COMPENG 2DX3
Final Deliverable Report

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

Ryan Li – li2323 – 400451015 – L01

As a future member of the engineering profession, the student is responsible for performing the required work in an honest manner, without plagiarism and cheating. Submitting this work with my name and student number is a statement and understanding that this work is our own and adheres to the Academic Integrity Policy of McMaster University and the Code of Conduct of the Professional Engineers of Ontario. Submitted by **[Ryan Li, li2323, 400451015]**

Device Overview: Features

The system provides a simple interface, consisting of only 1 external push button, and 2 onboard LEDs that indicate the state(s) of the system.

Texas Instrument MSP-EXP432E401Y

- Features a 32-bit Arm Cortex M4F Processor Core
- 256 Kb SRam
- 1024 Kb Flash Memory
- Bus speed of 30 MHz
- On-board LEDs and GPIO pins

28BYJ-48 Stepper Motor

- 4 Phases per step
- 2048 steps (Equivalent to 512) for 360 degrees rotation
- 4 On-board LEDs to indicate current phase
- 5 - 12V supply

VL53L1X Time of Flight (ToF) Sensor

- 2.6 – 3.5V operating voltage
- 50 Hz operating frequency
- Top end range of 4m

One off-board Push Button

- 5V supply voltage
- Button connected to PM0 of microcontroller enables full operation of device, performs a 360-degree rotation and taking information every 11.25 degrees.
- PN1 (LED1) blinks every 11.25 degrees to indicate measurement and PF4 (LED3) toggles device status (On or off)

Serial Communication

- I^2C protocol used for communication between ToF sensor and microcontroller
- UART protocol used for communication between microcontroller and PC
- 115200 BPS baud rate used for UART communication

3D Visualization

- Python used for serial communication between PC and microcontroller
- Python reads data, plots points, and constructs 3D model through Open3D

Device Overview: General Description

This device is an integrated 3D scanning system that records distances along the YZ plane every 11.25 degrees. The recorded data is then processed to produce a 3D visualization of the space that was mapped. The entire operation only requires the use of one push button.

The system is comprised of a microcontroller, a stepper motor, and a time-of-flight (ToF) sensor. The microcontroller manages all operations of the system besides the 3D visualization. The microcontroller provides power to other components of the system and transmit recorded data to an external device via serial communication. The microcontroller dictates a bus speed of 30 MHz. The stepper motor provides a full 360-degree range of motion, allowing for the mounted ToF sensor to record distance measurements in a full vertical plane. Each time the ToF sensor captures data, LED1 (PN1) will flash on the board. To prevent wires from getting caught, the stepper motor rotates 360-degrees in the opposite direction after scanning 360-degrees.

The ToF sensor determines the distance measurements by emitting a beam of light and measures the time it takes for the beam to reflect to the sensor. This can be mathematically represented through the formula:

$$\text{Measured Distance} = \frac{Photon\ Travel\ Time}{2}\ x\ Speed\ of\ Light$$

The read measurement data is sent to the board via I2C communication and then sent to an external PC via UART communication. The measurement data is then read through a Python code that extracts the YZ components through trigonometry while X coordinates are manually incremented, converted to XYZ coordinates and stored in a .xyz file, then produces a 3D mapping and visualization.

Device Overview: Block Diagram

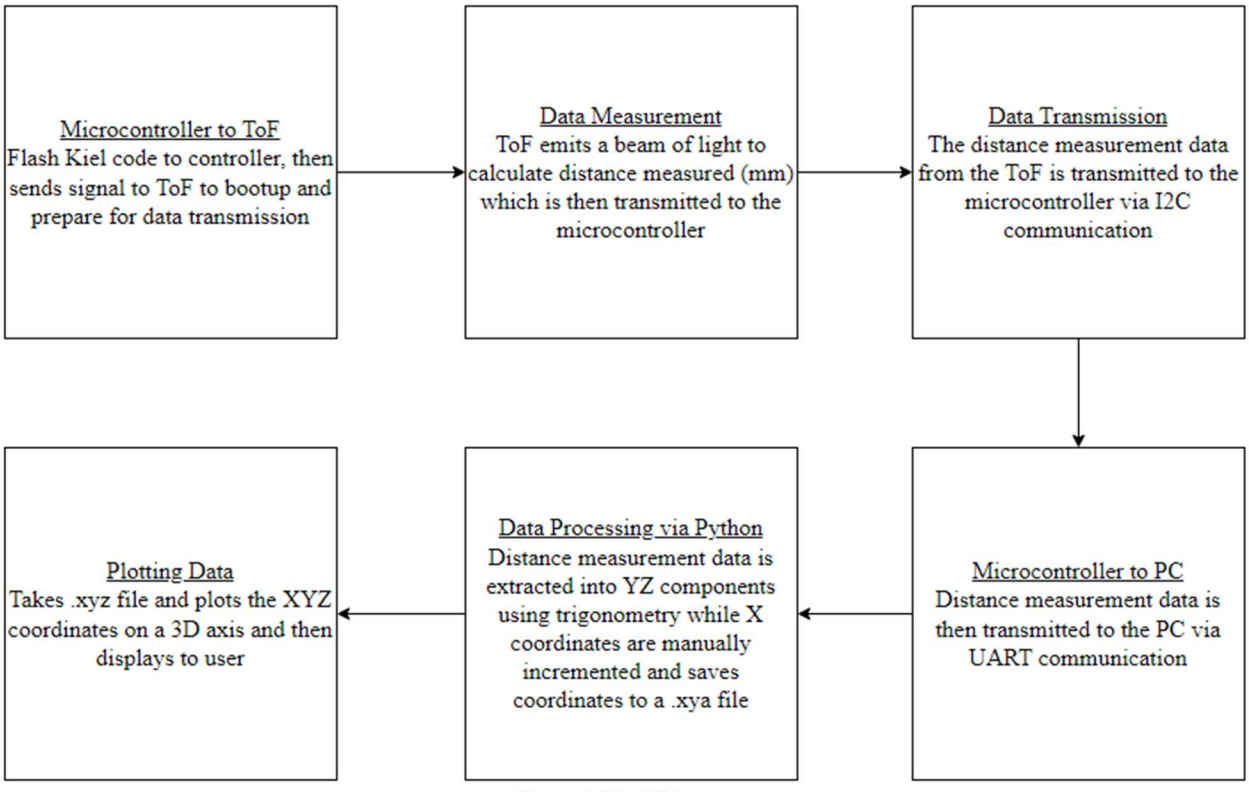

Figure 1: Block Diagram

Device Characteristics

MSP432E401Y		28BYJ-48 Motor		VL53L1X ToF	
Feature	Detail	Device Pin	Microcontroller Pin	Device Pin	Microcontroller Pin
Bus Speed	30 MHz	V+	5V	VDD	
Serial Port	COM4	V-	GND	VIN	3.3V
Baud Rate	115200	In1	PH0	GND	GND
Measurement and Status LED(s)	PN1, PF4	In2	PH1	SDA	PB3
		In3	PH2	SCL	PB2
		In4	PH3	XSHUT	
				GPIO1	

Table 1: Device Characteristics

Detailed Description: Distance Measurement

Distance measurements are performed by the VL53L1X Time-of-Flight sensor. The ToF sensor takes a measurement of every step of the stepper motor while the motor performs a 360 degrees rotation. This allows the ToF sensor to measure a full 360 degrees plane.

The ToF sensor performs distance measurements by emitting pulses of infrared light. This beam of light reflects off a surface nearby and back to its receiver and measures the time delay between emitting the beam of light and reception of the beam. The ToF sensor obtains the distance measurement through the formula $D = \frac{Flight\ Time}{2} x\ speed\ of\ light$. Following this measurement, the ToF sensor performs a variety of operations including transduction, conditioning, and Analog-to-Digital Conversion. This results in a digital representation of the analog distance measurement. The data is then transmitted to the microcontroller through the I^2C serial communication protocol.

The ToF settings used in the initialization of the sensor are the default configurations provided in the VL53L1 Core API. The ToF functions used in the system are provided in the VL53L1 Core API distributed by STMicroelectronics or in a base code provided by the course coordinators, and implementation of the functions remain unmodified.

In the initial microcontroller initialization process, the I^2C capabilities of the microcontroller are initialized to allow the microcontroller to communicate with the ToF sensor. The necessary variables used for the ToF functions are also initialized. To activate the the ToF sensor and initiate ranging, the ToF sensor boot function is invoked in the Kiel C program. Although the sensor is active and ready to take measurements, the process does not occur immediately. The microcontroller waits for a trigger from its peripherals through polling, with the use of only 1 external push button connected to PM0. The push button actives the stepper motor to rotate the stepper motor a full 360 degrees and toggles the additional LED status. During this process, the ToF takes measurements every 11.25 degrees, such that we receive 32 steps of measurements in a 360 degrees plane.

In the main block of the C code, we can control the number of distance measurements in each 360-degree plane and the depth. The first check is to determine whether the motor has rotated 11.25 degrees. This is done by tracking the total number of steps with the modulo of 64. Although one 360-degrees rotation is 512 steps, the code increments the steps by 1 for each phase in the stepper motor, therefore the number of total steps is quadruple. This method is optional but in this case, we will receive a total of 2048 steps for a full 360-degrees rotation. Through calculations, 2048 divided by 64 is 32, which is the 32 measurements taken for each 360-degrees. The program checks whether the motor has rotated 11.25 degrees. If it passes, the measurement LED status flashes and takes a distance measurement. Within Keil, the data transmitted is only the Distance, as information such as RangeStatus and SpadNum are not relevant. Therefore, only the Distance measurement is transmitted to the board through I^2C, which is then transmitted to the PC via UART. The scans variable controls the depth or the number of 360-degrees rotations the program and stepper motor runs. Once all the scans are complete, the program itself will stop.

The measurement data is processed by the PC through the **data_collect_display.py** python script to write the data into .xyz file format. The python script receives the data sent through UART from the microcontroller by using the pySerial library. The pyserial is connected to COM4 port of the computer, which is the port of the microcontroller for serial communication. The Y and Z components of the data are calculated using distance measurements by applying simple trigonometry on the data. The angle of the motor can be determined by counting the number of steps recorded and incrementing by 11.25 each time, then multiplied by $\pi/180$. The Y and Z components were calculated by $b\,cos\theta$ & $b\sin\theta$ respectively. The X component is manually incremented and can be changed based on preference in the python code.

Detailed Description: Visualization

The 3D visualization occurs within the same python code of **data_collect_display.py**. The 3D visualization portion is run after all the data collection is finished and converted into the .xyz file. The Open3D library's functions are used to visualize the data, which includes functions to read and format the data to an appropriate format for visualization. A point cloud was created using an Open3D function. The point cloud contains all the points that will be presented in the visualization. The point cloud is used to create a line set that contains all the points in the point cloud and connects the points by appending the points with each point such that the lines are present in the visualization to display a 3D figure. The lines contain all individual points in one plane or slice, with several lines connecting slices together. The resulting visualization is shown further into the report. All the data collected from the scan can be found in the file 2dx3-Copy.xyz along with two python codes, one to run data collection and 3D visualization and the other to only run 3D visualization if data collection already happened.

Application Note

The integrated lidar scanning system uses infrared laser light to measure distances and create 3D mappings of environments. A primary application of the lidar system is in mapping and surveying, where it can create high-resolution digital elevation models. This can be applicable to floor modelling, urban planning, and resource management. This may also be applied to autonomous vehicles as real-time data of the vehicle's surroundings may aid in obstacle detection and increased safety.

Instructions

The instructions and setup procedures assume that the user has all the necessary software and applications installed and configured for their system. This includes Kiel Development Environment, Python, pyserial, an IDE for Python, Open3D, and configuration and setup of the MSP-EXP432E401Y microcontroller.

1) Plug microcontroller into the computer and determine COM port. To determine COM port, go into 'Device Manager' and scroll down to 'Ports (COM & LPT)' and expand by clicking the arrow. Take note of the # in the statement 'XDS110 Application/User UART (COM #)'

2) Open the python script **data_collect_display.py** in a desired IDE. Change the value of COM# in line 4 'ser = serial.Serial('COM#', 115200, timeout = 10)' such that # is the port number you are connected to. Once changed, serial communication is set up.

3) Configure circuit by following **Table 1: Device Characteristics**. May also follow the **Figure 4: Circuit Schematic**.

4) Once properly connected and wired, open the Kiel project.

5) In both the Kiel C code and python code, change to desired depth. In the Python code this can be found on line 14 and in Kiel on line 224. The numbers should match or else the code does not work. (eg. Kiel: while(scans < #) and Python: depth = #)

6) Translate, build, and load the main C code. This can be found near the top left section of the interface.

7) Flash the project onto the board by clicking the reset button.

8) Run Python Code. Make sure there isn't a duplicate 2dx3data.xyz file. If so, rename the file or delete.

9) Press the external push button to initiate the entire process. No further actions are needed. The entire system will stop on its own after running for the # of steps set.

10) Once finished, the point cloud displays. Close that window for 3D visualization with lines.

Expected Output

The expected output of the device can be visualized below by comparing my assigned hallway location to the 3D visualization output. The assigned scan location was location F, located in the JHE second floor to Gerald Hatch Centre as displayed in Figure 2.

As seen by the final scan Figure 3, the system can capture the details of the location. The 3D visualization highlights the depth of the elevator on the left, and the narrower hallway past the elevator.

Figure 2: Assigned Scan Location

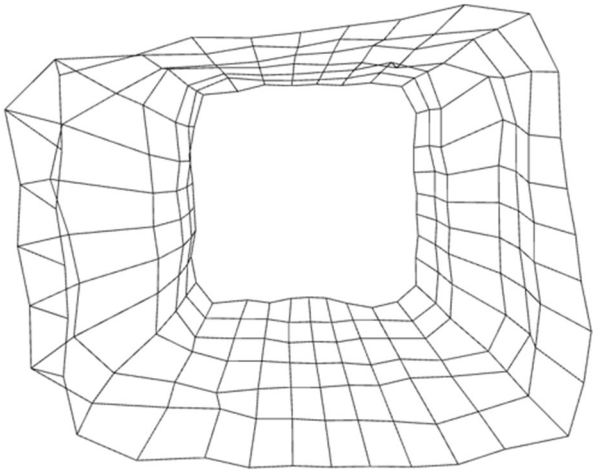

Figure 3: Open3D Visualization

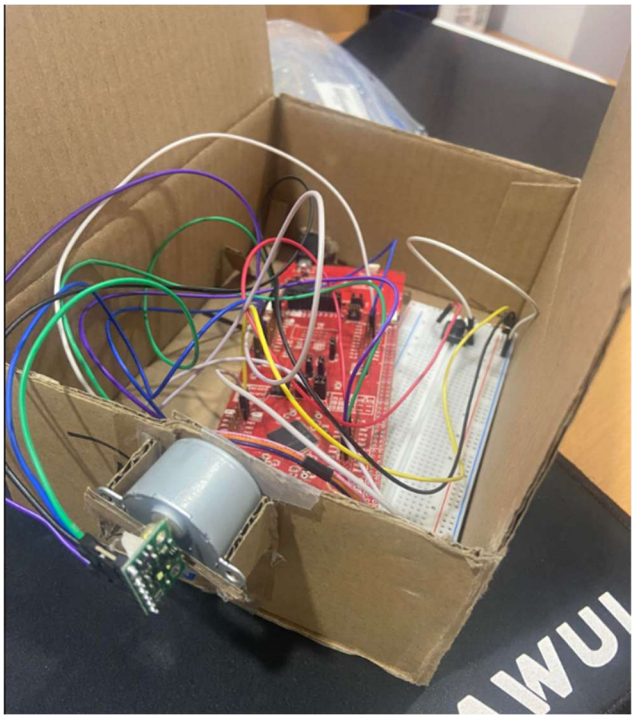

Figure 4: Physical Mechanism fully assembled

Limitations

1) The microcontroller is equipped with a Floating-Point Unit (FPU) that supports single-precision (32-bit) simple mathematical operations. The FPU can also perform conversions between floating-point data. There should be no issues with the use of trigonometric functions as they can be performed on float variables from the Python math.h library.

2) The maximum quantization error for the ToF can be calculated with $Max\ Quantization\ Error = \frac{Max\ Reading}{2^{ADC\ bits}}$. Therefore, the maximum quantization error of the ToF sensor is $\frac{4000mm}{2^{16}} = 6.10\ x\ 10^{-2}mm$

3) The maximum standard serial communication rate that can be implemented with the PC is 128000 bps. This was observed in the PC's device manager and clicking on the properties of the XDS110 UART Port.

4) The communication between the microcontroller and the ToF module uses I^2C serial communication. The ToF sensor has a maximum transmission speed of 50Hz.

5) There are also limitations on the speed. This includes the stepper motor and the ToF sensor. The stepper motor speed limitation can be adjusted through the delay between each step, but based on the configuration of the current code it is not much of a limitation. The ToF in this system scanned decently quickly but that is primarily due to

the assigned bus speed and set delay in the code. This can differ very much based on other configurations.

6) The assigned bus speed for this system is 30MHz. To configure this bus speed, the first approach is through the **PLL.h** file within the source code folder. To set the assigned bus speed, the correct PSYDIV number is used. After implementing the correct bus speed, a change in the **SysTick_Wait10ms()** function under **SysTick.c** is required. To receive a delay of 10ms with the assigned bus speed, the calculation of **SysTick_Wait()** is required. Since the bus speed is 30MHz, the delay parameter is $\frac{1}{30MHz} = 33.33ns$. To acquire 10ms, $33.33ns * x = 0.01s$ where x is found to be 300,000. This value of x is then used within the parameters of **SysTick_Wait()** under the **SysTick_Wait10ms()** function to acquire a 10ms delay.

Circuit Schematic

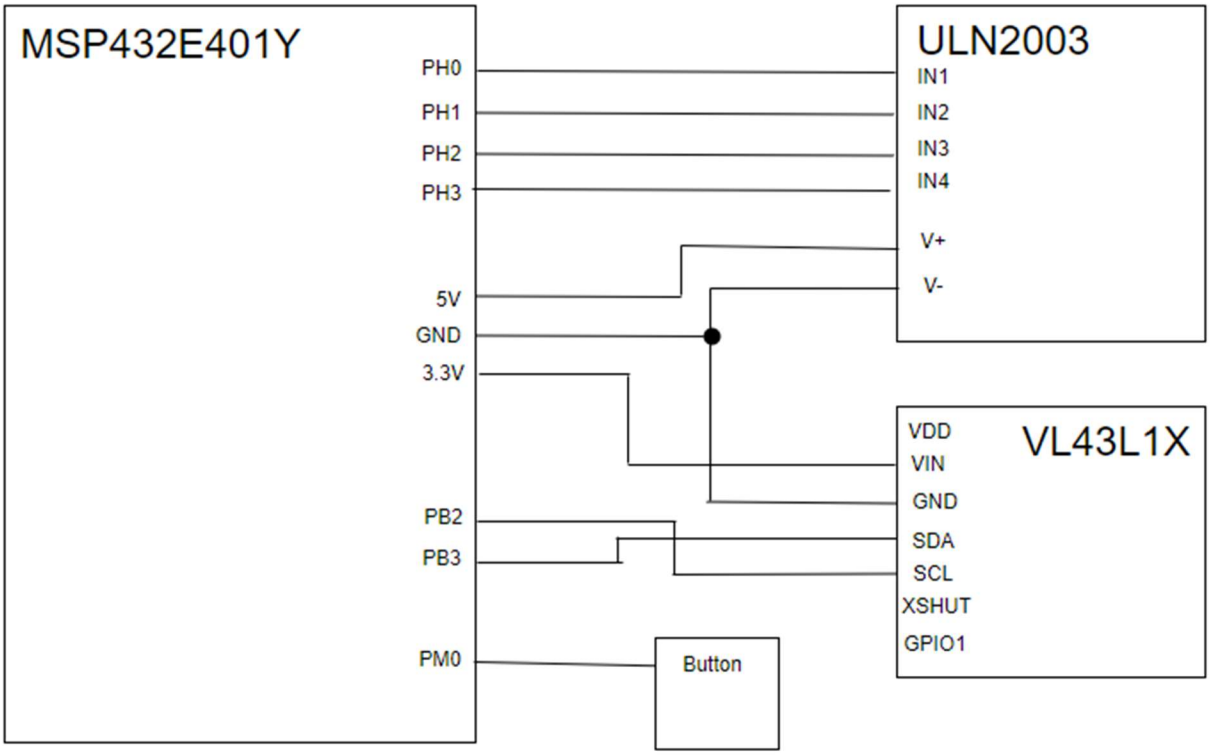

Figure 4: Circuit Schematic

Programming Logic Flowchart

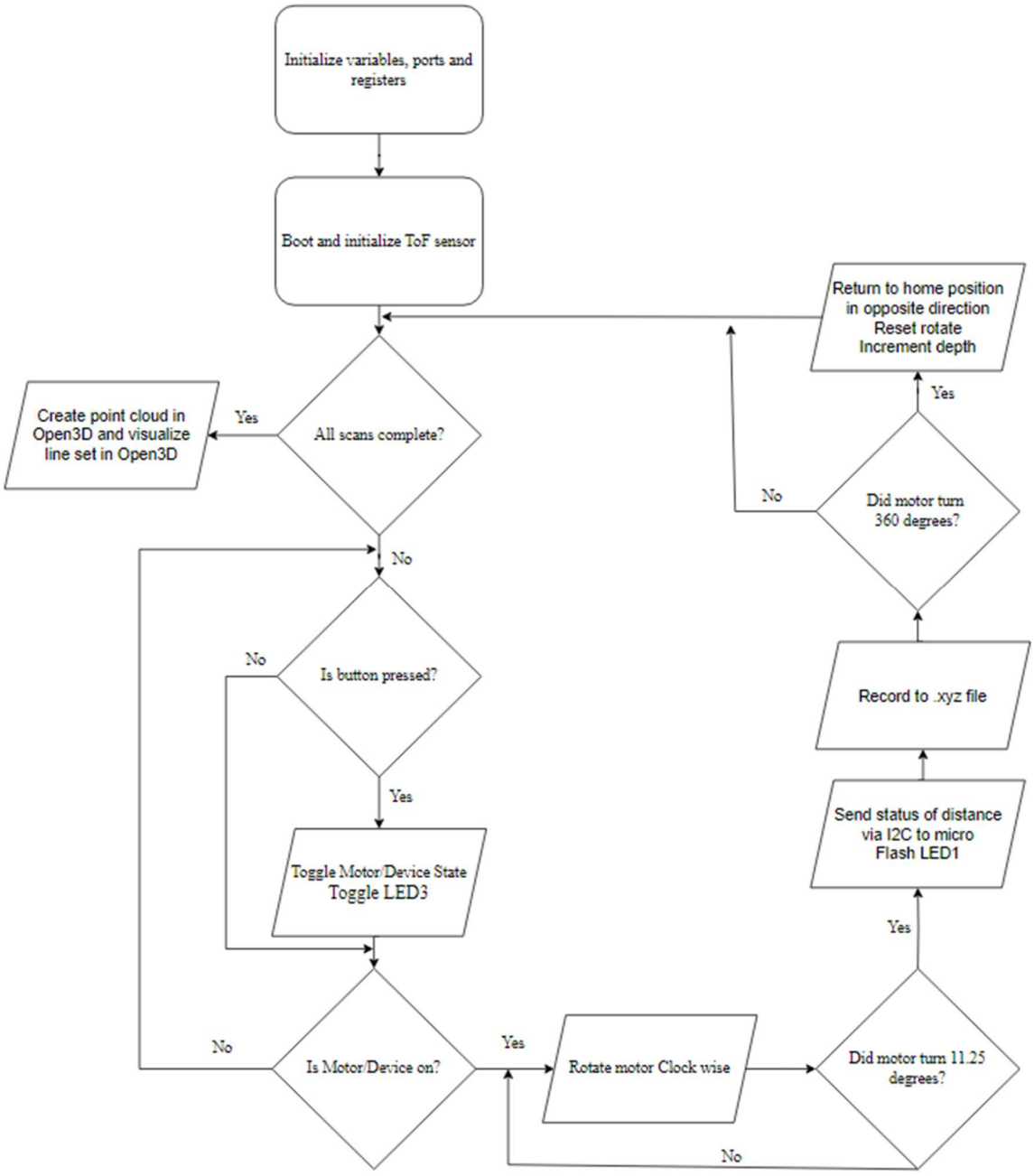

Figure 5: Programming Logic Flowchart