Final Project Report

COMPENG 2DX3

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

Adin Aviv – aviva

2024-04-16

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 **[Adin Aviv, 400368990]**

Device Overview

Features

- Comprehensive LIDAR System made for indoor environments, supplying in-depth LIDAR scanning and visualization abilities such as the ability to connect with a PC visualizing scanned data.
- The Texas Instruments MSP432E401Y SimpleLink™ Ethernet Microcontroller includes an Arm Cortex-M4F Processor Core, runs at a bus speed of 80 MHz, and is has 1024KB of Flash Memory, 256KB of SRAM, and 6KB of EEPROM. Its power supply ranges from 4.75 VDC to 5.25 VDC through an XDS-110 USB Micro-B connection and has status LEDs to indicate when it is ready for data scanning and when it is transmitting data to a PC. It can be programmed using C, Assembly or C++. [datasheet micro]
- The VL53L1X Time-Of-Flight Sensor can measure a maximum distance of 400 cm with a maximum ranging frequency of 50 Hz. It interfaces with devices through I²C with a maximum speed of 400 KHz. It needs a power input that's within a range of 2.6 to 5.5 VDC.
- The MOT-28BYJ48 Stepper Motor contains a ULN2003 Driver. It needs an input voltage of 5 to 12 VDC and has 512 steps per rotation for precise movement.
- Python 3.8 (64-bit) IDLE is used for visualization along with the Open3D package for constructing three-dimensional models based on the scanned data.
- I2C and UART are used for inter-system communication. I2C is for data transfer between the microcontroller and the Time-of-Flight sensor. UART is for communication between the microcontroller and a PC at a baud rate of 115,200 bps.

General Description

Our integrated distance measurement system operates by scanning indoor areas and constructs a 3D model of it. This system performs 360-degree spatial measurements in the YZ plane, and for every additional 360-degree measurement they are shifted manually in the X-plane. The step size of the motor determines how many samples are taken per 360-degree measurement, which impacts the resolution of the finalized 3D model. We send this information to a PC for visualization using the Python module Open3D.

The system hardware is comprised of the MSP432E401Y microcontroller board for operating the integrated system, the ToF sensor for distance measurements, the stepper motor with the driver for full rotation scans, a pair of pushbuttons which take user input, and a pair of onboard LEDs. We use C to program the device for operating the motor and scan function and for communication between the PC and the ToF. We use Python to reconstruct a 3D model using the Open3D module for mesh creation with the received information.

The ToF sensor package determines the spatial measurements from the time of flight of a pulse of light. The angle of the sensor is adaptable and can be changed by rotating the motor using the step size that we want. The information is obtained using I2C and is stored as local data on the integrated system, and we repeat this process for the number of measurements that we want in the X-plane. After this is done, the information is sent to the PC using the UART protocol so it can be visualized with the Open3D package. [1]

Block Diagram

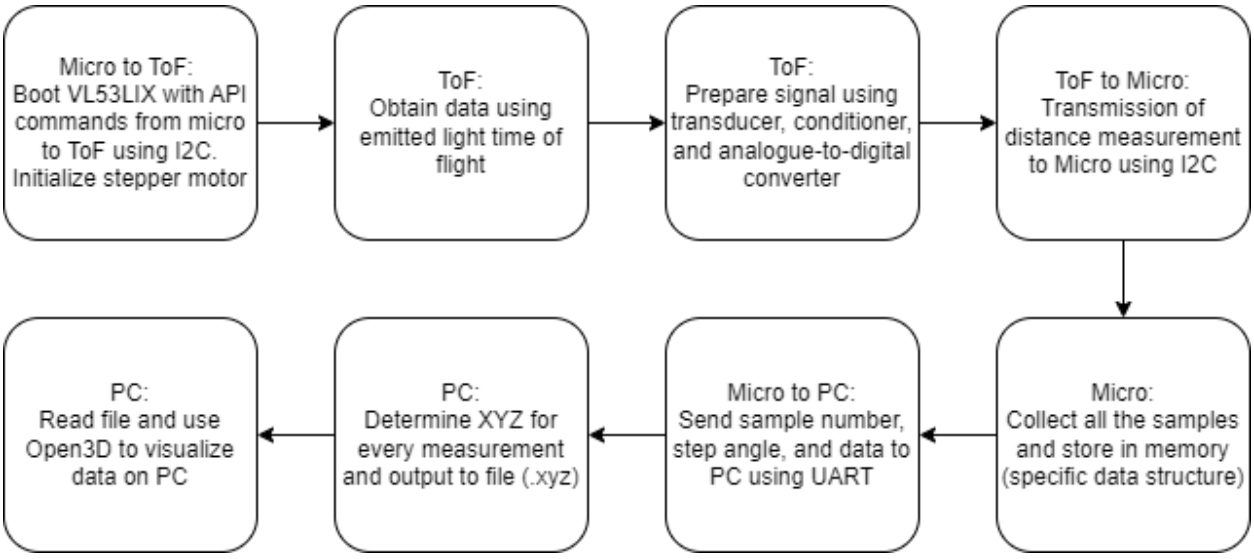

Figure 1: Block Diagram

Device Characteristics Table

General System Setup	
Feature	Description
Bus Speed	80MHz
UART baud rate	115200bps
Python version	3.8 (64-bit)
COM Port	COM4
Microcontroller Power	USB (5VDC)
VL53LIX Setup	
VL53LIX	Microcontroller
VDD	-
VIN	5V
GND	GND
SDA	PB3
SCL	PB2
XSHUT	-
GPIO1	-
ULN2003 Setup	
ULN2003	Microcontroller
+	5V
-	GND
IN1	PH0
IN2	PH1
IN3	PH2
IN4	PH3
LED Setup	
User LED	Assignment
PN0	Measurement status
PN1	Additional status
Pushbutton Configuration	
Microcontroller Port	Assignment
PL0	Start data acquisition
PM0	Initiate data transfer

Figure 2: Device Characteristics Table

Detailed Description

Distance Measurement

The integrated distance measurement system includes a VK53L1X LIDAR ToF sensor for obtaining spatial measurements. This component comes in a completely integrated package which contains the condition, transducer stage, and analogue-to-digital conversion. The ToF sensor depends on the time of flight of 940 nm light pulses to calculate the distance that is relative to the sensor. It considers the time needed for the light to travel and come back from the object to approximate the distance, and it also considers the speed of light. The next step in the process is for the information to be transmitted to the microcontroller using the I2C communication protocol. In this process, the embedded system uses the UM2510 API with the ToF sensor. Several functions can be used to initialize the device to the desired specification and also it can obtain distance data in terms of the corresponding measurement. The system measures the confidence of the measurement using the variable for range status. A value of 0 is returned if there is no error, a value of 1 or 2 if there is a warning, and a value of 4 or 7 if an error has occurred. At each stage of the scanning process, the system software verifies that no errors exist in the range status, so an accurate reading is obtained. After verification, the distance reading is stored (mm) in its corresponding spot in the 2D array to be transmitted later.

The sensor is mounted to the stepper motor using a 3D-printed mount. The motor is used to alter the angle of the sensor each time a measurement is taken. The ToF calculates the distance by starting to face 90 degrees horizontally. The final 3D model has a resolution proportional to how many measurements are taken per displacement in the X-plane, or it is inversely proportional to the step angle of the motor. For each measurement the ToF takes, the motor must step at a lower angle for the best resolution. Furthermore, the device firmware utilizes an interrupt-based program. At first, the system is configured by initializing the system clock, UART, I2C, LEDs, motor, sensor, and buttons. After this, interrupts are configured for the push buttons on the breadboard and wired to PM0 and PL0. The measurement status LED is connected to PL0 because this is the pin that the push button for measurement is connected to. The additional status LED is connected to PM0 since this is the pin that the push button for controlling the motor rotation is connected to.

By default, the system is in an idle state until the user takes further action, which is due to the nature of the interrupt-driven system. Onboard LED D2 flashes every step of the motor while it rotates, while LED D1 turns on if measurements are being taken. When the user desires to start obtaining distance data, button PL0 is pressed, which causes a GPIO interrupt to be triggered. The main loop in the Keil program realizes the change, and it executes a function which scans the YZ plane using the stepper motor and the ToF sensor. The distance information is obtained using I2C communication to the ToF from the microcontroller. After the motor completes a full rotation of 360 degrees, it goes back to the home position by spinning 360 degrees in the other direction, and the program goes back to the prior rest state. The user must manually increment the system's location in the X-direction so that more scans can be obtained in the X-plane. It is imperative to keep the relative location in the Z-plane from prior scans the same. When the user has moved to the new location, it is required to do all of this again by pressing the measurement status button again.

```
void FlashI2CTx(){
  FlashLED2(2);
}
void FlashI2CRx(){
  FlashLED2(2);
}
```
Figure 3: LED code block 2

```
// Toggle LED from on to off and vice versa with a delay
GPIO_PORTN_DATA_R ^= 0b00001;
SysTick_Wait(9500000);
GPIO_PORTN_DATA_R ^= 0b00001;
// If button is pressed during operation, break out of the loop
if (!currently_running) {
  break;
}
```
Figure 4: LED code block 2

Visualization

After the distance information is obtained using the process discussed above, it is required for it to be sent to the user's PC for the visualization process. The code for the visualization process was made in Python 3.8, and the file containing the code should be run using IDLE. It is easiest to open IDLE directly from the Windows menu and paste the directory of the file into Explorer when opening the file from the shell.

At the point when the number of scans we want is reached, the system enters a rest state. To send data to the PC, the additional status button must be pressed – the button which is connected to PM0. This will cause the LED D2 to flash for every rotation of the stepper motor, which indicates that the system is in a state of transmission at this point. Additionally, the main while loop will detect the change and will call a function that will wait to confirm that the user is ready to receive the data. With the IDLE shell running on the PC, the user does not need to perform any action using the keyboard, because the system will automatically be ready for the next transmission. The only step the user must take is to use the button which controls the motor which is located on the breadboard. The number of measurements in the X-plane, the number of samples for every X-plane measurement and every 16-bit distance measurement from the entirety of the samples using UART. It is important to note that every distance measurement has a length of 2 bytes, and UART sends one byte individually. The first to be transmitted are the 8 least significant bits, and then the 8 most significant bits are transmitted, for every distance measurement. The system is at this point reset and goes back to its initial state and is ready for additional measurements to be taken.

Using the Open3D, Math, NumPy, and PySerial packages, the Python code now can receive these UART frames and put the relevant data in memory. The data that is stored is the number of measurements and the number of samples for every measurement. By making use of these variables, the system iterates through a specific number of times to obtain each sample, rebuilding the two bytes into a single integer value every time. It is at this point that the distance information must be converted into a coordinate for the visualization process. The Python code uses the number of samples for every measurement to determine the degree between scans which was declared in the firmware. The system assumes an initial angle of 90 degrees in the horizontal direction and converts the distance and angle to a cartesian coordinate using rudimentary trigonometry. It is important to note that for every distance measurement sample, the angle is

lowered by the degree, which is between scans, which was determined before. Every set of the measurements in the X-plane is assumed to have a constant distance between them in the X-direction, and it is set to be 600 mm as the default, but it can be altered. Thus, for every X-plane distance and angle, the coordinate is obtained and stored on a new line of an XYZ file.

The file is now made and updated to contain the coordinates. This means that the Open3D package is utilized for the visualization process. A point cloud is initially created to present the relative position of every point obtained by the system. After we exit this window, the line visualization is displayed, which connects all the points in the frame to the points which are next to it. A loop is made for every set of measurements in the X-plane, which connects every point to the two points that are next to it. Additionally, lines are made between the points that are adjacent in differing X-locations. This occurs by iterating through every point and in the following frame the script connects the adjacent points. The user can study all of this, which the PC outputs.

Application Note, Instructions, and Expected Output

Application Note

The purpose of the integrated distance measurement system is to capture its surroundings and reconstruct a 3D model. Some considerations that must be considered are if the area we want to scan contains reflective or glass surfaces, which will cause disruptions in the 3D model that is generated. The areas that are scanned are indoors and cannot be too large, as the sensor cannot take measurements if the displacement is too high. When scanning, it is critical to evenly space out the distance between scans. The system applies to many different fields such as interior design, architecture, and virtual reality. It is truly revolutionizing how we can build a map of our environment using a tiny device such as the ToF sensor. One issue to make note of is that when the motor spins, the wires which connect the sensor to the microcontroller twist in a problematic manner. The wires partially obstruct the sensor; however, this hardware issue isn't concerning enough to alter the design because there is no apparent effect on the scan.

There are several system requirements which we must consider for our embedded system. Our system is designed to be embedded and integrated with components which are interconnected and can communicate with each other. Inter-system communication is critical for functionality, as we are using serial communication and other protocols to transfer data between the PC, sensor, and microcontroller. The microcontroller is multipurpose. It not only provides the means to control the sensor movement, which in other words is the stepper motor, but it does much more. Even without flashing the microcontroller, we are still able to use it as a power source and for ground as well. Furthermore, we can use its onboard LEDs in our designs, and we can program the built-in user input buttons as well. It is important to note that the program does not solely depend on hardware, as it is also software-based. We need to connect the PC to the microcontroller not solely as a power source, but because we need to run Python, specifically PySerial for communication between the PC, microcontroller, and sensor. We also need to run Open3D for visualization. We must make note of safety precautions. This is important to prevent damage to the integrated system and its components, as well as harm to the user. One warning sign to watch out for is if the motor overheats.

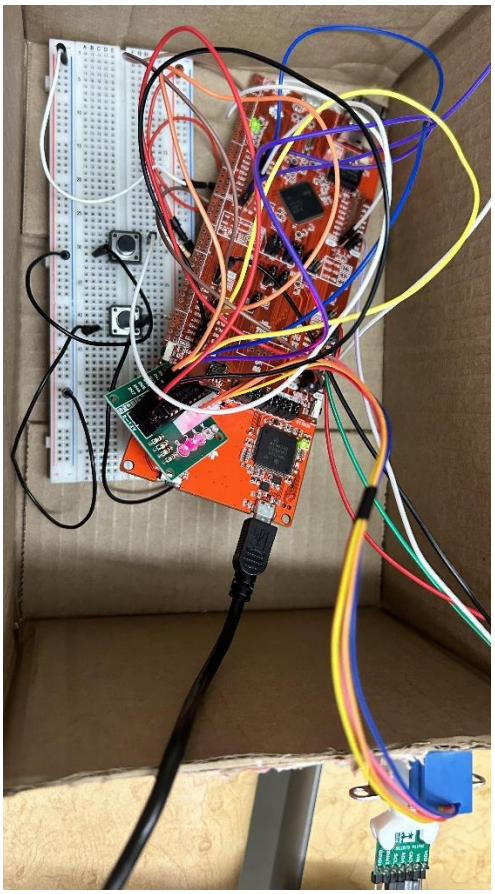

Figure 5: Picture of Entire System

Instructions

The first step is the wiring as shown in earlier sections of the report. The embedded system contains the VL53L1X ToF sensor. The VIN pin should be wired to the 3V3 pin on the microcontroller. Connect ground to ground. The SDA pin should be connected to PB3 and the SCL pin should be connected to PB2. We don't wire the VDD, XSHUT, and GPIO1 pins. Our code configures PN0 and PN1 as the two LEDs to be used for measurement status and data transmission status. The push button for measurement should be connected to PL0, and the one for starting the motor should be connected to PM0. For the stepper motor, the four input pins are connected to four pins from Port M according to their corresponding number. We connect the + to the 5V on the microcontroller, and the – to GND. The system requires the ToF sensor to be attached to the stepper motor, which is achieved using a 3D-printed mount that attaches to the rod part of the motor and holds the side of the circuit board of the sensor.

Before running anything, go to the device manager and find the COM port after connecting the microcontroller to the PC. This characteristic is device-dependent. Go into the Python file and change the COM port number at the top of the code to the correct value. We must use a version of Python between 3.7 and 3.11, otherwise it is required to reinstall an older or newer version of Python, depending on the circumstances. Open the Python file from within IDLE and be sure to save any changes after modifying the COM port number. Open3D and PySerial are essential for our system to work, so ensure they are both installed on the PC that is

being used. This can be done by going into the command prompt and using the “pip install” command.

To modify the LEDs, use the onboardLEDs.c file. FlashLEDX is called twice at the bottom of the file, and select the appropriate value for X. Keep the busSpeed() function call in the main loop commented unless intending to check the bus speed with the AD2. If it is uncommented, the entire system functionality will be broken. If we want to modify the bus speed, go into PLL.h and the comments contain the details on what to change in PLL_innit(). The formula for bus speed frequency is “480MHz/(PSYSDIV+1)” and see the below screenshot for the look-up table. Use the table to select the correct value to define PSYSDIV. To measure the bus speed, connect the positive probe for channel 1 from the AD2 to PE0 on the microcontroller, and ensure to ground it.

PSYSDIV	SysClk (Hz)
3	120,000,000
4	96,000,000
5	80,000,000
7	60,000,000
9	48,000,000
15	30,000,000
19	24,000,000
29	16,000,000
39	12,000,000
79	6,000,000

Figure 6: Look up table for bus speed

After all the connections are made and the required software is installed, we can begin the scanning process. Be sure to mount the sensor to the stepper motor, and when scanning, it is required that the motor be facing outwards on the side of the area that is to be scanned. Run the Python file in IDLE, and follow the prompts given. The program prompts the user for the number of scans and the displacement. After this is done, we can start using the hardware of the system. Press the button connected to PL0 to enable measurements to be taken, and to confirm this is true check the terminal when running the Python file in IDLE and be sure to press the button again if the message “no measurement taken” is displayed. To turn start the motor rotation process, press the button connected to PM0. Sometimes the button is finicky, so a double press may be required. This button starts and can also stop the rotation of the stepper.

If we want to scan an area such as a hallway, we must divide it up into steps that can be scanned. It is useful to use footsteps as a reference. Using a chair to hold the PC and the box containing are embedded system is helpful. Ensure no people or unwanted objects are in the area threat is being scanned. Take steps forward to extend the scan, and ensure the steps are evenly spaced for maximum accuracy. Once the scan is complete, the Python program will automatically open a colour-coded dot map of the 3D object in a new window. Exit this window and the mesh view of the scan will open in another window.

Expected Output

With glass features in the hallway, which was assigned to be scanned, achieving a perfect result is difficult. The below image is a side-by-side comparison of the hallway and the 3D scan which the embedded system reconstructed. The triangular feature at the end of the mesh figure is

the groove in the front of the hallway which contains the water fountain. One side of the 3D model is slightly triangular, which is most likely because of the glass on the left. It took twelve scans to achieve this 3D figure and two attempts for the scanning process. In the first attempt, the sensor fell from the box, which unfortunately meant that the process had to be restarted. If any space is scanned and the resulting 3D model contains strange and unwanted features, consider the surroundings and if glass or reflective surfaces are in the area. If not, revisit the stability of the device while it is being scanned, and how evenly spaced out the scans are.

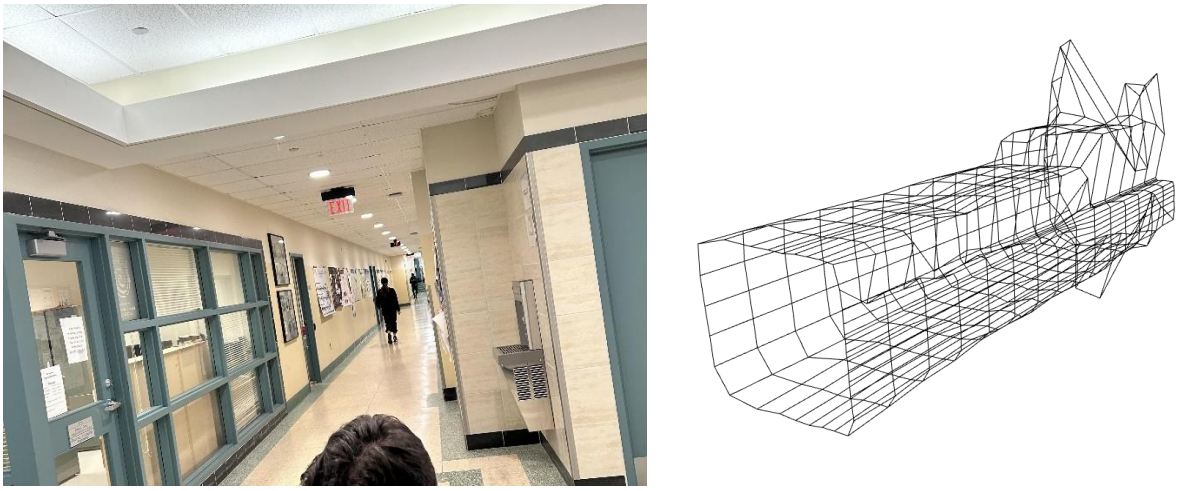

Figure 7: Side by side comparison of hall picture and scan

Limitations

Summarize any limitations of the microcontroller floating point capability and use of trigonometric functions.
The microcontroller that we use is limited when it comes to its floating-point abilities. FPU stands for floating-point unit. The device's FPU only supports single-precision floating point operations and values. It takes 32 memory bits to store a single-precision floating point value. Of all these bits, 23 of them are used for the fractional part of the number in memory. This does not mean that the possible error that results from a floating-point approximation in the microcontroller is a problem within the device. The microcontroller firmware decares integer variables of solely 8 or 16 bits in size for the data flow. The error comes from the visualization software on the PC. This is because trigonometric functions and floating-point values are used solely in the Python code. For this reason, the system is making an approximation because the cartesian coordinates are stored in floating-point values, which is in effect a source of error. When the system calculates the XYZ coordinates from the distance integer value, it approximates pi to 15 decimal places based on the math package. The cosine and sine function are also approximated. Consequently, the final image reconstruction has more possible inaccuracies.

Calculate your maximum quantization error for each of the ToF module.
The quantization error is another source of error for the information received by the ToF sensor. The maximum quantization error is equal to 4000 mm divided by 2^16, which equals 0.0610 mm. The percentage of the full-scale voltage is equal to the max quantization error divided by 4000 mm and then all of this multiplied by 3.3 V. This equals 5.035*10^(-5) %.

What is the maximum standard serial communication rate you can implement with the PC. What speed did you implement and how did you verify?
The value 128,000 bps is the highest standard baud rate which can be put into effect by the PC. We looked at the characteristics of the COM port in the device manager to obtain this value for the speed. The value was deemed to be standard because the maximum standard baud rate mentioned above was the biggest baud rate which was in the drop-down selection. For the project, we implemented a different baud rate of 115,200 bps for the UART communication protocol. We confirmed this value using the RealTerm software by choosing the baud rate and ensuring communication was in effect between the PC and microcontroller. The microcontroller had the same rate configured for UART.

What were the communication method(s) and speed used between the microcontroller and the ToF modules?
The communication protocol used to transfer data between the microcontroller and the ToF modules was I2C, and the speed was 100 kbps.

Reviewing the entire system, which element is the primary limitation on system speed? How did you test this?
Reviewing the whole system, the element which is the main limitation on system speed is obtaining the distance data from the sensor and motor setup. The sensor includes a programmable timing budge which can be configured from 20 ms to 1000 ms. The budget is 140

ms for long-distance mode, meaning the duration of ranging for every measurement is 140 ms. Furthermore, the motor requires time for rotation while it is cycling through positions. After considering these timings, when combined they are bigger than the required timing for serial communication by a significant amount. For this reason, they are the most impactful on the system speed. To test this we needed to look at how much data is being transmitted over serial, and we also need to check the amount of time needed for the transmission to be completed. It is notably quicker than the distance reading process because of the high speed of transmission.

Show the steps and calculations to configure your assigned system Bus Speed from Table 1.
My assigned bus speed was 80 MHz, which was configured in PLL.h by using the look-up table. The speed matched the value “5” for PSYSDIV, so no calculation was required. The steps to check the bus speed with the AD2 were previously discussed.

Circuit Schematic

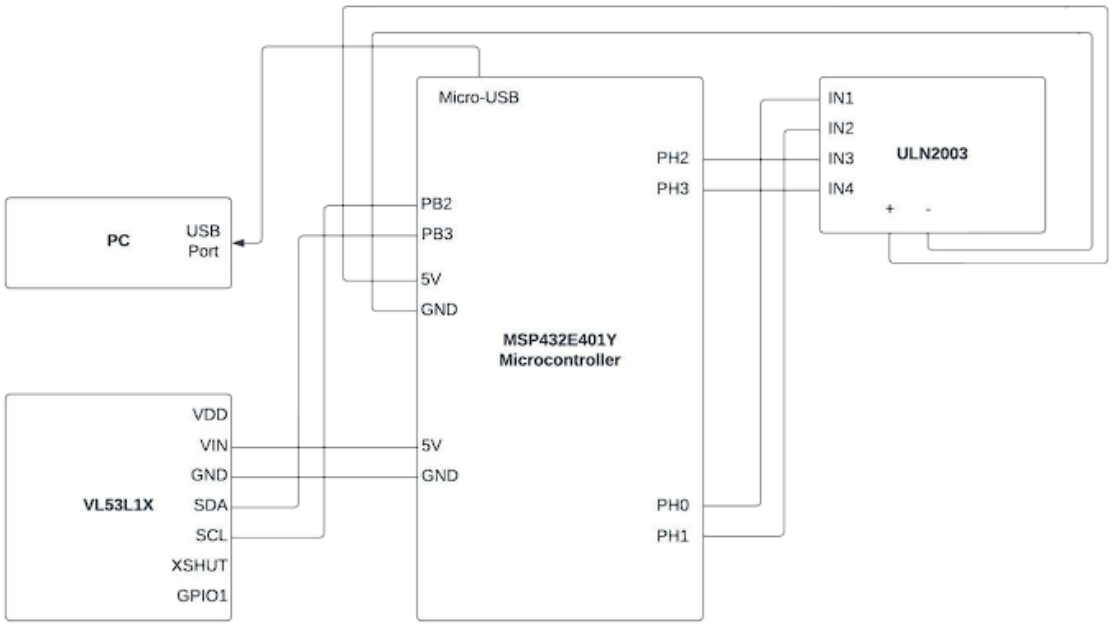

Figure 8: Circuit Schematic

Programming Logic Flowcharts

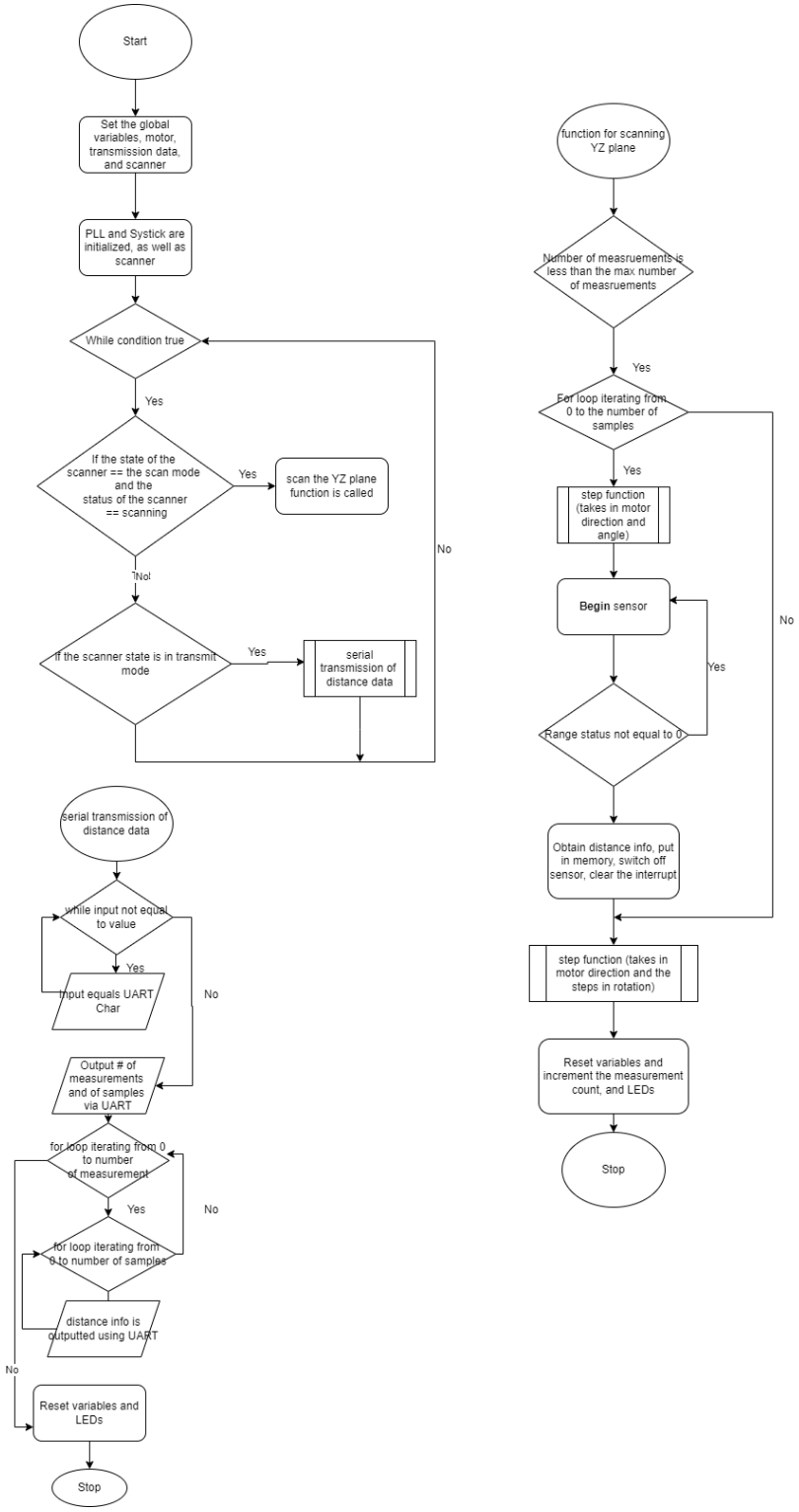

Figure 9: Flowchart 1 for C code

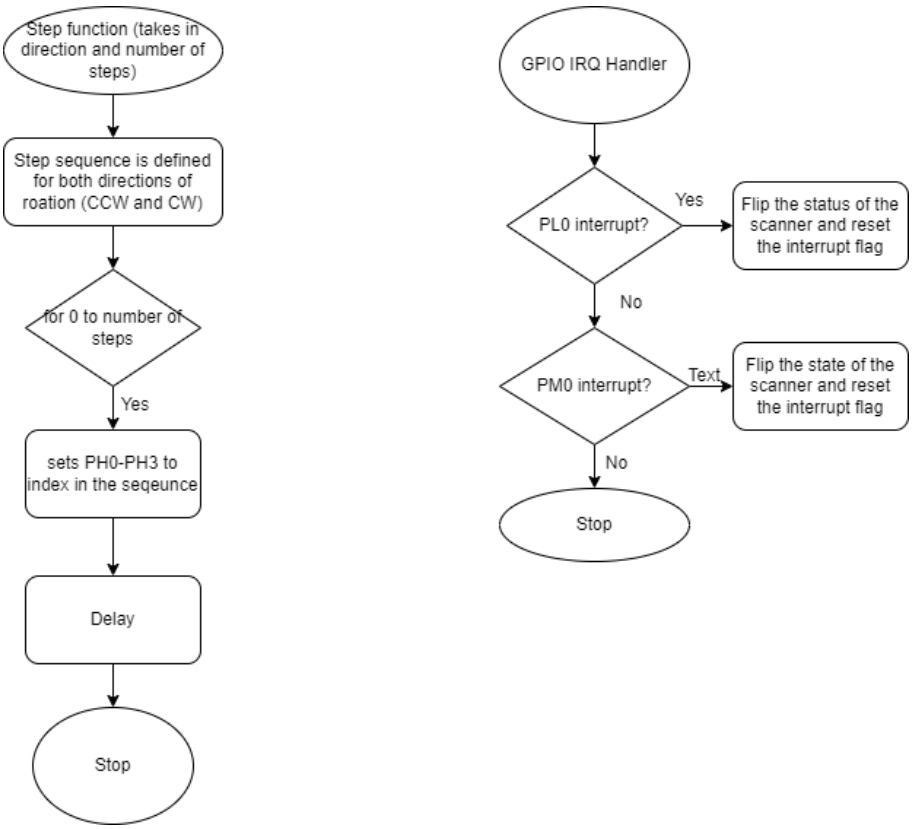

Figure 10: Flowchart 2 for C code

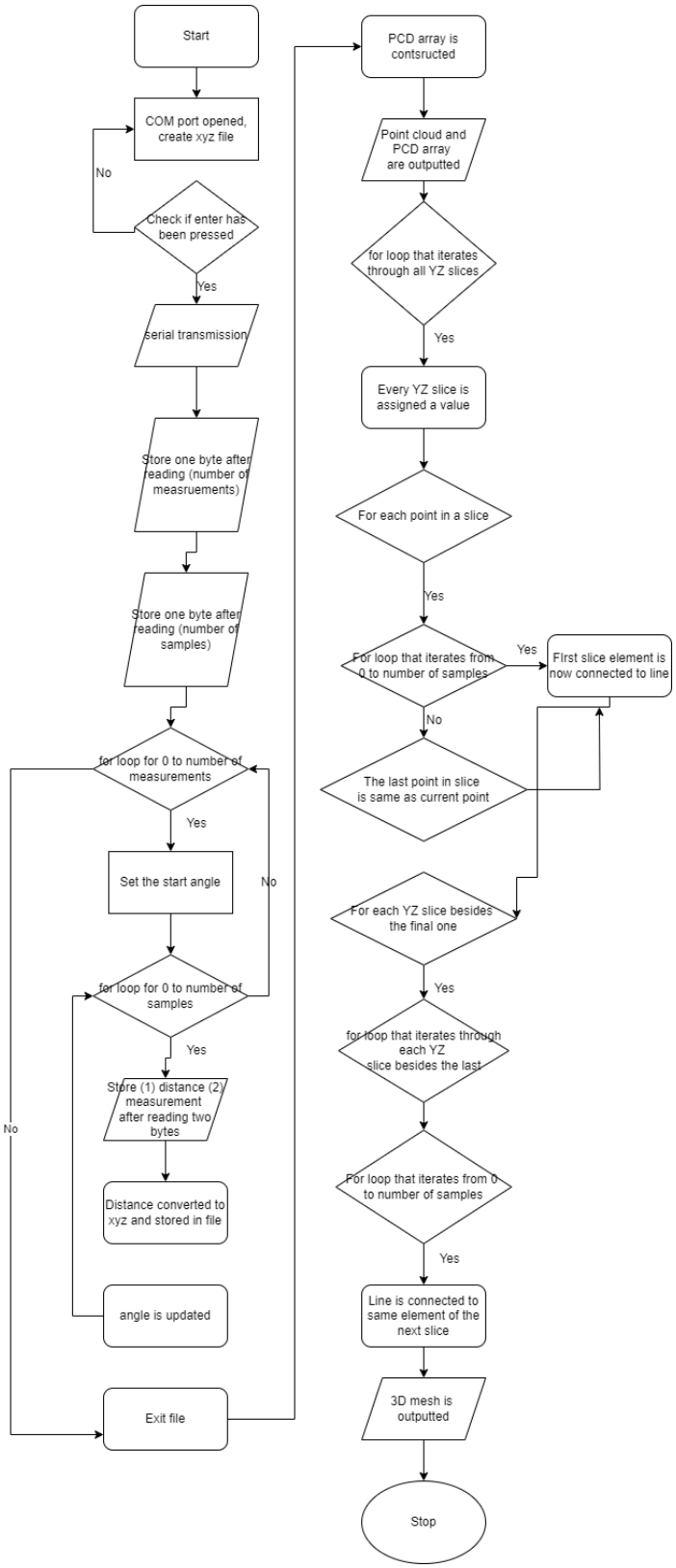

Figure 11: Flowchart for Python code

References

[1] S. Athar, T. E. Doyle, and Y. Haddara, *Computer Engineering 2023_2024_2DX3_Project_Specification.* Hamilton: DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING, MCMASTER UNIVERSITY, 2024.