

2DX: Microprocessor Systems

Final Project

Instructors: Drs. Boursalie, Doyle, and Haddara

Arji Thaiyib – thaiyiba – 400336020 – 2DX3 – Thursday Evening – L04

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 my own and adheres to the Academic Integrity Policy of McMaster University and the Code of Conduct of the Professional Engineers of Ontario. Submitted by [**Arji Thaiyib, thaiyiba, 400336020**]

1. Device Overview

1.1 Features: The spatial mapping using ToF device was composed of the following:

Simple Link MSP-EXP432E401Y Microcontroller:

- **Cost:** \$65.53
- **Bus Speed:** 16 MHz using PIOSC and 25MHz using MOSC (can go up to 120MHz using the PLL)
- **Operating Voltage:** 3.3V
- **Terminator:** 1 independent reset button for the MCU (**double check if this is classified as a terminator**)
- **Connector Ports:** Micro USB 2.0 A/B connector
- **User LEDs:** 4 onboard LEDs (D1-D4)
- **Why do we need it?** This serves as the motherboard of the device.

User Switch (onboard button, PJ1):

- **What does it do?** This button was used to start the measurement readings every x (displacement) iteration.

VL53LIX Time-of-Flight Distance Sensor:

- **Cost:** \$11.95
- **Communication Protocol:** I²C
- **Operating Voltage:** 3.3V
- **What does it do?** The ToF sensor is used to take distance measurements.

MOT-28BYJ48 Stepper Motor:

- **Cost:** \$8.03
- **Operating Voltage:** 5V
- **Why do we need it?** Stepper Motor controller
- **What does it do?** The stepper motor rotates the ToF sensor in a 360° revolution.

Active LO Button Configuration:

- **Operating Voltage:** 3.3V
- **Why do we need it?** Serves as a stop button (reset, not pause) for the ToF sensor's readings and stepper motor's rotation.

Open3D Graphing Software:

- **Why do we need it?** This software is used to visualize the ToF sensors' readings.

Communication Used from MCU to Computer via UART and Python:

- **Language Used:** Python

- **Baud Rate:** 115 200
- **What does it do?** The UART communication protocol is used to store measurements on my personal computer from the MCU.

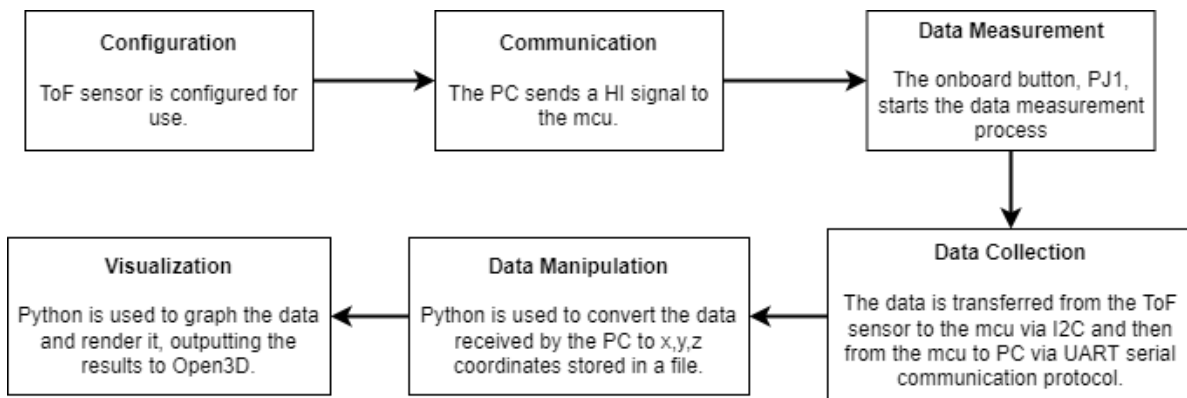
1.2 General Descriptions:

As mentioned above, the motherboard for this specific design is comprised of the Simple Link MSP-EXP432E401Y which is a 32-bit ARM Cortex-M4F based microcontroller and can perform multiple required tasks. It enables the configuration of the ToF sensor and by using the I²C communication protocol the ToF sensor's distance measurement readings can be communicated with and collected by the microcontroller. Every 45° rotation of the stepper motor the sensor collects data for a total of 8 readings with the onboard LED (specific to my student number, D3) lighting up with each iteration, as a representation of the ToF's sensor's status during this process. This by convention would also mean the MCU is responsible for the stepper motor, as it will rotate at a specific frequency, for a total of 3 complete 360° revolutions.

When the onboard button, PJ1, is pressed it starts the next set of readings after a complete rotation has finished and 8 readings collected (at a new set displacement, or x-value). Moreover, the MCU is able to communicate with my personal PC via UART communication protocol. Lastly, I also implemented a logic LO push button, connected to a 3.3V source, to completely stop the stepper motor and ToF sensor, effectively being a reset button.

The MOT-28BYJ48 Stepper Motor, connected to a 5V input voltage and Port H, is also used in the design configuration to rotate the ToF sensor a complete 360° revolution. Speaking of which, the VL53LIX Time-of-Flight sensor is used to measure distance using LIDAR technology. It calculates the amount of time a photon of light takes to bounce off of an object and be reflected back; and from this time measurement the distance can further be calculated. The sensor as mentioned previously will be attached the stepper motor and take measurements every 45°.

1.3 Block Diagram:



Arji Thaiyb
thaiyiba

1.4 Picture of Entire Device Put Together and Displaying Information:

***Note:** I am currently away from Hamilton and did not have my setup readily available, so I had to use the images for the setup and results taken prior, for my Interview Demo. I apologize for the inconvenience.

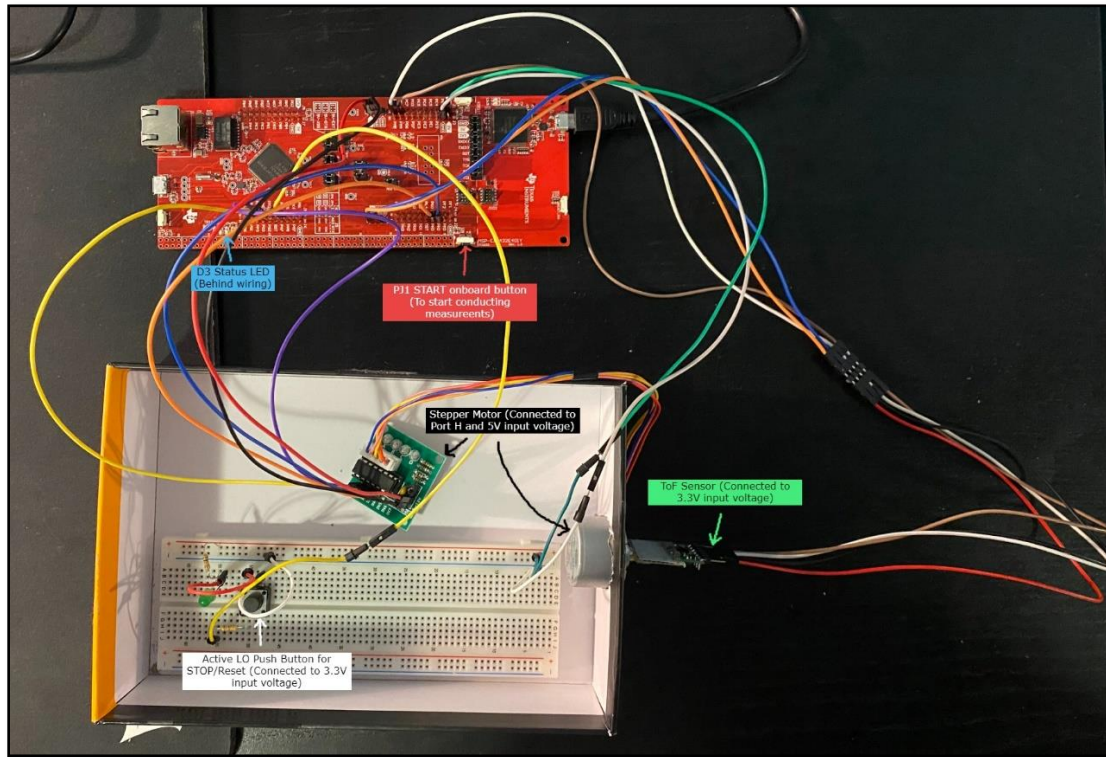


Figure 1: Circuit Configuration with all Components

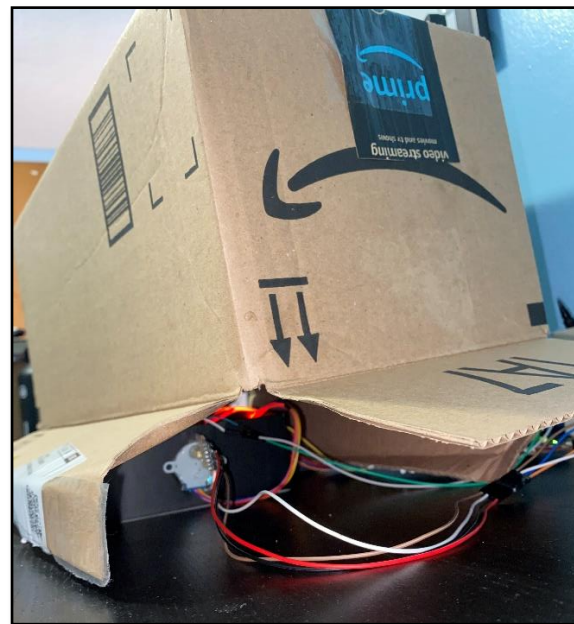


Figure 2: Cardboard Box Used as a Makeshift Hallway

1.5 Screenshot of PC Display:

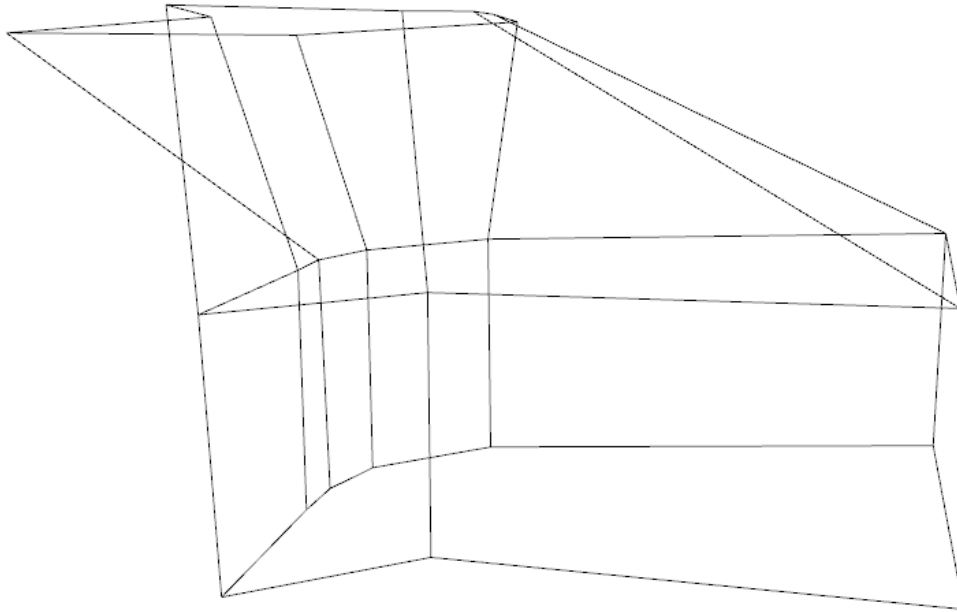


Figure 3.1: Open3D Render (View 1)

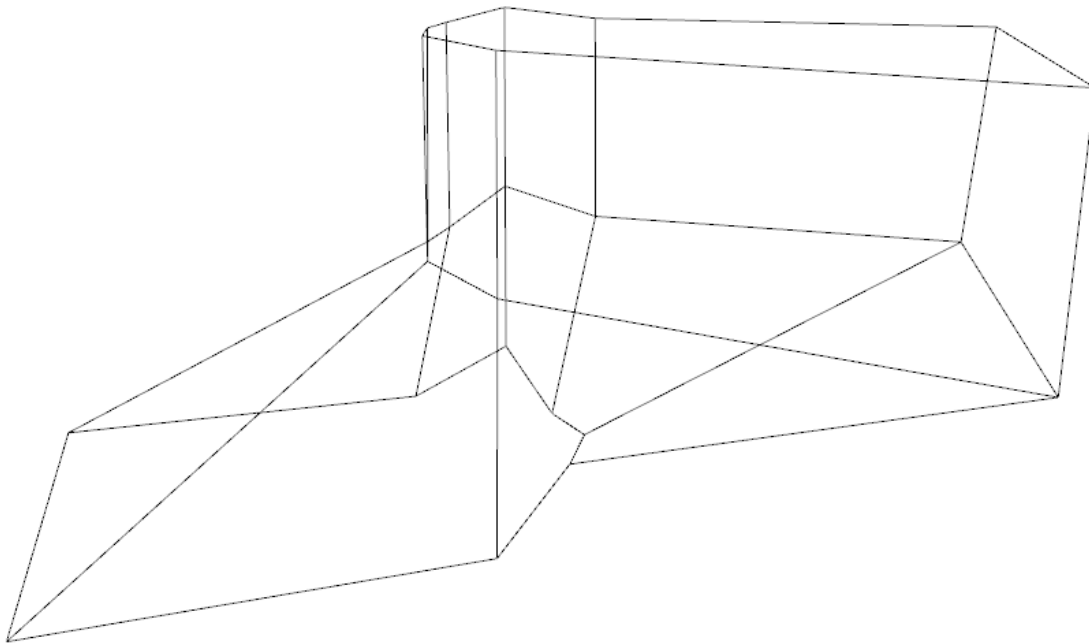


Figure 3.2: Open3D Render (View 2)

2. Device Characteristics Table

Feature	Description
Active LO Push Button	Pins: PM0 Voltage: 3.3V and GND
Stepper Motor	Pins: IN1=PH0, IN2=PH1, IN3=PH2, IN4=PH3 Voltage(s): 5V and GND Clock Speed: 60MHz
Time-of-Flight Sensor	Pins: SCL=PB2, SDA=PB3 Voltage: 3.3V and GND Clock Speed: 60MHz
Microcontroller	Clock Speed: 60MHz Status LED: D3 Onboard LED Bus Speed: 16MHz PIOSC or 25MHz MOSC
UART Communication Protocol	Pins: Serial Communication through Python Communication Speed: 115 200 bps (Baud Rate) Software: Python, Serial, Open3D

3. Detailed Description

3.1 Distance Measurement:

There are multiple steps involved with the process of gathering distance measurements from the Time-of-Flight sensor. Firstly, after setting up the microcontroller according to the schematic and characteristic table above, the Keil code must be loaded onto the MCU. Then the reset button must be pressed to configure the ToF sensor via I2C Serial Communication protocol. Although this may take some time, the process will be complete when all the onboard LEDs stop flashing.

Following this configuration step, you should run the Python code for data processing. The resulting output will be a HI to the MCU, this process is often referred to as a half-duplex configuration, and so the micro will wait until a button is pressed to begin taking distance measurements.

When the onboard PJ1 button is pressed, the stepper motor will begin rotating until a complete 360° revolution is completed. Every 45° the stepper motor will pause momentarily, so the ToF sensor can take distance measurements of its surroundings which are output to Python, for a total of 8 readings. It should also be noted that 3 complete rotations will be conducted in alternating clockwise and counterclockwise directions to avoid the wires from tangling.

By using Serial Communication in the form of UART, this collected data can be transferred from the MCU to the PC. The Python code then uses these measurements to obtain the x, y and z coordinates from the following formulas:

$$y = distance \times \cos(\theta)$$

$$z = distance \times \sin(\theta)$$

x, the displacement is preset to 100, 200 and 300 for each respective complete rotation.

These coordinates are then added to either a “y” or “z” list, respectively. The program then waits for another PJ1 button press to begin the next set of measurements. It is also important to note that if the STOP push button is pressed during or after one revolution has been completed the entire program will be stopped/terminated and we will need to begin from the beginning.

3.2 Visualization:

The computer used to communicate with the microcontroller was a Legion 5 laptop with a Windows OS as well as the system specs outlined below. Additionally, Python 3.9 was the program of choice, used to run the code and output the Open3D visualization.

Device specifications

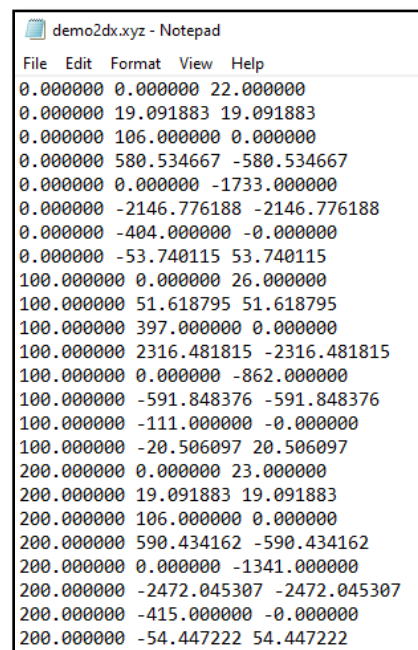
Device name	Arji
Processor	AMD Ryzen 7 5800H with Radeon Graphics 3.20 GHz
Installed RAM	16.0 GB (15.9 GB usable)
Device ID	15CD6A81-DAC3-417A-8E5C-D96DD5F62B76
Product ID	00342-20758-97759-AAOEM
System type	64-bit operating system, x64-based processor
Pen and touch	No pen or touch input is available for this display

- **What functions/libraries did you use?**

The functions/libraries used were import serial, numpy, open3d, math and python file handling.

- **How is the data stored and processed?**

For my computer specifically, the data goes through serial port COM3 and after the data processing Python program is ran, the distance measurements are stored in a .xyz file (demo2dx.xyz). This file, as the name suggests, contains the x, y and z measurements for each respective displacement measurement (100, 200 and 300) taken. As a result, for each plane, the file will include 8 points and when the device is moved forwards (to a new displacement value), this new plane will be appended to the file as well. Since 3 plane measurements or complete rotations are done, this will result in 24 total coordinates.



```
demo2dx.xyz - Notepad
File Edit Format View Help
0.000000 0.000000 22.000000
0.000000 19.091883 19.091883
0.000000 106.000000 0.000000
0.000000 580.534667 -580.534667
0.000000 0.000000 -1733.000000
0.000000 -2146.776188 -2146.776188
0.000000 -404.000000 -0.000000
0.000000 -53.740115 53.740115
100.000000 0.000000 26.000000
100.000000 51.618795 51.618795
100.000000 397.000000 0.000000
100.000000 2316.481815 -2316.481815
100.000000 0.000000 -862.000000
100.000000 -591.848376 -591.848376
100.000000 -111.000000 -0.000000
100.000000 -20.506097 20.506097
200.000000 0.000000 23.000000
200.000000 19.091883 19.091883
200.000000 106.000000 0.000000
200.000000 590.434162 -590.434162
200.000000 0.000000 -1341.000000
200.000000 -2472.045307 -2472.045307
200.000000 -415.000000 -0.000000
200.000000 -54.447222 54.447222
```

Figure 4: XYZ File

- **How does it visualize?**

To visualize the data in the .xyz file, Open3D library was used. Using the data in the measurements file, a point cloud was created and printed onto the screen. This plots all the coordinates onto a graph and was done using the code provided in Lecture 9. Although this point cloud may be useful to visualize the different axis present, it does not show the entire shape of the hallway. In order to achieve a full render of the surroundings, using a nested for loop, all the points on the different planes were connected and then all these “slices” were also connected effectively creating layers and the full shape of the hallway. Nested for loops are also used to append the lines array provided in the program code, and the connects of these lines of these lines are further appended to this array, effectively helping visualize a final image. To achieve this image, methods from the open3d library including `read_point_cloud`, `append`, `line_set`, and `visualize.draw_geometries` were used.

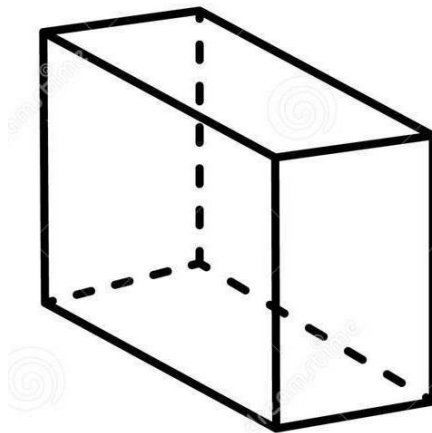


Figure 5: A rough dimensioned isometric sketch of the space.

4. Application Example & User Guide

Acquiring Signal Mapping:

1. Run the “Python Code.py” file once the MCU has finished configuring the ToF sensor (when all the onboard LEDs are off). When the serial port is opened, the program will output, “Opening: COM#”.
2. Once the serial port has opened, press PJ1 and the measurements will begin to be output to the Python console in the following way (Distance x-coord y-coord z-coord).
3. Once the first 8 measurements are taken (each 45° apart for a total of 360°), move the device forward and press PJ1 once more to begin taking the next set of readings.
4. Repeat steps 2 and 3 for three samples.
5. All the measurements will now be in the “demo2dx.xyz” file.
6. Close the initial tab that pops up and the Open3D file window should show the visualized data.

Device Setup:

1. Connect and configure the circuit as shown in *Figure 1*.
2. Connect the MCU to the PC with its USB cable.
3. Attach the ToF sensor to the stepper motor.
4. Load the Keil code onto the MCU.
5. Run the Python code (“Python Code.py”), ensure to change the “COM#” in the code to your specific one.
6. Press the reset button and wait for the onboard LED lights to stop flashing.
7. Press “Enter” on your keyboard to begin the Python program execution.
8. Press the onboard PJ1 button, so the stepper motor can begin rotating and wait until a complete revolution is completed.
9. Move the device forwards and repeat Step 8 for three samples.
10. If you wish to stop data collection at any point, press and hold the STOP button (external active LO push button).

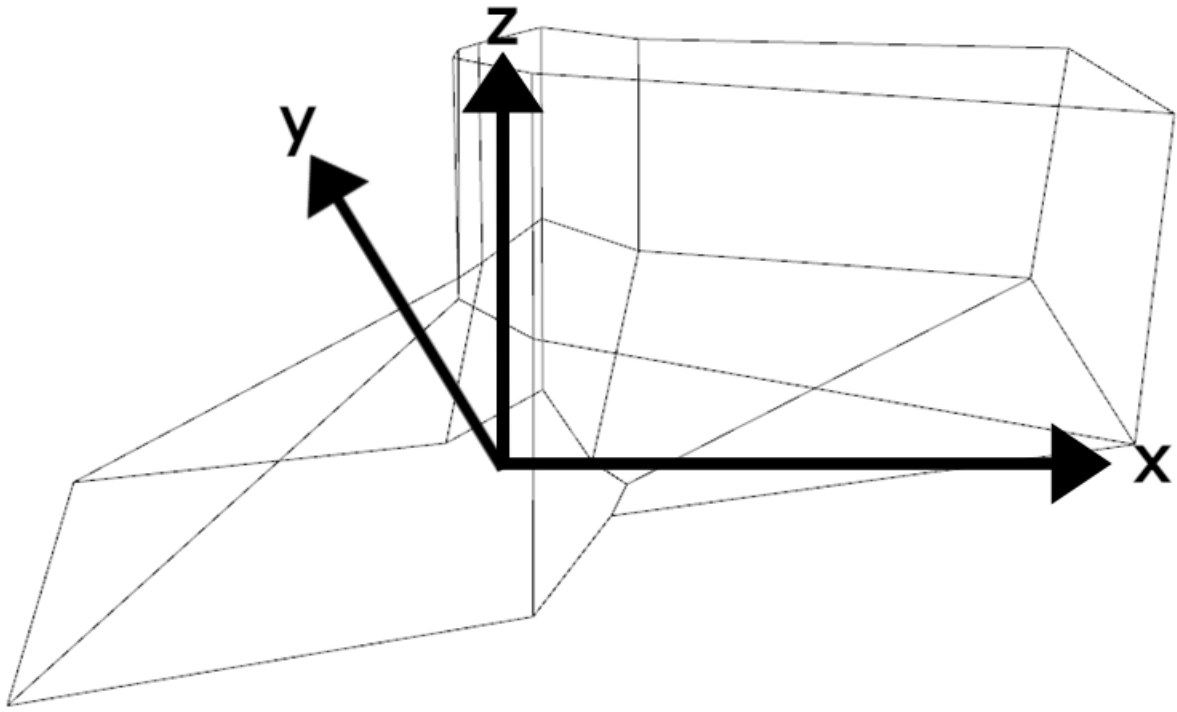


Figure 6: Open3D Renger with Dimensions.

As seen above, the figure shows all the axis planes. The x plane (displacement) is representative of the device moving forward, while the y and z planes (spatial) are vertical slices which the sensor reads.

5. Limitations

1) Summarize any limitations of the microcontroller floating point capability and use of trigonometric functions:

A limitation of the MCU is its ability to round since the ToF sensor rounds to the nearest whole number, meaning that the measurements will be less accurate and not exactly what they should be. As a result, the MCU transfers whole numbers for the distance measurements hindering the overall precision of the device. Moreover, the way that the ToF sensor is wired means that while rotating they overlap with one another and possibly move the ToF sensor. This could result in a loss of accuracy and lessen the precision of the design, which is also why I made the stepper motor alternate between CW and CCW rotations.

2) Calculate your maximum quantization error for each of the ToF module:

Given that the ToF sensor uses 8 bits to communicate, and the full-scale voltage is 3.3V, the maximum quantization error can be calculated as follows:

$$\begin{aligned}\text{Quantization Error} &= \frac{V_{FS}}{2^n} \\ &= \frac{3.3V}{2^8} \\ &= \mathbf{0.0129V}\end{aligned}$$

Therefore, the maximum quantization error for each of the ToF module is 0.0129V.

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

The maximum baud rate for UART found in the data sheet was 5 Mbps. That being said the MCU is limited to 115 200 bps as given in the data sheet. Therefore, the maximum standard serial communication rate which can be implemented with the PC and our design is 115 200 bps.

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

The ToF sensors use I2C serial communication protocol to communicate with the MCU, and in this particular case the MCU is the master (leader) and the ToF sensor is the slave (follower). By looking at the datasheet for the ToF sensor, it can also be noted that it takes measurements at a speed of 100 kbps.

5) Reviewing the entire system, which element is the primary limitation on speed? How did you test this?

After reviewing the entire system, it can be concluded that the primary limitation on speed is the stepper motor, as it operates very slowly. Moreover, the ToF sensor's measurements are virtually instant and not as noticeable in regard to the time it takes to output. The same thing can be said for the communication time. That being said, the only aspect of this design which does take a noticeable amount of time to operate, or do a full 360° revolution in this case, is the stepper motor. As a result, this impacts the time it takes for the entire system to function. This can be tested by changing both the bus and clock speeds respectively, and when they were decreased the stepper motor rotates quicker all while preserving the accuracy of the measurements. This demonstrates the constraints the stepper motor, or more specifically the specific bus and clock speed I am required to follow, has on the overall speed at which this design functions.

6. Circuit Schematic

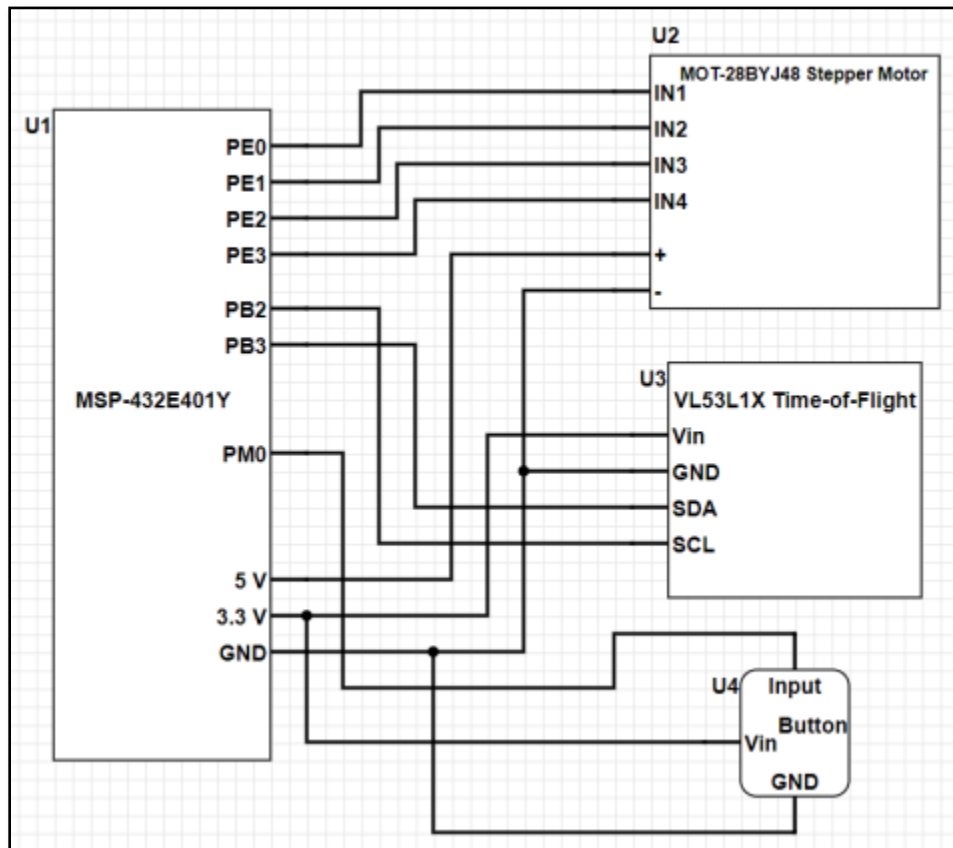


Figure 7: Circuit Schematic

7. Programming Logic Flow

Arji Thaiyib
400336020

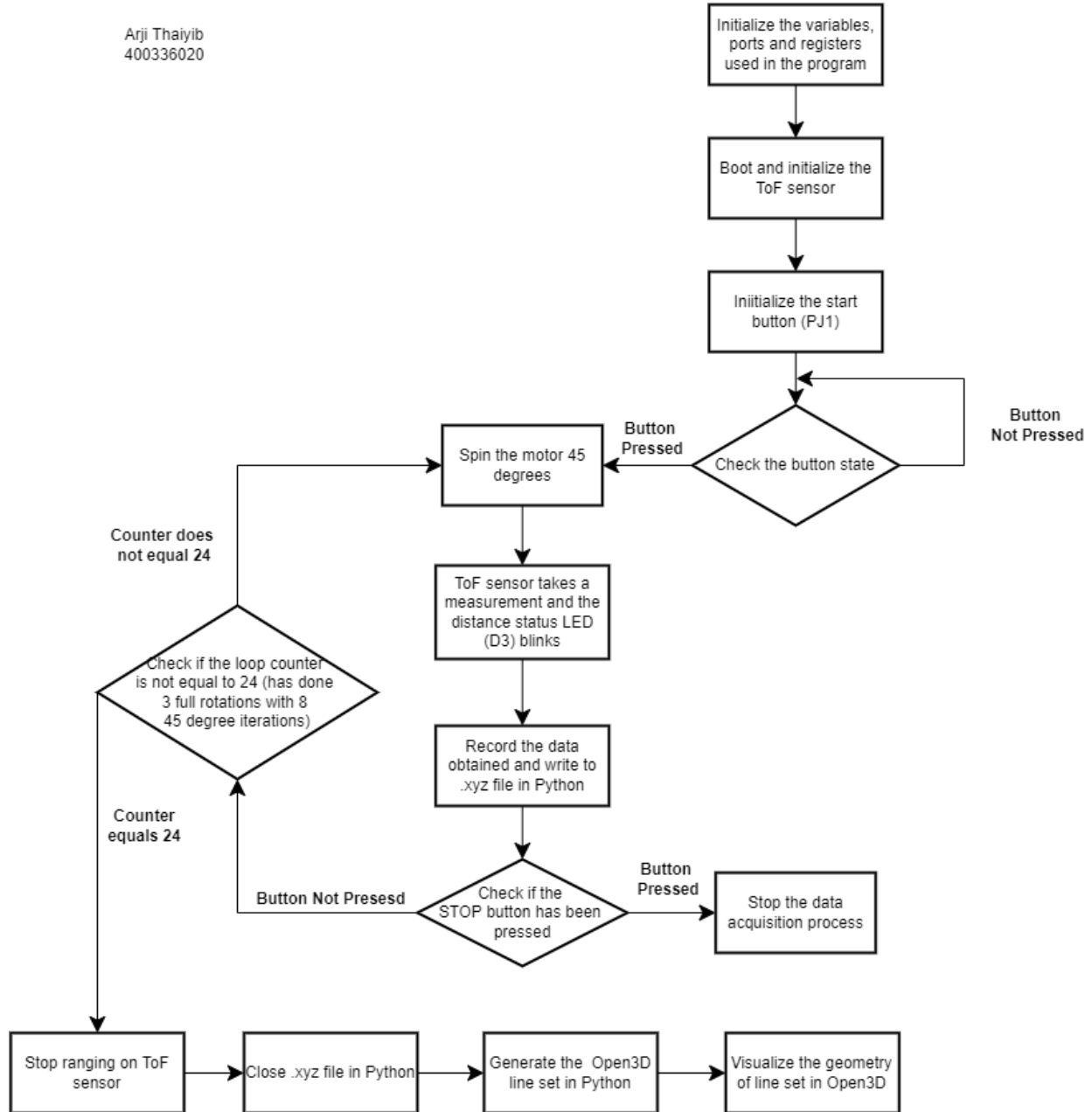


Figure 8: Programming Logic Flowchart for MCU and Python Code