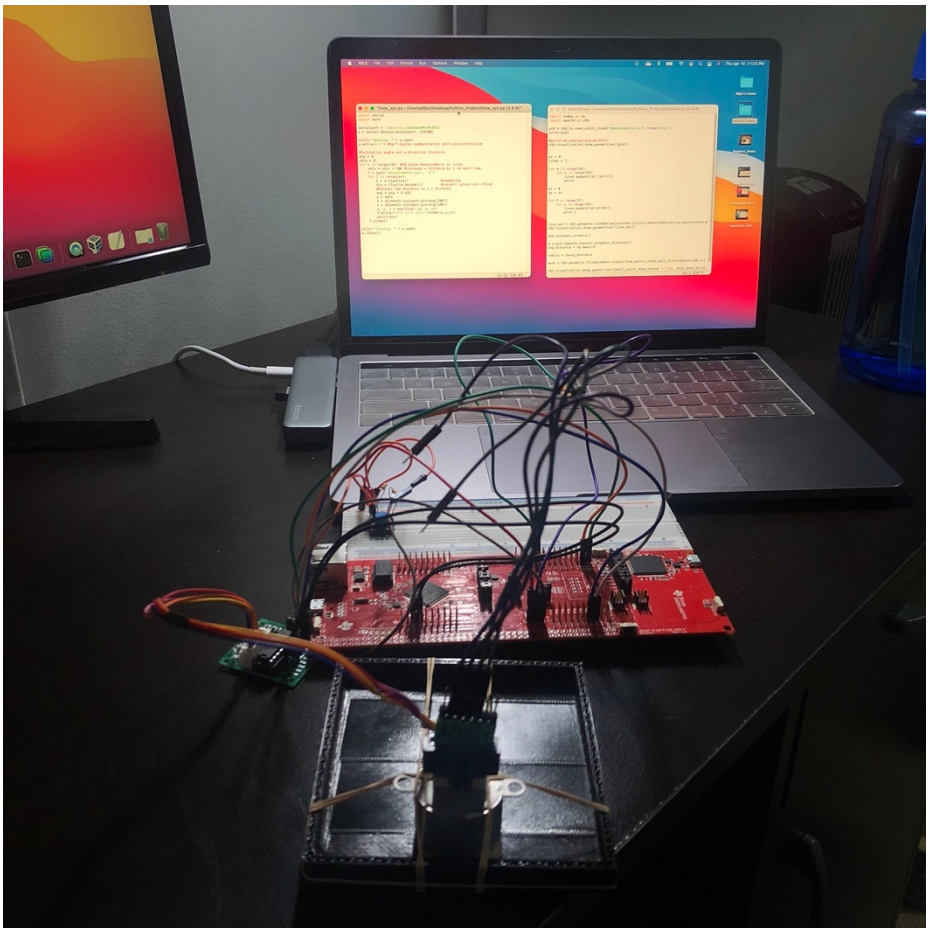
2DX4: Microprocessor Systems Project
Final Project

Instructors: Dr. Haddara, Dr. Doyle, Dr. Shirani

Diljot Dola Saini – L04 – dolasaid – 400252842

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 Diljot Dola Saini, 400252842

Project Video



Demo:

https://drive.google.com/file/d/15MeWUKRfnaQnStuu2_eZklhhJYEYZpAr/view?usp=sharing

Question 1:

https://drive.google.com/file/d/1310yjBZK8vlt-_qD0lnFIc1M925mCyZi/view?usp=sharing

Question 2:

https://drive.google.com/file/d/1iKqcxvUfJFUvPbFXyf5sZToGjXUAYKb0/view?usp=sharing

Question 3:

https://drive.google.com/file/d/1bQ5qwBo2xAG4L4d53B-ehXivbm96evTW/view?usp=sharing

1 Device Overview

1.1 Features

The environment mapping device includes the following features:

- SimpleLink MSP432E401Y microcontroller ~ $65.53
    - 4 on-board status LEDs
        - Sensor configuration
    - Communicates with sensor and PC, Rotate stepper motor
    - 1 microcontroller reset switch
    - USB 2.0 Micro A/B connector
    - Bus Speed: 48 MHz
- (ToF) - Time of Flight Sensor (VL43L1X) ~ $11.95
    - Operating voltage: 3.3V
    - Collects distance measurements (64 measurements/rotation)
    - I2C communication
    - Maximum distance measurement range: 400 cm
    - Maximum frequency ranging: 50 Hz
- MOT-28BYJ48 Stepper Motor ~ $8.03
    - Operating voltage: 5V
    - Used for collecting a 360° scan by rotating ToF sensor
- Distance measurement LED status
- Python 3.8
    - Libraries: Open3d, PySerial, NumPy
    - Process data and create 3D rendering
    - Serial communication (UART) with microcontroller using PySerial library
        - Baud rate: 115200
        - Store measurements from sensor to the PC

1.2 General Description

The environment mapping device is an embedded system which creates a 3D representation of the environment in which the system is situated in. There are various components of this embedded system including the microcontroller, ToF sensor, stepper motor, push-button, and PC data processing.

(a) Microcontroller

This microcontroller configures the ToF sensor via I2C protocols with the click of the on-board reset button. It also serially communicates with it via I2C to collect distance measurements. Then it is used to perform digital input and output with an external push button. This microcontroller is also used to serially communicate with the PC via UART. The on-board LEDs are used to display the ToF sensor configurations status as well as the distance measurement status. Lastly, this microcontroller is used rotate the stepper motor to achieve a full 360° rotation.

(b) Time of Flight Sensor (ToF)

This component is mounted to the stepper motor and it communicates with the microcontroller via I2C to collect distance data. The sensor uses LIDAR technology to measure how long it takes for the emitted pulse of light to reach the nearest object and be reflected back to the detector. It performs the distance calculation using the speed of light and the time taken for the reflection.

(c) Stepper Motor & Push Button

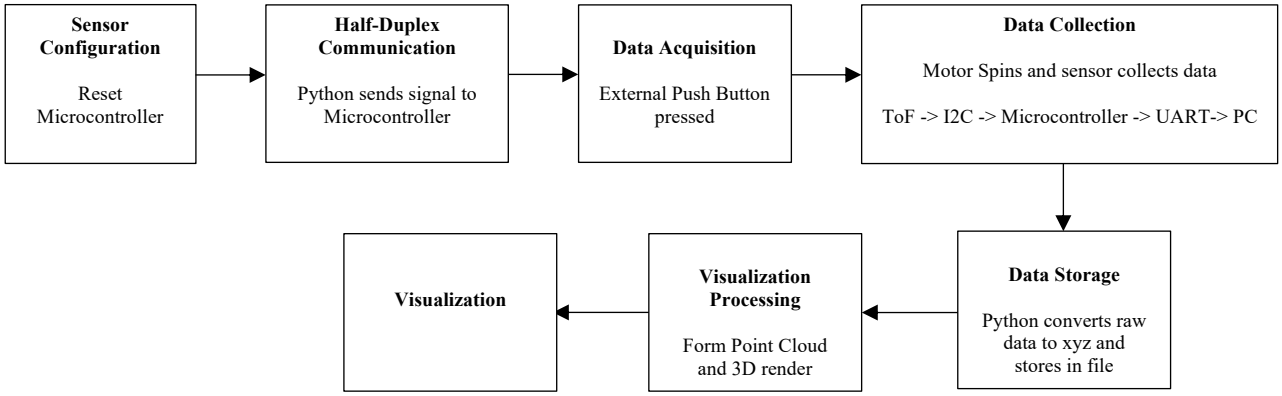
The push button and stepper both work with the microcontroller via GPIO input and output. The push button is the component of the device that starts the collection of distance measurements. Once the button is pressed, it indicates the microcontroller to start rotating the motor to allow the sensor to get a 360° scan of the environment around it.

(d) PC data processing

The data collected from the sensor is serially communicated from the microcontroller to the PC via UART and it is processed on the PC via Python 3.8 using the PySerial library.

1.3 Block Diagram

Figure 2. Block Diagram



2 Device Characteristic Table

Feature	Description
Push Button	**Pin Connections:** PM0, 3.3V, GND
Stepper Motor	**Pin Connections:** PH0 -> IN1, PH1 -> IN2, PH2 -> IN3, PH3 -> IN4, 5V, GND **Clock Speed:** 240,000 in Systick.c_Wait10ms
Time of Flight Sensor (ToF)	**Pin Connections:** PB2 -> SCL, PB3 -> SDA, 3.3V, GND
UART communication	**Serial Port:** COM3 **Communication Speed:** 115200 bps (baud rate)
Microcontroller	**Bus Speed:** 48 MHz

3 Detailed Description

3.1 Distance measurement

To collect the distance measurements, there are a sequence of steps. The first step is to push the on-board reset button on the microcontroller to configure the ToF sensor via I2C (refer to Figure 5). The configuration will start with booting the ToF chip and flashing all the on-board LEDS. Then it will flash LED D1, D2, and D4 four times to indicate the user that the sensor is configured with the default settings and ranging needed.

Acquisition & Transmission

Next, half-duplex communication will be used to transmit/receive data from the microcontroller via UART. Now, the python script using the PySerial library must be run which will send a '1' (refer to Figure 4) to the microcontroller indicating it that it can begin checking for whether the external push button is pressed or not to begin data measurement. When the button is pressed, the stepper motor will begin to rotate and the ToF sensor mounted on top of the stepper motor will begin to collect data. The system is programmed to collect measurements every 5.625° providing 64 measurements/360° scan.

The ToF sensor is programmed to collect data through the API's provided by the manufacturer. Using LIDAR technology, the sensor physically collects data. The sensor uses light as a pulsed laser to measure ranges of distance where a pulse of light is emitted. This pulse then hits an object and returns back to the sensor. The sensor measures how long it takes for the emitted pulse of light to reach the nearest object and be reflected back to the detector.

Conversion

The data collection and convertion is a simultaneous process. As the sensor is collecting data the python script is reading in the raw distance bytes and converts those measurements into strings which are then casted to floats. This data is now available for data processing. To get the x, y, z distance measurements trigonometry is used. Since the sensor is being displaced in the positive x-direction, this value is updated after every 360° scan is completed. The measurements in the y, z direction is achieved by using the following formulae:

$$y = distance\ measured * (\sin(\theta)), z = distance\ measured * (\cos(\theta))$$

The angle θ is incremented by 5.625° every time a measurement is taken. When the data is processed as explained, we are able to achieve the x, y, z distances all at once per measurement taken.

e.g., if the raw distance measured is 203 mm from the sensor and we are on the 1st measurement of the 1st plane, these are the distances written to the .xyz file:

$$x = 100\ mm$$

$$y = (203\ mm) * (\sin(5.625°)) = 19.9\ \text{mm}$$

$$y = (203\ mm) * (\cos(5.625°)) = 202\ \text{mm}$$

3.2 Visualization

OS, Computer Specs, Program Version & Libraries

The PC being used with this device is a MacBook Pro operating on macOS Big Sur, with an Intel Core i5 processor, and 8 GB RAM. The program being used to collect and visualize the data is Python 3.8.8. The libraries that were used to collect and visualize the data are PySerial, Math, Open3d, and NumPy. The libraries used for data collection were PySerial and Math and the libraries used for visualization were Open3d and NumPy.

Data Storage

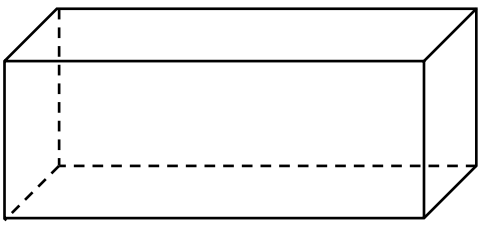
To visualize the converted measurements, the x, y, z measurements had to be stored in a '. xyz' file format. Thus, the converted x, y, z measurements were written to a file in the '. xyz' format each time a measurement is taken. Every time we switch to the next plane and x is displaced, the file is appended, and it is ready to take the next set of x, y, z distance measurements. The python program will close once the device has gathered measurements in 10 planes. All the measurements being taken end up in the '. xyz' file containing 640 points ((64 points/rotation) *10).

Visualization

Now, the code written for visualization is accessed via another python script in which the file containing the stored data is processed. In this python script, the Open3d library is used to create a point cloud, from the '. xyz' file obtained in the distance measurements. This was achievable by searching for the right methods and classes in the Open3d documentation. To achieve a complete rendered 3D display of the measured environment, the points in each plane had to be connected with each other and the planes had to be connected with each other. To obtain this I had to create a list in which pairs of points would be stored. For getting this list I had to use to for loops, one was for the points in the planes, and the other one was for connecting each plane. Once I obtained my list of point pairs, I was able to use the 'geometry', 'LineSet', and 'visualization' method from this library to show the 3d rendered output.

Space Recreation

The space that was recreated was the inside of a rectangular prism show box.



4 Application

Device Set-Up

To set-up the device, the components of the environment mapping device have to be connected to the microcontroller and the PC. (Refer to Figure Circuit Schematic)

1) Connect stepper motor to pins PH0, PH1, PH2, PH3, 5V and GND on the microcontroller
2) Connect the push button circuit to the microcontroller on pins PM0, 3.3V and GND
3) Connect the ToF sensor PB2 -> SCL, PB3 -> SDA, 3.3V, and GND on the microcontroller
4) Mount the ToF sensor on top of the stepper motor
5) Connect the microcontroller to the PC

Program Set-Up

6) Load the code from Keil to the microcontroller by first compiling and building the code to check if there are no errors
7) Before you move on to the next step, you must check which ports are available on your PC for serial communication through the command prompt window
8) Open the Python script “Data_xyz.py” for collecting and storing the data, modify the COM port if needed

Gathering Data

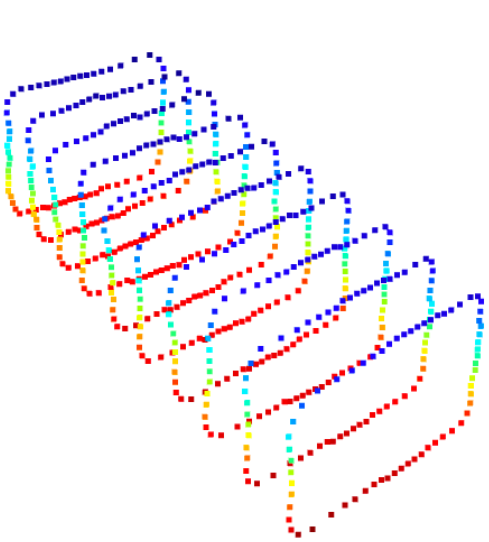
9) Place the environment mapping device in a space you would like to map in 3D
10) On the microcontroller press the on-board reset button and wait for the configuration of the sensor to be complete
11) Once the sensor is configured, run the Python script “Data_xyz.py”. You should be seeing “Opening: COM PORT”, where COM PORT is the port, you entered in step 8
12) Next, press the external push button to begin collecting distance measurements
13) Displace the stepper motor with the sensor attached to it in the x direction (1 cm each time) from an origin point after each 360° scan (For each plane we are measuring in the y-z plane and the x-axis is the direction for the forward displacement of the device)
14) Once you have gathered 10 measurements the python script will terminate, and the data will be stored in the file named “measurements.xyz”
15) Open the “measurements.xyz” file to verify that the data has been stored

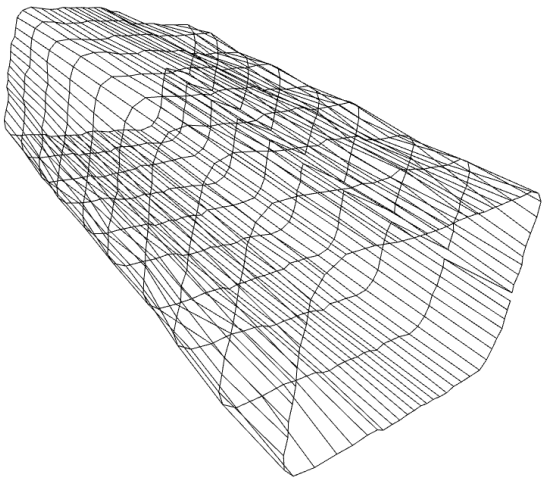
Visualization

16) Now open the Python script named “Open3d.py” and run the script
17) You should be able to see a point cloud, followed by a 3d render of the mapped environement

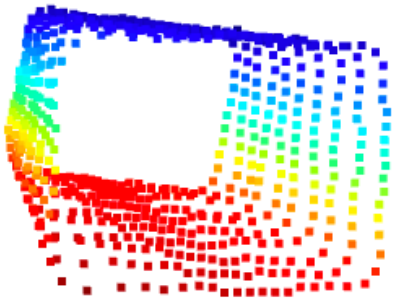
Expected Output

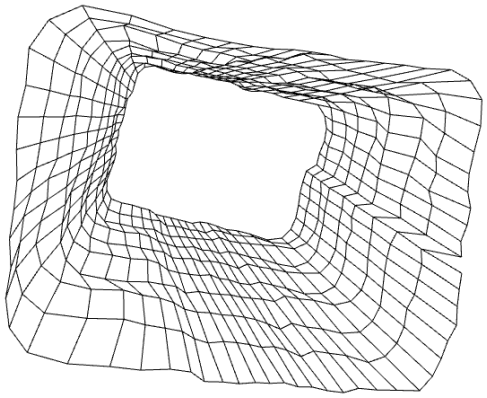
Displaying y-z planes and view





Displaying the x-direction view





As you can see the measurements obtained from the 360 scans are in the y-z plane and the sensor is displaced in the x-direction.

5 Limitations

Question 1

Since the raw data collected by the sensor is transmitted/received via UART, all the raw distance measurements were rounded to integers. This leads to a lack of precision in measurements because we are using the raw distance measurements and we are converting this measurement into y and z distance measurements using sin and cosine functions. Even if the range of the sensor was increased, there would still be some lack of precision due to the rounding done by the microcontroller during ADC and the UART transmission.

Question 2

To calculate the maximum quantization error, we must do V_{FS}/2^m where m = data bits. Thus, the maximum quantization error of the ToF is 3.3V/2^8 = 0.012891V.

Question 3

The maximum standard serial communication rate you can implement with the PC is 115200 bps, this was verified from the data sheet of this microcontroller. This is a limitation because we can actually do UART communication at a maximum of 5 Mbps. Thus, we could have gotten a faster data transfer, but this was limited by our microcontroller.

Question 4

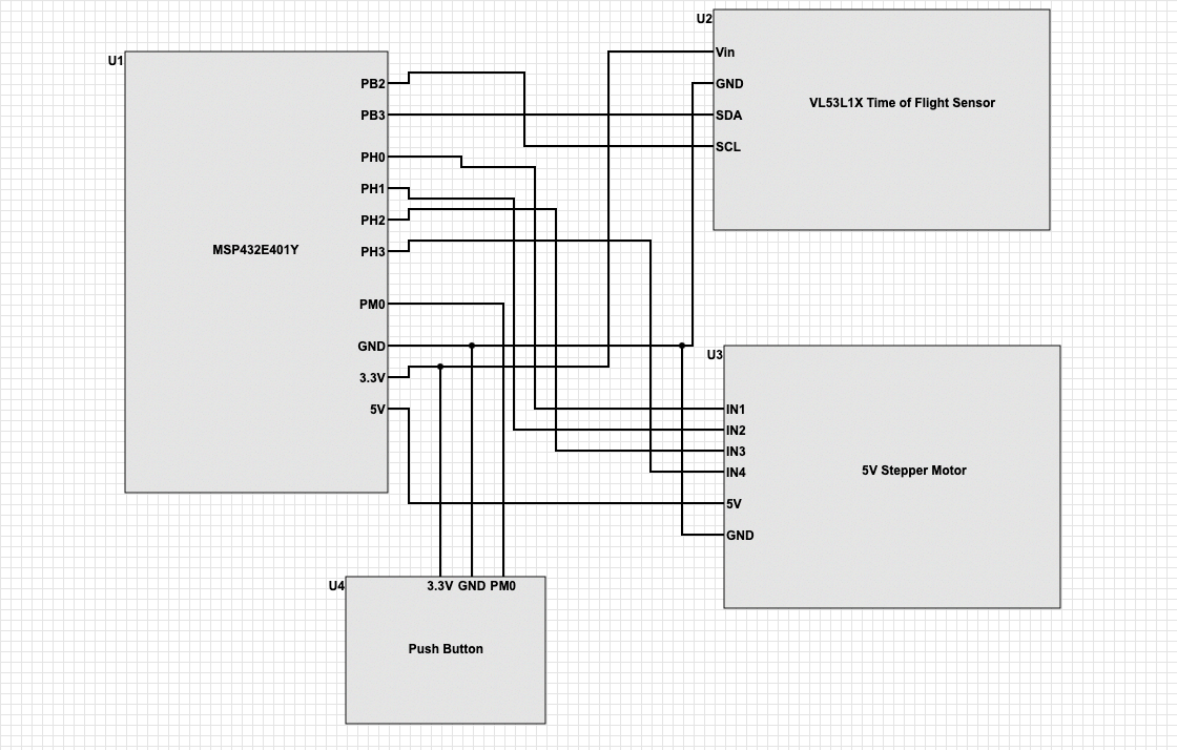
The communication methods used were I²C and UART. The ToF communicated with the microcontroller via I²C at a speed of 100 kbps (obtained via the datasheet of the sensor). The microcontroller communicates to the PC via UART at a speed of 115200 bps.

Question 5

Reviewing the system, I can clearly see that the primary limitation on the speed is the ToF sensor. This is because the sensor itself needs a maximum of 60 ns to measure the emitted light pulse, thus this cuts down the speed at which measurements can be collected. Thus, achieving a lot more samples of measurements at a faster clock speed will be limited due to the sensor. I verified this by testing the collection of a large sample of measurements over a small clock speed. This was too fast for the sensor, and it got overload with measurement requests and did not work.

6 Circuit Schematic

Figure 3 Python Program Flowchart



7 Flowcharts

Figure 4 Python Program Flowchart

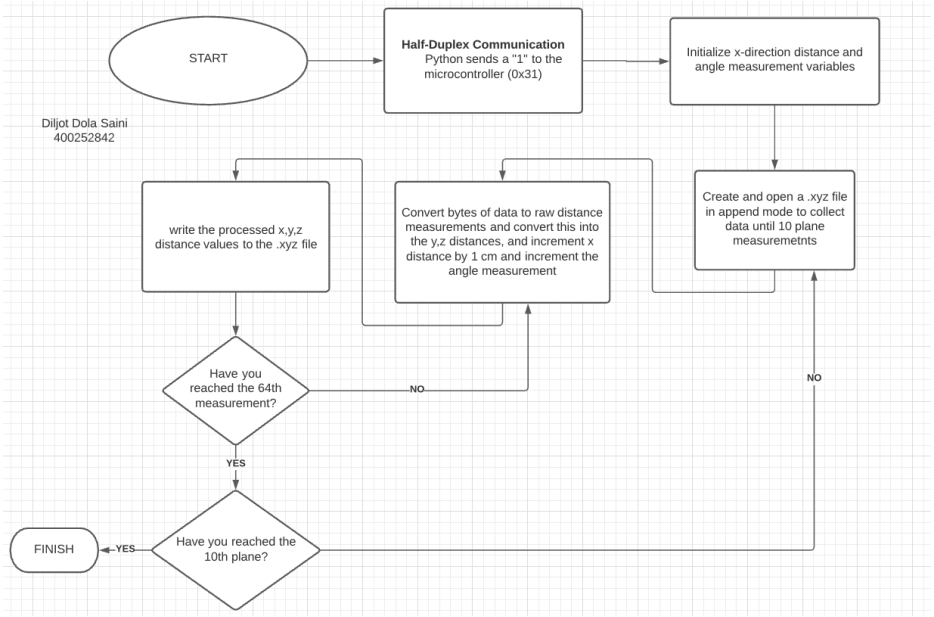


Figure 5 Microcontroller Program Flowchart