COMPENG 2DX3 Spatial Mapping Project Report

1 Device Overview

1.1 Features

- LIDAR System
  - Appropriate for exploring and navigating indoors
  - Data acquisition system
- Texas Instruments MSP432E401Y Microcontroller:
  - ARM Cortex-M4F Processor Core
  - Operating voltage range of 2.5V-5.5V
  - Bus/clock speed of 120 MHz
  - Two 12-bit SAR-Based ADC modules, each supporting 2Msps
  - Eight Universal Asynchronous Receivers/Transmitters (UARTs)
  - Ten Inter-Integrated Circuit (I^2C) modules
  - Baud rate of 115200
  - 1024KB of flash memory
  - 256KB of SRAM
  - 6KB of EEPROM
  - C/C++ complier
  - Approximate cost of $39.99 CAD via Texas Instruments (for only the microcontroller, one micro-USB cable, and one quick start guide)
- ULN2003 Driver Board for the MOT-28BYJ-48 Stepper Motor
  - 512 steps for a 360-degree rotation
  - MOT-28BYJ-48 Stepper Motor has an operating voltage of 5V
  - Approximate cost of $6.95 CAD via ABRA Electronics (for the driver board and stepper motor)
- VL53L1X Time-of-Flight (ToF) Sensor
  - Size: 4.9x2.5x1.56mm
  - I^2C interface
  - Maximum distance measurement of 4000mm
  - Up to 50Hz ranging frequency
  - Emitter: 940nm invisible laser
  - Operating voltage range of 2.6V-3.5V
  - Approximate cost of $18.95 CAD via Pololu (for one sensor)
- Python 3.9
  - Python 3.9 IDLE compatible with Open3D Python library add-on
  - Libraries used include math, serial, numpy, and open3d
- Data Communication: I^2C Serial Communication
  - Using I^2C serial communication between the MSP432E401Y Microcontroller and the VL53L1X Time-of-Flight (ToF) Sensor
  - PB2 and PB3 pins for I^2C
- Data Communication: UART Serial Communication
  - UART serial communication to the PC and using Python
  - UART serial communication baud rate of 115200
- Visualization
  - Mapping out distance measurement data through 3D visualization
  - Completed using Python 3.9 IDLE
  - 3D visualization using Open3D Python add-on for 3D data processing

1.2 General Description

For this project, I am using the Texas Instruments MSP432E401Y microcontroller, the VL53L1X Time-of-Flight (ToF) sensor, and the MOT-28BYJ-48 stepper motor to obtain distance measurements of a hallway in the Engineering Technology Building at McMaster University, which will then be graphed in a 3D graphing software. This project uses the above components to take distance measurements of the surrounding environment. This is then converted into xyz coordinates, which are then used to create a 3D presentation of the scanned location. One Keil program file was used, which is named "2dx_studio_8c." Additionally, two Python files were used which were "2dX3_FinalProject.py" and "O3D_FinalProject.py."

In this project, the VL53L1X Time-of-Flight (ToF) sensor handles obtaining all of the distance data measurements. The sensor will emit a 940nm laser pulse into its surroundings and it will be reflected off of the closest object, which then returns back to the sensor. The distance is than calculated by the formula $Distance\ Measured = \frac{Photo\ Travel\ Time}{2} \times Speed\ of\ Light$. The sensor also handles the analog to digital conversion component of the project. The transducer captures the signal, which is then processed from being analog to a digital signal. The sensor converts the time measurement into an analog signal, which represents the distance to the object the emitted pulse touched. The signal is them amplified and filtered to remove any interference. It is then processed into a digital value, which is processed by the microcontroller to get the distance measurements.

The communication method used between the microcontroller and the ToF sensor was I^2C. I^2C communication protocol was used to communicate the data between the ToF sensor and the microcontroller. To do this, I connected SCL to PB2. I also connected SDA to PB3. The data from the sensor would go through the microcontroller, and then it would go to the PC via UART and the COM port used at the specified baud rate of 115200.

The visualization of the project is done using Open3D in Python 3.9. First, the distance measurements from the sensor are sent to the microcontroller and then to the PC, and finally to Python. Using the "2dX3_FinalProject.py" file, the data was obtained from the sensor are converted into xyz coordinates. The formulas used to obtain the y vertical coordinate and the z horizontal coordinate using the distance measured and the angle measurement are the following:

$$Angle = \left(\frac{Number\ of\ Steps}{Steps\ in\ the\ 360°\ Rotation}\right) \times 2\pi$$

$$Y = r \times \sin\left(angle\right)$$

$$Z = r \times \cos(angle)$$

$$X = X$$

Then, using the "O3D_FinalProject.py" file, the point cloud and the final 3D visual representation are generated.

1.3 Block Diagram

Figure 1 below displays the block diagram for this project, displaying the major parts involved to understand the system.

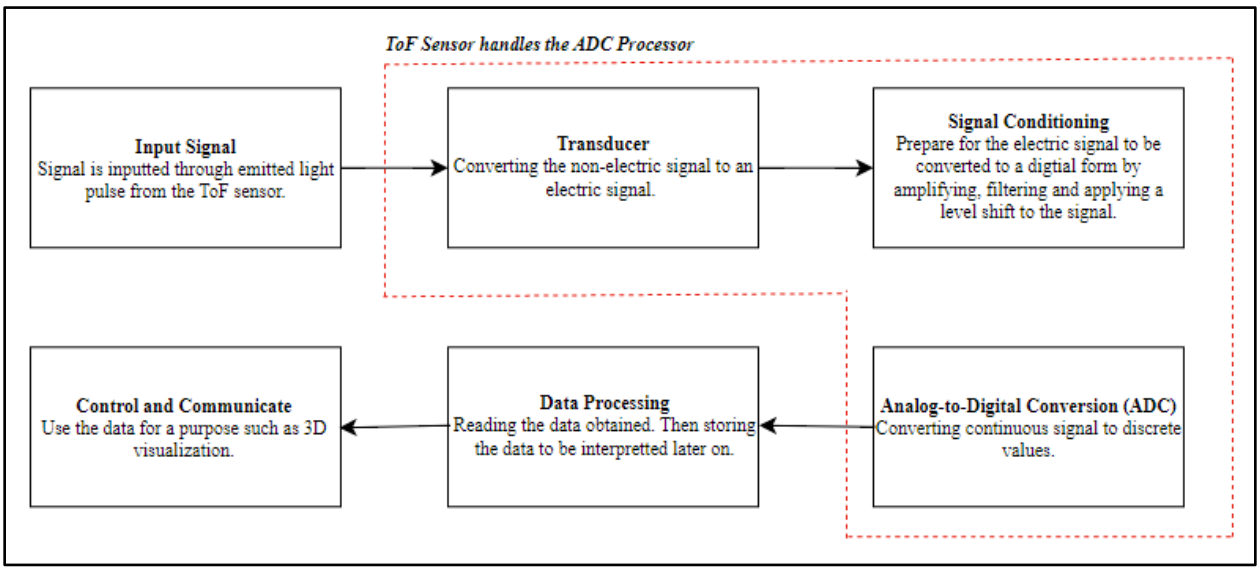

Figure 1. Project Block Diagram

2 Device Characteristics Table

General System Setup	
Student Number Assigned Feature	Assigned LEDs/Values
Assigned LEDs	PF4, PF0
Assigned Bus Speed	16 MHz
Onboard Pushbutton Setup	
Onboard Pushbutton	Detail
PJ1	Controls the start/stop of the data acquisition process, configured with interrupts
LED Setup	
Onboard LED	Detail
PF4 → D3	Port F, LED for distance measurement status indication
PF0 → D4	Port F, LED for troubleshooting status indication
VL53L1X ToF Sensor Setup	
VL53L1X ToF Sensor	Microcontroller Connections
VDD	N/A
VIN	3.3V
GND	GND
SDA	PB3
SCL	PB2
XSHUT	N/A
GPIO1	N/A
ULN2003 Driver Board Setup	
ULN2003 Driver Board	Microcontroller Connections
IN1	PH0
IN2	PH1
IN3	PH2
IN4	PH3
-	GND
+	5V
Python 3.9 (64-bit) Setup	
Python 3.9 (64-bit)	Detail
Import Library: math	Math library used
Import Library: serial	Serial library used
Import Library: numpy	Numpy library used
Import Library: open3d	Open3D library used
UART Setup	
UART	Detail
Baud Rate	115200 bps
COM Port	COM3

Table 1. Device Characteristics Tables

3 Detailed Description

3.1 Distance Measurement

For this project, I used the VL53L1X ToF sensor which measures distance. This sensor uses an emitter to emit a 940nm invisible laser into the environment, which is reflected off of the closest object. It then returns back to the sensor, which is noted by the detector/receiver. This sensor uses a single photo avalanche diode, or SPAD, receiver array with an integrated lens as the receiver. The sensor records the time taken for the emitted light pulse to be reflected back and then multiplies the time delay by the speed of light divided by two. This can be written as the following formula: $Distance\ Measured = \frac{Photo\ Travel\ Time}{2} \times Speed\ of\ Light$, where the speed of light is 299 792 458 m /s. This sensor also obtains distance measurements in the units of millimetres. This is how the ToF sensor acquires distance measurements.

To explain the project, this system starts off with initializing and configuring all of the pins and ports that are used within the circuit. This includes Port B for the I^2C module, Port H for the stepper motor, Port F for the onboard LEDs, and Port J for the onboard pushbutton. Once this has been done, the interrupts must be enabled for the onboard pushbutton PJ1 in Port J. This is required since it allows for the port to identify when the button has been pressed to either start or stop the project from functioning. The configuration of the interrupts used were done using the five main steps:

1. Arm the source of the interrupt
2. Enable the interrupt in the Nester Vectored Interrupt Controller (NVIC)
3. Enable the global interrupts
4. Set the interrupt priority level
5. Write the Interrupt Service Routine (ISR) that will handle the interrupt

Once the circuit is built and configuration have been done, we can begin the distance measurement process. The sensor and motor only start after the onboard pushbutton has been pressed. After pressing the pushbutton, the sensor initially boots its systems and configures with its default settings. Once this is done, the sensor will emit a 940nm laser pulse into its surroundings and it will be reflected off of the closest object, which then returns back to the sensor. The distance is than calculated by the formula previously stated. For my project, the ToF sensor takes measurements in units of millimetres every 22.5 degrees. This means that it takes a total of 16 measurements or scans 16 times (360/22.5 = 16) in one full rotation. During each distance measurement taken, the onboard LEDs will flash, which signals that a measurement has been acquired. Once a full rotation is completed, the system will wait until PJ1 is pressed again after displacing the entire circuit system and sensor in the positive x direction. Displacement is implemented into the project by first initializing the initial displacement in the x direction in the Python file. This is done in the “2dX3_FinalProject.py” file, where x is first set to a value of 0. After multiple scans, displacing in the x direction each time, a 3D visual representation is obtained.

In order for the sensor to communicate with the MSP432E401Y microcontroller, it uses I^2C protocol. This is why I have connected SCL to PB2 and SDA to PB3. These are pins that are a part of the I^2C module on the microcontroller. The data collection is first initialized by pressing the reset button on the microcontroller, and then pressing the onboard pushbutton PJ1. When each measurement is taken, the data from the sensor would go through the microcontroller, and

then it would go to the PC via UART and the COM port used at a baud rate of 115200 bps. Then, the data is obtained by Python. The created Python program retrieves the data and stores it to use it later on.

Once Python gets the data, we can convert the distance and angle measurements into cartesian coordinates on an xyz plane. These conversions are done within the "2dX3_FinalProject.py" Python file. The formulas used to obtain the y vertical coordinate and the z horizontal coordinate using the distance measured and the angle measurement are the following:

The angle of the stepper motor is determined by the number of steps it has done and the number of steps in one full rotation:

$$Angle = \left(\frac{Number\ of\ Steps}{Steps\ in\ the\ 360°\ Rotation}\right) \times 2\pi$$

$$Y = r \times \sin(angle)$$

$$Z = r \times \cos(angle)$$

$$X = X$$

where r represents the distance

Example Calculation:

X = 2, r = 403.457mm

$$Angle = \left(\frac{Number\ of\ Steps}{512}\right) \times 2\pi$$

$$Angle = \left(\frac{32}{512}\right) \times 2\pi$$

$$Angle = 22.5°$$

$$Y = r \times \sin(angle)$$

$$Y = (403.457) \times \sin(22.5)$$

$$Y = 154.39630957$$

$$Z = r \times \cos(angle)$$

$$Z = (403.457) \times \cos(22.5)$$

$$Z = 372.74566454$$

$$X = X$$

$$X = 2$$

Coordinate = (2, 154.39630957, 372.74566454)

This is how the distance and angle measurements are converted to the xyz plane.

3.2 Visualization

For the visualization component of this project, I used Open3D in Python 3.9. Open3D is a Python library add-on that is used for 3D data processing and visualization. Open3D offers tools like point cloud registration and 3D data visualization, which are the two tools I have used in this project. Open3D is compatible with Python 3.9, which is the version I downloaded and used. I used two files in Python for this project. One file I used called "2dX3_FinalProject.py," was used to deal with obtaining the data from the sensor and converting it to xyz coordinates. The second file I used was called "O3D_FinalProject.py," which was used to generate the point cloud and the final 3D visual representation. For testing, the programs created were run on a HP Envy x360 15.6" laptop, with an Intel Core i5 and 16GB RAM.

In Python, the libraries that were incorporated includes math, serial, numpy, and Open3D. The math library was required in order to use the value of pi in the conversion calculations when changing the distance measurements into xyz coordinates. The serial library was used since serial communication is being used through the COM port and the baud rate, which in my case was COM 3 and 115200. The numpy library was used to allow for working with arrays of any dimensions. Lastly, the Open3D library was used to create the actual 3D visual of the scanned area using the xyz coordinates.

Variable "s" in the Python code represents the serial port. This port gets opened and cleared of any leftover data. Then, a .xyz file is created to be written in. Once the sensor finishing scanning and the data it sent to Python, the xyz coordinates of the data are written in the .xyz file after the xyz coordinates are determined by the formulas previously measured. After each scan taken, the number of scans taken are incremented by one. The process will end once the number of scans desired to be done which is inputted into the code, are completed. To read the point cloud from the .xyz data file, I used the function "o3d.io.read_point_cloud." This reads the data from the .xyz file. The function used to obtain the final 3D visual representation is "o3d.visualization.draw_geometries," which will connect the data points from the point cloud and include the 3D coordinate vertices of each slice/scan taken. This is how the visualization process works.

Figure 2 and 3 below are examples of the point cloud and 3D visual representation taken from a scan of a skinny cup:

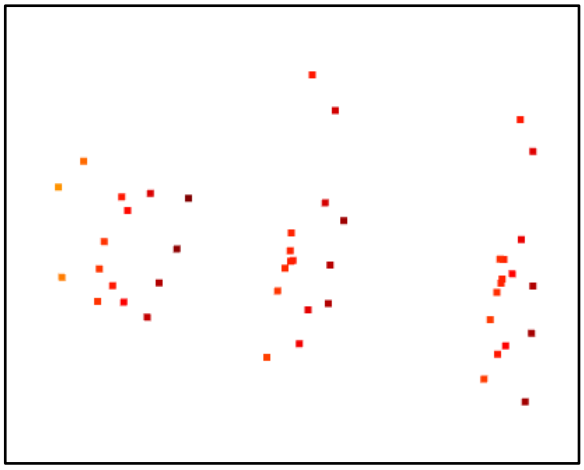

Figure 2. Point Cloud

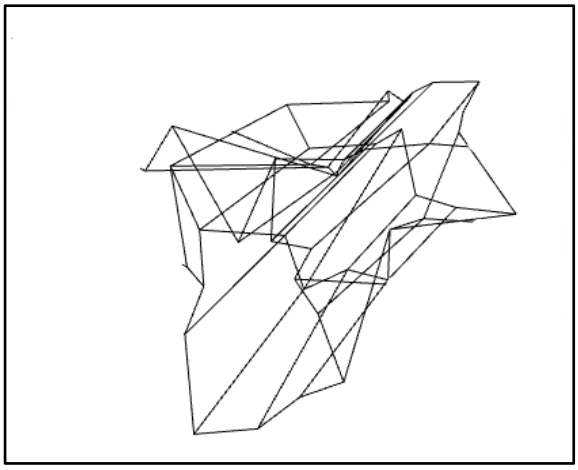

Figure 3. 3D Visual Representation

4 Application Example, Instructions, and Expected Output

To utilize this created system for this project, there is a setup process and a set of instructions to be followed to ensure that any user can use this system and its corresponding code. In this setup process, downloading and installing software are required, as well as physically setting up and building the circuit used. The following setup instructions take you from the start to the end, making it easy for anyone to follow.

Setup Instructions:

1. Download and install Keil (MDK Microcontroller Development Kit) using this link: https://developer.arm.com/Tools%20and%20Software/Keil%20MDK. After licking this link, click “Download Keil MDK,” and fill out the information on the next page prompted and click submit.
2. Once Keil is installed, the Pack Installer will open automatically. If it does not, open Keil and select the Pack Installer icon (which is visible directly beneath the help tab). Under the “Devices” tab, find and select the first “MSP432E401Y” microcontroller under the Texas Instruments tab. In the right panel, go to the “Packs” tab and expand “Device specific,” then click install for the device family pack. Close the pack installer.
3. Download and install the TI Emulation Pack software using this link: https://software-dl.ti.com/ccs/esd/documents/xdsdebugprobes/emu_xds_software_package_download.html. Install this using all the default settings.
4. Plug the microcontroller into your PC. Open the command prompt. This can be done by searching for “Command Prompt” in the search bar of your PC. Copy and paste the following into the command prompt one by one:
   cd /d C:\ti\ccs_base\common\uscif\xds110
   xdsdfu –m
   xdsdfu –f firmware_3.0.0.13.bin –r
   Wait until the command line reappears, and then close the command line.
5. In Keil, open the “Options for Target.” Then under the “Target” tab, change the “Arm Compiler” to the “default compiler.” Then, go to the “Debug” tab and change the debugger to “CMSIS-DAP Debugger.” Beside this there is a “Settings” button, click it. Change the “Adapter” to “XDS110 with CMSIS-DAP.” Then click “OK.”
6. Download a release of Python 3.9. Python 3.9 can be downloaded using this link for your operating system: https://www.python.org/downloads/release/python-390/. Once it has been downloaded, open and execute the file. Run the setup and follow the prompted instructions, ensuring to select the “Add Python 3.9 to path” option during the installation process.
7. The link provided above will also download an IDLE within the default Python 3.9 installation, which is what we will be using for this project.
8. After Python 3.9 has been successfully installed, open the command prompt. To check if Python 3.9 has been installed, type “python --version” and you should get the

output “Python 3.9.0.” If you did not get this message, try to restart your PC and try again. If you still do not get this message, try to reinstall Python 3.9 again.

9. In the command prompt, we will install pip, Open3D and pySerial. To install pip, copy and paste the following: curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py
10. Then, type “python get-pip.py” to install pip. If pip was successfully installed, there should be a statement such as “successfully installed pip.”
11. Now that pip is installed, we can install Open3D. To do this, type “pip install open3d” in the command prompt. This may take a few minutes to fully install all the packages. If Open3D was successfully installed, there should be a statement such as “successfully installed open3d.”
12. Finally, we need to install pySerial. To do this, type “pip install pyserial” in the command prompt. If pySerial was successfully installed, there should be a statement such as “successfully installed pyserial.”

Now your PC has all the necessary software required to run this project. Now you can start the data collection process after a few more steps stated below:

13. Right click the python file called “2dX3_FinalProject.py” and select “Edit with IDLE.”
14. Now, you will need to check your COM port and change it in the Python file. This is for the UART serial communication being used. To determine your COM port, open the “Device Manager” on your PC. This can be found by searching for “Device Manager” in the search bar of your PC. Then, under the “View” tab, select “Show hidden devices.” Click the dropdown arrow next to “Ports (COM & LPT).” Your COM port is the port written in brackets next to “XDS110 Class Application/User UART.” Go back to the “2dX3_FinalProject.py” Python file, and change the COM port in line 6 according to your determined COM port. Save the file so your changes are saved, and close the file.
15. Build the circuit using the MSP432E401Y microcontroller, the VL53L1X ToF sensor, the ULN2003 driver board, and the MOT-28BYJ-48 stepper motor. The pin connections and wiring are explained the “2 Device Characteristics Table” section of this report. A full circuit schematic is provided in the “6 Circuit Schematic” section of this report, which visually displays the circuit connections and pins. Figure 4 at the end of the instructions also displays the circuit I built. Ensure the microcontroller is connected to your PC using the micro-USB cable. Ensure all connections and wires are secured and not loose, as this will affect the execution of the code. You will also need a cardboard box with a hole for the stepper motor.
16. After the entire circuit has been built, open the “2dx_studio_8c” Keil file from the “Final_Project_patem156” folder. Translate, build, and load this code onto the microcontroller. No errors or warnings should be present.
17. Press the onboard reset button on the microcontroller.
18. Double-click the “2dX3_FinalProject.py” Python file and follow the prompted message. The prompted message will say “Enter Number of Scans,” to which you will enter an integer value for how many scans you will be taking. Hit the enter key.

19. The system is ready to start. Move to your desired scanning location, ensuring the sensor is facing the direction you want to scan.
20. Some key things to remember are:
    PJ1: Starts and stops the data collection process
    LED D3: Measurement status LED (flashes when every measurement is taken every 22.5 degrees)
    LED D4: Additional status LED that also flashes
21. When you are ready to start scanning, press the onboard pushbutton PJ1.
22. In the Python file already opened, you will see each distance measurement taken by the sensor.
23. Once a full rotation is completed, the stepper motor and sensor will stop and the distance measurements will stop being collected. For the program I created, xyz is defined in a specific manner. In this program, yz is used for vertical distance slices, and x is used for displacement.
24. If you are doing more than one scan, press the onboard pushbutton PJ1 again after displacing the entire circuit setup to start the data collection again. Do this for however many scans you are taking.
25. Once you are done scanning, double-click the “O3D_FinalProject.py” Python file found in the same “Final_Project_patem156” folder. A message will appear asking you to “Enter Number of Scans,” which is where you will enter how many scans you took. After entering an integer value, click the enter key. This will display the obtained point cloud, which will display dots. You can click and drag on the window to manipulate the view of the point cloud.
26. Close the point cloud by clicking the “X” in the top right corner. After you close this, you will see the final 3D visualization. This is where the dots in the point cloud are connected. You can click and drag on the window to manipulate the view of the 3D visual.
27. Repeat steps 16 to 26 every time you would like to take a new scan to obtain a new point cloud along with a new 3D visual representation.

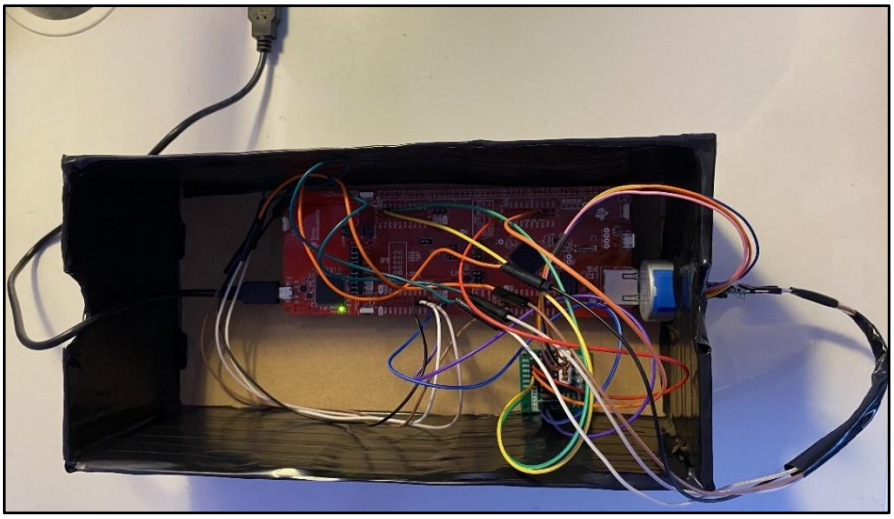

Figure 4. Built Circuit

Application Example following the Provided Instructions:

An application example of this project is the hallway scan I performed in the Engineering Technology Building of hallway section F. First, I found the location I wanted to scan within the building, which is section F. I had previously already done steps 1 to 14 before arriving at the building. I then built the circuit, following the circuit schematic I created for my project. This is step 15 in my instructions. Following this, I proceeded to follow and complete steps 16 to 26. Below you can see the obtained 3D visual representation and the hallway I scanned.

Figure 5. ETB Hallway F

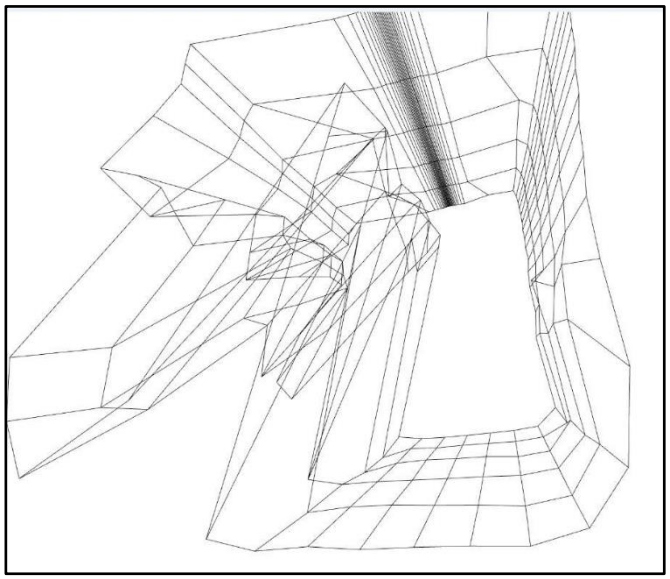

Figure 6. ETB Hallway F 3D Visual Representation

An alternate application example where you can use this project is to scan the inside of an object to create a 3D model that would fit inside it. To do this, follow steps 1 to 27 stated in the instructions. After obtaining the 3D visual representation graph, you can then use MeshLab to generate a .stl file for 3D printing. MeshLab is another software for processing and editing 3D meshes. In MeshLab, you can import the .xyz file and obtain the .stl file.

Expected Output:

After translating, building, and loading the “2dx_studio_8c.c” file onto the microcontroller, running the “2dX3_FinalProject.py” file, entering the number of scans, and pressing the PJ1 onboard pushbutton, the stepper motor will begin rotating and the sensor will begin the data collection process. When the data collection process starts, the user will see the obtained distance measurements on their screen. For the program I created, xyz is defined in a specific manner. In this program, yz is used for vertical distance slices, and x is used for displacement. Once all scans have been completed, the user will obtain a point cloud with all the plotted points. This point cloud will have the data displayed as unconnected dots. Once you close this window, you will see the 3D visual representations as all the data points will be connected with lines.

5 Limitations

1. There are some limitations to the microcontroller's floating-point capability and the use of trigonometric functions. The MSP432E401Y microcontroller has a 32-bit ARM Cortex-M4F processor core. This includes the floating-point unit (FPU). This microcontroller has an IEEE 754 32-BIT FPU integrated within the core. This means the microcontroller supports floating points up to a precision of 32 bits for addition, subtraction, multiplication, division, and square root operations. The microcontroller is able to perform the calculations with the trigonometric functions, but for the ease of this project, these were done within Python. The trigonometric functions within the math.h library used in Python is able to perform the necessary calculations with float variables. There was no errors or issues caused by the combination of these limitations since the device and application used C language. This means that there are float data types that can still retain accuracy.

2. The maximum quantization error can be calculated for each of the ToF modules. This can be done using the formula: $Maximum\ Quantization\ Error = \frac{V_{FS}}{(2^m)}$. In this equation, VFS is the full-scale voltage, and m is the number of bits.

$$ToF\ Maximum\ Quantization\ Error = \frac{V_{FS}}{(2^m)}$$

$$ToF\ Maximum\ Quantization\ Error = \frac{3.3V}{(2^8)}$$

$$ToF\ Maximum\ Quantization\ Error = 12.891mV$$

$$MCU\ Maximum\ Quantization\ Error = \frac{V_{FS}}{(2^m)}$$

$$MCU\ Maximum\ Quantization\ Error = \frac{5V}{(2^{12})}$$

$$MCU\ Maximum\ Quantization\ Error = 1.221mV$$

$$IMU\ Maximum\ Quantization\ Error = \frac{V_{FS}}{(2^m)}$$

$$IMU\ Maximum\ Quantization\ Error = \frac{5V}{(2^{16})}$$

$$IMU\ Maximum\ Quantization\ Error = 76.294uV$$

3. The maximum standard serial communication rate I can implement with the PC is 115200 bits per second (bps). This was verified through the device manager by checking the properties of XDS110 Class Application/User UART (COM3). The speed I implemented was 115200 bits per second (bps), which was also verified by checking the properties of the port I used.

4. The communication method used between the microcontroller and the ToF modules was I^2C. I^2C communication protocol was used to communicate the data between the ToF sensor and the microcontroller. To do this, I connected SCL to PB2. I also connected SDA to PB3. The data from the sensor would go through the microcontroller, and then it would go to the PC via UART and the COM port used at a baud rate of 115200 bps. The speed used between the microcontroller and the ToF modules was 115200 bits per second.

5. Reviewing the entire system, the element that is the primary limitation on the system speed is variable PSYSDIV, the speed of the stepper motor, and the speed of the ToF sensor being used. The PSYSDIV variable limits the speed since the microcontrollers bus speed is 120MHz, and my assigned bus speed is 16MHz. I tested this by using the Analog Discovery 2 and the Waveforms 2015 software. The stepper motor has a speed of 100Hz and the sensor has a speed up to 400kHz. This bottlenecks the system due to the fact that they are both a lot smaller than the bus speed, which is 120MHz for the microcontroller and 16MHz for the assigned bus speed.

6 Circuit Schematic

Figure 7 below displays the full circuit schematic for this project. The MSP432E401Y microcontroller, the VL53L1X ToF sensor, ULN2003 driver board for the MOT-28BYJ-48 stepper motor, and the PC are incorporated in this circuit schematic. The circuit schematic shows all the connections between each device used. Since I used the onboard pushbutton PJ1, PJ1 is included in the schematic, but it is not connected or wired to any of the components.

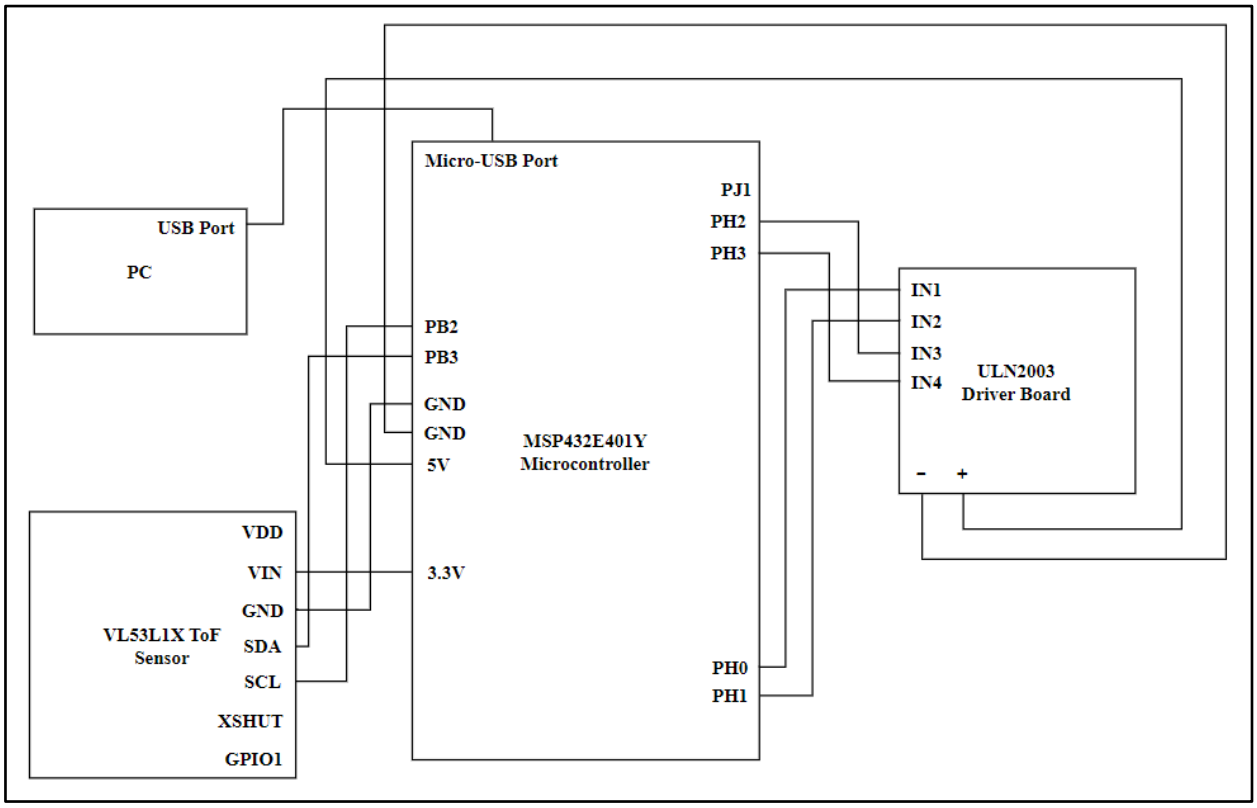

Figure 7. Circuit Schematic

6.1 Programming Logic Flowchart

Figure 8 below displays the programming logic flowchart for this project, which displays how the project works.

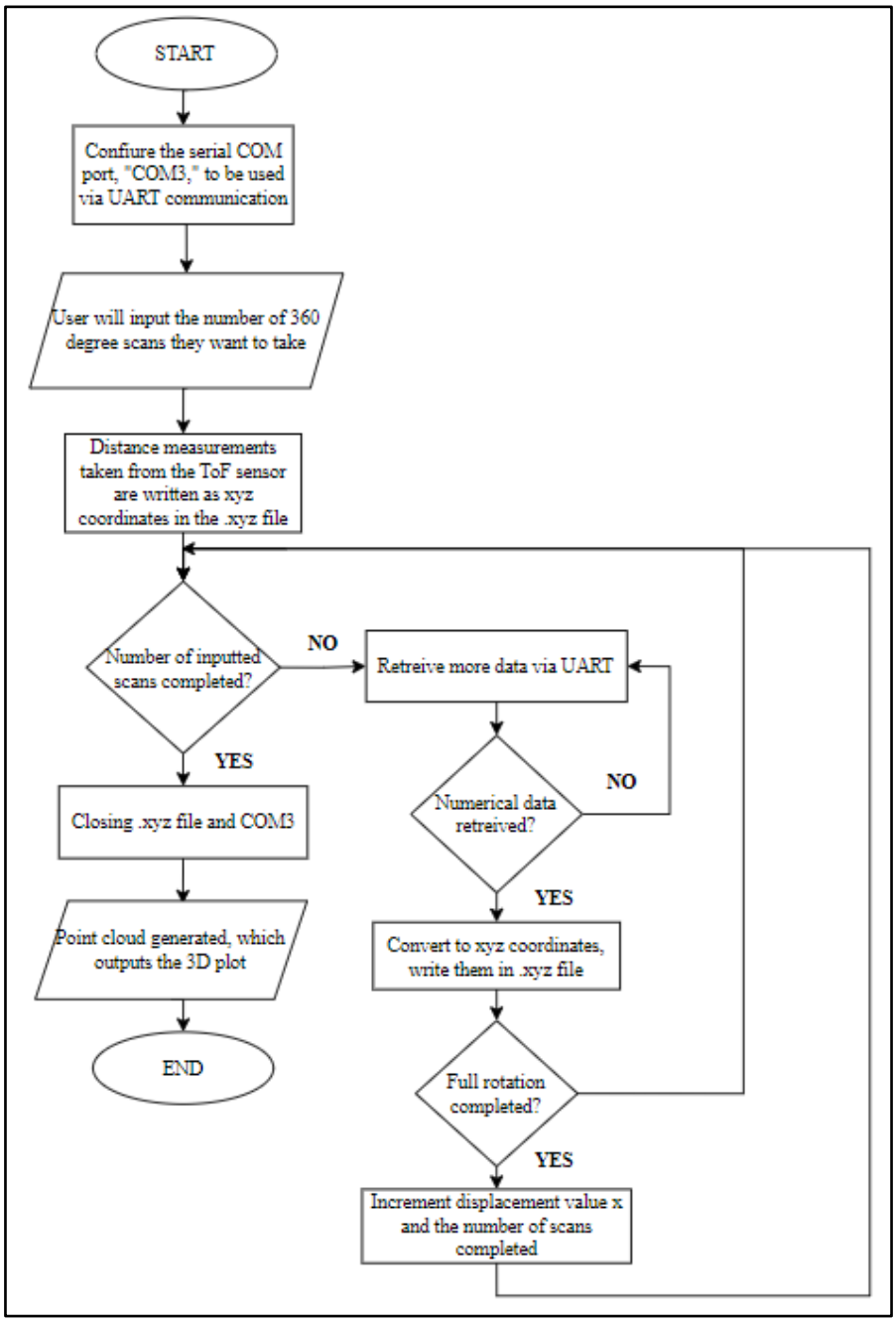

Figure 8. Programming Logic Flowchart