horizontal onboard jumper positioning).
- It also uses Python for serial communication to read data from the microcontroller, allowing the collected distance measurements to be transferred efficiently to the PC. For visualization and 3D Mapping, the Open3D library is utilized.

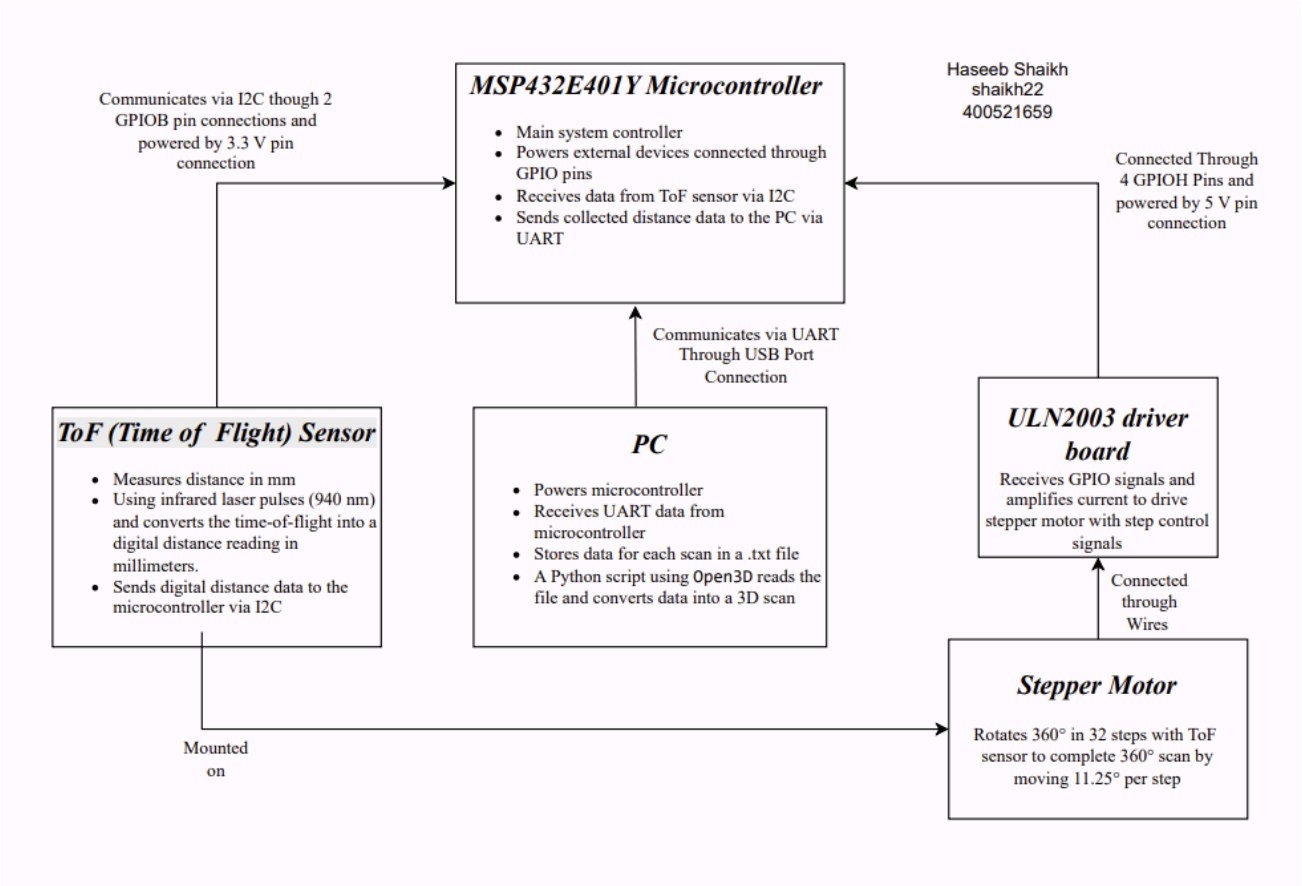
General Description

Commercial LIDAR systems are often bulky and expensive, making them impractical for many simple indoor applications. This project focuses on creating a smaller, more affordable alternative that's ideal for indoor exploration and navigation. In many engineering scenarios, data collection is essential, but available systems can be overkill either too costly, overly complex, or not quite suited to the task. The goal is to measure real-world data accurately without unnecessary expense or complication.

The spatial scanning system consists of a PC, a microcontroller, a stepper motor with its driver board, and a ToF (Time-of-Flight) sensor. These components are interconnected through GPIO and use I^2C and UART for data transfer. Assigned bus speed was 12 MHz.
Here's how the system works: The VL53L1X sensor is mounted onto a stepper motor through a 3D printed mounter which ensures it securely rotates the ToF sensor , allowing it to acquire distance measurements as it rotates. Each full 360° scan of a vertical plane (the y-z plane) consists of 32 measurements, each taken at 11.25° increments. For the 3D spatial map, the setup is moved manually along the x-axis at fixed intervals. The ToF sensor functions by emitting laser light pulses from the emitter, which travels and hits an object or surface and reflects back to the detector. By measuring the time it takes for the light to travel to the object and back, the system can calculate the distance using the formula: Measured Distance = (Time / 2) × Speed of Light.

These measurements are transmitted via I^2C from the ToF sensor to the microcontroller, then forwarded via UART to the PC, where they're saved as a file. Python is used to convert the distance measurements into y and z coordinates using trigonometric functions. The y and z coordinates are computed by applying sine and cosine functions to the angular increments of each step, and the x coordinate is incremented manually in the code. To build a 3D model, the system performs multiple vertical plane scans, shifting along the x-axis for each new layer. These layers are combined into a 3D point cloud using Python and Open3D library, allowing for full 3D mapping and visualization.

Block Diagram (Data flow graph)



Device Characteristics Table

MSP432E401Y Microcontroller	
Bus Clock Speed	12 MHz
Push Button Input	PJ1 onboard push button to start scanning operation
Measurement Status LED	Onboard LED 4 (PF0)
UART Transmission LED	Onboard LED 1 (PN0)
Additional Status LED	Onboard LED 2 (PN1): Turns on when motor rotates CCW to return to home position after scanning operation
UART Baud Rate	115200 bps – used for serial data transmission to PC (Serial port: COM4)

VL53L1X Time-of-Flight (ToF) Sensor	
Vin	3.3 V pin of microcontroller
GND (Ground)	GND pin on microcontroller
SDA	PB3 pin on microcontroller (I2C data, 0x29 address)
SCL	PB2 pin on microcontroller (I2C clock)
Measurement	Max Range: 4 m or 4000 mm (long distance mode) Units: mm (16-bit output) Angular Resolution: 11.25° (32 steps per rotation) Displacement: User Defined (e.g. 100 mm)

Stepper Motor Driver: ULN2003 Board	
Vin	3.3 V pin of microcontroller
GND (Ground)	GND pin on microcontroller
IN1-IN4	PH0-PH3 pins on microcontroller

Software	
Programming MSP432E401Y microcontroller	Keil uVision5 (C Programming language)
Python Script	3.8 (Compatible with 3.6–3.9)
Serial Communication Module (Python)	PySerial
3D Visualization Module (Python)	Open3D

Detailed Description

Distance Measurement

The VL53L1X Time-of-Flight sensor which is the core component responsible for collecting distance data is mounted to a 28BYJ-48 stepper motor, which performs a full 360° sweep in 32 steps, each separated by 11.25°. Once the ToF sensor is booted, the system initializes the sensor with its default settings and enables ranging, allowing the ToF sensor to begin measuring distances.

When the onboard pushbutton (PJ1) is pressed, an interrupt routine is triggered and the microcontroller begins the scanning process, rotating the motor clockwise step-by-step (every 16 steps out of 512 steps per loop or 64 steps out of 2048 steps). The microcontroller collects a distance measurement at each angle in 11.25° increments (32 total measurements) via I2C serial communication to obtain distance measurements from the ToF sensor which is stored in a 32 element array and the onboard LED 4 flashes after every measurement collected for indication of every successful distance measurement. After the scan is complete, the motor rotates back counter clockwise (CCW) and while it rotates CCW, LED 1 turns on to indicate that the motor is returning to its home position to prevent wire tangling. A manually incremented **x** variable, along with the newly acquired distance measurements, are stored in their respective 32-element arrays for later transmission. The **x** variable is incremented each time the interrupt is triggered and a 360 scan is completed, ensuring that the distance is accurately recorded for each slice of the scanned room. Additionally, an "indexNum" variable is incremented every 11.25 degrees and reset at the end of the scanning process. This ensures that the measurements are properly indexed and stored in the correct slots within the arrays.

After 360 scan is successfully completed, the program checks for measurement readiness, reads the distance, and when a user has entered the enter key on the PC, it transmits data to the PC using UART serial communication at 115200 bps and at the onboard LED 2 flashes at every data transmission to indicate a successful data transmission. When the onboard pushbutton (PJ1) is pressed again, the same scanning process takes place again once the data is transmitted to PC via UART and if the number of scans are still remaining. Once all scans are completed, the data is then visualized on a 3D plot using Python script and Open3D library.

The ToF sensor measures distance using Light Detection and Ranging (aka LIDAR) by emitting 940 nm infrared light from the emitter. It calculates the time it takes for the light to hit an object (usually a wall, floor or ceiling) and reflect back to the detector sensor and based on the known speed of light, the core formula used is:
Measured Distance = (Time / 2) × Speed of Light as shown below in Figure 1. For example,if a ToF sensor measures a time of 10 microseconds for the infrared light to travel to the object and back. To calculate the distance, we use the formula and get 1.5 meters so the object is located 1.5 meters from the sensor.

$$Distance = \frac{10\,x10^{-6}}{2} x\,(3.00x10^{8}) = 1500\,mm = 1.5\,m$$

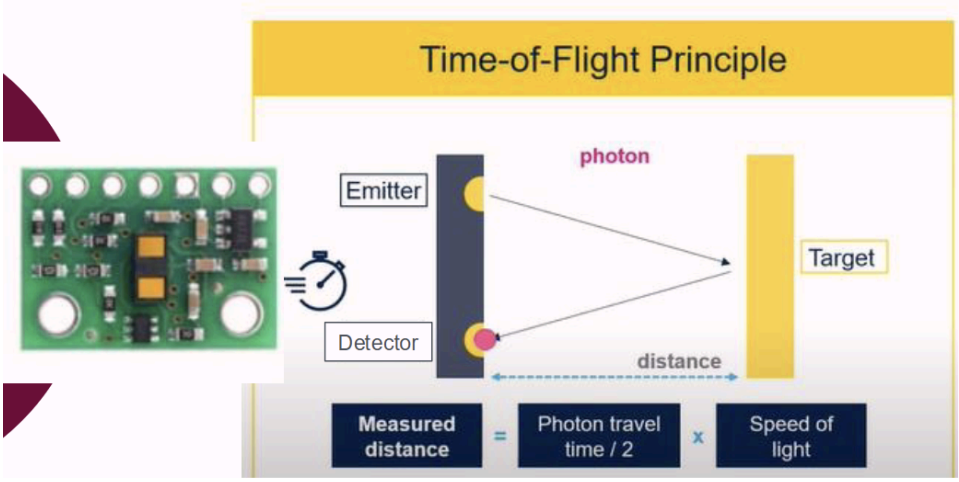


Figure 1: Time of Flight Principle Concept With formula [2]

Internally, the sensor performs signal conditioning, noise filtering, and analog-to-digital conversion, outputting a 16-bit digital value in millimeters. This ADC process is outlined below in Figure 2. The ToF sensor supports different ranging modes such as short, medium, and long-distance modes. For this project, long-distance mode is used to enable detection of walls and objects up to 4 meters away. It also provides additional data like Range Status, Signal Rate, and Ambient Rate, though only the raw distance data is used in this application. Communication with the sensor is handled via the I2C protocol, using PB2 (SCL) and PB3 (SDA) on the MSP432E401Y. The sensor's default I2C address is 0x29. It's also important to note that the Texas Instruments MSP-EXP432E401Y microcontroller operates at an assigned bus speed of 12 MHz.

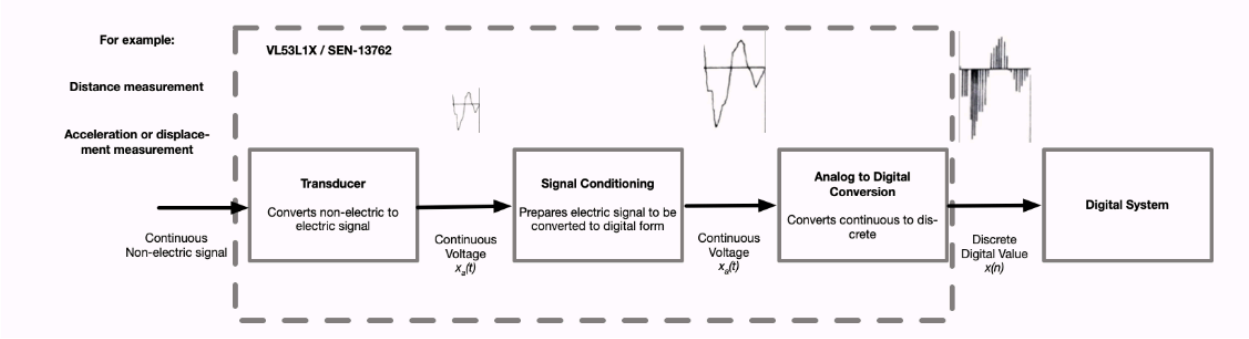


Figure 2. ADC Process [2]

Visualization

Once all 32 measurements are collected and the motor returns back to the initial position by rotation counter clockwise, the data is sent from the microcontroller to the PC via UART at 115200 bps baud rate through the COM4 serial port (USB connection) after the user presses the enter key on the computer to enable receiving of data. The visualization process starts on the PC

side, A Python script running on the computer reads the data using the pySerial library. Once the data is received, error checking is performed to ignore the first three data entries, as they simply confirm the data transmission. The remaining 32 data points are parsed into **x** and distance measurements and stored in their respective arrays. Each measurement is associated with an angle θ (starting at 0° and increasing by 11.25° for every measurement so 32 measurements to complete a 360° scan) and the angle, starting from 11.25 degrees, is manually updated for each iteration using a modulus operation. By using the distance, and angle θ, trigonometric functions convert polar coordinates into y-z cartesian coordinate pair by using the equations:

$y = \text{distance} \times \cos(\theta)$

$z = \text{distance} \times \sin(\theta)$

These (y, z) values are paired with a manually preassigned x-value that increases with each scan to indicate a horizontal displacement. After each scan, the user moves the sensor along the x-axis (contributing to x-displacement). These points are now written to a file in xyz format which is saved on the PC (Hard drive of computer). The file is then opened by the same Python script and reads all the data and prints it on the screen to see all the data allowing the system to use all 32 data points in xyz format into a 3D point cloud. The serial, numpy, Open3D, and math libraries were utilized to handle the data and create the plot. The resulting data uses the Open3D Python library to connect all the point cloud data points using line sets appropriately to get a 3D visualization mapping effectively outlining the shape of the scanned area.

Application Note

LiDAR scanning, a technology that uses laser light to measure distances, is widely used to generate 3D models of scanned areas. It plays a crucial role in various engineering fields, particularly in autonomous vehicles. These vehicles rely on LiDAR to create accurate 3D maps of their surroundings, enabling them to detect obstacles, track pedestrians, and perform tasks like lane detection and lane changing. Beyond vehicles, LiDAR is also used in drones and aerial mapping to monitor large agricultural areas. This spatial scanning system is a practical and affordable way to bring LiDAR-like functionality into hands-on learning, prototyping, and small-scale projects. It captures accurate distance measurements using laser-based time-of-flight sensing and builds 3D models of indoor environments. Because it's lightweight and simple to use, it's especially useful in places where bulky or expensive commercial systems just aren't practical like classrooms, hallway mapping, or early-stage robotics experiments.The system works well in GPS-limited areas like buildings and basements, and mimics how industrial LiDAR systems operate by scanning in layers. That means it can be used to simulate real-world applications like urban planning, basic terrain mapping, or even archaeological modeling in controlled spaces. With accessible tools like Python and Keil uVision combined with modular hardware, this system serves as a great introduction to embedded systems and remote sensing, allowing users to explore LiDAR applications in a cost-effective manner.

Instructions

Follow the following steps to produce a successful 3D scan.

1. Ensure all hardware connections match the circuit schematic in Figure 5 so all devices are connected as required.
2. Install Keil uVision, Python (3.6–3.9), and required libraries (pySerial, Open3D).
3. Download and open the Keil.project file 2DX3Project_haseeb of the type 'UVision5 Project'.
4. In the Keil software, navigate through the following Target 1 -> left click on Source Group 1 -> Add Existing Files to Group ->click on 2DX3Project_haseeb.c-> Add -> close Add Files window -> click on 2DX3Project_haseeb.c tab.
5. On line 261, replace the number incremented by x to your displacement value (default: 100 mm) : x += (value of x displacement)
6. Click the following in this order: a) Translate, b) Build, c) Download.
7. Press the RESET button on the microcontroller to flash the program onto the microcontroller.
8. Open the Python script 2DX3_python_haseeb.py.
9. In line 7, change COM4 to match the active UART port on your PC. To find the COM number, open Device Manager, expand the Ports menu, and look for the line labeled "XDS110 Class Application/User UART (COM #)". The number following "COM"

represents the COM port assigned to the device. To change the number of scans (Default: 3 Scans), change the value of the “NumScans” variable.

10. Place your hardware setup to the scanning position by considering the position of the ToF sensor mounted on the motor.
11. Run the Python file once you are ready to start the scanning process.
12. When prompted, press PJ1 on the microcontroller to start scanning. The motor will rotate in 11.25° increments and take one distance reading per step (32 total).Wait until the ToF Sensor stops rotating or wait until LED 1 turns off indicating the motor has now returned to home position
13. Press the enter key on the keyboard of the PC to start communication and verify that the transmitted data is displayed correctly in the output, listing each data point as expected.
14. Move the hardware setup (sensor) forward to the next scan location (default: 100 mm).
15. Repeat steps 12–14 until the total number of scans are completed. A point cloud of all readings will be displayed once all scans are completed.
16. Close the plot window when finished. A second window appears connecting all points using lines. This is the final 3D visualization mapping of the scanned area.

Expected Output

The expected result of the system is a 3D that closely visualizes the scanned environment. In this case, a real hallway was scanned, and the scan result was compared to physical observations of the same location. My assigned scan site was location J and with horizontal displacement set to 400 mm for each scan, and its results were compared directly to a real image of the hallway. Structurally, the scan had a few inconsistencies but captured the major architectural features well such as Benches on the left side showed up as slight bumps in the scan, while a vertical structure representing the pole on the right was also clearly present.

What stood out was the system’s ability to detect objects at distances that approached the 4-meter maximum range of the VL53L1X sensor. This occurred in the widest section of the hallway and was likely helped by clean line-of-sight. In a few areas, the infrared light emitted from the ToF Sensor seemed to pass through glass or bounce off window panels, leading to inaccuracies like extended depth readings though this actually helped demonstrate the sensor’s sensitivity and potential for environmental detail.

Even with these challenges, the scan nailed a few key points: the floor and ceiling lines were consistent and parallel, and the vertical stepping created a clean structure from end to end. One thing that made this hallway difficult to scan was its unconventional shape; it wasn't just a straight rectangular corridor, but had a middle section that opened up significantly, testing the limits of both the sensor’s range and the scanning software. Despite that, the final scan looked stable and captured the essence of the space. Overall, this demonstration wasn’t perfect, but it definitely worked. It's evident that the system can reliably recreate the physical environment with enough detail for spatial analysis and real-world navigation planning using accessible, low-cost hardware.

Figure 3: Picture of Assigned Hallway that was Scanned

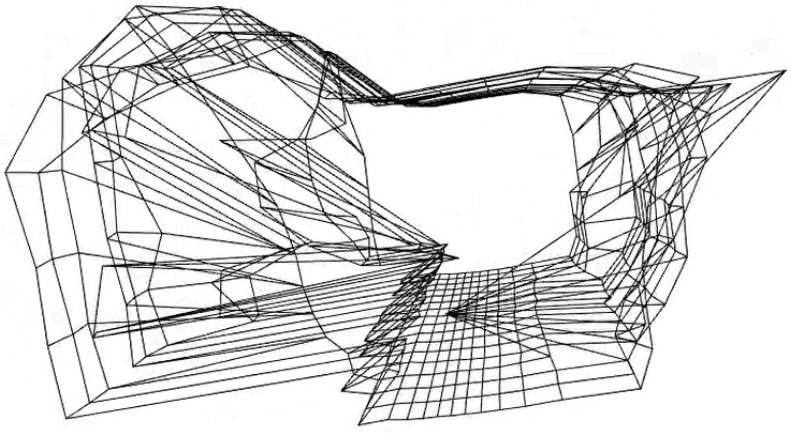


Figure 4: 3D Visualization of Scanned Hallway

Limitations

Floating Point and Trigonometric Calculations: The process of converting distance measurements into **y** and **z** coordinates for plotting requires trigonometric calculations. If these calculations were handled directly on the microcontroller, the long decimal portions of the resulting coordinates would present a challenge due to the microcontroller's limited precision. The Texas Instruments MSP-EXP432E401Y microcontroller has 32-bit single-precision registers within its Floating Point Unit (FPU), enabling floating-point operations. However, this precision is limited by the size of the registers, which can impact the accuracy of the trigonometric calculations for distance conversion. To overcome this limitation, the trigonometric operations were offloaded to Python running on the PC. While this solution simplified development and debugging, future versions of the system could benefit from moving these operations to the microcontroller for better performance and precision.

System Clock and Delays:

Two key factors contribute to the overall speed of the system. First, the stepper motor's rotation speed is influenced by the delay between each full-step movement. If the delay is too short, the motor will not properly rotate and may simply vibrate. Testing revealed that a 10 ms delay between steps was the minimum required to ensure smooth operation. If the delay was reduced further, the motor's movement became erratic. Second, the ToF sensor takes time to boot up and begin its ranging process. The sensor has a maximum frequency of 50 Hz, meaning it can only generate one reading every 20 milliseconds. This results in a delay between motor steps as the system waits for the sensor to complete each measurement. While the motor can rotate quickly, the scan speed is primarily limited by the time required for the sensor to gather and process data.Therefore, the speed of Stepper motor and ToF Sensor can be considered bottle necks of the system.

The system bus was configured to 12 MHz by setting MINT to its default value and adjusting N accordingly in PLL.c. This bus speed is sufficient for basic functionality and stability. However, it imposes some limitations when the system is handling multiple tasks simultaneously, such as UART communication and polling the ToF sensor. Increasing the bus speed to 20 MHz or higher could improve the system's overall responsiveness by reducing delays and enhancing data processing efficiency.

Quantization Error: The maximum quantization error can be determined by examining the resolution of the VL53L1X ToF sensor. Based on this, the maximum quantization error is found to be 1 mm. The VL53L1X sensor provides 16-bit distance measurements. Although this level of precision is more than sufficient for general 3D scanning tasks like hallway mapping or obstacle detection, it introduces a small degree of uncertainty. For applications that require highly accurate absolute distance measurements—such as precision engineering or scientific research—this error may become a limiting factor. However, for most general scanning applications, this level of error had minimal impact.

Communication Protocol Speeds:

The communication between the microcontroller and the ToF sensor is handled via the I2C protocol, while data is transmitted from the microcontroller to the PC using the UART protocol. The UART communication operates at a baud rate of 115200 BPS, which is sufficient for single data transfers. However, the PC's COM port has a maximum supported baud rate of 128,000 BPS, as confirmed through the Device Manager through the details of the XDS110 Class Application/User UART port. Although the current baud rate is stable, increasing it could improve data transfer speeds if larger data sets or higher scan resolutions are used in the future. The I2C protocol typically operates at a speed of 50 Hz, While this speed is sufficient for transferring single measurements, the limitations of communication speed could become a bottleneck as data volume increases or scan resolution improves.

Circuit Schematic

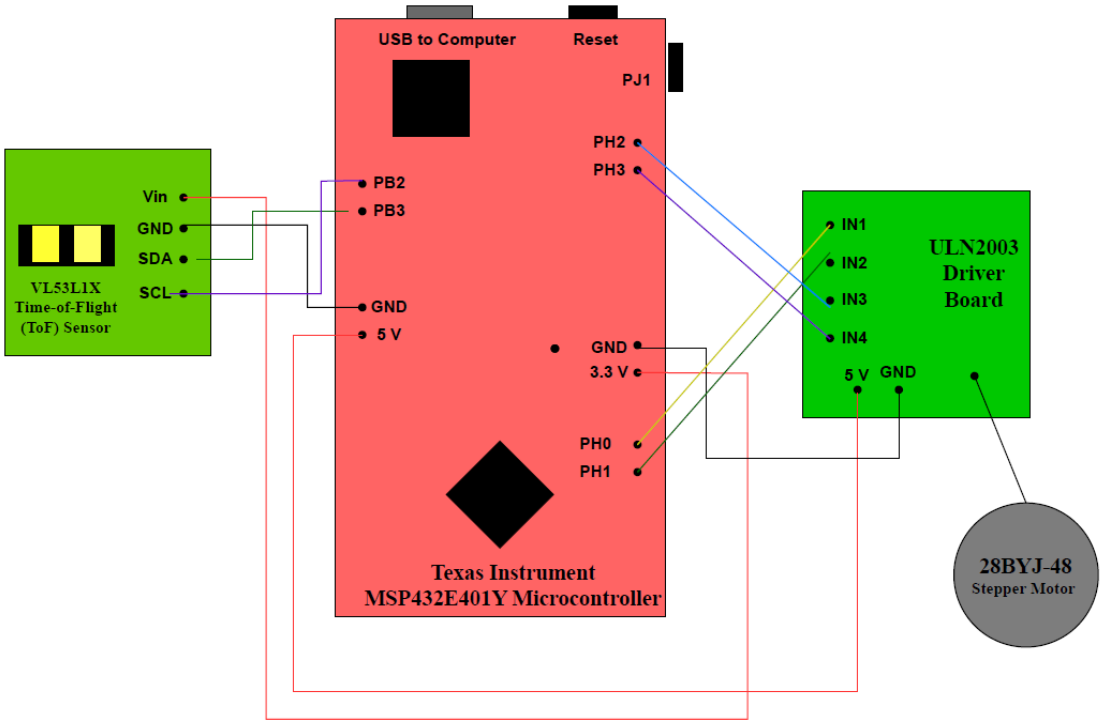


Figure 5: Circuit Schematic showing all Physical Connections

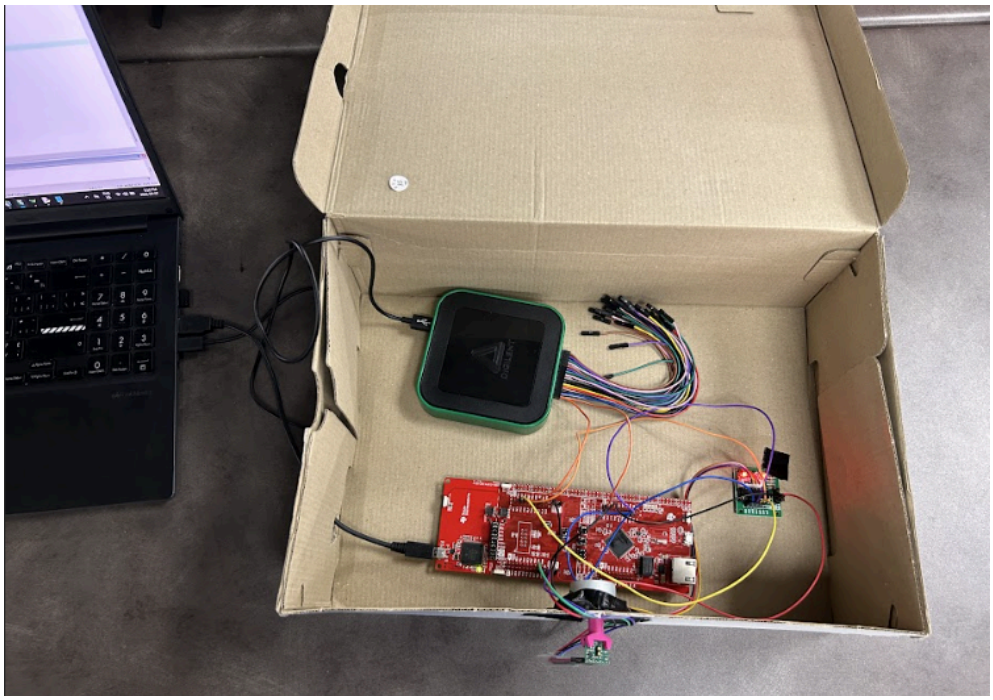


Figure 6: Hardware Setup (AD3 connected to Microcontroller for demonstrating bus speed purposes (not needed))

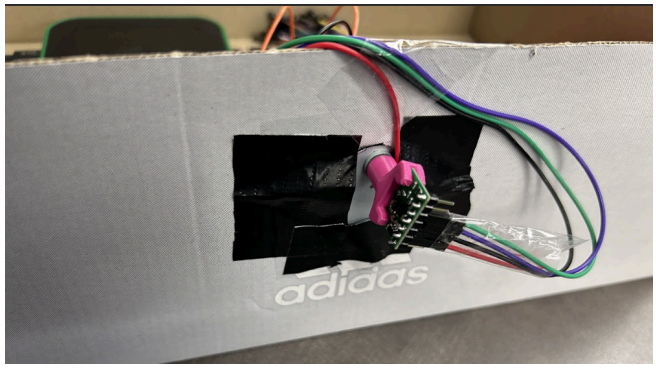
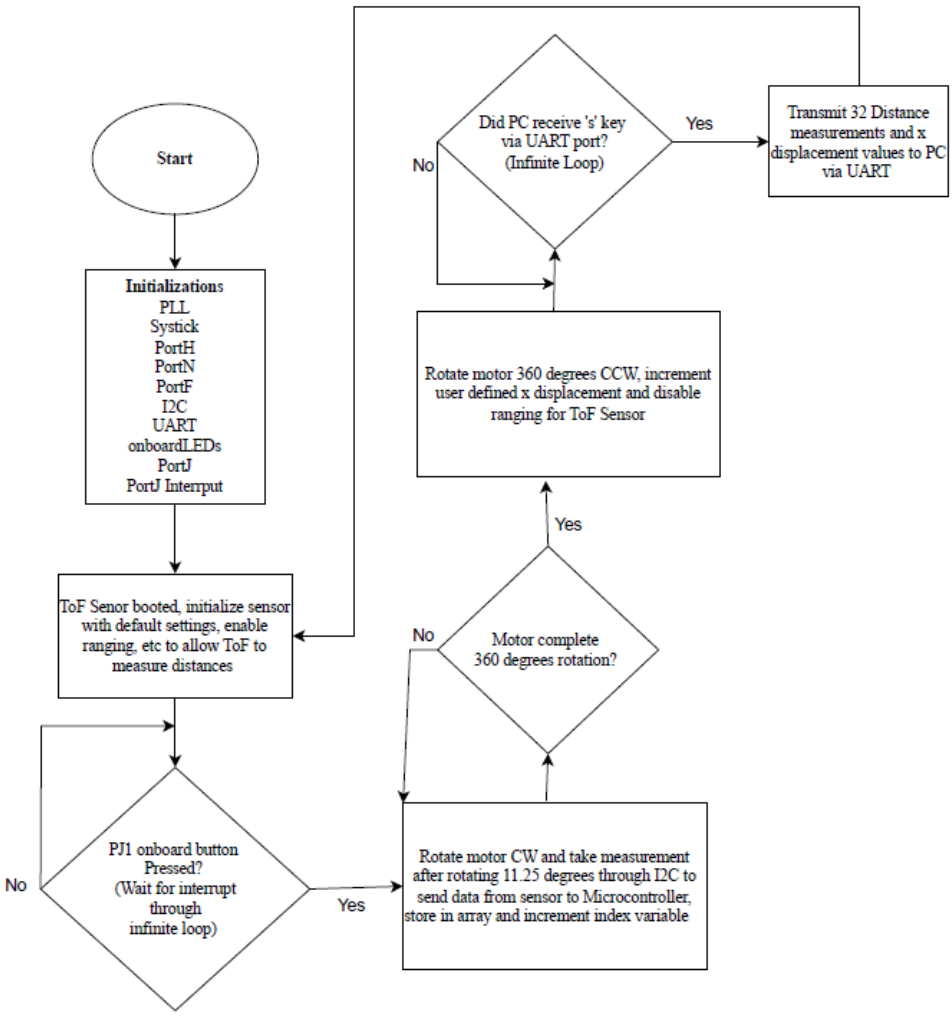


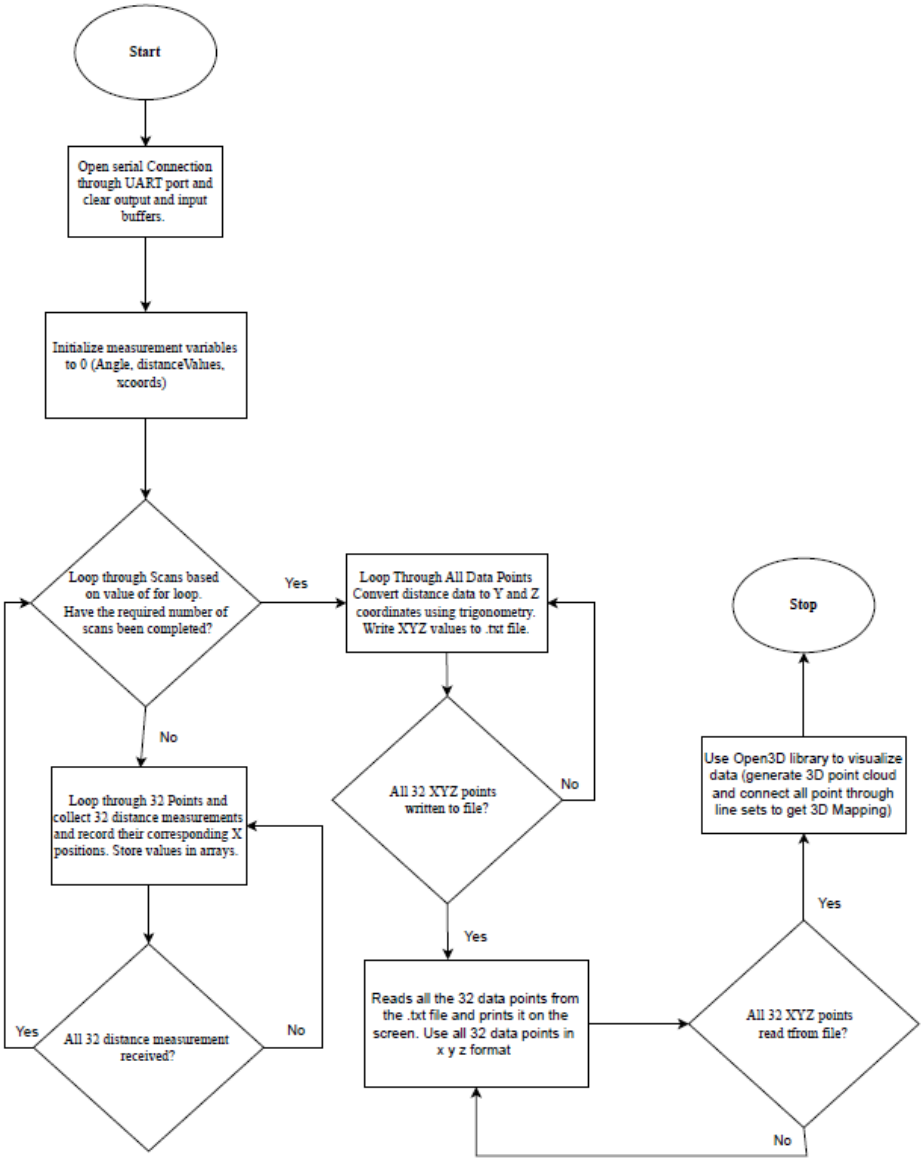
Figure 7: ToF Mounted on Stepper Motor through 3D Printed Mounter

Programming Logic Flowchart(s)

C Code in Keil uVision5 (Programming Microcontroller)



Python Code Programming



References

[1] S. Athar, T. E. Doyle, and Y. Haddara, "*Computer Engineering 2DX3 2023-2024 Laboratory Manual*." McMaster University, Hamilton, Ontario, Jan. 29, 2025

[2] S. Athar, T. E. Doyle, Y. Haddara, "*2023–2024 2DX3 Project Specification – Observe, Reason, Act: Spatial Mapping Using Time-of-Flight*", McMaster University, Hamilton, Canada, 2025

[3] Texas Instruments. (2018). "*MSP432 SimpleLink™ microcontrollers – Technical reference manual*", (Chapter 4, 17, pp. 326–327, 1201–1252).