EMBEDDED 3D MAPPING SYSTEM USING ToF SENSING

COMP ENG 2DX3 - Microprocessor Systems Project

Name: Shiv Patel
Student ID: 400530101
MacID: pates302
Section: L07

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

Submitted On: April 9, 2025

Table of Contents

Device Overview

General Description

This project presents the design and implementation of a compact, cost-effective, and fully functional Light Detection and Ranging (LiDAR) system [1]. Using a VL53LX1 Time-of-Flight (ToF) sensor, a stepper motor, and a Texas Instruments MSP432E401Y microcontroller [3], this project captures 3D spatial data by rotating the stepper motor and the sensor, which is attached to the motor, in a full 360-degree vertical scan (y-z) plane and outputting the scan in the form of a 3D plot using Python and its various libraries.

The main goal of this project was to create a spatial mapping system that can mimic a LiDAR system but at a reduced cost and size while maintaining the overview and function, which is maintaining accurate data acquisition and real-time visualization. The VL53L1X sensor measures distance using emitted infrared light and time-of-flight calculations (see **Equation 1**), which are collected via I^{2}C and processed by the microcontroller. The sensor data is transferred via I^{2}C communication, followed by the data being transmitted to a personal computer (PC) over Universal Asynchronous Receiver-Transmitter (UART), where it is rendered into a 3D point cloud using a Python script and available Python libraries, Matplotlib and NumPy. The data points are plotted on a 3D graph and are connected to recreate the scanned environment, with arbitrary displacement increments (*x*-axis) of 33 cm, providing a full scan of its environment.

$$distance = \frac{(Time\ of\ Flight\ of\ Photon\ Emitted) \times (Speed\ of\ Light)}{2} \tag{1}$$

Overall, this project showcases the real-world application of embedded systems, microprocessor interfacing, sensor integration, communication protocols, and data visualization.

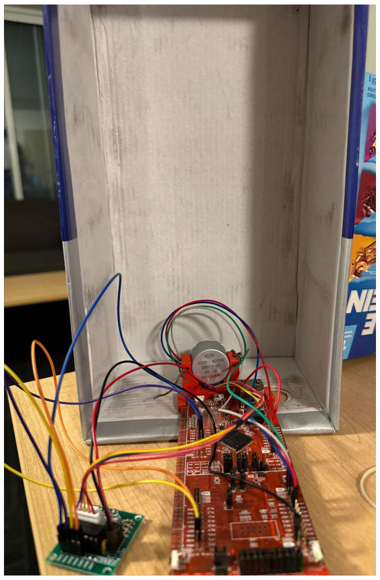


Figure 1: Device Setup

Features

Table 1: All devices used, their features, and their costs

Device Type	**Device Name**	**Device Features** [2][4][5][3]	**Cost** [6][7] ($CAD)
ToF Sensor	VL53L1X	● Size: 4.9x2.5x1.56 mm ● Up to 4 m (400 cm) of range ● 50 Hz Sampling Frequency ● I^2C communication up to 400 kHz ● Typical Full Field of View (FoV) of 27° ● 940 nm invisible laser emitted ● SPAD (Single Photon Avalanche Diode) ● Typical operating voltage: 2.6 V to 3.5 V	$24.85
Stepper Motor	28BYJ-48	● 5V DC Input ● 64:1 gear ratio ● 5.625°/64 stride angle ● Operating frequency: 100 Hz ● 512 steps/rev	$3.95
Stepper Driver	ULN2003	● 7 Darlington transistor pairs ● Each transistor pair is capable of up to 500mA and 50 V loads ● LED Step indicators ● Overcurrent protection	$2.19
Microcontroller + Microprocessor	TI MSP432E401Y + MSP-EXP432E401Y	● 32-bit ARM Cortex-M4F with Floating-Point Unit (FPU) ● 12 MHz default clock speed (up to 120 MHz) ● 256 KB SRAM ● 1MB Flash RAM ● 60 MHz Bus Speed ● 5V, 3.3V volt sources ● 5V micro-usb for power and programming ● GPIO with interrupt, PWM, and ADC support ● Onboard pushbuttons and LEDs ● I^2C/UART/Ethernet Support	$39.99

		● UART at 115200 bits/sec for PC communication ● C or Assembly Language operated (via Keil)	
Python	Python 3.10.0 (64-bit)	● Threading for input/output (I/O) ● Math for trigonometry ● NumPy for vector handling ● Queue (data structure) ● PySerial for UART ● Matplotlib for 3D Plotting	N/A
Device Mount	-	● 3D Printed	$2.50
Components Kit	2DX3 Components Kit	● Male-to-Male and Female-to-Female wires	$10.54

Block Diagram (Data Flow Graph)

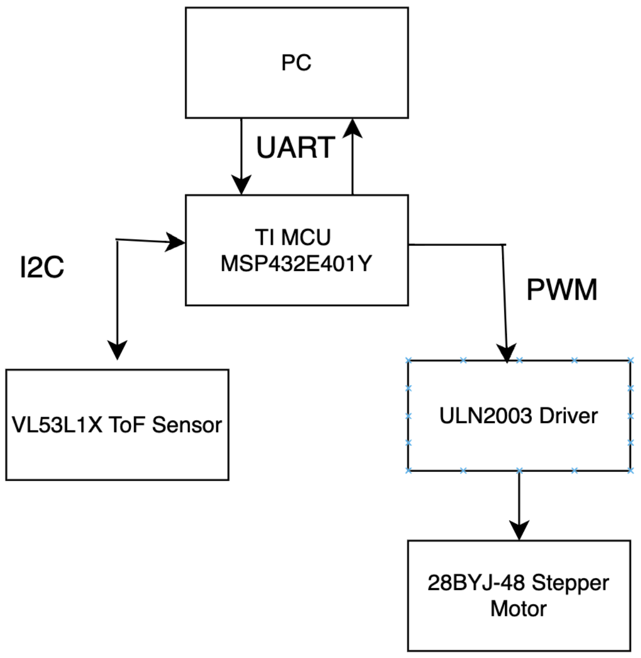


Figure 2: Block Diagram

Device Characteristics

Table 2: Device characteristics table

Characteristic	Specification
Serial Port (UART)	COM4
Baud Rate (bps)	115200
Python Version	3.10.0, 64-bit
Bus Speed (MHz)	16 (Student number based; Least Significant Digit (LSD): 1)
Measurement Status LED	PF4 (Student number based; 2nd Least Significant Digit (LSD): 0)
UART Tx	PN1 (Student number based; 2nd Least Significant Digit (LSD): 0)
Additional Status (UART Rx)	PF0 (Student number based; 2nd Least Significant Digit (LSD): 0)
Reset/On Indicator	PF0, PF4, PN0, PN1 (All on together)
Onboard Button (Start/Stop)	PJ1
ToF Sensor Pin Mapping	

Pin	Microcontroller
V_{DD}	N/A
V_{IN}	3.3 V
GND	GND
SDA	PB3
SCL	PB2
XSHUT	N/A
GPIO1	N/A

ULN2003	

Pin	Microcontroller
IN1	PH0
IN2	PH1
IN3	PH2
IN4	PH3
+	5V
-	GND

Detailed Description

Distance Measurement

The core of the project and its process relies on distance acquisition, which is done by the VL53L1X Time-of-Flight (ToF) sensor. This sensor allows for non-contact distance measurement by emitting 940nm of infrared light (IR) and calculating the time it takes for the light to reflect off the surface and return to the detector. The formula it uses to calculate that distance is:

$$distance = \frac{(Time\ of\ Flight\ of\ Photon\ Emitted) \times (Speed\ of\ Light)}{2} \qquad (1)$$

Where distance is the *distance* to the object, *t* is the round-trip travel time of the IR, and *c* is the speed of light ($c = 3 \cdot 10^8 m/s$)). The device uses a Single Photon Avalanche Diode (SPAD), which captures the photon's return time by a SPAD array, digitized and filtered to give a distance value in millimeters (mm), which is also known as Analog-to-Digital Conversion (ADC). This value is eventually transmitted to the microcontroller via the I^{2}C communication bus.

For example, if a sample ToF was measured to be approximately $3.34 \cdot 10^{-9}$ $seconds$, the distance would be

$$distance = \frac{(3.34 \cdot 10^{-9}) \times (3 \cdot 10^{8})}{2}$$

$$distance = 0.501\, m$$

Based on this formula and the chosen ToF value, the calculated distance would be approximately 0.501 m or 501 mm.

Moving on, when the system is booted up, the Microcontroller Unit (MCU) enables the I^{2}C peripheral and initializes the ToF sensor using specific API calls. These functions activate the sensor, setting up its configuration and default address, and begin the distance collecting process. Additionally, the sensor is also configured to operate in a single-ranging mode, which allows for a new distance measurement per cycle. This is ideal for this project, as it allows for a consistent distance sampling.

While scanning, the code constantly checks if the ToF Sensor has new data ready to be communicated. When it is, it reads the distance and status, clears the interrupt, and sends the result over UART, with a configured baud rate of 115200 baud to the PC. The data is sent in a string format, which is "*x-value*, distance, angle", with the *x-value* being the arbitrary x-value, the distance being the polar coordinate (in mm), and the angle the point was captured at (in degrees).

After each reading, the stepper motor moves a step ahead (clockwise) using the ULN2003 driver. The full rotation is broken into 512 steps (which is approximately 0.70° each). Every 64 steps (approximately 45°), the onboard LED (PN1) flashes for visual confirmation. This continues until the full 360° is achieved, so a total of 8 flashes occur during every scan.

Once a full scan is done, the system resets the step counter and increases the pre-set *x*-distance value (default at 333 mm), which can be implemented physically by having the user move the device forward. It also rotates back in the opposite direction, returning to its home position (counter-clockwise) to ensure that wires aren't tangled, which could result in interference in future scans. To continue with the scan, press the onboard button (PJ1) can be pressed again, and the entire process will repeat.

Visualization

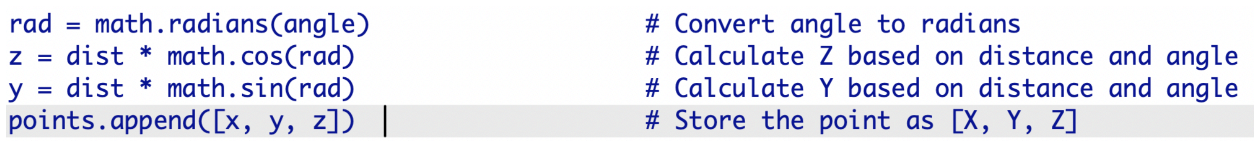
The visualization part of the project is done on a PC using a Python 3 script. The script is used to read the serial data transmitted by the microcontroller, transforming the polar coordinate data into cartesian coordinates and outputting a 3D plot. The script uses various common Python libraries to help manage real-time communication, data transformation, and visualization, including PySerial, NumPy, Matplotlib, and other built-in default libraries, such as Math and Threading. Additionally, the script is primarily implemented using a queue-type data structure.

Before the data is visualized, it is received over a UART interface. The microcontroller reads all of the data points collected from the ToF sensor and sends the scan points in the form of a string formatted as “x, distance, angle”. These values represent the displacement (x), which is arbitrary, and the other values (x and y) are calculated using trigonometric ratios (see **Figure 2**). The values communicated represent the distance (in millimeters) and the current angular position (in degrees). The MCU sends this data over UART with a baud rate configured at 115200 bps.

On the PC side, the script establishes a serial communication to a port (i.e., COM4), configuring it with the baud rate used by the MCU. To ensure that the data transmitted is complete and read correctly, a class named “SerialLineReader” is implemented, as it verifies that only complete data is processed and any noise or outlier values are discarded. Next, all the incoming serial lines are stored in a queue, which helps to ensure the data is processed accordingly and in the order they were transmitted. This class and queue implementation allows for a safe data acquisition and processing, as well as preventing any I/O blocking that could cause discrepancies in the final plot.

Next, each data line that is received is received as 3 values: displacement (*x*), distance (*d*), and the angle (Θ). These polar coordinates are transformed into cartesian coordinates (see **Figure 2**) using these trigonometric functions in the script. As the *x*-value is arbitrary and set by the user, it is pre-defined for that run, and the rest of the values (*y* and *z*) are calculated. The script determines if all of the readings are acquired or not by checking a conditional statement checking if the angle value has surpassed 359.2° (512 steps, which means approximately 0.71° per step or reading) and if the expected number of scans has been met or not.

Lastly, after all of the data is acquired, the list of points is converted into a NumPy array for efficient and easy data manipulation. The script uses “matplotlib.pyplot” with its “mplot3d” toolkit to render the points into a 3D plot. The plot consists of all the data points in blue, which are connected by red lines (within a scan), and the scans are connected by green lines (see **Figure**).



```
rad = math.radians(angle)                     # Convert angle to radians
z = dist * math.cos(rad)                      # Calculate Z based on distance and angle
y = dist * math.sin(rad)                      # Calculate Y based on distance and angle
points.append([x, y, z])                      # Store the point as [X, Y, Z]
```

Figure 2: The use of trigonometric ratios to calculate y and z values

Application Note, Instructions, and Expected Output

Instructions

PC (Windows 10)

1. Download a version of Python 3.6.0 to 3.10.0 (Python 3.10.0 was used throughout the demonstration of this project)
2. Install the following libraries on Python: PySerial, NumPy, and Matplotlib
3. Next Stage: **Python**

Python

1. Open the Python file "2DX3_Plotting.py" file in the Python 3.10.0 IDLE
2. Change your "COM" port accordingly, which can be verified by accessing the computer's terminal and typing "python3 -m serial.tools.list_ports –v", which should return a list of the ports being used and the COM address (COM#)
    a. If no list pops up, there may be issues with your drivers; please resolve those first (try plugging it into a different USB Port on your PC)
3. Enter that port address into Line 42 of the Python code
4. To change the number of scans (default is 1) or distance per scan (default is 333 mm), modify line 60 and line 61 accordingly
5. Once all the modifications are completed, run the Python code
6. Next Stage: **Physical Setup**

Physical Setup

1. Connect the microcotroller using the micro-USB wire to the PC being used
2. Ensure the wiring is configured accordingly to the circuit schematic (see **Figure 6**)
3. Place the sensor into the sensor mount, followed by the motor and the microcontroller into their respective mounts (see **Figure 1)**
4. Now, open the project file "main.c" and open it using Keil uVision IDE
5. Using the top 3 icons on the top left corner (see **Figure 3**) and press the left most, then the second left most, followed by the "LOAD" icon. Give it a few seconds, and the code should load, followed by the confirmation message being printed.
6. Using the onboard "RESET" button (beside the micro-USB input)
7. Next Stage: **Entire Project**

Entire Project

1. The Python terminal tab should open up and begin communicating now, and some prompts should have popped up
    a. If no messages pop up, there may be issues with the PC's drivers; please resolve those first (try plugging it into a different USB Port on your PC)
2. Start the scanning process by pressing the onboard button, PJ1
3. The scanning process will begin, and once the process is complete, the sensor will return to its home position (where it initially started)

4. Depending on the number of scans you have set in the Python Script, once the prompt comes up again, press PJ1 to scan once again
5. Once all of your scans are complete, the 3D graph will appear on your screen and can be observed.

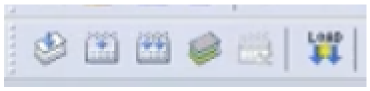


Figure 3: Keil uVision IDE interface

Expected Output

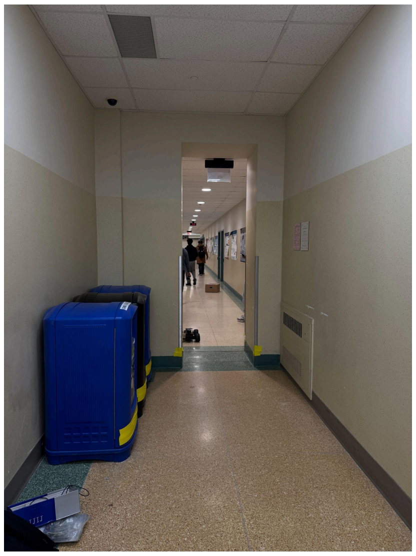


Figure 4: Hallway scanning location (ITB/Location B)

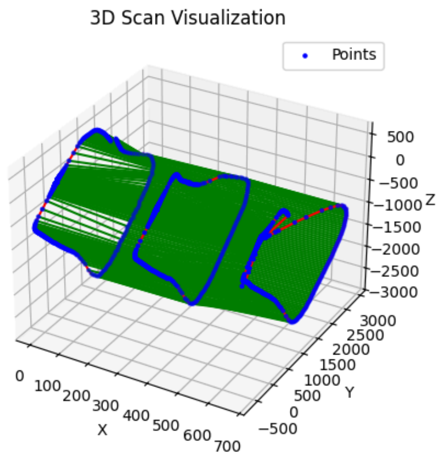


Figure 5: Python scanning result

Limitations

1. *Summarize any limitations of the microcontroller floating point capability and use of trigonometric functions.*

   The microcontroller used in this project (TI MSP432E401Y) has an ARM-based microprocessor that has a Floating-Point Unit (FPU). This FPU can handle single-precision (32-bit) math operations (i.e. multiplication, division, addition, subtraction, etc.). Since it supports single-precision and with the **math.h** library using trigonometric functions, such as cosine, sine, tangent, these are functions can only operate on values that float type. Given the FPU available with the devices used, there is a chance of rounding errors when acquiring and dealing with the data, specifically the mathematical data. This is because the FPU available is only 32-bits, and any operations exceeding that (i.e. 64-bit or double type), would be forcefully rounded, leading to a potential rounding error. But given this project and purpose, since the tolerance range isn't as small, these errors wouldn't result in discrepancy in data values.

2. *Calculate your maximum quantization error for the ToF module.*

   The VL53L1X sensor data sheet states that the distance measurements are returned in whole millimeters. This indicates that the sensor has a digital resolution of 1 mm, meaning it can only represent distance values in 1mm increments. As a result, the maximum quantization error, which is half of the resolution step, is ± 0.5 mm, but due to rounding, it becomes 1 mm.

3. *What is the maximum standard serial communication rate you can implement with the PC. How did you verify?*

   In terms of standard UART communication, the maximum baud rate is 921600 bps (or baud). But for this project's purpose, a default and more of a standard rate of 115200 was chosen. For the PC that was used during this project, to verify and check the maximum baud rate, it was done by accessing Device Manager, then the port being used (i.e COM4), and then checking its properties and port settings. This value was listed as 12800 bps, but this value could vary depending on the hardware (PC) being used.

4. *What were the communication method(s) and speed used between the microcontroller and the ToF modules?*

   The communication method used was I^2C protocol, using the data and clock lines on the sensor (SDA and SCL). As stated in the data sheet, the communication speed was set to 100 kHz, resulting in a 100kbps transfer rate for this case, but the ToF can support up to 400 kHz.

5. *Reviewing the entire system, which elements are the primary limitation on speed? How did you test this?*

   The elements that are the primary limitation on speed are the stepper motor and the sensor, which is partially related to the motor. For the motor, delays are required to avoid skipping or missing any steps within a scan and to ensure that the torque is maintained throughout. Adding a delay to the motor spinning section of the code ensured that the step sizes were verified and adhered to. Additionally, for the sensor, given that some locations have an abundance of light, a delay of a significant time was necessary, mostly just to ensure and avoid that any extra noise (excessive light) isn't captured and that the actual photon is captured and received to ensure data accuracy.

Circuit Schematic

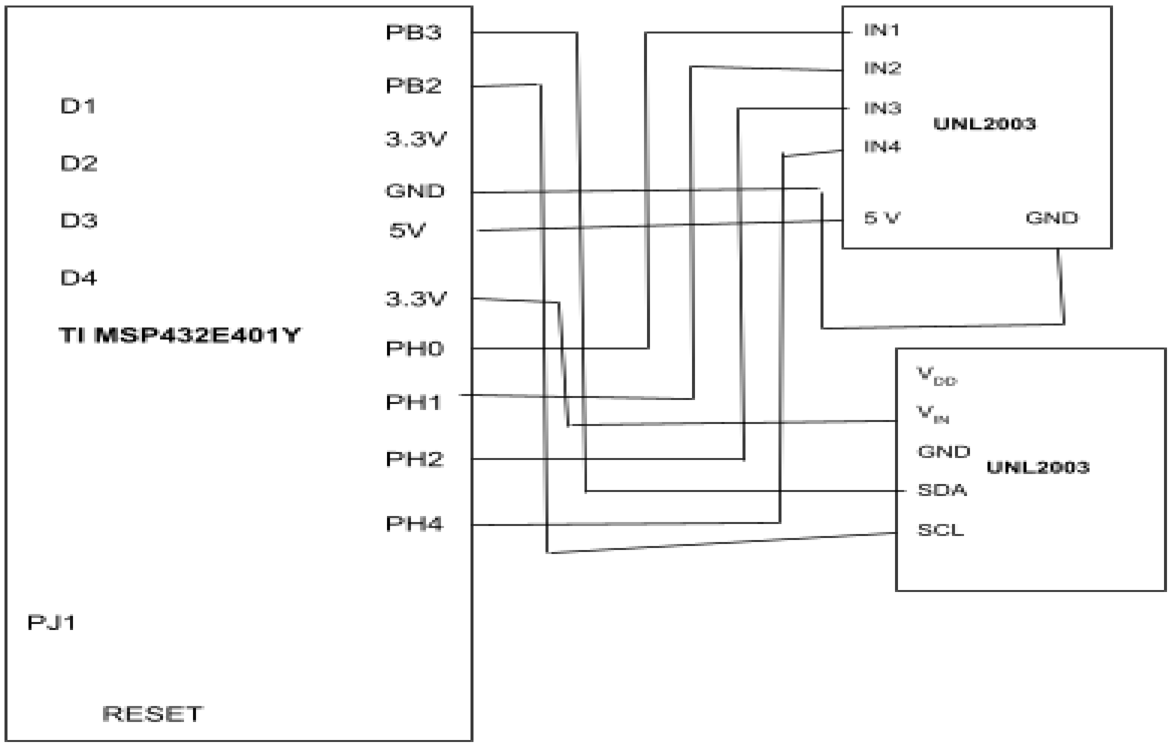


Figure 6: Circuit Schematic

Programming Logic Flowcharts

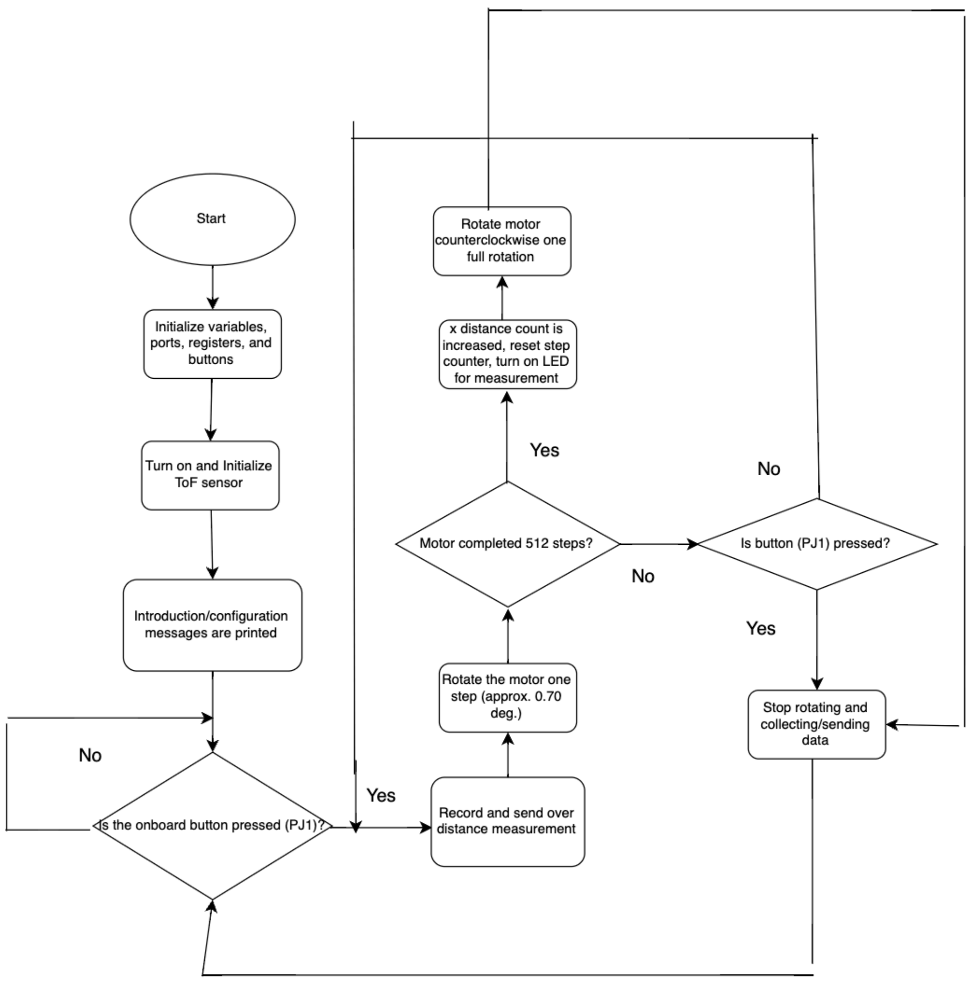

Start
Initialize variables, ports, registers, and buttons
Turn on and Initialize ToF sensor
Introduction/configuration messages are printed
No
Is the onboard button pressed (PJ1)?
Yes
Record and send over distance measurement
Rotate the motor one step (approx. 0.70 deg.)
Motor completed 512 steps?
Yes
x distance count is increased, reset step counter, turn on LED for measurement
Rotate motor counterclockwise one full rotation
No
Is button (PJ1) pressed?
No
Yes
Stop rotating and collecting/sending data

Figure 7: C Code Flowchart

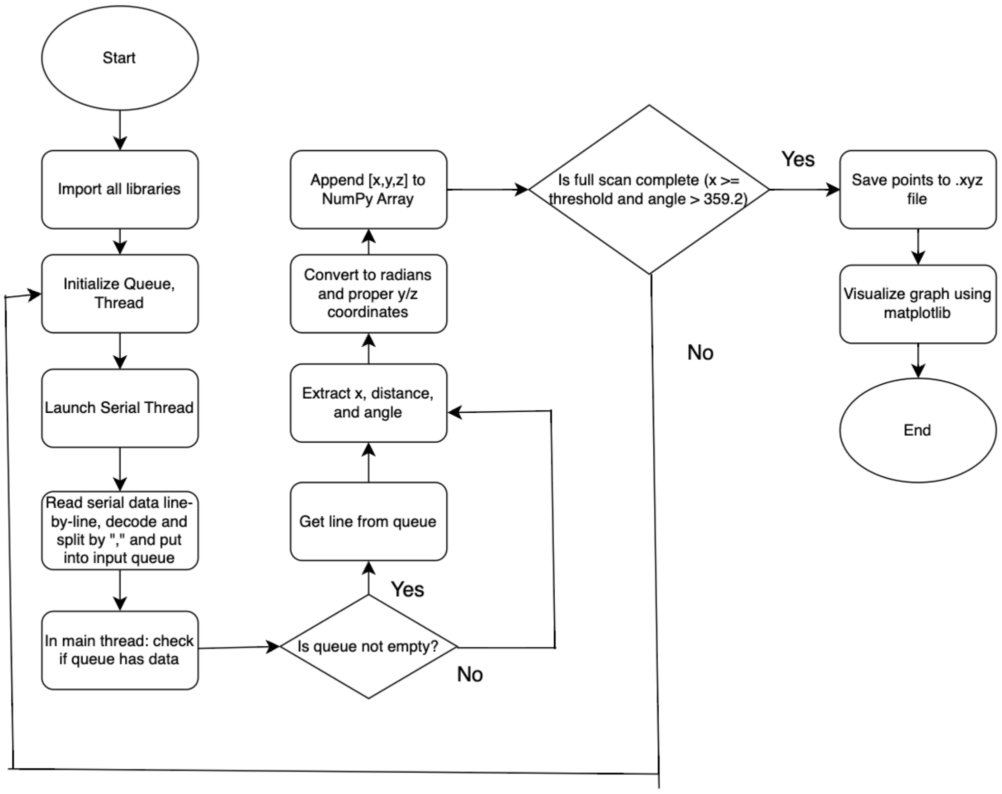

Start
Import all libraries
Initialize Queue, Thread
Launch Serial Thread
Read serial data line-by-line, decode and split by "," and put into input queue
In main thread: check if queue has data
Is queue not empty?
Yes
No
Get line from queue
Extract x, distance, and angle
Convert to radians and proper y/z coordinates
Append [x,y,z] to NumPy Array
Is full scan complete (x >= threshold and angle > 359.2)
Yes
No
Save points to .xyz file
Visualize graph using matplotlib
End

Figure 8: Python Code Flowchart

References

[1] McMaster University, *COMPENG 2DX3: Microprocessor Systems Project — Project Specification Document*, Winter 2025. Available: https://avenue.cllmcmaster.ca/d2l/le/content/638924/viewContent/5019689/View [Accessed: Apr. 7, 2025].

[2] STMicroelectronics, *VL53L1X Time-of-Flight Ranging Sensor Datasheet*, DocID031281 Rev 3, Nov. 2018. Available: https://avenue.cllmcmaster.ca/d2l/le/content/638924/viewContent/4995093/View [Accessed: Apr. 7, 2025].

[3] Texas Instruments, *MSP432E401Y SimpleLink™ Ethernet Microcontroller Datasheet*, SLASEN5, Oct. 2017. Available: https://www.ti.com/lit/ds/symlink/msp432e401y.pdf?ts=1744223308814 [Accessed: Apr. 7, 2025].

[4] STEPPERONLINE, *StepD-01 Stepper Motor Driver Datasheet*, [Online]. Available: https://www.mouser.com/datasheet/2/758/stepd-01-data-sheet-1143075.pdf. [Accessed: Apr. 7, 2025].

[5] Hadex, *Stepper Motor M513 Datasheet*, [Online]. Available: https://www.hadex.cz/spec/m513.pdf. [Accessed: Apr. 7, 2025].

[6] Abra Electronics, *JW-MM-20-6 Jumper Wires Connected 6" M-M (20 pack)*, [Online]. Available: https://abra-electronics.com/wire-cable-accessories/wirecable/jumper-wire-assembly/single-row-2-54mm-0-1/jw-mm-20-6-jumper-wires-connected-6-m-m-20-pack-jw-mm-20-6.html. [Accessed: Apr. 7, 2025].

[7] Texas Instruments, *SimpleLink™ Ethernet MSP432E401Y MCU LaunchPad™ Development Kit User's Guide*, SLAU748B, Rev. B, Sept. 2018. Available: https://www.ti.com/tool/MSP-EXP432E401Y [Accessed: Apr. 7, 2025].