2DX4: Microprocessor Systems Processor

Final Project

Instructors: Drs. Doyle, Haddara, and Shirani

Jil Shah – L09 – shahj29 - 400252316

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 **Jil Shah, shahj29, 400252316**

Video Links

Question 1: Question 1 Answer

Question 2: Question 2 Answer

Question 3: Question 3 Answer

Demonstration of the working code: Final Demonstration

(The demonstration has one point which was faulty due to the wire interfering with the sensor, the point has been manually fixed as the error was from an external source.)

Device Overview

a) Features

- All in one LIDAR System which is cost-effective.
    - Indoor and Outdoor purposes
    - 3D visualization of data that is collected in Python from the serial monitor
- TI MSP432E401Y Microcontroller
    - Arm Cortex-M4F Processor Core
    - SRAM size: 256KB
    - Programmed using Keil in C and Assembly Language
    - Clock Speed: 12Mhz
    - Cost: $39.99
- Stepper Motor: 28BYJ-48 and ULN2003 Driver Board
    - LEDs Used for Step Movements
    - Source: 5V-12V
    - 360-degree rotation = 512 steps
    - Cost: approx. $5 /per motor, on average a pack of 5 costs $20.99
- Time of Flight Sensor (VL53LIX)
    - Accurate ranging up to 4m
    - Input Voltage Range: 2.6V to 5.5V
    - Distance measurements can be read from the I2C
    - Nearly 50Hz of ranging frequency
    - Cost: approx. $16.99
- Communication of Data
    - I2C serial communication between the Time-of-Flight Sensor and the microcontroller
    - UART communicates from the microcontroller to the PC
    - The project is currently specific to port 7 Serial Communication
    - Baud Rate: 115.2kbps
- 3D Visualization
    - Packages and Libraries: Open3D, math, NumPy, and serial
    - Python 3 v 3.8.9
    - Updates as new data is being processed

b) General Description

Designed and built an embedded spatial measurement system using a time-of-flight sensor to acquire information of the surrounding environment. This uses a stepper motor to provide a 360-degree measurement of distances within a single geometric plane. The system is moved 20cm every complete rotation. Every 20cm, twenty-four 15 degree turns occur in which data is collected and sent into the microcontroller; a distance

measurement (in mm). The mapped spatial data is sent to the pc via USB (COM 7). This information is converted into x, y, z coordinates and then used to form a 3D visualization using python.

The system uses a stepper motor, microcontroller, time-of-flight sensor. The time-of-Flight sensor is mounted onto the stepper motor shaft using cardboard and twist ties, to provide accurate horizontal data.

The time-of-flight sensor being used simply emits pulses of infrared light and determines the distance by measuring the amount of time of how long the pulse is in motion. This is converted into a distance measurement and then sent to the microcontroller. The data is read in the serial monitor and then converted to 3d coordinates and visualized in open3D.

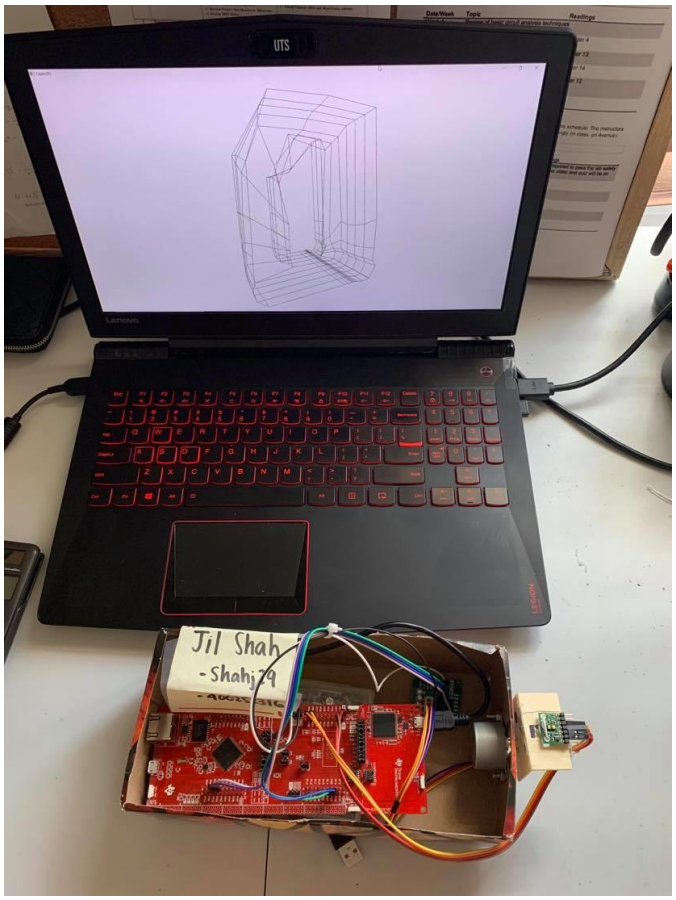

Figure 1. The LIDAR System including the microcontroller, sensor, and the 3D model displayed on the PC.

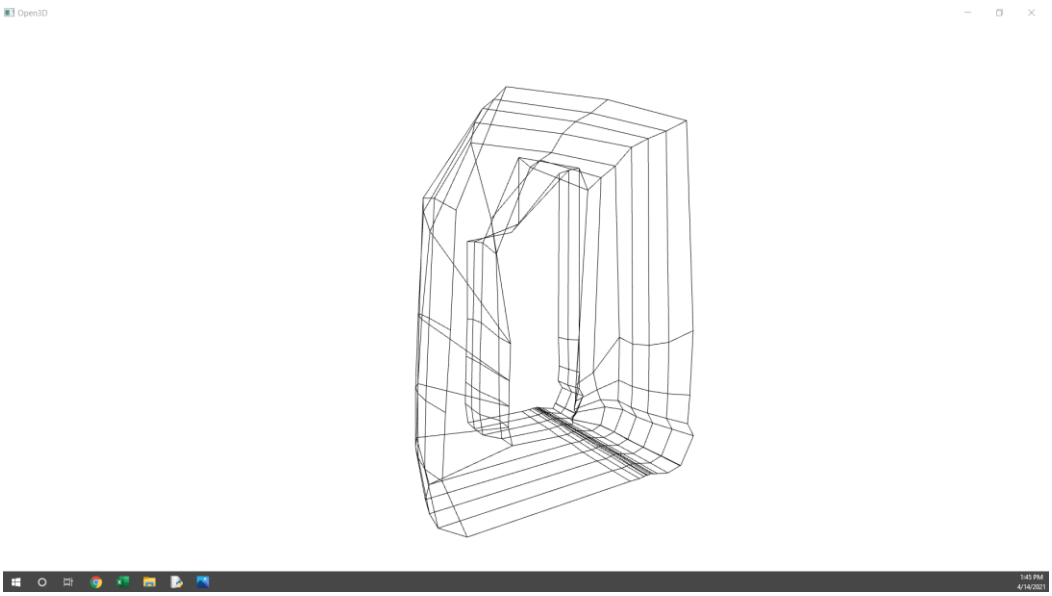

Figure 2. Screenshot of PC with the captured visualization

c) Block Diagram

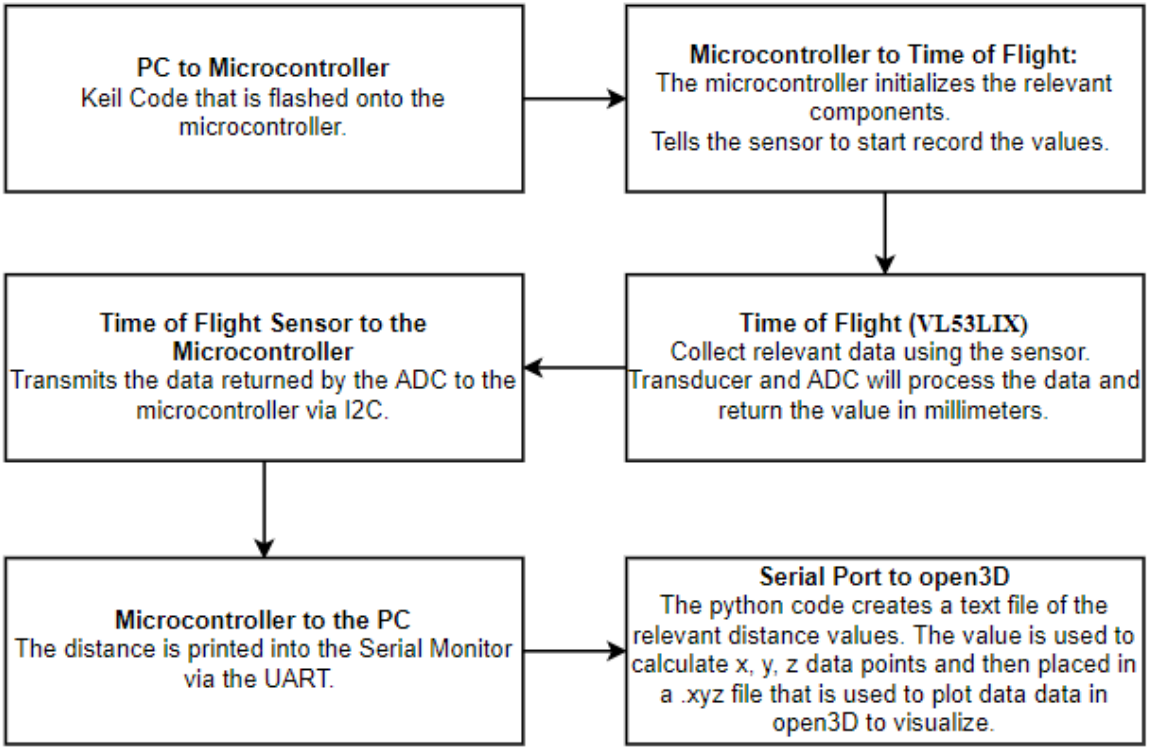

Figure 3. System Block Diagram

Device Characteristics Table

Specification	Details
Serial Communication Port	COM7
Python Version	3.8.9
Bus Speed	12MHz
LED Flash	PN0 = LED D2
Motor ULN2003 Driver Pinout	Device → Microcontroller IN1-4 → PH0-3 (respectively) V++ → 5V GND → GND
ToF Sensor	Device → Microcontroller VIN → 3.3V GND → GND SDA → PB3 SCL → PB2

Table 1. Device Specifications.

Detailed Description

a) Distance Measurement

The VL53L1X time-of-flight sensor is used as the tool to determine the different distances from the center point. The sensor is mounted directly on a stepper motor such that the information that the sensor sends to the microcontroller is of one entire plane.

Specifically, the ToF sensors work to measure the time it takes for a pulse to travel a distance through a medium and then return to the sensor after reflecting off of an object. A general equation to determine the distance from the time is $d = (c * t)/2$, c would be the speed of light, t is the time of flight and the reason we divide by the 2 is due to the time of travel includes the pulse travelling to the object and from the object back to the sensor. Following the flowchart in *Figure 9*, the sensor will instantaneously calculate the distance by converting the time from analog to digital and substituting the relevant value into the equation. The sensor will then use the I2C to send the data to the microcontroller.

For this specific project, the time-of-flight settings are the default configuration settings provided by the professors and the VL43L1X user manual. The distance measurement code for this project includes 24 unique pieces of data sent from the sensor at every 360-degree rotation. The portion of the program specific to this sensor begins with enabling the I2C; which allows the microcontroller to communicate with the sensor.

The sensor is initially booted up, initialized, and starts ranging to acquire data at each interval.

The interrupt system used in this project is called the GPIO Interrupt System. In this case, the interrupt function is enabled, and the GPIO PJ1 button is used as a toggle to start and stop the time-of-flight sensor data transfers. When the button is pressed, the motor will begin to turn and the ToF sensor will begin receiving values. If the GPIO PJ1 button is pressed during a rotation, then the interrupt is activated which means the code will go in standby mode, this basically is pausing the program until the button is pressed once again.

After each data transfer, the motor rotates 15 degrees ($15 * 512/360 = 21.3\ steps$) and flashes the distance LED D2. Once the motor rotates 360 degrees (512 steps) the motor rotates back to the original position by rotating counterclockwise, the individual will then manually move the device 20cm to get the next reading on a different plane. The data is transmitted to the microcontroller using the I2C protocol and then transmitted to the PC using the UART for every measurement calculated. This data is printed into the serial monitor and ends when all the required number of measurements for the current number of planes is printed. Following the flowchart of the python code in Figure *8*, we see that the script receives the data from the serial monitor using the Serial Library and then writes the relevant data values in a text file called 'output.txt'. This text file is then used to place all measurements in an array and then used to calculate the Y, and Z coordinates using trigonometry and the Math Python Library (angle changes by 15 degrees at each measurement and converts the angle to radians). The X coordinate is calculated by adding 20cm every 24 measurements since the plane is shifting.

$$x \mathrel{+}= 20$$
$$y = distance * sin(\Theta)$$
$$z = distance * cos(\Theta)$$

These X, Y and Z coordinates as floats are then written to the .xyz file, this file will contain all points, each in a new line. This file is later used to visualize all the coordinates as a 3D model.

c) Visualization

The computer that this was initially tested on is a Windows 10 OS, Lenovo Legion Y520. The processor is the Intel® i7 CPU @ 2.8GHz, the graphics card is the NVIDIA GeForce GTX 1060 3GB. The python program on the computer is Python 3.8.9 and all the libraries have been updated.

The 3D visualization of the data is performed by using the .xyz data file of all the points. To perform this visualization, it is important to recognize that the relevant libraries need to be included; NumPy and open3D. Using the open3D **read_point_cloud()** function, the python code reads a point cloud from the .xyz file of the points, this is done by decoding the file from the extension name.

Initially, the Open3D visualizer is initialized such that the geometrics can be updated in real-time over time. To visualize these coordinates, it is necessary to create sets of lines that connect all the points. This is done by initializing an empty array, then using one loop to connect all 24 points within each plane and adding an array with two coordinates into the empty array of lines. The second loop will connect each point with the same angle point on the next plane. Next, using the open3D **geometry.LineSet()** function, the **line_set** is created using the points and the lines array. Once the line set is created, each line set geometry is added to the visualizer, removed, and then moved to the next one. The planes are rendered into open3D, the visualization is now available in the window and can be moved around to see all the different angles of the 3D model.

This window will remain open and can only be closed manually.

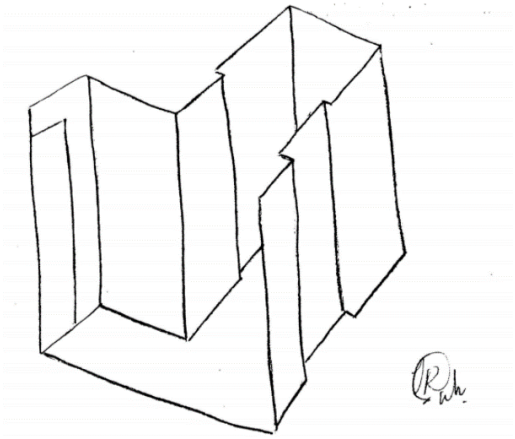

Figure 4. Isometric Drawing of the surrounding space.

Figure 5 Image of the surrounding space.

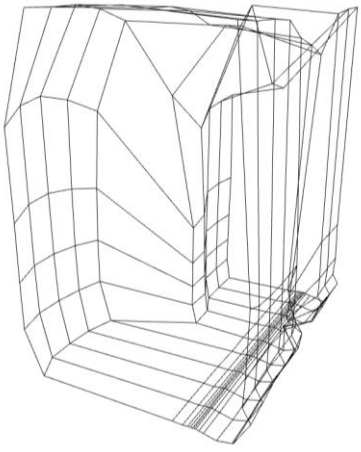

Figure 6. 3D visualization using open3D.

Application Example with Expected Output

To use this system, the steps are listed below, starting with the software, hardware and then the code changes that are required for different computers. The current set-up process is for machines running Windows 10.

1. Download Keil: Microcontroller Development Kit for the Arm Microcontroller (MDK ARM V5.33). Select MSP432E401Y and install the packs.
2. Download Python 3.8.9: we will be using python IDLE (Integrated Development and Learning Environment). Confirm the version number and if the application is currently downloaded
3. Install the different python packages: begin with opening up the Command Line Interface and install PySerial, NumPy, and Open3D. This is done by typing '*pip install pyserial*', '*pip install numpy*' and '*pip install open3d*' individually and sending it into CLI.
4. Hardware Setup: Build the circuit, follow the circuit diagram in Figure *7*.

After the setup, the user's PC will have all the required software and packages to successfully run the different programs.

1. Connect the microcontroller to your PC using the micro-USB port, this will cause the microcontroller to turn on and your PC to act as the power supply.
2. Open the python file called, 'LIDAR-Python Program', this is the python code that will complete most of the tasks that are required for the project to actually visualize the values in open3D. However, there are changes required in the code:
    a. From the CLI, determine the serial port that is connected to the microcontroller by using the command '*command -m serial.tools.list_ports -v'* make note of the port being used for your communication. One of the two will work, confirmation of the port can be done by using RealTerm or by just testing both using the baud rate of 152000. Change the 'serial_port' variable on **line 24** to be equal to the COM port for your specific computer.
    b. Change **lines 31 and 48** and change the file path names to be specific to anything you prefer to call the text file names, remember both names must be the same.
    c. **Lines 27-29** must be changed according to the Main.C file in the Keil Folder; **line 27** is $360/15 = 24$, **line 28** is the total number of planes for initial testing purposes it is important to lower the number, and **line 29** is the partial angle shifts. Lines 29 and 27 should remain the same, such that the number of points and lines do not change.
3. Open up the Keil project that is submitted with this report by downloading and extracting the compressed ZIP folder. Once this is opened, you can compile and

flash the code onto the microcontroller. Lines **244 and 245** can be changed to a different angle and plane but these changes should be reflected on the Python code.

4. The GPIO PJ1 button press will allow the code to begin running in which the sensor will boot up and let the user know in the serial communication that the time-of-flight sensor is ready to take measurements. The motor will begin to turn and the user can now hold the system in one plane and shift the system 20cm in a different direction while the motor rotates counterclockwise to move back to the original position. The y and z are calculated using trigonometry and the measurement calculated by the sensor.
5. Once all these values are measured, the code will automatically display the visualization of the measurements on Open3D.

Limitations

1. The microcontroller, MSP432E401Y has a FPU (floating point unit), this means it can only support single precision operations (32 bits only). Since, the FPU has can only support 32 bits wide, and 64 bit double precision operations which will take more than one instruction as the FPU may need to split the data into 32 bit words. The FPU is also capable of performing conversions between the fixed point data formats and floating point constant instructions. The trigonometric functions are used by accessing the math.h library and applying the functions in the library to variables that are of type float.
2.
    - The maximum quantization error for the time-of-flight module is $6.1 * 10^{-2} mm$ $(4000mm/2^{16})$.
    - Maximum quantization error for IMU is $4.79 * 10^{-3} m/s^2$ $(32 * 9.81/2^{16})$
3. The maximum standard serial communication rate is 128000 bits/second that can be implemented with the PC. This is verified checking the port settings for UART Port in the device manager of the PC.
4. The I2C protocol has a clock speed that is configured to 100KHz and the transfer rate is at 100kbps.
5. The primary limitation on the speed of the system is the ranging of time-of-flight sensor. This is because the sensor we are using has a budget in time that can be set to any value between 20 to 100ms. The minimum distance that the sensor allows is 4m in the longest distance mode. We can look at the C code in Keil to test and determine why the ToF sensor is the slowest aspect of the system. After running the code, we can see that each individual measurement that is acquired every 15-degree motor rotation, there is a delay. This delay is approximately 300ms which in the end will accumulate to nearly 72000ms $(300ms * 240\ measurements)$.

However, when we try decreasing this delay, this affects the time-of flight sensor as there will be no data acquisition and the system will stop working. The system will continuously increase in time as we increase the accuracy by increasing the number of measurements per rotation.

Circuit Schematic

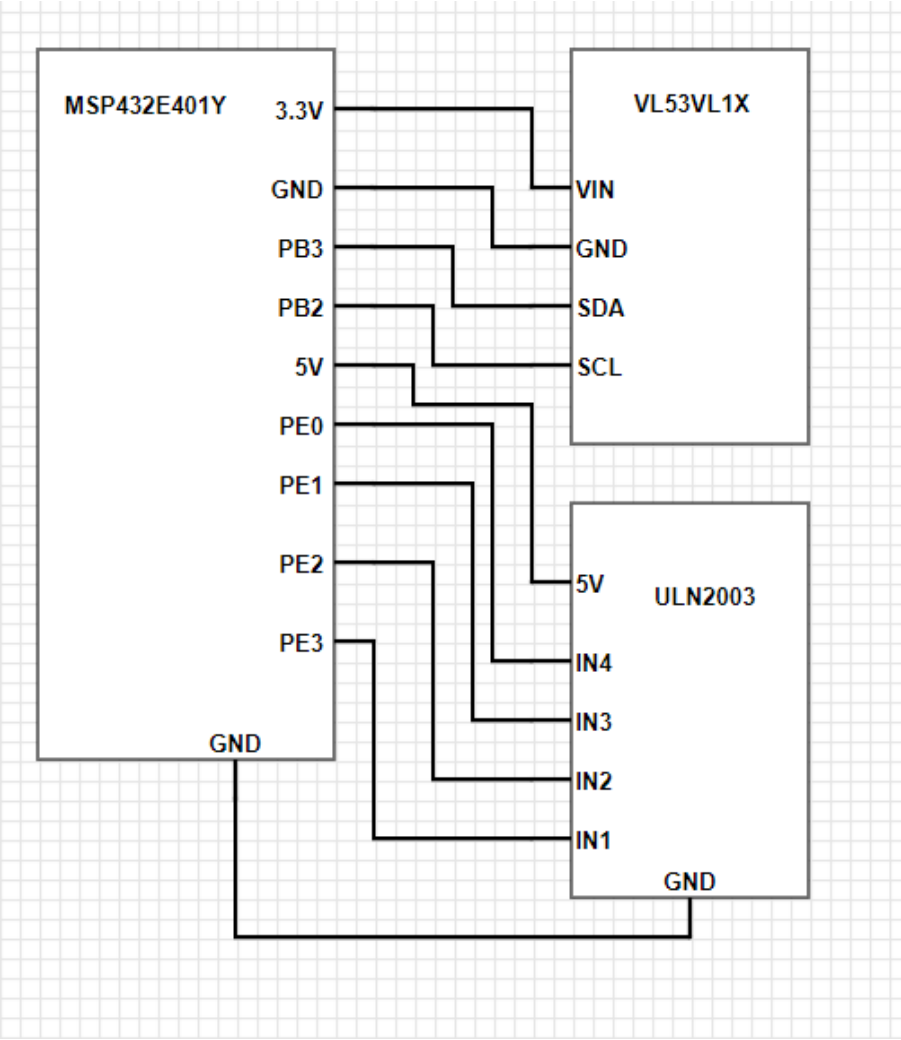

Figure 7. Circuit Schematic of the System

Programming Logic Flowcharts

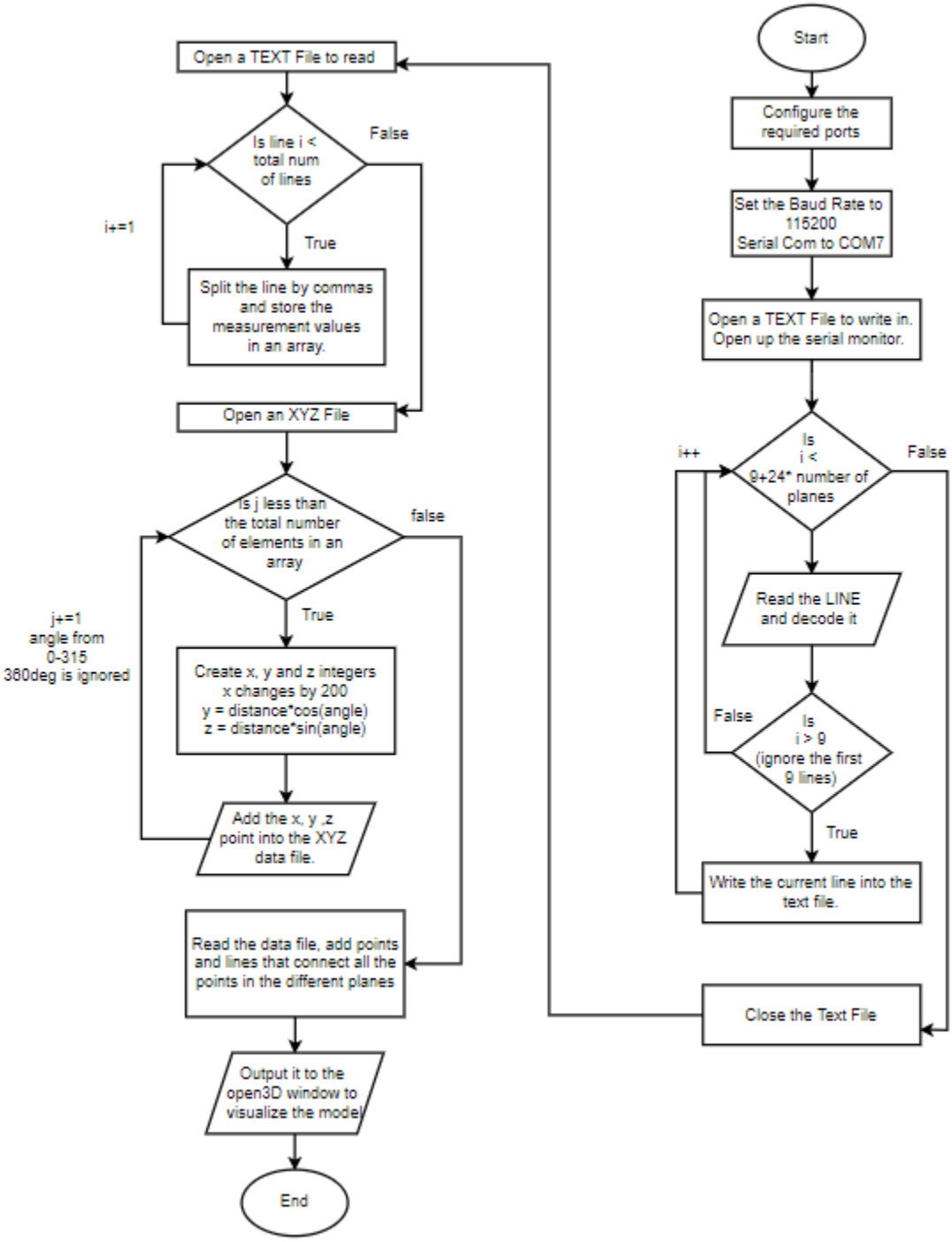

Figure 8. Flowchart of Python Code.

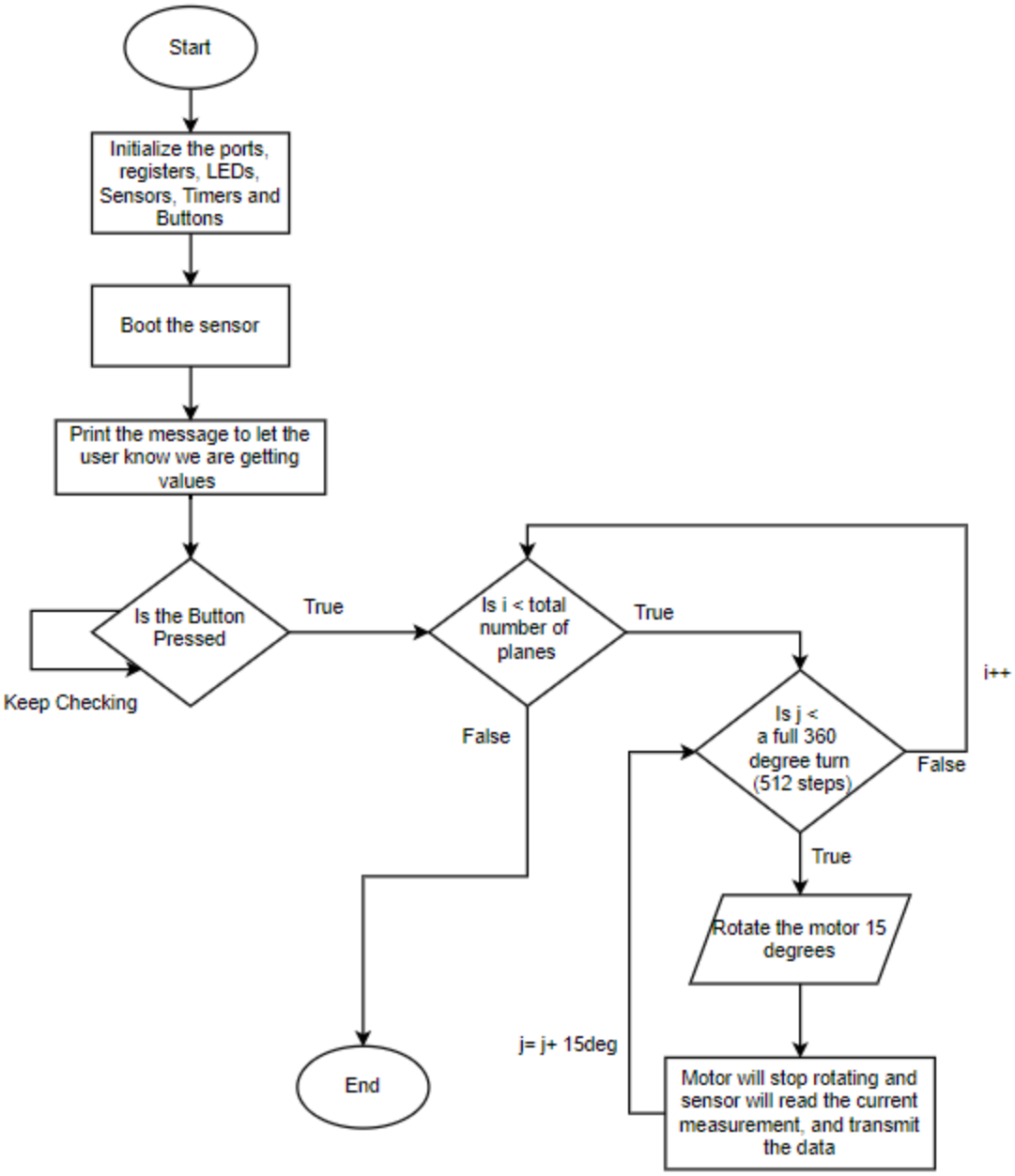

Figure 9. Flowchart of Main.C code in Keil

Works Cited

[1] D. D. D. H. Dr. Shirani, "Avenue To Learn: COMPENG 2DX4: 2021 Microprocessor Systems Project," [Online]. Available: https://avenue.cllmcmaster.ca/. [Accessed 12 April 2021].