Spatial Mapping Using Time of Flight

Diego Verdin

400504430

COMPENG 2DX3

Date: April 9, 2025

Statement of Originality: I hereby certify that all the work described within this report is the original work of the authors. Any published (or unpublished) ideas and/or techniques from the work of others are fully acknowledged in accordance with the standard referencing practices, and we have obtained the necessary permissions from the company to submit this report.

Table of Contents

1 Device Overview

1.1 Features

The device provides the user with an easy and simple of way of capturing and plotting 360-degree distance data using LiDAR.

VL531X Time-of-Flight Sensor for high precision distance and measurement:

- Accurate distance measurements with ± 20mm
- Up to 4 meters of range
- 2.6-3.5 Operating Voltage
- Uses I2C communication to transmit data to microcontroller
- 20-25$ CAD

MSP432E401Y Microcontroller

- Arm Cortex-M4F Processor
- 12MHz Bus speed
- 3.3V Operating Voltage
- 3 On-Board status LEDs
- 2 On-board push buttons
- About 60$ CAD

ULN2003 Driver Board and Motor

- 2048 steps for one rotation of the motor shaft
- 32 steps for one rotation of inner rotor
- Controlled rotation to perform y-z plane distance measurements

Serial Communication

- Microcontroller uses UART communication protocol to transmit data to PC
- 115200 BPS baud rate

3D Imaging

- Python script collects data from UART
- MATLAB plot library used to create a 3D reconstruction of the given data

1.2 General Description

This intelligent system performs 3D spatial mapping by rotating a ToF that vertically using a stepper motor and recording distance measurements across multiple, equally separated x-axis positions. The ToF is attached to the stepper motor using a custom designed and 3D printed piece. The microcontroller communicates with the ToF sensor via I2C protocol and the it then transmits the data to the PC over UART. On-Board LEDs provided visual feedback for several statuses: motor on/ff, data collection on/ff, data transmitting pulses. Each measurement cycle is manually triggered after horizontal displacement using on-board push buttons. Data is processed on a PC using python to produce a 3D model of the surrounding environment.

1.3 Block Diagram

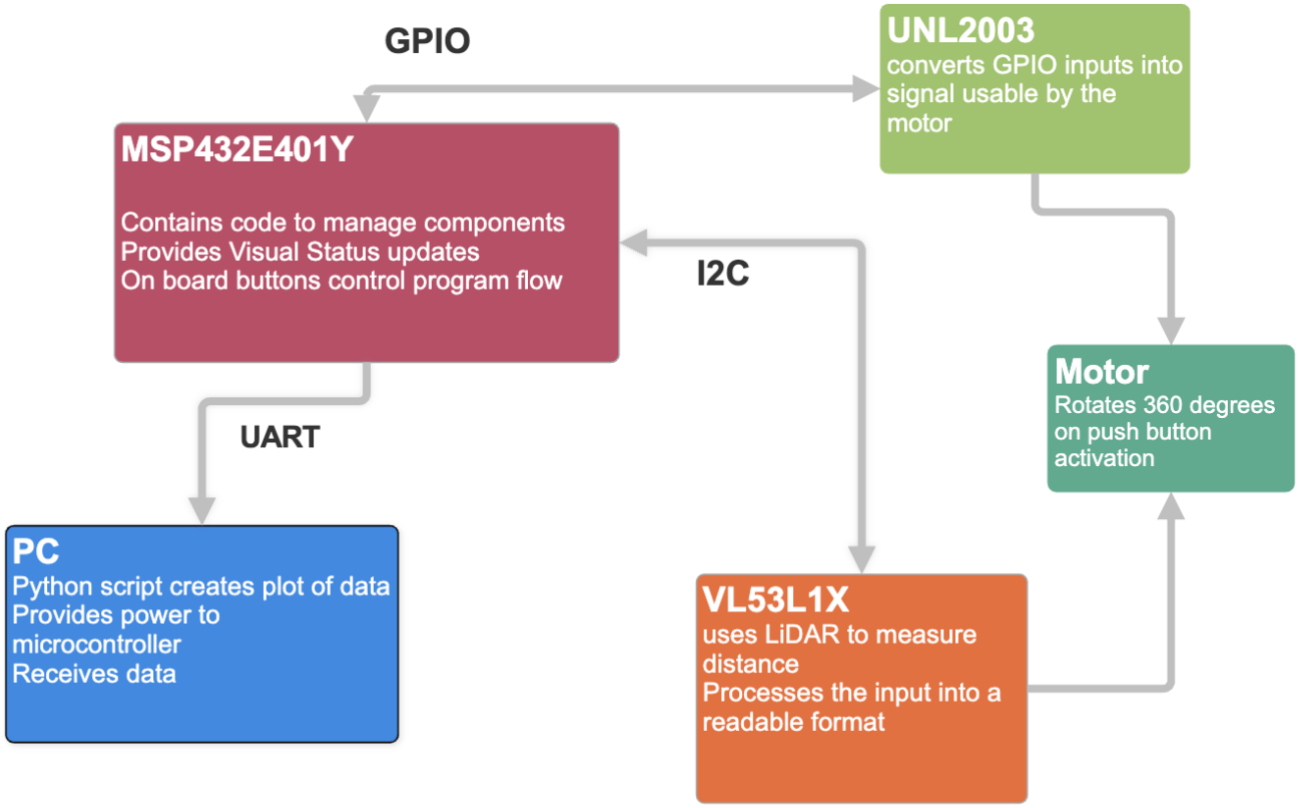

Figure 1: Block Diagram for the Device

2 Device Characteristics Table

Table 1 Device Characteristics

MSP432E401Y		UNL2003		VL53L1X	
Bus Speed	12MHz	**VDD**	5V	**VIN**	3.3V
UART Baud Rate	115200	**GND**	GND	**GND**	GND
Measurement Status	PF4	**IN1**	PM0	**SDA**	PB3
UART Tx	PF0	**IN2**	PM1	**SCL**	PB2
Motor Status	PN1	**IN3**	PM2	**SCL**	PB2
Motor ON/OFF	PJ1	**IN4**	PM3		
Measurement ON/OFFF	PJ0				

3 Detailed Description

3.1 Distance Measurements

TheVL53L1X Time-of-Flight sensor is compact in size has reliable accuracy and the ability to measure up to 4 meters. The sensor works by emitting an infrared wave out of it "emitter" and timing how long it takes for the wave to reflect back of a surface and arrive to the "receiver" which is located right under the emitter. The data acquisition system used a long-distance mode which allows to measure the max distance, but it is significantly affected by ambient light.

Each measurement begins when the user presses a button on the microcontroller. The button triggers the beginning of a 360-degree vertical scan using the stepper motor. The motor is driven through 32 evenly spaced angular positions (11.25-degree increments). Pausing momentarily at each position to allow for the sensor to collect data. The measured data, together with the corresponding angle, relative to the initial position, is sent to the PC through UART. Once one rotation finishes, the motor stops and the users must manually move the device a constant distance forward along the x-axis. The button to start the scanning needs to be pressed again to collect the next set of data.

On the microcontroller, the stepper motor is controlled using GPIO toggling. A stepping sequence energizes the coils in a half-stepping pattern. The system delays between steps to allow the motor to settle.

The ToF sensor is accessed over I2C. Configuration registers are written to set the distance mode, timing budget, and measurement frequency. A measurement command is issued, and the returned value is read using I2C read commands. The measurement command is formatted with the corresponding angle and sent to the PC. All ToF sensor control functions used are provided in the VL53L1 Core API provided by STMicroelectronics, specifically the platform code provided by the course coordinator, this code is unmodified.

An On-Board LED is used to flash every time there is a successful measurement that is sent through the UART.

The x-distance measurement is handled outside of the ToF and microcontroller logic. It assumes that the user is moving in constant intervals and said interval is modifiable in the python file which collects and interprets the data from the microcontroller.

3.2 Visualization

The raw data that the PC receives is parsed using Python. The initial distance and angle format of the data is not viable for the reconstruction. In order to address this the python script uses trigonometry to modify the data into cartesian coordinates:

```
# Convert to Cartesian
            angle_rad = math.radians(angle_deg)
            y = distance * math.cos(angle_rad)
            z = distance * math.sin(angle_rad)
            x = x_position  # simulated x based on scan batch
```

Figure 2: Python Snippet Converting Data to Cartesian Coordinates

After the conversion, the calculated points are stored into an in the form [x, y, z] into an array. These values represent the point in space where a surface was detected by the ToF sensor. Each time the program is run, 32 points are added to the array. The array is then saved into a Pandas DataFrame once the scanning has stopped. The users then have the option to toggle whether the data is saved into csv file or not. As mentioned earlier, the x-distance is not measured but rather simulated, the step size can be modified, and it assumes that the steps are evenly spaced and orthogonal to the vertical scanning plane.

The final stage is data visualization. The python script uses the matplot library, specifically Axes3D, to create a 3D plot of all the collected points. For each scan the 32 vertical points are plotted using the ax.plot() function to form a vertical curve along the y-z axis. To simulate the wall and give more context to the data corresponding points between scans are connected using grey lines:

```
ax.plot([x1, x2], [y1, y2], [z1, z2], color='gray')
```

Figure 3: Python Snippet Connecting Corresponding points along the x-axis

This creates a mesh-like effect which connects the vertical slices into a more understandable representation of a hallway.

This format of displaying data shows both the shape of the hallway as well as proper scale. The user can rotate, pan and zoom the 3D plot. This allows for visually confirming the accuracy of the scan and increases the ability to identify abnormalities.

Finally, the PC side of the visualization process was designed to be resilient to incorrect/non-sensical data. Any invalid lines were caught using try-except blocks and skipped.

This ensured for the scan to continue and maintain its quality without requiring restarts making the design more user friendly.

The complete software was a modular design. The microcontroller C code handles real time control and raw data transmission while the python script was responsible for logging the data and the 3D visualization. This approach made it easier to debug and improve either side without risk of breaking something.

The result is a robust system capable of mapping physical indoor environments to a relatively high accuracy and with a simple user interface.

4 Application Note

This section provides a comprehensive walkthrough to set up, operate, and visualize the data collected using the 2DX3 Spatial mapping system.

4.1 Required Software and Tools

To be able to operate the device and produce the desired plots, ensure the following software is installed in the host PC:

1. Python 3.10+:

Download and install python from https://www.python.org/downloads/.

2. Python Libraries:

Download and install the required packages using pip, these are used for serial data, managing tables and generation 3D spaces:

```
pip install pyserial pandas matplotlib numpy
```

Figure 4: pip command to install required packages

3. Serial Monitor (Optional):

Although python handles serial communications, tools like RealTerm can help with debugging if changes are to be made.

4. Keil MDK (if code is to be reflashed or modified):

Download Keil or any compatible IDE to flash the program unto the microcontroller.

5. XDS Emulation Software (if code is to be reflashed or modified):

Download the XDS Emulation Software pack from https://software-dl.ti.com/ccs/esd/documents/xdsdebugprobes/emu_xds_software_package_download.html.

6. Python Script:

Ensure that the datamapper.py file script is downloaded and can run in IDE of choice.

4.2 Hardware Setup

1. Power Connections:
    - Connect the MSP432E401Y via USB to the PC
2. Wiring:

- Refer to table 1 to wire the VL531X and the ULN2003 to the correct ports in the MSP432E401Y.

3. Mounting
    - Attach the VL53L1X sensor to the 3D printed piece shown in figure 5 using screws.
    - Attach the stepper motor and the ULN2003 driver board to the 3D printed piece shown in figure 6.
    - Connect the ToF sensors piece into the stepper motor

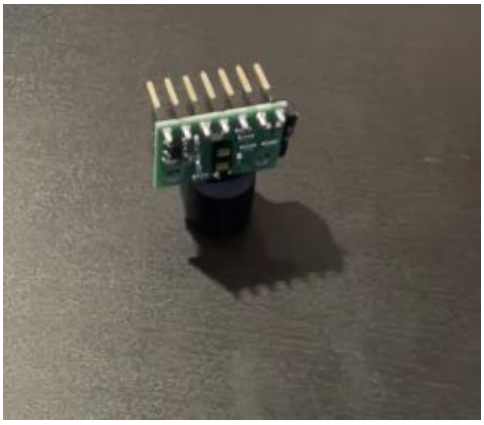

Figure 5 ToF Sensor Mount

Figure 6 Motor and ULN2003 Case

4.3 Operation Instructions

Step 1: Flash Microcontroller

- Open the firmware project in uKeil MDK.
- Compile and flash to MSP432E401Y using USB debugger.
- Press the Reset button on the microcontroller.

Step 2: Check UART Port

- Open device manager, check which COM Port UART is configured to

Step 3: Configure Python File

- Confirm that the COM port on datamapper.py line 7 is accurate.
- Change the X-step on line 10 to the horizontal displacement, in millimetres, that will be travelled between scans.
- Toggle line 11 to produce a csv file if wanted.

Step 4: Beginning Scanning

- Run the datamapper.py file
- Press button PJ0 to enable ToF measurement.
- Press button PJ1 to enable motor rotation.
- The motor will rotate vertically in 32 steps, collecting data at each. After that the rotation will stop and the python file will wait for user input.

Step 5: Displace the System Manually

- Move the assembly forward by the distance specified in step 3.
- Once in the position, press J0 to enable ToF measurement.
- On the PC press enter to resume the python script.
- Press J1 to start the rotation.

Step 6: Repeat

- Repeat steps 4 and 5 for as many horizontal displacements as needed to map the desired hallway.

Step 7: Stop Data Acquisition

- Press Ctrl+C in the terminal once the scanning is complete.
- A window showing the 3D map will be generated as well as a csv file with the data.

4.4 Expected Output

Live Terminal output:

```
Listening for ToF data... (press Ctrl+C to stop)
```

```
Raw line: '2940, 0'

Raw line: '3120, 11'

...

Advanced to x = 1000 mm

Press Enter to continue to next scan...
```

Figure 7: Expected Terminal Output when running the datamapper.py

CSV Output Format:

```
x,y,z

0,2894.7,0.0

0,3067.2,592.2

...
```

Figure 8: Sample CSV Output

3D Plot Output:

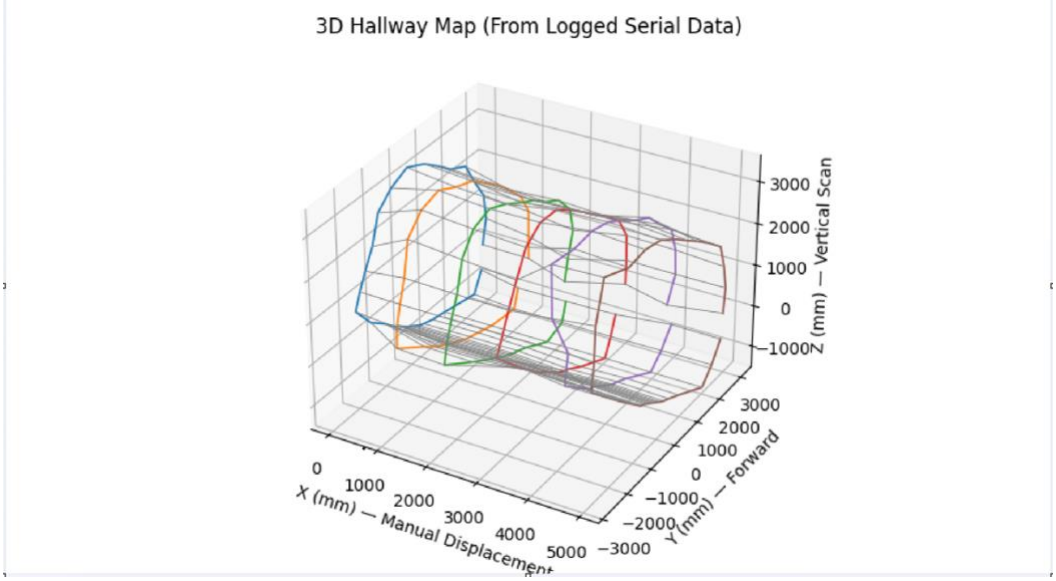

Figure 9: Scanned 3D Hallway

Figure 10: Hallway Scanned in Figure 7

5 Limitations

Trigonometric functions can be important through the math.h C library. However, trigonometric functions usually result in large decimal results which could be a problem for a

low-level systems. Fortunately, the microcontroller contains a Floating-Point Unit that allows 32-bit floating point operations. However, complex operations like trigonometric functions are computationally expensive and as such it was decided to perform said calculations in the host PC using python. It would be possible, however, to implement these operations in the microcontroller itself.

The maximum quantization error can be found with:

$$Max\ Quantization\ Error = \frac{Max\ Input}{2^{16}}$$

The bottom term is to the power of 16 as our ADC is 16 bits large. Since the ToF is set to long mode, the maximum input it can send is 4000mm. Therefore:

$$Max\ Quantization\ Error = \frac{40000}{2^{16}} = 0.061035$$

The maximum standard serial communication rate on PC is 128000 bits per second. This was verified through the device manager COM port settings. The speed that was used in the project is 115200 bps. This was verified by making sure the initialization of the UART

$$Baud\ Rate = \frac{PIOSC\ Clock}{16 \times (IBRD + \frac{FBRD}{64})}$$

```
UART0_CTL_R &= ~UART_CTL_UARTEN;      // disable UART
UART0_IBRD_R = 8;                     // IBRD = int(16,000,000 / (16 * 115,200)) = int(8.681)
UART0_FBRD_R = 44;                    // FBRD = round(0.6806 * 64) = 44
                                      // 8 bit word length (no parity bits, one stop bit, FIFOs)
UART0_LCRH_R = (UART_LCRH_WLEN_8|UART_LCRH_FEN);
                                      // UART gets its clock from the alternate clock source as defined by SYSCTL_ALTCLKCFG_R
UART0_CC_R = (UART0_CC_R&~UART_CC_CS_M)+UART_CC_CS_PIOSC;
                                      // the alternate clock source is the PIOSC (default)
SYSCTL_ALTCLKCFG_R = (SYSCTL_ALTCLKCFG_R&~SYSCTL_ALTCLKCFG_ALTCLK_M)+SYSCTL_ALTCLKCFG_ALTCLK_PIOSC;
```

Figure 11: UART_Init Snippet showing Baud Rate Parameters

$$Baud\ Rate = \frac{16{,}000{,}000}{16(8 + \frac{44}{64})} = 115107.9$$

Communication with the VL53L1X sensor was established via the I2C protocol. The I2C speed was configured at 100Kbps. This bandwidth is sufficient for infrequent reads.

The primary limitation on system speed is the stepper motor. The motor is the bottleneck as each call of the function that energizes the coil adds a delay of 10ms. This function is called

2048 times in one cycle and as such it is easily the slowest part of the process. If the delay is set any lower, so that the motor does not spin anymore, the program runs faster. As such this is the limitation. The ToF sensor also adds a significant delay, having the functionality to rotate the motor without taking measurements showed this; when not measuring a 360 degree rotation is smooth as it does not need to pause every 11.25 degrees to measure.

The system clock can be derived from the main crystal oscillator and the PLL.

$$System\ Clock = \frac{fVCO}{PSYDIV + 1}$$

Since the fVCO is defaulted to 480,000,000 and the assigned system clock is 12,000,000 PSYDIV was set to 39.

6 Circuit Schematic

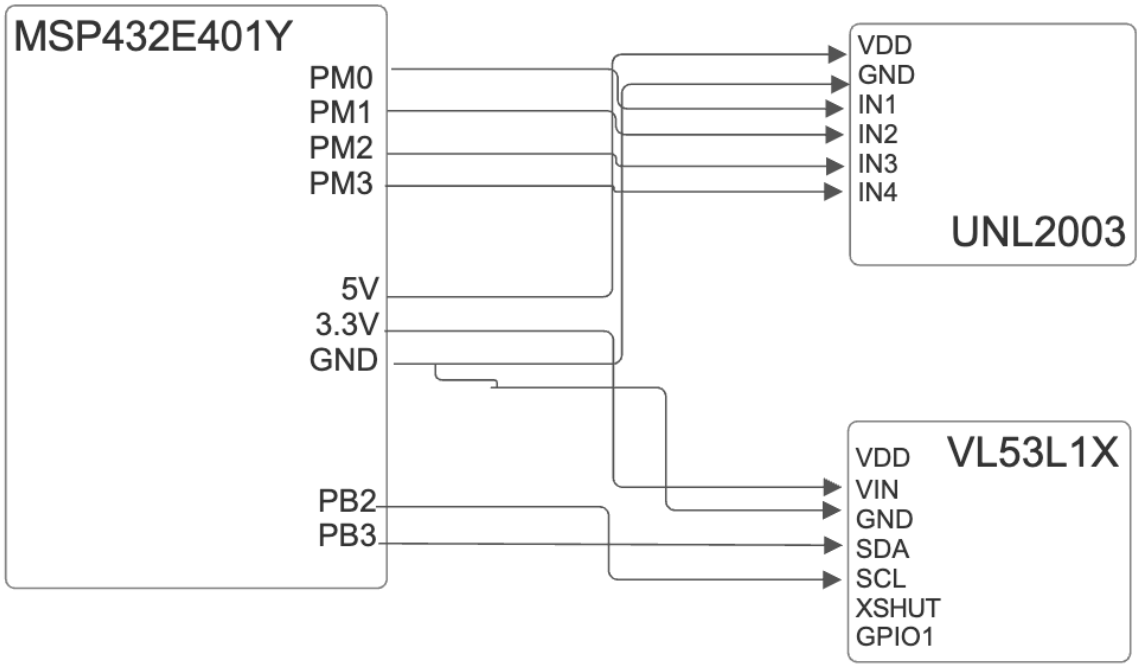

Figure 12: Circuit Schematic

7 Program Logic Flowchart

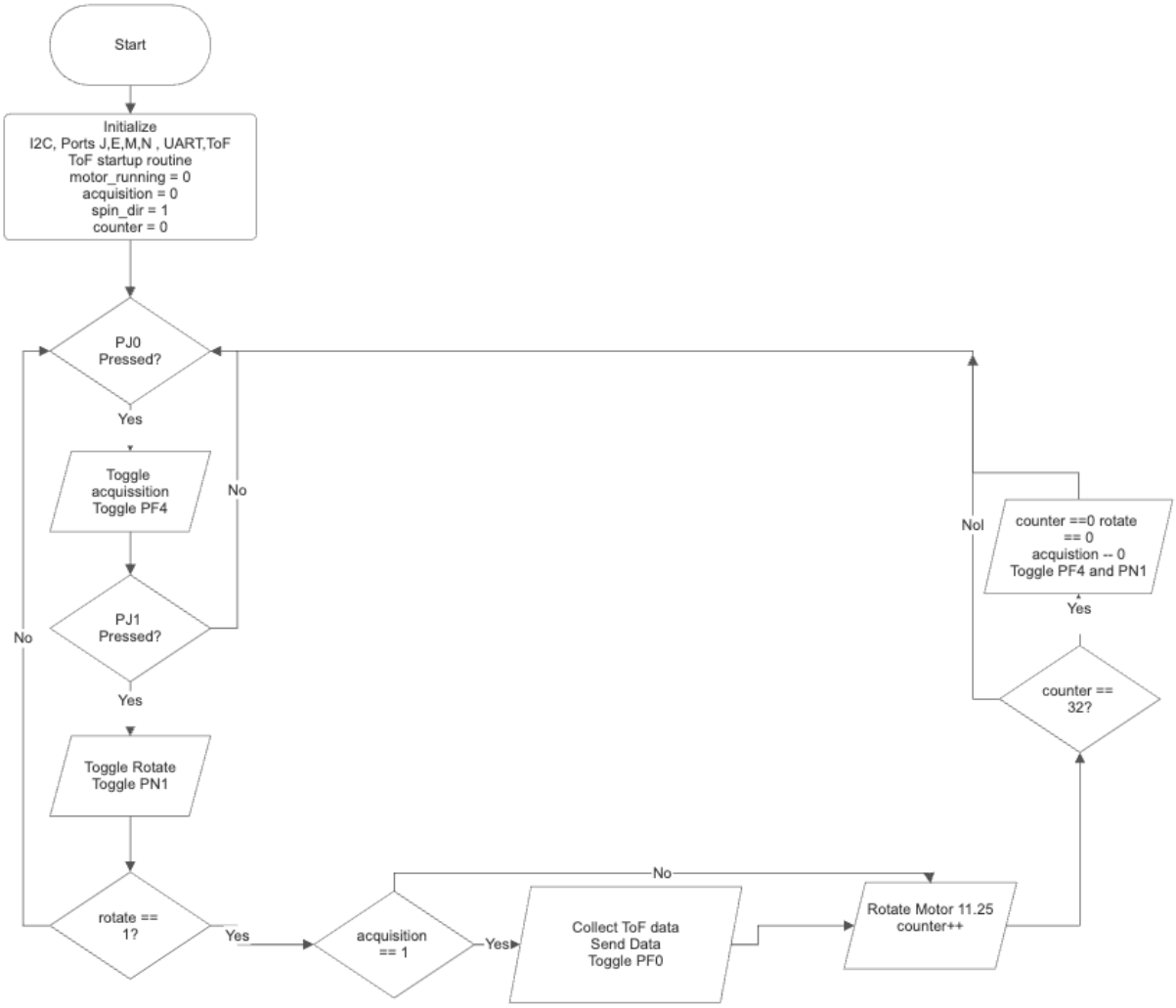

Figure 13: Program Logic Flowchart

8 References

1. Texas Instruments, *MSP432E401Y SimpleLink™ Microcontroller Technical Reference Manual*, Rev. C, Nov. 2021. [Online]. Available: https://www.ti.com/lit/pdf/spmu477
2. STMicroelectronics, *VL53L1X Time-of-Flight Ranging Sensor Datasheet*, Rev. 6, Jul. 2021. [Online]. Available: https://www.st.com/resource/en/datasheet/vl53l1x.pdf
3. STMicroelectronics, *VL53L1X API User Manual*, UM2555, Rev. 3, Oct. 2021. [Online]. Available: https://www.st.com/resource/en/user_manual/um2555-vl53l1x-api-user-manual-stmicroelectronics.pdf