COMPENG 2DX3 — Final Project Report
LiDAR Datasheet

Maheer Huq — 400508517

Instructors: Dr. Athar, Dr. Doyle, Dr. Haddara

Date: April 3rd, 2025

Table of Contents

Device Overview

(a) Features

This device is a rotating spatial measurement system that employs a Time-of-Flight (ToF) sensor (VL53L1X) mounted on a stepper motor (28BYJ-48) to capture distance measurements from multiple angles. An MSP-EXP432E401Y microcontroller (MCU) manages data processing and transmission for visualization, with an external push button for homing. Designed as a low-cost, customizable LiDAR alternative, this system is well-suited for robotics, navigation, and indoor environmental mapping applications. The components' specifications and features are detailed below, including information regarding the receiver-side programming language.

VL53L1X ToF Sensor (source of specifications: [1])

- Supports accurate ranging up to 400cm (4m) with a 27° field of view
    - Supports up to 50 Hz ranging frequency
- Operates at voltages between 2.6V and 3.5V
- Communicates with the MCU using the I^2C serial synchronized communication protocol
- The manufacturer's price for this device is CAD 4.99 [2]

28BYJ-48 Stepper Motor (source of specifications: [3])

- Total gear shaft ratio of 64:1, requiring 512 steps for a 360° rotation
- A four-phase unipolar motor operating in a full-step rotational configuration (adjustable to other configurations as needed)
- An operational voltage of 5 - 12V
- The price for this device is CAD 4.19 (including the ULN2003 board) [4]

Texas Instruments MSP-EXP432E401Y (source of specifications and price: [5])

- Functions as the central unit of the device, communicating with the ToF sensor via I^2C and the user PC via UART
- A default bus speed of 120 MHz (currently adjusted to 10 MHz)
- An operational voltage of 4.75 - 5.25V
- The manufacturer's price for this device is CAD 56.34
- Processes data at a baud rate of 115200 bps for UART communication
- Features 1024 KB of flash memory, 256 KB of SRAM, and 6KB EEPROM

External Push Button

- Handles homing functionality by returning the motor and sensor mount to a predefined home position
- Connected to a 3.3V voltage supply through a pull-up resistor
- This component can be found for CAD 0.15 [6]

Receiver-Side Programming Language

- Python is used as the primary receiver-side language, utilizing various libraries to process data transmitted to the UART port on the user's PC

- The following modules were imported into the program: *PySerial*, *math*, *numpy*, *Open3D*, and *time*
- Data is taken from the UART port using *PySerial*, processed using *math*, *numpy*, and *time*, and finally plotted using *Open3D*

(b) General Description

This embedded system allows users to map enclosed environments, such as hallways. It provides a low-cost and portable alternative to other spatial mapping systems on the market, though there are certain constraints to consider, as outlined above. Its purpose is to act as an additional component alongside other embedded projects that could benefit from mapping their surrounding environments. To understand each operation available to this device and how they work, refer to the details below:

- Start/Stop: onboard push button PJ0—begins distance measurement, with the motor halting every 11.25° to measure
  - The onboard LED0 (D1) will switch on when distance measurement is active, and off when not
- UART Transmission: onboard push button PJ1—to be pressed upon every completion of a 360° cycle; sends the collected data for a single cycle to the user's PC via UART
  - The onboard LED2 (D3) will briefly flicker to affirm that the measurements were successfully transmitted
- Homing Function: external momentary push button—returns the motor and ToF sensor to a predefined home position; only works if motor deactivated via PJ0
  - The onboard LED1 (D2) will rapidly flicker to signify the motor is returning home and will switch off when successful
  - A short message indicating a successful homing will be transmitted via UART to the user's PC

The system converts a distance measurement into an electrical signal via analog-to-digital conversion. A distance value is captured by the ToF sensor as it emits and receives a beam of light, using the following formula to scale how far it travelled:

$$\boxed{Distance\ (mm)\ =\ \frac{Photon\ travel\ time\ (s)}{2}\times Speed\ of\ light\ (m/s)}$$

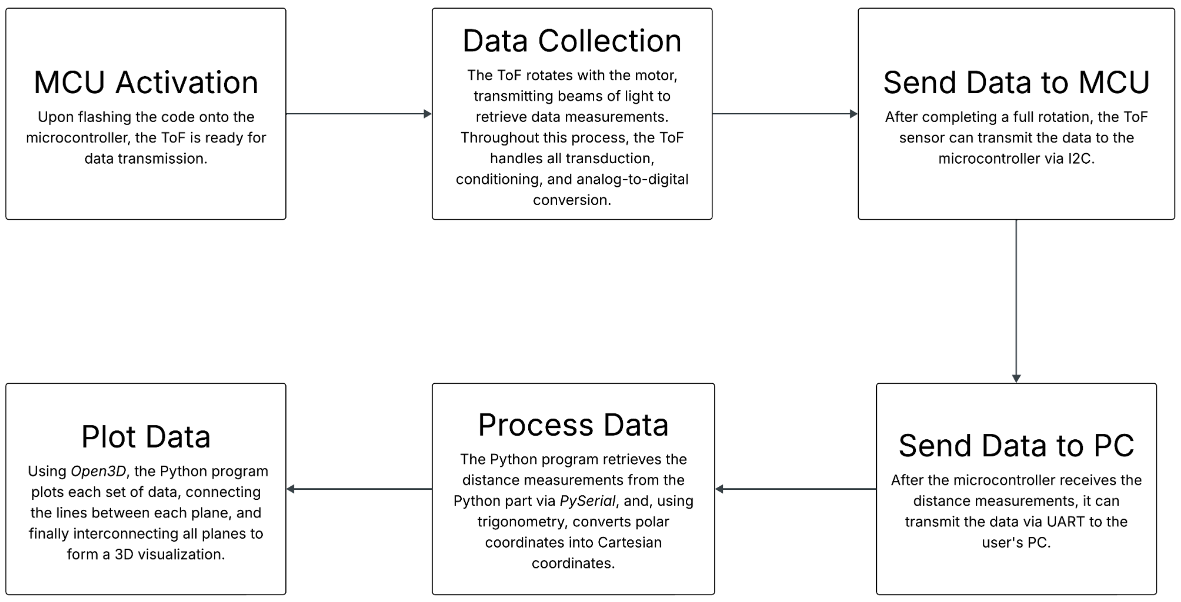
Equation 1 — ToF Distance Measurement Formula [1]

The VL53L1X sensor handles transduction, signal conditioning, and analog-to-digital conversion, significantly simplifying the translation process. Using the electronic representation of distance, several calculations were implemented to convert the values from polar to Cartesian coordinates, enabling the creation of a 3D map through Open3D.

After collecting distance measurements, the ToF sensor communicates serially with the MCU via I^2C to transfer the data values. The MCU then sends these values to the user's PC using UART, an asynchronous serial communication protocol. This works by physically connecting the MCU to the PC via a cable, which establishes a UART communication port on the PC.

Upon receiving these values, the receiver-side program, developed in Python, actively retrieves incoming data from the UART port and processes it for visualization. By combining the distance measurements from the ToF sensor with the corresponding angles, the program generates values in the y-z plane, while the user manually increments the x-values. This results in a set of XYZ coordinates that can be plotted.

(c) Block Diagram



MCU Activation
Upon flashing the code onto the microcontroller, the ToF is ready for data transmission.

Data Collection
The ToF rotates with the motor, transmitting beams of light to retrieve data measurements. Throughout this process, the ToF handles all transduction, conditioning, and analog-to-digital conversion.

Send Data to MCU
After completing a full rotation, the ToF sensor can transmit the data to the microcontroller via I2C.

Send Data to PC
After the microcontroller receives the distance measurements, it can transmit the data via UART to the user's PC.

Process Data
The Python program retrieves the distance measurements from the Python part via *PySerial*, and, using trigonometry, converts polar coordinates into Cartesian coordinates.

Plot Data
Using *Open3D*, the Python program plots each set of data, connecting the lines between each plane, and finally interconnecting all planes to form a 3D visualization.

Figure 1 — LiDAR Device Block Diagram

Device Characteristics

Table 1 — Device Characteristics

MSP-432E401Y MCU		**VL53L1X ToF**		**UL2003 Driver Board**	
Bus Speed	10 MHz	V_{IN}	3.3V	In1	PH0
Onboard Push Buttons	PJ0, PJ1	GND	GND	In2	PH1
Measurement Status LED	PN1	SDA	PB3	In3	PH2
UART Transmission LED	PF4	SCL	PB2	In4	PH3
Homing Function Pin	PM0			V_+	5V
Homing Status LED	PN0			V_-	GND
Baud Rate	115200 bps				
Serial Port	COM3				

Detailed Description

(a) Distance Measurement

As discussed in the *Device Overview*, the ToF sensor measures displacement (or distance) by emitting a beam of light and analyzing the time it takes for the light to return to the sensor. Displacement refers to the shortest straight-line distance between the starting point and the final position, along with the direction of the movement. Using basic kinematics, it can be calculated by multiplying the time it took for the light to travel by the velocity at which it travelled. In this case, the speed of light is used as the velocity, which is approximately 299,792,458 meters per second in a vacuum [7]. As the light is anticipated to return along an identical path, the desired distance is approximated by dividing the total travel time by two, yielding *Equation 1*. Using this equation, the ToF sensor provides a measurement in millimetres and sends it to the microcontroller via I^2C on a synchronized clock connected to pin PB2. To gather data across the full rotation (360°), the motor halts 32 times, at 11.25° intervals, to collect each measurement. A sample calculation employing *Equation 1* can be found below:

Sample Calculation:

To calculate the distance measured by the ToF sensor, we use the equation:

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

Where:

- The *photon travel time* is the time taken by the light to reach the desired object
- The *speed of light* is approximately 299,792,458 m/s

Given:

- $Photon\ travel\ time = 1.5\ ns$

Calculation:

$$Distance = \frac{1.5 \times 10^{-9} s}{2} \times 299,792,458\ m/s$$

$$Distance = 0.2248m \times \frac{1000mm}{1m}$$

$$Distance = 224.8\ mm$$

After the microcontroller receives these distance values, it transmits them to the user's PC via UART, where the receiver-side program retrieves the data using *PySerial*. Trigonometry is then applied within the Python program to convert the distance measurements and angles into xyz coordinates:

$$y = Distance \times sin(\theta)$$

$$z = Distance \times cos(\theta),$$

where y is the y-coordinate, z is the z-coordinate, and θ is the angle of the stepper motor relative to a predefined home position. It is important to note that the y-z plane in these measurements defines the vertical plane, while the x-coordinates are manually incremented to represent depth. The increment value for the x-coordinates can be customized within the Python program, and it is assumed that the user moves according to the set distance.

(b) Visualization

Once the data has been fully transferred to the user's PC, the provided Python program must be actively running to provide visualization. This script uses the *Open3D* library to visualize a 3D map of the measured environment by processing data from the ToF sensor. Initially, the script establishes serial communication with the sensor over UART, retrieving distance measurements sent in sets. These values are processed by converting the distance and angle measurements into XYZ coordinates using trigonometry, where the y and z coordinates are calculated using the formulas $y = Distance \times cos(\theta)$ and $z = Distance \times sin(\theta)$, with the *x* coordinate manually incrementing between scans to simulate depth. The data for each scan is stored in a file and later compiled into a 3D point cloud using *Open3D*'s *PointCloud* class. The script also generates line connections between the points, both within each scan and between adjacent scans, creating a connected structure. Visualization is handled using *Open3D*'s *Visualizer* class, allowing the user to interact with the 3D map. The script sets a custom viewing angle to ensure a proper display of the 3D structure, using *Open3D*'s functionalities to render the points and connections effectively. Important libraries used in this process include *PySerial* for UART communication, *math* for trigonometric calculations, *numpy* for data manipulation, *time* for stabilization and delay, and *Open3D* for visualization.

Application Note, Instructions, and Expected Output

(a) Application Note

This LiDAR-based device is a versatile tool that can be used in various fields to capture 3D spatial data, making it a valuable resource for several applications.

For instance, LiDAR technology has revolutionized precision agriculture, providing farmers with detailed, accurate 3D maps of their land [8]. These maps allow farmers to make impactful decisions using provided data, leading to optimization of crop yields and reducing general costs.

A specific use case is terrain mapping [9]. LiDAR sensors mounted on drones or ground vehicles can scan the land, capturing high-resolution 3D data on elevated territories. These scans let farmers develop topographic maps, which can reveal subtle variations in soil structure and terrain. By analyzing these maps, farmers are able to identify areas prone to flooding or areas that dry out too quickly, improving irrigation strategies. This ensures that water is distributed across the farm, minimizing waste and conserving resources.

Additionally, LiDAR is used to map the canopy structure of crops [10]. By analyzing the height and density of the canopy itself, farmers can assess the health of their crops and detect areas that might be experiencing stress from pests or diseases. Take, for example, if a field is found to have a denser canopy or varying crop height, then it could indicate where plants are competing for sunlight or nutrients, meaning a different planting or irrigation technique may be required.

(b) Instructions

It is assumed that the user has already properly installed and configured the following software and libraries before proceeding with these instructions:

- Keil µVision 5
- Python (Versions 3.7 - 3.12) with the following libraries installed/included
    - *Numpy*
    - *Open3D* (only supports up to 3.12 as of January 9th, 2025)
    - *PySerial*
    - *math*
    - *time*

1) Construct the Circuit

The first step is to construct the circuit as shown in *Circuit Schematic*, while also making sure to account for the external push button and pull-up resistor wired to pin PM0.

2) Connect the MCU to the PC and determine the UART port

The user should then connect the microcontroller to their PC via the USB cable, and then determine which COM port the UART communication occurs at. To do this, search for "Device Manager" in the Windows search query.

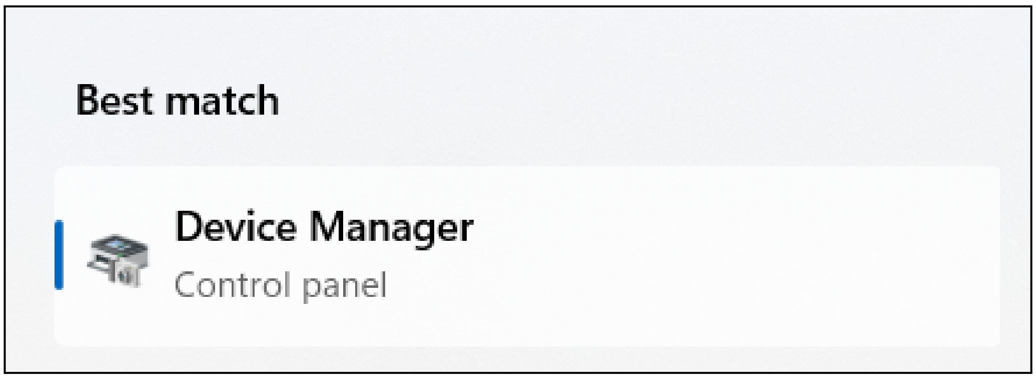


Figure 2 — Device Manager Search Result

Click on the shown search result and locate the "Ports (COM & LPT)" section in the list that appears.

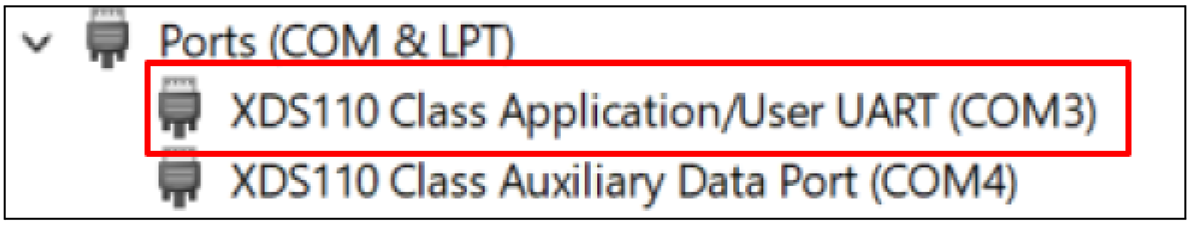

Ports (COM & LPT)
XDS110 Class Application/User UART (COM3)
XDS110 Class Auxiliary Data Port (COM4)

Figure 3 — Locating the COM Port

Take note of the port that your PC uses for UART communication. In the example above, it is seen that the UART port is on COM3, which is also what is established in the code provided. It is important to note that this port will vary by machine.

3) Software Setup

Extract the contents from the ‘2DX3 LiDAR Device.zip’ folder into an empty folder, enter the ‘Keil Project’ directory, and open the ‘LiDAR Program’ µVision 5 project. Navigate to ‘Deliverable 2’. Compile, build, and load the code to your microcontroller, refer to the image below for a visual indication.

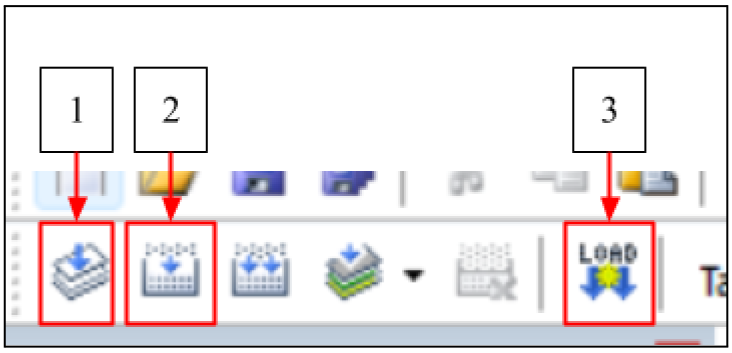


Figure 4 — Flashing the Code onto the MCU

Next, open the ‘UART_Input.py’ file from the ‘PC Code’ folder in your preferred IDE. In Line 11, update the COM port to match the one identified in the Device Manager.

4) Data Collection and Mapping

Before starting data collection, ensure that the location being measured is a closed environment. Adjust the number of scans to the preferred amount in Line 23 of the Python program. To begin scanning, run the Python program and wait for the startup message. On the microcontroller, press the reset button followed by PJ0 to initiate the process.

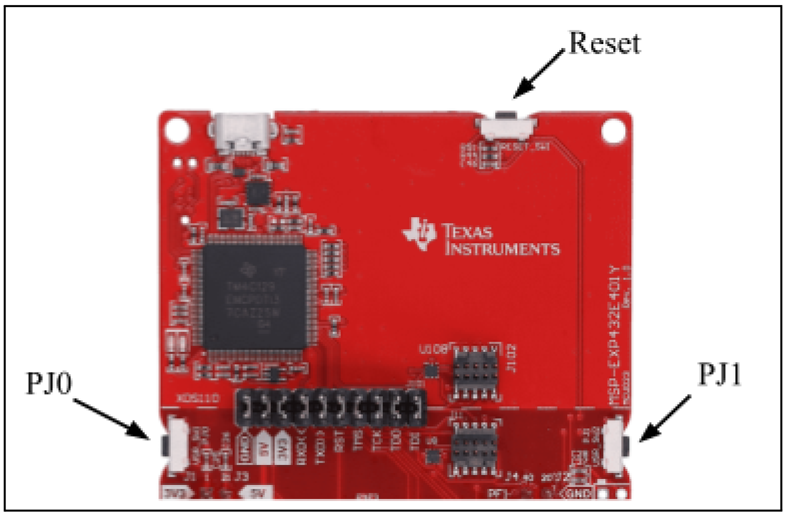


Figure 5 — PJ0, PJ1, and Reset on the MCU

A success message transmitted via UART to the IDE running the Python program will indicate whether the process was successful. If so, the motor will begin rotating and halting every 11.25° for measurement. After a full 360° the motor will begin to automatically rotate backwards to the home position at which it started. This feature was implemented to avoid wires tangling and causing stress to the system. Once the motor has completely halted, the user must press PJ1 to transmit the data collected during the cycle. This data will be displayed on the IDE's terminal and saved to a file. The user should also be mindful of the timing for data processing, as the Python program waits only a few seconds before proceeding to the next cycle. After finishing the defined amount of scans, the program will provide an exit message and automatically generate a 3D plot via *Open3D*.

(c) Expected Output

The expected output is a plot described by interconnected dots and lines resembling the area being measured. An example of the output in comparison to the region being mapped is found below.

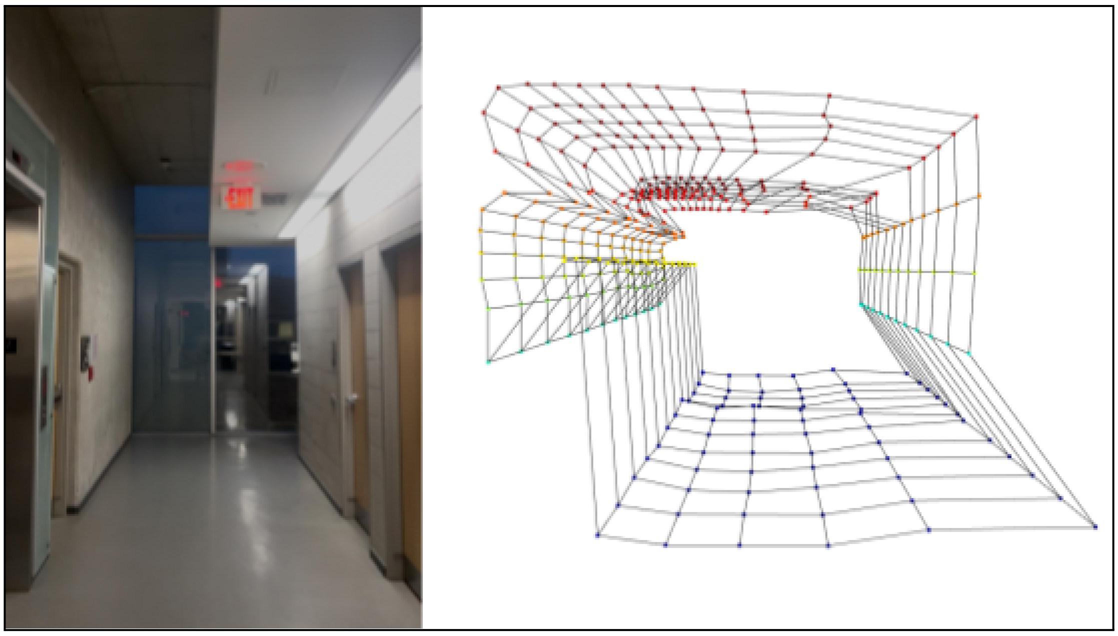


Figure 6 — Side-by-side Comparison of a Measured Room and its Output

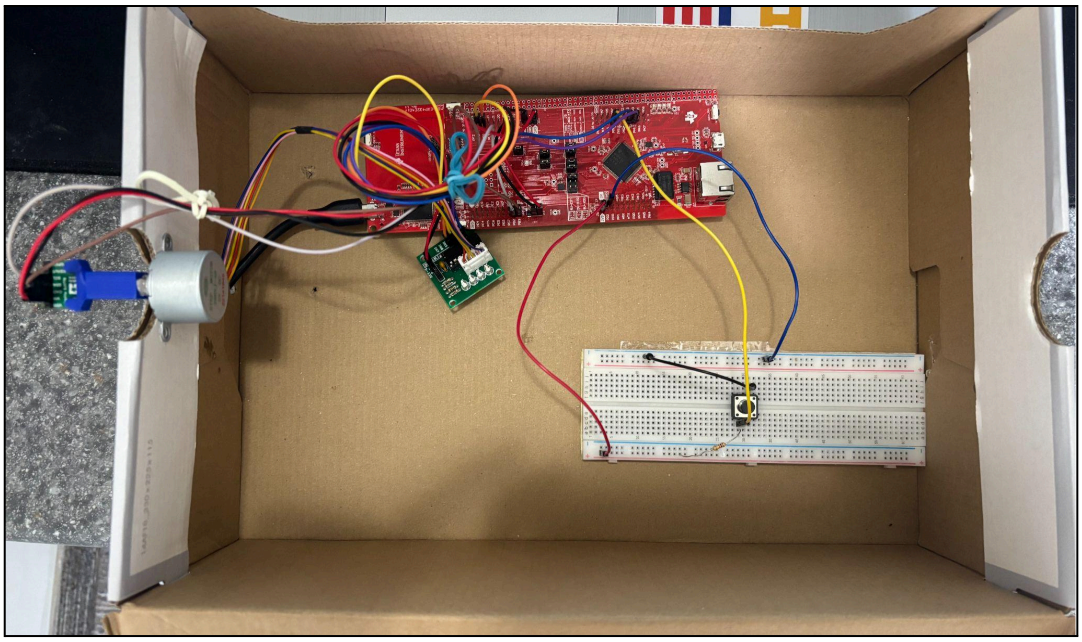


Figure 7 — Example Construction of Circuitry

It is important to note that several factors have the ability to affect the data retrieved by the ToF sensor. In the image above, mapping errors can be attributed to the irregular shape of the ceiling, as well as the lights located in the upper-right corner. As mentioned previously, the axes in the program are defined such that the y-z plane represents the vertical slices, while the x-measurements are simulated by manually moving forwards or backwards to introduce depth.

Limitations

(a) Limitations of microcontroller floating point capabilities and use of trigonometric functions

The MSP-EXP432E401Y features a 32-bit floating-point unit (FPU) that supports single-precision add, subtract, divide, multiply-and-accumulate, and square root operations [5]. However, these single-precision operations are 32-bit, meaning they are less precise in comparison to the 64-bit operations often found in PCs. This implies that accuracy is lost in complex calculations involving floating-point numbers, a common occurrence when using trigonometric functions.

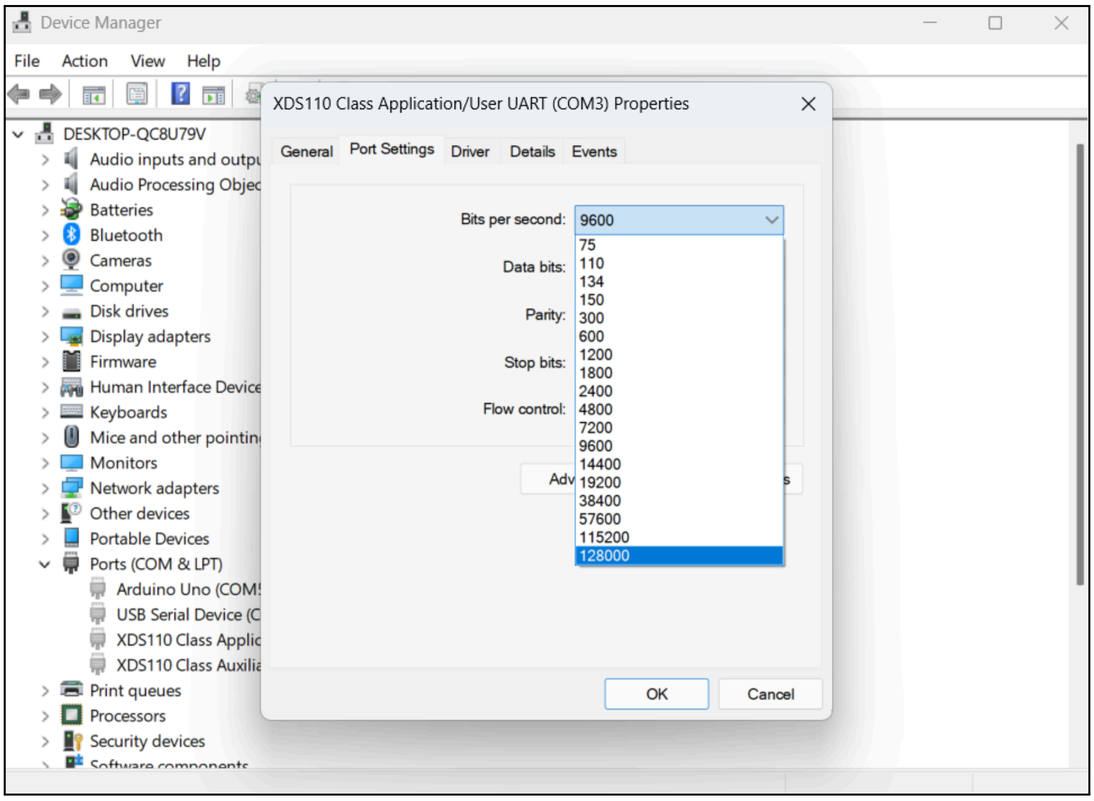
When converting from polar to Cartesian coordinates, the sine and cosine functions are employed. According to Niven's Theorem, it is known that, except for a specific set of angles, most values derived from the sine and cosine functions are irrational, meaning they continue indefinitely without repeating [11]. This becomes an issue when considering that such calculations would be slower and less precise on the MCU than on a typical PC. Additionally, the Python program performs various operations on these floating-point values, such as converting them to radians, which further introduces the potential for error. Had these values been handled on the PC initially, they would maintain twice the precision they currently have, which has significant implications, especially in industrial settings such as manufacturing where precision is critical.

(b) Maximum Quantization Error for the ToF Module

The maximum quantization error refers to the largest possible error introduced when a continuous signal is converted into a discrete representation during the process of quantization [12]. The VL53L1X ToF sensor lacks a typical ADC in its hardware, so its maximum quantization error can be determined by its resolution, which is 1 mm, the unit it measures in [13]. While this level of error is suitable for many applications, it can become a limitation when considering precision-critical tasks such as medical imaging, where even small discrepancies can have significant effects.

(c) The Maximum Standard Serial Communication Rate with the PC

The maximum standard serial communication rate available between the microcontroller and the PC is 128,000 bps. This value was verified by examining the available options under the 'Bits per second' setting in the UART port's Port Settings. Despite this, the device employed a Baud rate of 115200 bps given that it is the highest stable rate that is supported between the microcontroller and the PC.



XDS110 Class Application/User UART (COM3) Properties

Bits per second: 9600

75
110
134
150
300
600
1200
1800
2400
4800
7200
9600
14400
19200
38400
57600
115200
128000

Figure 8— Highest available Baud Rate between the PC and the MCU

(d) Communication Method and Speed Between the MCU and the ToF Modules

A communication type refers to the specific method or protocol used for transmitting data between two or more devices. The MSP-EXP432E401Y microcontroller and the ToF sensor communicate via the I^{2}C protocol, a synchronous serial communication method commonly used to connect microcontrollers to peripheral devices. I^{2}C allows several devices to operate on the same set of data and clock lines, leading to efficient data transmission when considering short distances. In the case of this device, it employs the standard mode of the I^{2}C protocol, operating at a data transfer rate of 100 kbps.

(e) The Primary Limitations on Speed and Testing

The system's speed is primarily limited by the stepper motor and the ToF sensor which act as bottlenecks. The ToF sensor has a limitation in that to accurately measure a distance point, it must employ a 100 ms timing budget [1]. This means that the majority of the limitation on speed can be adjusted through the stepper motor, which has been programmed such that there is a small delay between the four phases. To test how much quicker the device could operate, the delay between the motor phases was shortened until it no longer functioned. It was found that the quickest delay that could be put into place was approximately 7.5ms for the specific motor employed in the examples provided.

Circuit Schematic

Refer to *Figure 9* for a high-level overview of all circuitry involved in this device.

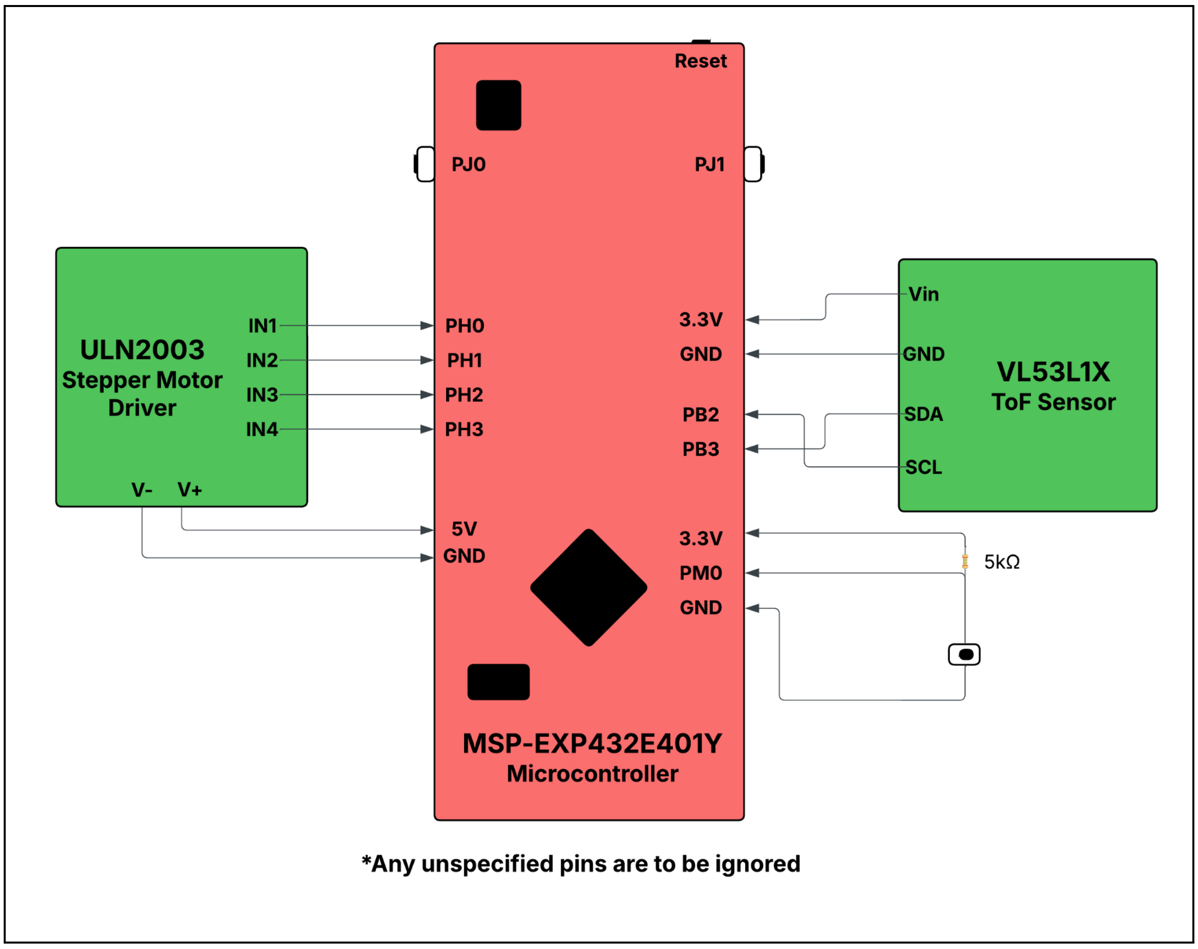


Figure 9 — Circuit Schematic

Programming Logic Flowcharts

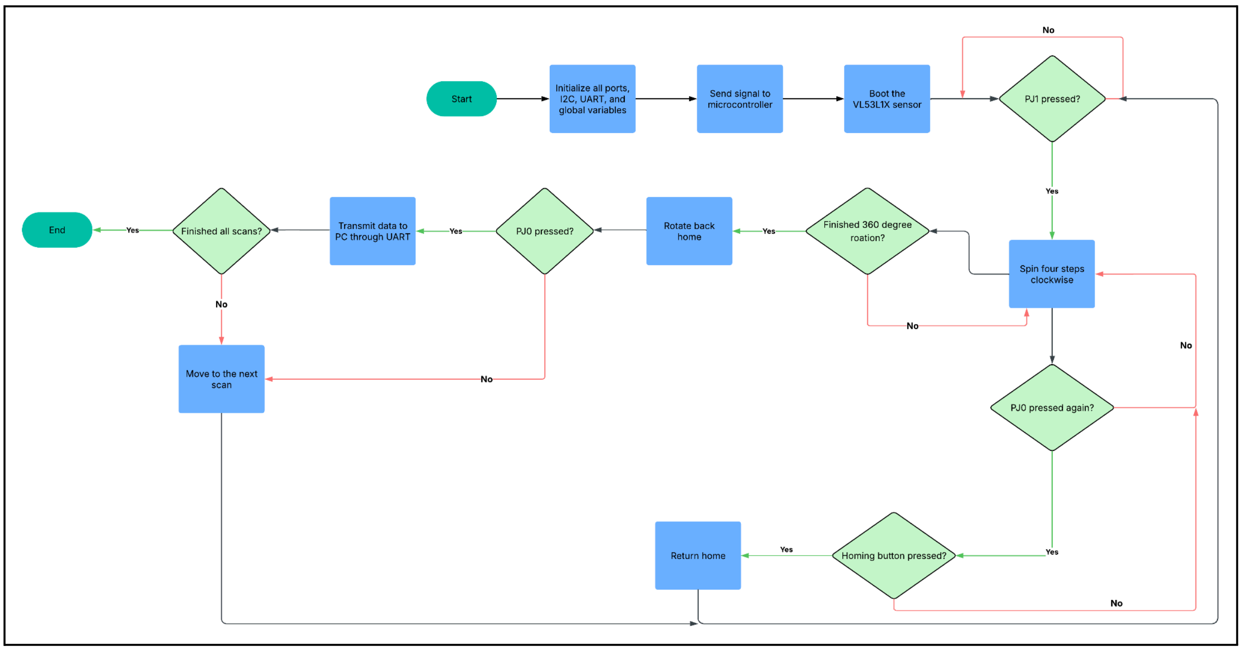


Figure 10 — C Program Logic Flowchart

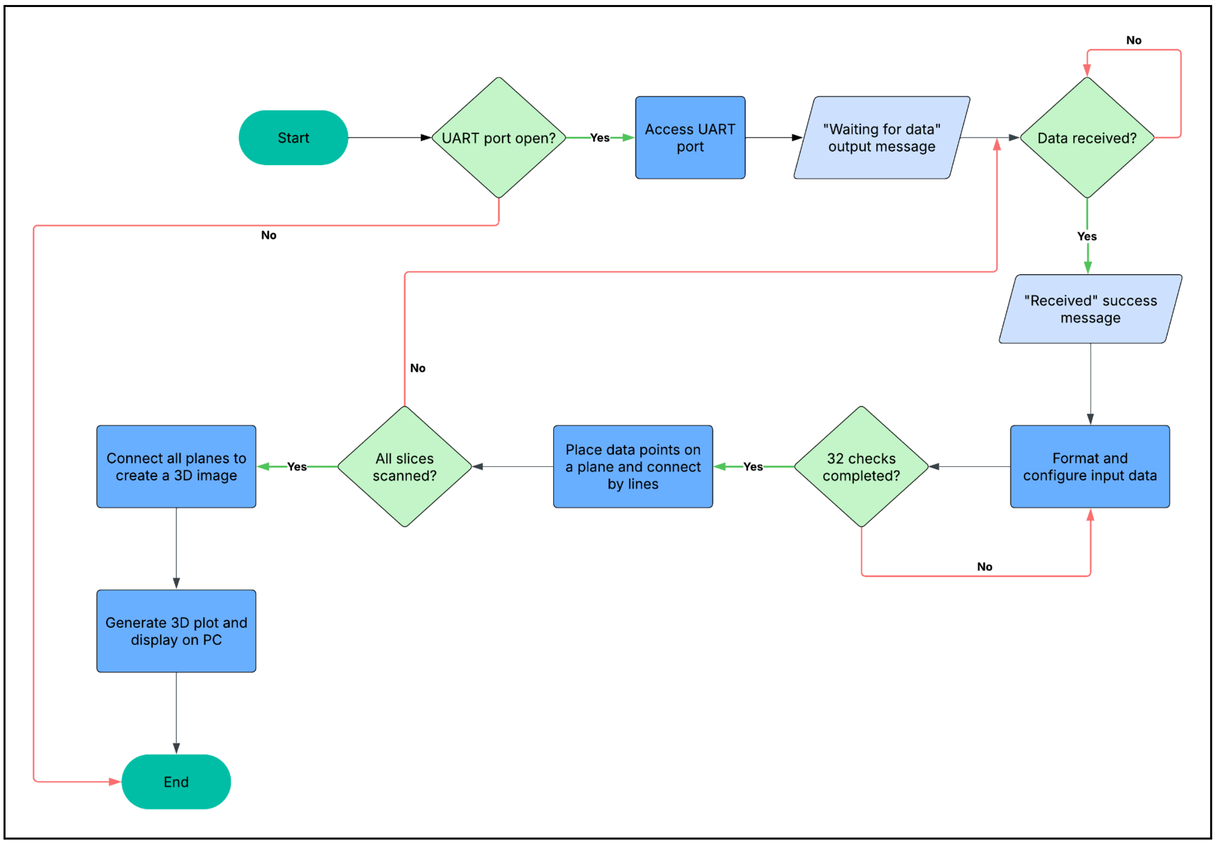

Start
UART port open?
Yes
Access UART port
"Waiting for data" output message
Data received?
No
Yes
"Received" success message
Format and configure input data
32 checks completed?
No
Yes
Place data points on a plane and connect by lines
All slices scanned?
No
Yes
Connect all planes to create a 3D image
Generate 3D plot and display on PC
No
End

Figure 11 — Python Program Logic Flowchart

References

[1] "This is information on a product in full production. VL53L1X," 2022. Available: https://www.st.com/resource/en/datasheet/vl53l1x.pdf

[2] "Buy VL53L1X - eStore - STMicroelectronics," *St.com*, 2025. https://estore.st.com/en/products/imaging-and-photonics-solutions/time-of-flight-sensors/vl53l1x.html

[3] "28BYJ-48 -5V Stepper Motor." Accessed: Apr. 06, 2025. [Online]. Available: https://www.mouser.com/datasheet/2/758/stepd-01-data-sheet-1143075.pdf?srsltid=AfmBOoqjpfJQ3QA4tvCcIEyQ5Zc6WBWjXTGZGiWBOqFjvdknX0yURP3J

[4] "Small Stepper Motor (Unipolar 5V) 28BYJ-48," *Canada Robotix*, 2019. https://www.canadarobotix.com/products/1926?srsltid=AfmBOor8uNkHAupY30BLK_3ZaQEtyWu5btQziK33l5qjEmCLbMvS0mzJ

[5] "SLASEN5 Data sheet | TI.com," *Ti.com*, 2017. https://www.ti.com/document-viewer/msp432e401y/datasheet

[6] "Electronic Components and Parts Search | DigiKey Electronics," *Digikey.ca*, 2025. https://www.digikey.ca/en/products/result?s=N4IgTCBcDaIA4FcDOALABAIwQF2wewDsQBdAXyA

[7] NIST, "CODATA Value: speed of light in vacuum," *Nist.gov*, 2018. https://physics.nist.gov/cgi-bin/cuu/Value?c

[8] "How Lidar Technology is Revolutionizing Agriculture," *Marketsandmarkets.com*, 2024. https://www.marketsandmarkets.com/blog/SE/lidar-technology-agriculture

[9] V. Hoffmann, "LiDAR for topographic mapping: Advantages and disadvantages," *Above*, Aug. 09, 2023. https://www.abovesurveying.com/lidar-for-topographic-mapping-advantages-and-disadvantages/

[10] M. A. Lefsky, W. B. Cohen, S. A. Acker, G. G. Parker, T. A. Spies, and D. Harding, "Lidar Remote Sensing of the Canopy Structure and Biophysical Properties of Douglas-Fir Western Hemlock Forests," *Remote Sensing of Environment*, vol. 70, no. 3, pp. 339–361, Dec. 1999, doi: https://doi.org/10.1016/s0034-4257(99)00052-8.

[11] C. Nunn, "A Proof of a Generalization of Niven's Theorem Using Algebraic Number Theory," *Rose-Hulman Scholar*, https://scholar.rose-hulman.edu/rhumj/vol22/iss2/3/

[12] A. Colagrossi, V. Pesce, S. Silvestrini, D. Gonzalez-Arjona, P. Hermosin, and M. Battilana, "Sensors," *Elsevier eBooks*, pp. 253–336, Jan. 2023, doi: https://doi.org/10.1016/b978-0-323-90916-7.00006-8.

[13] "Qwiic Distance Sensor (VL53L1X, VL53L4CD) Hookup Guide - SparkFun Learn," *Sparkfun.com*, 2017. https://learn.sparkfun.com/tutorials/qwiic-distance-sensor-vl53l1x-vl53l4cd-hookup-guide/all