

COMP ENG 2DX4: Final Project

Submitted By:

Areeb Jamal, jamala19, 400315588

Date Submitted:

April 17th, 2023

As (a) future member(s) 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.

Device Overview

Features

T.I MSP432E401Y Microcontroller

- ARM Cortex-M4F Processor Core.
- Static RAM (SRAM) of 256 Kb with around 2GB/s of bandwidth at 120 MHz clock frequency.
- Onboard LEDs used for measurement status and visualization.
- Onboard push button used to start, stop, and reset functions.
- Can be programmed in C, C++, or assembly language.

VL53L1X Time-of-Flight Sensor

- Using infrared laser light, collects up to 4000mm of distance measurements.
- Has 3 modes of operation: short, medium, long.
- Operating voltage of 2.6 V to 3.5 V.

28BYJ-48 Stepper Motor & ULN2003 Stepper Motor Driver

- 360-degree rotation captured using 512 turns/rotation.
- 4 connected LEDs indicate status of motor.
- Spinning knob used for visualization: capable of being the base to connect small components like the VL53L1X ToF sensor.
- Bidirectional rotations used to detangle wires.
- Operating voltage of 5 V to 12 V.

Data Communication Protocols

- I²C communication used between microcontroller and ToF sensor.
- UART communication at 115200 bps baud rate used between microcontroller and PC through COM8.

Data Plotting

- Python3 code written through IDLE.
- 3D graph generated through matplotlib library.
- .txt file used to store values.

Total Cost (approximately)

- Microcontroller: \$70
- ToF Sensor: \$9
- Stepper Motor: \$3
- Total: \$82

General Description

To collect spatial distance and create a graphical 3D representation of the surroundings, the integrated lidar system followed a general flow of tasks. The MSP432E402Y microcontroller first performs essential tasks such as controlling the stepper motor, collecting distance measurements from the VL53L1X TOF sensor, indicating measurement status through onboard LED D4, and managing operation via the onboard push button. These tasks done by the microcontroller are initialized through code flashed from Keil uVision5. The TOF sensor then utilizes an ADC process to capture non-electric signals and converts them into digital measurements in millimeters. The microcontroller initialized the sensor capabilities through I²C

communication. The sensor is fixed on the stepper motor knob to allow a complete 360 scan, taking 64 measurements per rotation, and visualizing up to 10 full rotation scans (640 measurements). The distances gathered are transmitted to the microcontroller through I²C communication, which transforms them into meters and sends them to Python via UART. The measurements are saved by Python into a text file for storage until prompted by a user input for a graphical representation. It will then take the measurement information given and translate it to a cartesian x, y, z coordinate system using trigonometric functions. The result of this produces an accurate 3D visualization of the environment.

Block Diagram

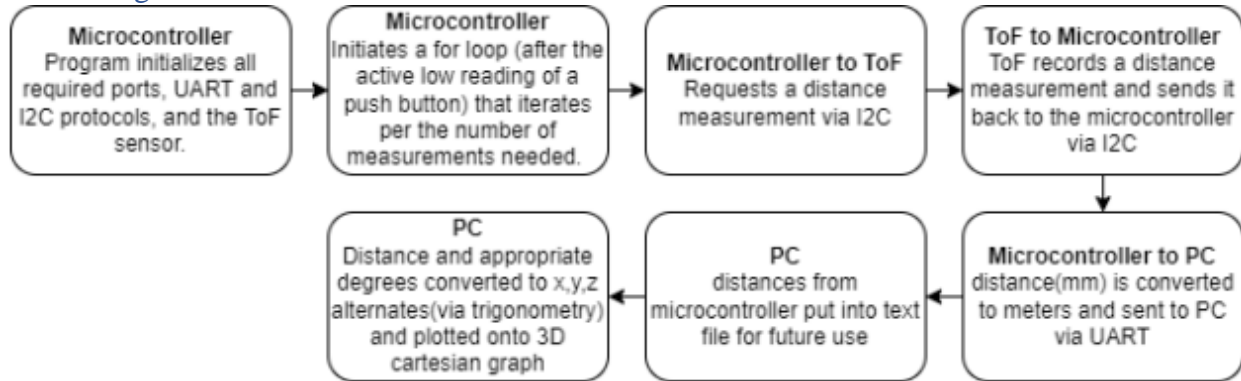


Figure 1: Block diagram outlining the flow of the project component connections.

Device Characteristics Table

	Specification	Detail
MSP432E401Y Microcontroller	Bus Speed	50 MHz
	Serial Port (UART)	COM8
	Onboard LED	PF0/D4
Python	Version	3.10.11
	Baud Rate	115200 bps
	Libraries	matplotlib & numpy
	Device Pin	Microcontroller Pin
VL53L1X ToF Sensor	VIN	3.3 V
	GND	GND
	SDA	PB3
	SCL	PB2
ULN2003 Stepper Motor Driver	+	5 V
	-	GND
	IN1	PH0
	IN2	PH1
	IN3	PH2
	IN4	PH3

Table 1: Technical details and port connections of devices.

Detailed Description

Distance Measurement

The distance measurements were taken by the VL53L1X ToF sensor mounted on top of the 28BYJ-48 stepper motor. After flashing the MSP432E401Y microcontroller with instructions written in Kiel, the microcontroller controlled the stepper motor according to the onboard push button status to obtain 360 degrees of measurements. The microcontroller communicated to the ToF sensor to boot up via I²C and the ToF sensor returns the measured distances back. The microcontroller then sent the measured values to the PC via UART communication where it was displayed using the matplotlib library in Python.

The Keil uVision5 program was written based on studio 8C template code. The bus speed was set to the value assigned to based on student number. In the case of this project, 50 MHz was the assigned bus speed value for student numbers with the least significant digit of 8. Looking at the PLL.h file, the PSYSDIV value that attributed to 50 MHz bus speed was calculated using Equation 1.

$$480 \text{ MHz} / (\text{PSYSDIV} + 1) = 50 \text{ MHz}$$

Equation 1: Calculating PSYSDIV value.

The PSYSDIV value was calculated to be 8.6 and this value was written in the PLL.h file, as seen in Figure 2.

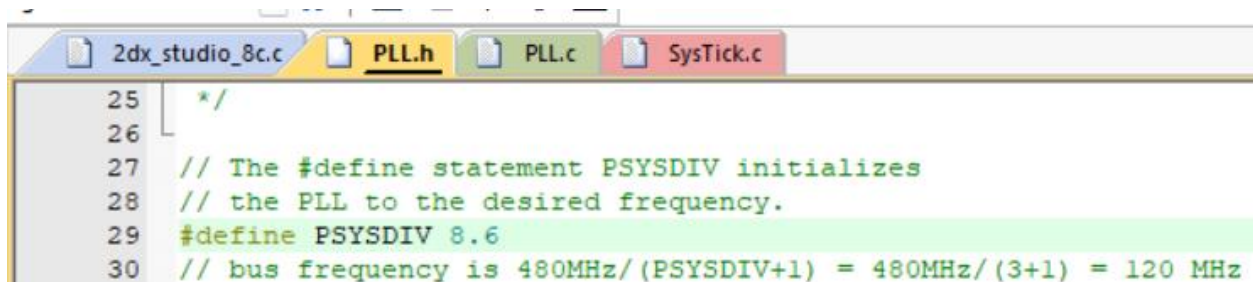


Figure 2: PSYSDIV value according to assigned bus speed.

The onboard push button PJ1 was used to start, stop, and reset the measurement scanning of this project. Once the button was pressed, the loop of 64 steps was begun and the data would be collected and stored for an entire 360 rotation. This if statement to check if the button was pressed utilized bit masking to check the active low configuration of the onboard push button. After each button press, the system is programmed to assume the next measurement will occur 10 cm forward horizontally.

The ToF sensor measures distances by sending out an infrared light to determine the distance a target location is from it. It does this by also receiving back the same light sent out and measuring the time it takes for the light to travel to the target and back. Once the total time of light travel is measured, the distance of the target is calculated using Equation 2.

$$\text{Distance} = (\text{Speed of light}) * (\text{Time travelled}) / 2$$

Equation 2: Equation used by ToF sensor to calculate distance of target object.

This formula is used for each measurement taken and then communicated back to the microcontroller via I²C as previously mentioned. The distance range of the sensor varies depending on the operation mode the sensor is in. The sensor has a short, medium, and long mode. For the entirety of this project, the sensor was in long mode (capable of measuring 4000mm away) to ensure precision in measurements, as seen in Figure 3. After being set, a function named GetDistance was called to obtain the measurements from the ToF sensor. This function was restated every 5.625 degrees of the stepper motor rotation to continuously obtain measurements, as seen in Figure 4.

```
180 //Long mode is used to ensure precise measurements
181 status = VL53L1X_SetDistanceMode(dev, 2);
```

Figure 3: Setting the ToF sensor to long mode.

```
200 //I2C communication to obtain distance measurement from ToF sensor
201 //LED flash every measurement
202 dataReady = 0;
203 status = VL53L1X_GetDistance(dev, &Distance);
204 FlashLED4(1);
```

Figure 4: Calling ToF sensor to get measurement.

The stepper motor was controlled through KEIL code to step through a full rotation using 64 increments to match 5.625 degrees of rotation. This ratio was determined using Equation 3.

$$1/64 = 5.625/360$$

Equation 3: Ratio used to determine steps.

This means that every step up until 64, one distance measurement can be taken to obtain total data equaling each 5.625th interval of the total 360-degree rotation. The stepper motor was set to rotate in the counterclockwise direction by default for this project. The status LED required to flash on every measurement taken was based on individual student numbers. In the case of this project, D4 (PF0) was the assigned LED for student numbers with the least significant digit of 8. After the entire loop was finished, the motor would have taken 64 measurements and will have rotated a full 360-degree rotation. The motor is set to reset back to the home position after each scan to prevent wire tangling.

For actual data collection in terms of communication and port setup, this project utilized I²C communication protocols between the ToF sensor and the microcontroller, as well as UART communication between the microcontroller and the PC. Once the ToF communicated the measurement values to the microcontroller in millimeters, the microcontroller converts the measurements to meters and serially transmits it to the PC through UART communication COM8. The data is displayed and stored on Python which is then used to display on a 3D visualization.

For the bonus 1 implementation, while the stepper motor is obtaining the 64 measurements, the onboard push button is constantly being checked every iteration (polling method) to see if it has been pressed during data collection. If the button is pressed, the stepper motor is programmed to spin back in the reverse direction the number of times it has currently already measured, as seen in Figure 5. The assigned additional LED according to student number was used to test this implementation. In the case of this project, D2 (PN0) was the assigned additional LED for student numbers with the least significant digit of 8. The LED was flashed each return rotation to ensure the stepper motor reaches the home position. Once back at the home position, the Python code prompts the user to determine further actions to be taken.

```
//---- BONUS IMPLEMENTATION ----
//Check if button is pressed at any moment during measurement calculations (active low config) during measurement collection
if((GPIO_PORTJ_DATA_R40b00000010) == 0) {
    //Assigned troubleshooting LED used
    FlashLED2(1);
    //Spin the reverse direction the amount of times already measured and start over
    for(int j = 0; j < i; j++) {
        spin(1);
    }
    goto Restart;
}
```

Figure 5: Bonus 1 implementation in the Keil program.

Visualization

After obtaining the measurements (max 640 due to 10 scan max), the Python program uses a MATLAB extension to display the data in 3D. The python code reads the distances saved in a text file and using trigonometric functions, converts them to cartesian coordinates to use with the matplotlib library for 3D visualization.

The programming for the visualization was done on a ACER Aspire A515-45 with Windows 11. The PC has 12 GB of ram with a AMD Ryzen 5 5500U processor with Radeon Graphics. It is a 64-bit operating system with a x64-based processor. The python version installed was 3.10.11 and the additional libraries imported were serial, matplotlib, numpy and math.

The 3D visualization begins with a user input of “P” to plot the data. This allows the Python to enter the if statement containing the plotting instructions. The other possible entries a user may input are “R” to erase and restart all current data and “C” to continue scanning without overwriting previous scans.

Once in the plotting phase, the Python reads the text file that contains the measurements communicated from the microcontroller to the PC via UART port COM8 at 115200 bps. This file is read using the readline() function, appending each measurement to a new .txt file called data. At this stage, the program also checks to see how many scans have occurred previously to determine what the x coordinate is assigned to, as the program assumes after each scan that it is moved 10 cm horizontally. The x coordinate is equal to $0.1 \times \text{number of scans}$, as seen in Figure 6.

The y and z coordinates are found using trigonometry. The angle that each value is compared to is multiplied by the sin and cosine of the specific angle, as seen in Figure 7. The 0.098 value comes from 5.625 in radians.

```

#Plot data
if(action == 'P'):
    count2 = 0
    #Assuming 10cm forward measurements and 10 total scans max
    while(count2 != len(plotdata)):
        if(count2 == 64):
            z += 0.1
        elif(count2 == 128):
            z += 0.1
        elif(count2 == 192):
            z += 0.1
        elif(count2 == 256):
            z += 0.1
        elif(count2 == 320):
            z += 0.1
        elif(count2 == 384):
            z += 0.1
        elif(count2 == 448):
            z += 0.1
        elif(count2 == 512):
            z += 0.1
        elif(count2 == 576):
            z += 0.1
        elif(count2 == 640):
            z += 0.1

```

Figure 6: X coordinate set in python code

```

count3 = 0
#While loop used for vertical lines
while(count3 < len(plotdata)-64):
    #Print iteration number to see how many data points are being plotted
    print(count)
    #Check for what scan number was present (10 max), indicating z value
    if(0 <= count3 < 64):
        z = 0
        x_line = [(plotdata[count3])*cos((count3)*0.098), (plotdata[count3+64])*cos(count3*0.098)]
        y_line = [(plotdata[count3])*sin((count3)*0.098), (plotdata[count3+64])*sin(count3*0.098)]
        z_line = [0,0.1]
        #Make a line between point and next point, add to plot
        ax.plot(x_line ,y_line,z_line, color = 'red')
    elif(64 <= count3 < 128):

```

Figure 7: Setting y and z values using trigonometry

Application Example, Instructions and Expected Output

To setup the LIDAR system required for this project, there are specific steps that need to be taken:

1. Download a Python version more recent than version 3.6 from <https://www.python.org/downloads/windows/> to avoid any issues with 3D visualization. This specific project was run using Python version 3.10.11. During the setup, select “Add Python 3.X.XX to PATH” for personalization.
2. Launch the Command Prompt on Windows using the search bar. Install Pyserial by typing in “pip3 install pyserial”. Type “python” to confirm successful installation. To exit python, type “exit()”.
3. Once exited out of python, to ensure the specific libraries are available on the PC at use, type in “python3 -m pip install -U matplotlib”. This library is used for 3D visualization.
4. To setup the UART communication with the microcontroller, connect the microcontroller to the PC USB port. By using the following command in the command prompt: “python3 -m serial.tools.list_ports -v”, note the “COM#” value associated with the UART of the PC. This specific project was connected through user UART COM8, as seen in Figure 8.

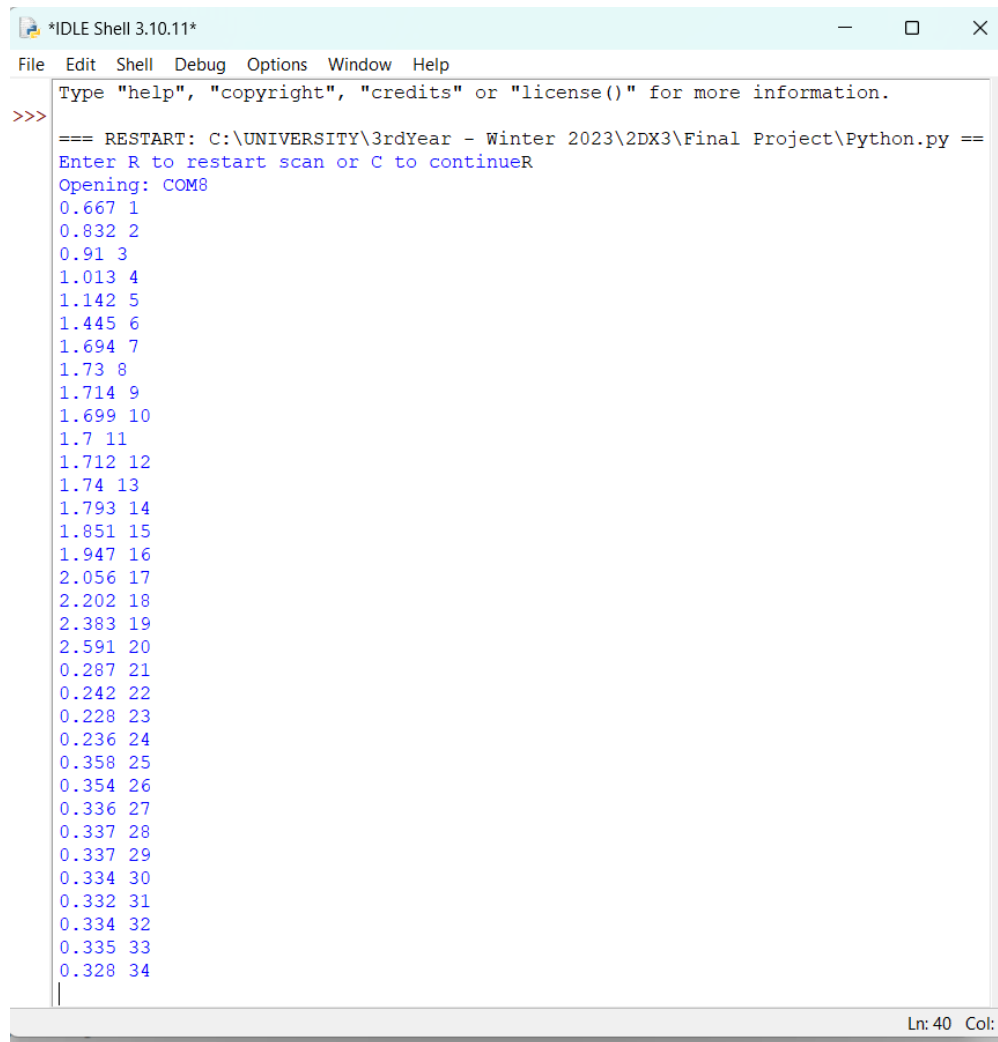
```
C:\Users\areeb>python3 -m serial.tools.list_ports -v
COM8
  desc: XDS110 Class Application/User UART (COM8)
  hwid: USB VID:PID=0451:BEF3 SER=ME401023
COM9
  desc: XDS110 Class Auxiliary Data Port (COM9)
  hwid: USB VID:PID=0451:BEF3 SER=ME401023 LOCATION=1-2:x.3
2 ports found
```

Figure 8: User UART COM#.

5. Open the Keil project, PLL.h file and navigate to line 29. This number must be changed according to the least significant digit of the required student number. The PSYSDIV value can be calculated using Equation 1. This specific project was required to have a bus speed of 50 MHz, therefore equaling a PSYSDIV value of 8.6.
6. While the microcontroller is still connected, translate, build, and download the Keil code to the microcontroller.

The PC now has all the necessary software and libraries to run the Python program, and the next step is to collect, transfer, and plot the data measurements. There are specific steps that need to be taken:

1. While the microcontroller is still connected and flashed with the Keil program, open the Python program in an Integrated Development Environment. This specific project used Python IDLE. Ensure the python code on line 16 has the appropriate UART COM# port written in it.
2. Save the Python program and run it, but do not answer the Reset or Continue question displayed on the screen. The program will wait for an answer before proceeding.
3. Press the reset button on the microcontroller, located near the micro-USB connection and wait for the onboard LEDs to stop blinking to indicate that the boot-up process has finished.
4. Ensure that the ToF sensor is in the preferred location to map, with no points more than 4m away from the sensor. Ensure that the wiring does not obstruct the sensor and that it is perpendicular to the box or mechanism it is attached to.
5. Type "R" and hit enter in the Python program while simultaneously pressing the onboard PJ1 button. The expected output of the program should be distance measurements in meters and the number of completed measurements, as seen in Figure 9.



```
*IDLE Shell 3.10.11*
File Edit Shell Debug Options Window Help
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=== RESTART: C:\UNIVERSITY\3rdYear - Winter 2023\2DX3\Final Project\Python.py ==
Enter R to restart scan or C to continueR
Opening: COM8
0.667 1
0.832 2
0.91 3
1.013 4
1.142 5
1.445 6
1.694 7
1.73 8
1.714 9
1.699 10
1.7 11
1.712 12
1.74 13
1.793 14
1.851 15
1.947 16
2.056 17
2.202 18
2.383 19
2.591 20
0.287 21
0.242 22
0.228 23
0.236 24
0.358 25
0.354 26
0.336 27
0.337 28
0.337 29
0.334 30
0.332 31
0.334 32
0.335 33
0.328 34
Ln: 40 Col: 1
```

Figure 9: Mid-run of the Python code obtaining distance measurements.

6. Repeat step 5, 9 times except with the input “C” to continue to plot more data and to avoid erasing any previous data. Ensure to move the system forward 10 cm each time.
7. After the 10th scan, input “P” into the Python code to obtain the final 3D output of the mapped hallway.

Figure 10 is the assigned hallway that was scanned, and Figure 11 is the expected output of this project.



Figure 10: The assigned hallway to be mapped using the LIDAR system.

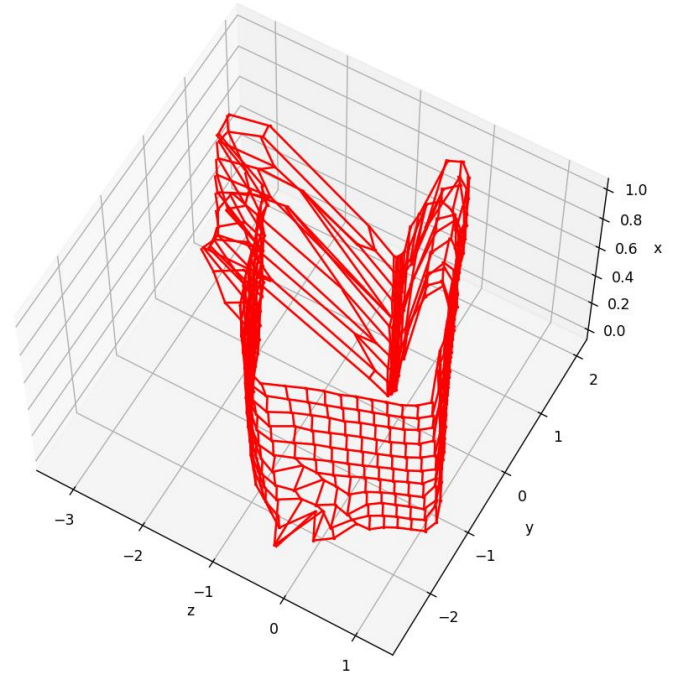


Figure 11: 3D visualization of hallway of 640 using the measurements.

The x-axis corresponds to the 10 cm of forward movements every scan. The y-axis is the height, and the z-axis is the distance away the walls are to the lateral sides of the sensor.

Limitations

This project had some limitations to it as the microcontroller and ToF sensor had unique settings that can not pass specific values.

1. The MSP432E401Y microcontroller's Floating-Point Unit (FPU) supports 32-bit operations such as addition and subtraction, however, the trigonometric used implement double precision floating-point format, which requires 64 bits. As a result, the FPU must be divided by 2, which is a limitation of the microcontroller.
2. The maximum quantization error for each ToF sensor can be determined using Equation 4. Where V_{FS} is the full-scale voltage and N is the number of bits. For the ToF sensor used in this project, the full-scale distance is 4000 mm and 16 is the number of bits. Using Equation 4, we can solve for the maximum quantization error of each ToF module to be 6.10×10^{-2} mm.

$$\text{Change (Max Quantization Error)} = V_{FS} / 2^N$$

Equation 4: Maximum quantization error.

3. The PC can implement a maximum standard serial communication rate of 128000 bps. This value was verified through the device manager on the PC and looking at the XDS110 UART port settings.
4. In terms of the I²C communication between the microcontroller and the ToF sensor, the speed of communication is 100 kbps due to the ports used having a clock speed of 100 kHz.
5. In terms of speed, the ToF had the largest limitation on speed. It has a sampling rate equal to 50 Hz, allowing a maximum of 50 scans per minute. Raising this frequency will alter the accuracy of data collection 4000 mm away and may possibly rise errors. The stepper motor also had a limitation on speed. The maximum speed of the 28BYJ-48 stepper motor is roughly 15 rpm at 5V. This max speed was tested during lab implementation as well as outlined in the manufacturer settings of the device.

Circuit Schematic

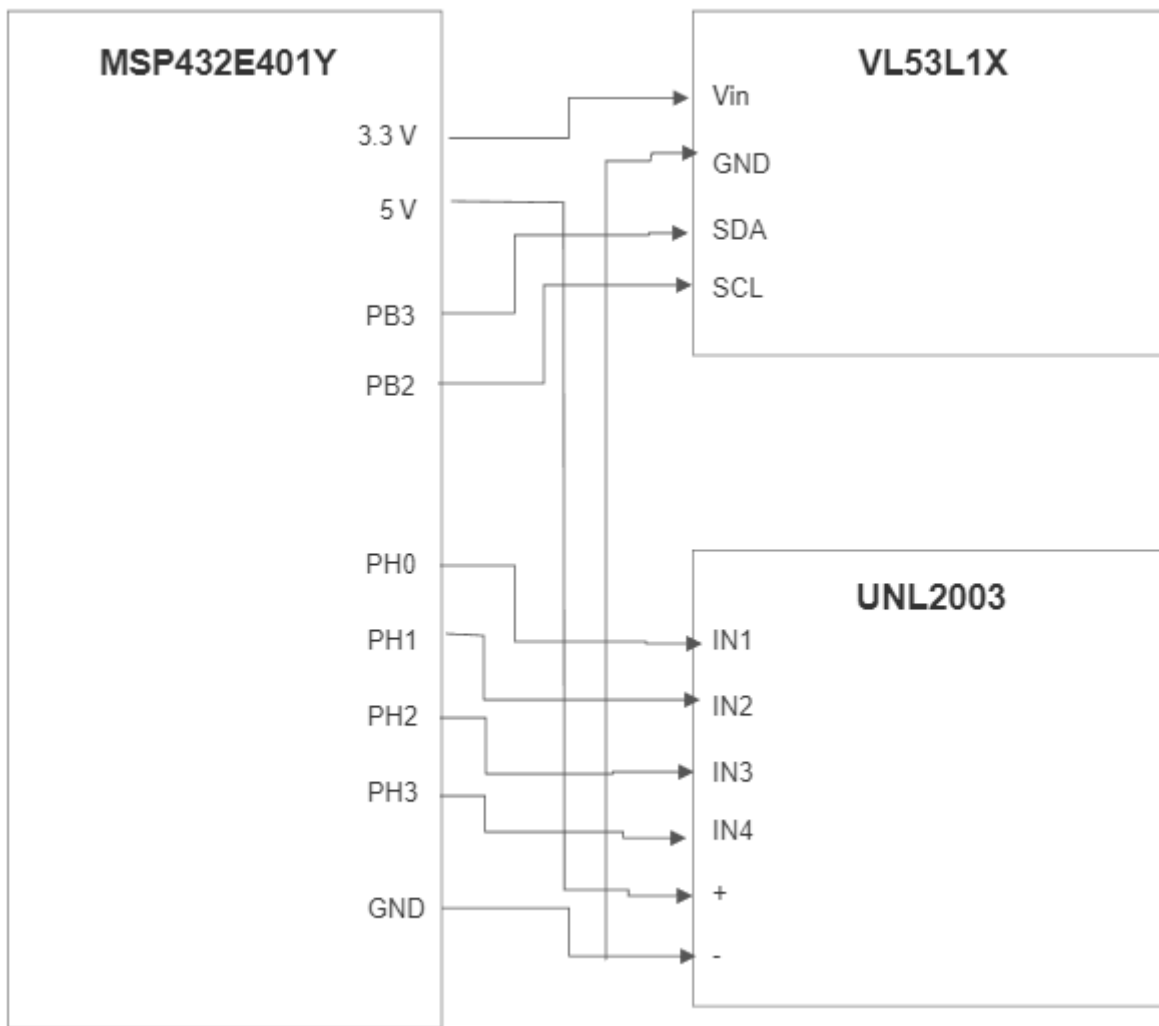


Figure 12: Schematic of the entire system connected to the microcontroller.

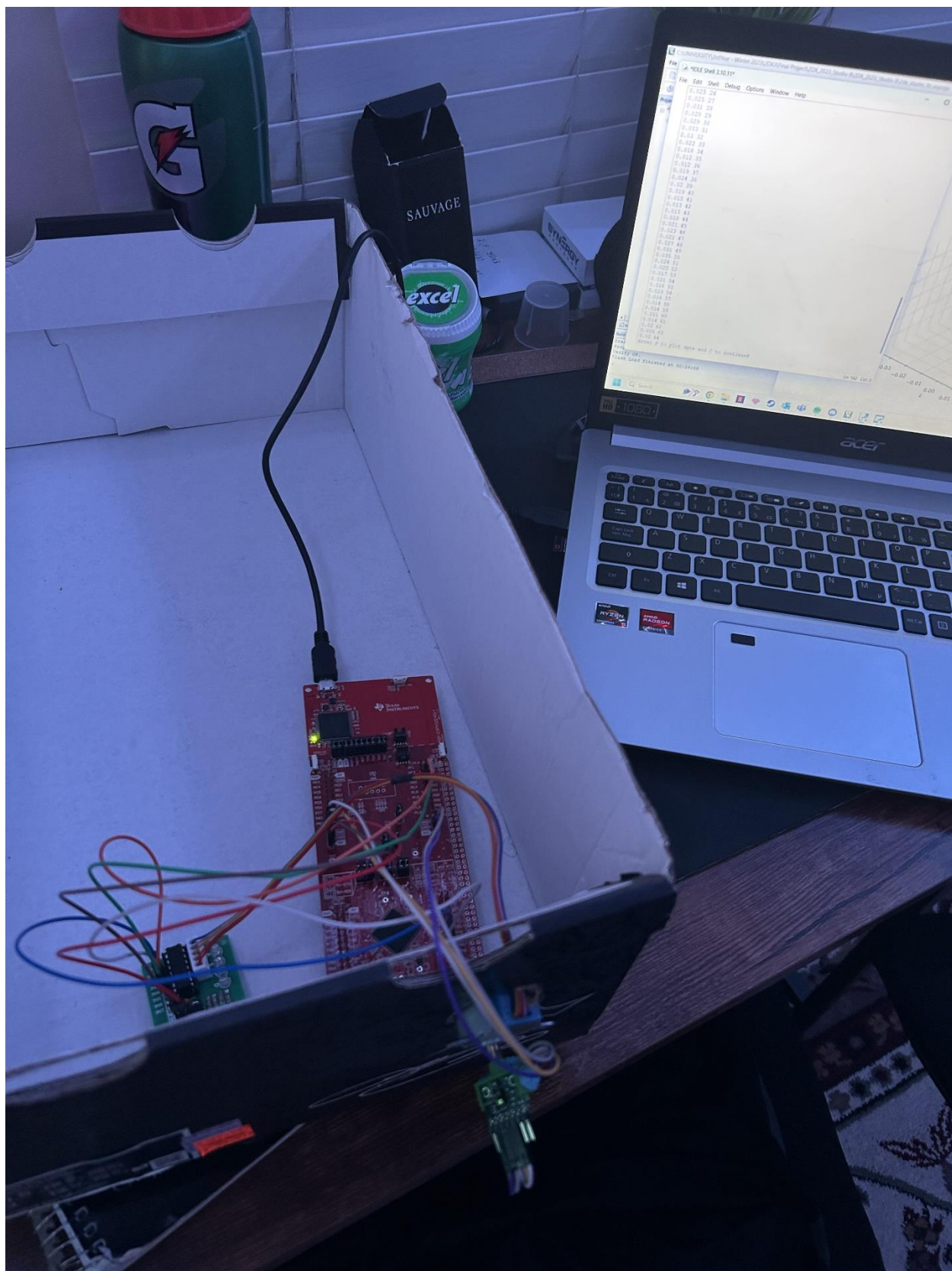


Figure 13: Physical build of final project.

Programming Logic Flowchart

Python Code

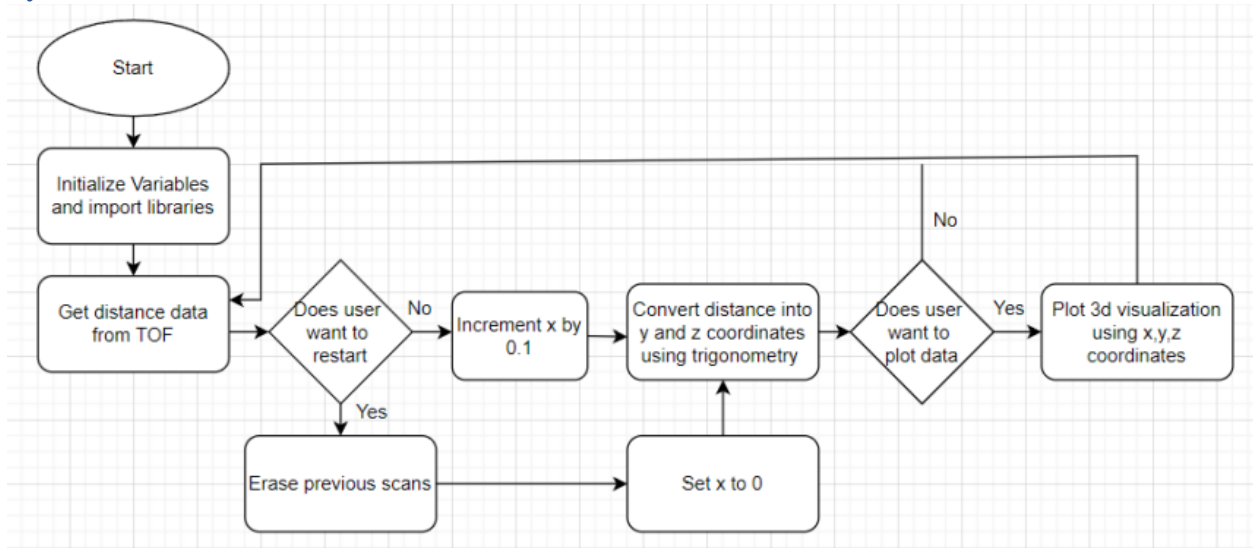


Figure 14: Flowchart of Keil programming logic.

Keil Code

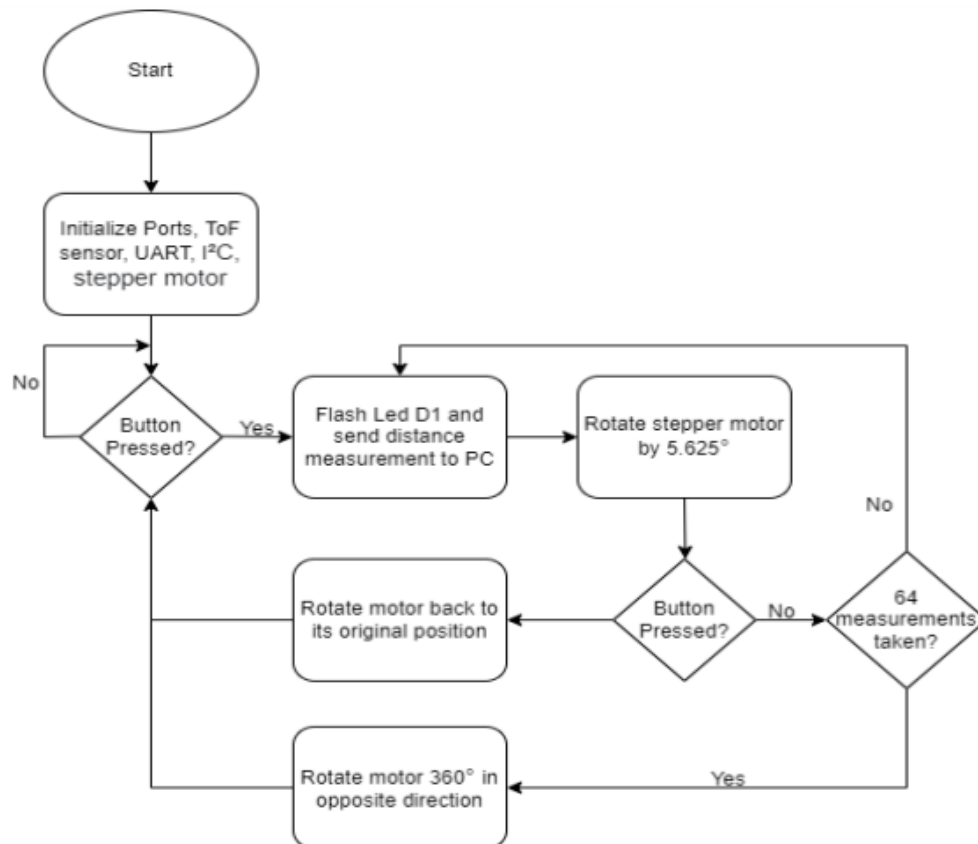


Figure 15: Flowchart of Python programming logic.