2DX4: Microprocessor Systems Project
Final Project

Instructors: Dr. Bruce, Dr. Haddara, Dr. Hranilovic, Dr. Shirani

Abd Elelah Arafah – L01 – arafaha - 400197623

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 **[Abd Elelah Arafah, arafaha, 400197623]**

0. The Video Links for the Final interview

First question Google drive link: https://drive.google.com/open?id=1uc0y5kyF5qLX6ARK-BALCO4Dqgxa6uup

Second question Google drive link: https://drive.google.com/open?id=1smHmdFnslKgzbAd_J2oY8bBWPHmSg-jX

Third question Google drive link: https://drive.google.com/open?id=1J_Pinr7bGdY6HbX-9tPIVjmOx94YhB4k

(The links can only be accessed by McMaster emails to ensure privacy)

1. Device Overview

a) Features:

The device acts as a lidar in which it employs multiple devices and electronics to allow it to produce a computer 3D image of a room. This device is needed as it is a more costly efficient device than a specialized lidar machine. The device involves specifically three devices which are listed with their relevant features below:

1- SimpleLink™ Ethernet MSP432E401Y Microcontroller

- 120 MHZ Arm Cotex-M4F processor
- Bus speed is set as 120 MHZ by default (is set 48 MHZ for this system)
- Operating voltage is around 5 Vdc
- Analog to Digital converter
  - Two 12-bit Sar-based ADC modules
  - Each with up to two million samples per second
- Memory
  - 256 KB of SRAM (Bandwidth at 120 MHZ)
  - 1024 KB of Flash memory
  - 6 KB EEPROM
  - Internal ROM
- Language
  - Assembly and mainly C programming languages
- Costs around $39.99 (USD)
- Serial Communication
  - Baud rate of 115200 Signal per second

2- 28BYJ-48 stepper motor and ULN2003A driver

- **Operating voltage of 5 Vdc**
- **Costs around $5.50 (USD) for the motor and $3.99 (USD) for the board**
- **Moves by sending voltage to specific coils**

3- Time-of-Flight Sensor VL53L1X

- **I^2C bus speed of 400 KBITS/second**
- **Operating voltage of around 3.3 V**
- **Non-volatile memory**
- **Cost is around $21.95**
- **Language used is C programming language**

b) General Description:

The device is an accurately functional lidar system as it employs some of the best electronics in terms of performance and cost efficiency.

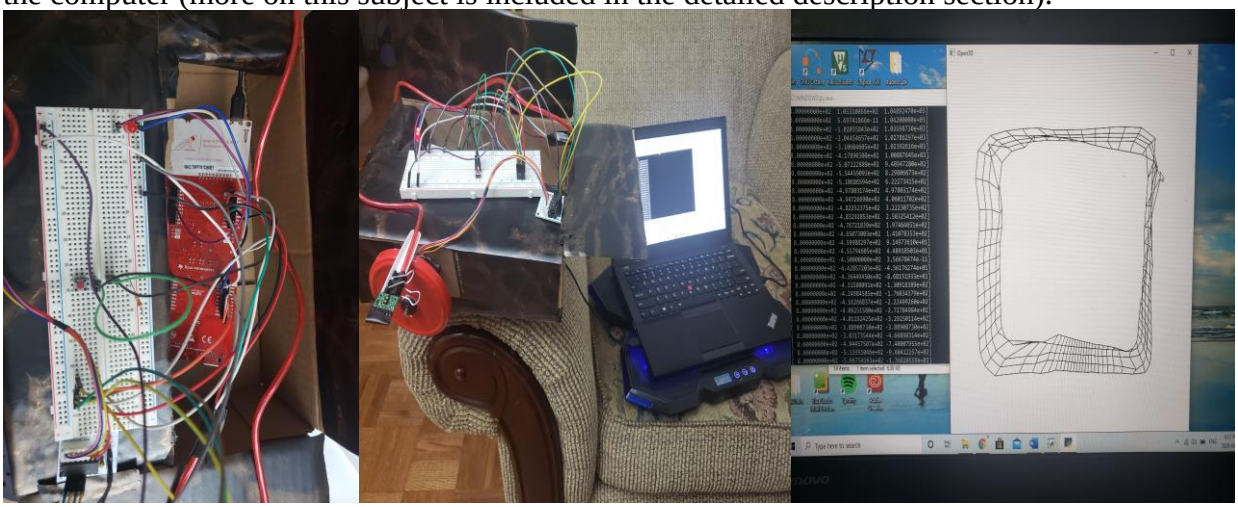
The microcontroller used in this device provides top performance and advanced integration of a great range of rich communication features. With that being said it is clear that the microcontroller is the main component of the system as it technically acts as a bridge to allow data to get transferred from the ToF VL531X sensor to the computer that the system is connected to. As well as control the motor's movement.

The motor acts as a very important component of the system as well as it moves the sensor (that is mounted on the motor) with accurate angles to allow for accurate data calculations.

The ToF sensor is the component in the system in which is responsible for acquiring data. The sensor uses laser ranging in order to achieve its function (more on its function is included in the detailed description section).

The entirety of the system is shown in Figure 1. And a block diagram of the main stages of functionality is shown in Figure 2.

In figure 2 it is shown how the device is put together as well as how it uses the python code so serially communicate with the computer to transfer data and eventually create an image from the computer (more on this subject is included in the detailed description section).



a) b) c)

Figure 1. a) Showing the top view of the system. b) Showing the System conducting data while connected to a PC c) The results of the visualization on the PC

c) Block Diagram

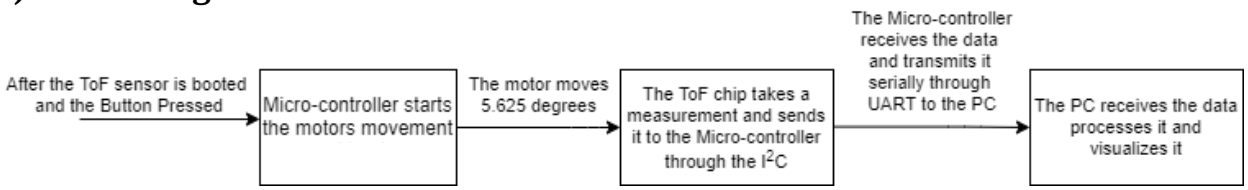


Figure 2. The block diagram showing the main stages of the functionality of the system

2. Device characteristic Table:

Feature	Description
Micro-controller pins used	
Pin F0	Connected to the on-board led
Pin L4	Connected to the external led
Pin N2 – N5	Connected to the motor
Pin B2 & B3 (I^2C)	Connected to the ToF sensor
PM0 & PE0	Connected to the button
Bus Speed	48 MHz
Serial Port	Port 5
Serial Communication Speed	115.2 kbps 8N1

3. Detailed Description:

In order to start conducting data and use the system. Firstly, the keil code needs to be translated, built and loaded. After loading the code, the external button must be pressed for the device to start functioning. In this device the sensor is mounted on the stepper motor on the outer side of the jar's cap. Meaning that every rotational movement by the motor is also applied as well to the ToF sensor.

a) Distance Measurement:

Before moving forward about explaining how measurements are acquired it is important to explain in more detail how the ToF sensor operates.

The ToF sensor is a sensor that employs laser-ranging to acquire measurements. The ToF integrates the use of a receiving array of SPAD, optics and infrared filters, as well as a 940 nm laser emitter to function properly, efficiently and accurately. Finally, the ToF is the component of the system in which does the transduction, preparation of signal and ADC.

Now that the working of the sensor is established a more detailed description is introduced. After loading the keil code onto the microcontroller and refreshing the microcontroller. The code (briefly represented in figure 6) starts off by printing some welcoming and starting statements to the user of the device by sending to the UART channel by the microcontroller the strings written in the code in the form of print statements (i.e. the code printing "Program starts"). Following that, the code goes on to starts calling basic I^2C (serial communication with the microcontroller serially bit by bit. The microcontroller being the master and the ToF sensor being the slave) read functions to check the I^2C functions of the sensor. It does that by calling different Api functions such as VL53L1_RdByte as well as VL53L1_GetSensorId and passing the default address of the sensor (seen by the microcontroller) of 0x52 as a parameter. Which then sends the model id and module id of the sensor to the UART. Now that after the sensor is properly read by the I^2C. The code checks that the sensor state is achieved and then calls the Api function VL53L1_BootState to boot up the sensor also passing the address as a parameter. Following that another Api function (VL53L1X_ClearInterrupt) is called in order to clear the interrupt to enable the next interrupt to occur. The last Api function called before ranging is VL53L1X_SensorInit in which sets up the sensor with default settings. Now, to start ranging the Api function VL53L1X_StartRanging is called. Now that everything is set up. To allow for any ToF measurements the button must be pressed (polling procedure).

The press of the button allows for a 360 degrees rotation of the stepper motor as well as the sensor because it is mounted on the motor as mentioned previously.

To use this device correctly. The sensor must have a starting position of the sensor pointing to the ceiling of the room at 0 degrees angle with the vertical axis. This is important as the data gets calculated based on this assumption (Explained later in the python code explanation). Following the press of the button with the sensor at the correct position. The stepper motor moves for 5.625 degrees and takes 1 measurement. Meaning that every 360 degrees rotation of the motor the sensor takes 64 measurements of distance from the device to the walls of the room. This is implemented in the keil code in a for loop that increments for 512 steps and it checks when only 8 steps are reached by the motor. Once that condition is met the onboard F0 LED flashes and the sensor takes a measurement to the wall. By calling functions like VL53L1X_CheckForDataReady (checks when data is ready for the sensor to measure) as well as it calls VL53L1X_GetDistance which is the most important command that actually tells the sensor to take a distance measurement to the wall. After this command is called and the measurement is taken. This measurement gets transferred from the slave (sensor) to the master (microcontroller) through the I^2C serial protocol bit by bit.

Now that the data is being acquired, it must be processed and manipulate it in order to transfer this data into useful information that is later visualized. Meaning that the micro controller now takes this data it received through the I^2C and transfers it to the computer it is connected to through a serial port through the UART communication channel. For this communication to be useful a python code is needed to extract only the raw measured distance data.

In the python code it is firstly important to import all the necessary libraries that are needed for this device to operate properly. The libraries used in the python code are serial, array, string, NumPy, open3d and math. When the button is pressed, and the python code is launched. The microcontroller communicates with the computer the data serially bit by bit by initiating the serial connection with the correct port 5 for the used PC and the baud rate of 115200 signal/second (First line of code after import of libraries). After the serial connection has been established the code then uses different python functions that read the data coming in from the microcontroller and decoding it. The code uses manipulation of different variables as well as typecasting to create measurements that are then stored in an array element. An array is used to store the data in order from the starting position and ending at the starting position after rotating 360 degrees. Meaning that after a successful 360 degrees spin by the stepper motor and the sensor, the array would have 64 elements each being a distance measurement. It is also important to mention that the data is being set from the microcontroller with a measurement followed by a comma which is read and decoded by the python code to realize that this measurement is done which means it can be added to the array and it would move on to receive the next measurement to store it in the next index from the variable.

Now onto the main parts to this procedure which is definition of axis and coordinates calculation. The axis for this system is defined as seen in figure 3 below.

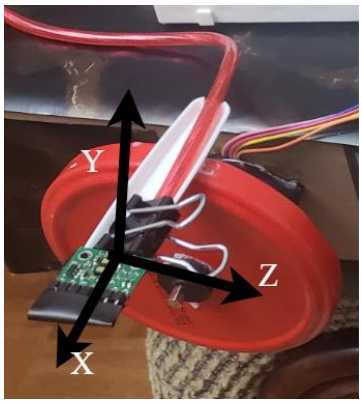


Figure 3. Axis definition of the system

This means that in order to use the device properly one must press the button and obtain measurements then procced 20cm in the +X-direction (i.e. Move the device forward) then press the button again until five 360 degree rotations Is achieved that's when we have 320 points (64*5) and the code is exited. As for the coordinates calculation the code takes each measurement in its specific order matches it with the angle the motor is at and it generates a y and z component of the point by using simple trigonometry. Meaning as seen in figure 3, the equation for the coordinates is: y = Distance*cos(angle of motor) and z = Distance*sin(angle of motor). And the x displacement of the device is done manually by the user. As an example, let's say the motor turns to get its first measurement at 5.625 degrees. The sensor creates a 5.625 degree acute angle with the vertical y axis meaning that the y coordinate for that point would

equal Distance*cos(5.625) and the z coordinate for that point would be Distance*sin(5.625). Now it is also important to mention that the sensor doesn't spin exactly in it's place meaning that mounting the sensor on the outside of the cap would produce a very slight error within the data as the origin is slightly changing (discussed more at the end of section 4).

b) Displacement measurement

In order to make this system more accurate and self dependent in terms of functionality a displacement variable could have been introduced using the MSP-9250 IMU sensor as it is a great sensor that could measure its own displacement. Firstly, to properly use the IMU it must be also mounted on the motor just near the ToF sensor to measure any change in its position or its angle (moved up to down at a angle). Meaning that instead of only sending only the Distance through the UART to the computer it must also now send another value of that being the displacement. In which then can be programmed in the python code to compute the x-axis coordinates associated with the y and z coordinates provided by the ToF sensor. Also since the SPI has been blocked by the manufacturer, the chip must also utilize the I^2C for it's serial communication with the microcontroller.

c) Visualization

The computer device that was used to test/use the system was a Lenovo x240 Notebook. Which is a machine running the latest version of Windows 10 Pro. The computer contains an intel I7-4600U processor clocked @ 2.10 GHZ and has 2 cores. The machine also has a total of 8 GB of memory and has an integrated graphics card. The system is also a x64 based PC. As for the program used for visualization is Python specifically version 3.6.8 (64-bit).

At the beginning of the python code, the program imports two libraries: NumPy and open3d. Which are both responsible for the visualization later on at the end of the code.

To be able to use this device one must have a compatible computer in which is able to download the open3d library without any issues. Meaning that any computer that is used to be connected to the system must be x64 based as any device that is x32 based can not download the open3d library successfully.

As for the implementation of the visuals (As seen in figure 8 in the form of a flowchart). The python code starts off by importing all the libraries needed including open3d as mentioned earlier. After that the code goes on to open/create a new file called "tof_lidar_visual.xyz" as a write-to command. Then later on in the code a f.write command is called to format the coordinates calculated earlier from the distance coding implementation. Specifically, the write function prints a x coordinate followed by a space and the y coordinate followed by a space and the z coordinate and then moves on to the next line to add the other points. What this is doing is that it's formatting the file opened earlier to the correct .xyz format so that the open3d library successfully reads the points. This process is then repeated for the next four 360 degree measurements (from 20cm on the positive x-axis to 80cm). After all, 320 points are written successfully onto the file the file is closed. Which then the program calls for the read_Point_cloud command in which opens the file and confirms the points that are registered and formatted correctly. Following that these points are stored as a NumPy "np" array for now. Then a pointer definition of 64 points is done (symbolic of 64 points each 'run' of the device) which these pointers are then used in a for loop to create the 5 planes/slices of data produced from the data (Lines.append function used). The pointers are then reset to their original values and then are used in a for loop again to connect these 5 planes together. This line data set is then converted to 3D vectors and is finally visualized by calling the o3d.visualization.draw_geometries function.

4. Application Example and expected output

To properly use this device please refer to the step by step guide below:
1)Mount the sensor onto the stepper motor in which its starting position is the ToF sensor pointing to the ceiling and the sensor turns around the same vertical axis of the rotor in a clockwise motion and it ends the rotation by also pointing at the ceiling. This is the case because as seen in figure 3. The axis is defined were the ToF is conducting points on the y-z plane and the x-axis is being handled manually by the user. Since the y axis is define as the vertical axis of the system the equation to calculate the y coordinates is Distance*cos(angle) While it is Distance*sin(angle) for the z coordinates.
2) Download the Keil, Python 3.6.8, and the open3d library
3) Extract the zip file containing the Keil code and open the project
4)Translate, build then load your microcontroller with the code after it is connected to the computer
5) For a guide on how to wire up the system please follow the schematic in figure 6.
6) Extract the zip file containing the python code.
7) Right click on the icon of the file and navigate to Edit with IDLE > Python 3.6.8
8) Now it is needed to identify the port in which the computer is connected to the microcontroller with. Which can be done by typing the command “python -m serial.tools.list_ports -v” in cmd.
9) Looking at the results of the command the port that says UART is the port needed for the serial connection with the microcontroller
10) Navigate back to the python window and modify the line of code “s = serial.Serial("COM5", 115200)” specifically to the port that says UART in the command window. To expand on this line of code what this code does is that it is the line of code that establishes this communication channel (UART) between the microcontroller and the computer. The modification of the COM# part of the code is very important as the port defers from a computer to another. The number in that same line of code sets the baud rate (in signal per second) that the serial connection is based on.
11) Now run the python code and refresh the microcontroller by pressing the onboard refresh button.
12) Kindly wait until the external LED is lit up (This takes about 30 seconds as the ToF chip needs its time to bootup correctly) and then press the button
13) Following the press of the button observe the device from a far enough distance so that the data doesn’t mistake the user as a wall in the room and mess with the visualization.
14) once the motor stops turning you have completed a full 360 degree slice visualization of the room
15) Repeat steps 12 to 14 four more times to obtain 5 slices of the room. However, before starting the system again you must move the sensor part roughly around 20 cm in the positive X-axis direction each time before pressing the button (as defined in figure 3).
16) After 5 successful runs a couple of messages appear on the computer screen showing that the points have been read and detected by the code successfully (Also prints the points and saves them in the file). As well as a open3d window opens up to show the visualization of the data acquired by the 5 runs of this system. The expected output is shown in figure 4 below.

a)

b)

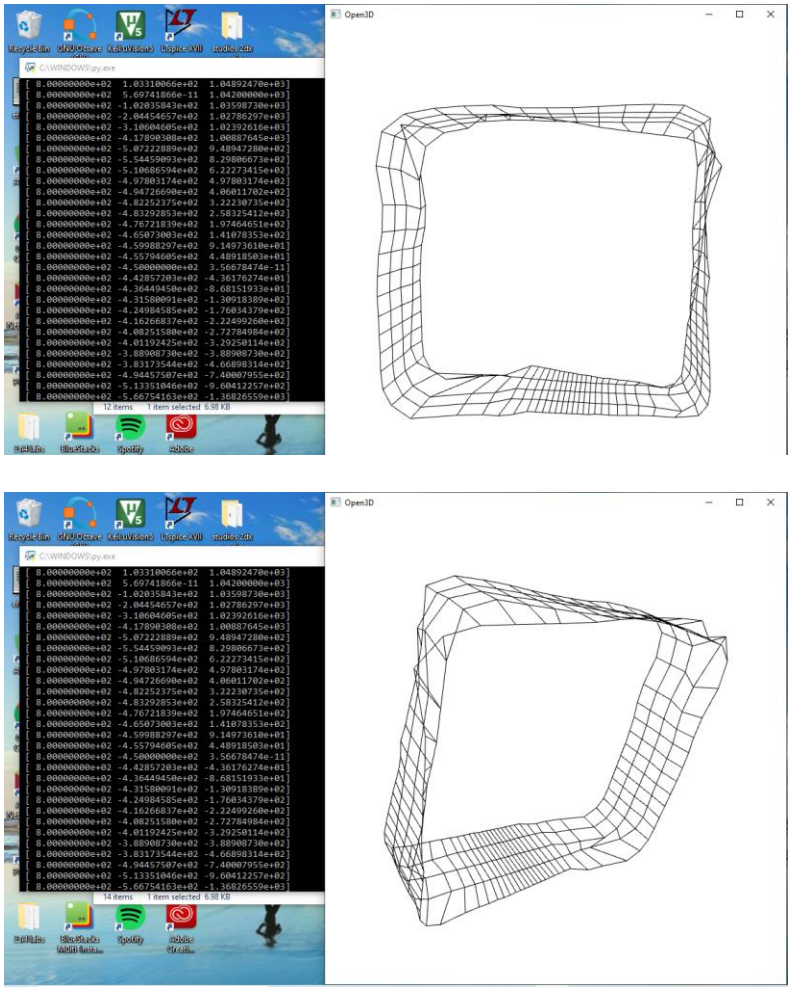


Figure 4. An expected output from 5 runs of the system

As seen in figure 4, the observed visualization produced is a fairly accurate representation of the room's shape and dimensions however, as seen in the figure the visualization has some distorted and inaccurate points.

This could have occurred because of how to sensor is mounted onto the motor. We can see that the sensor rotates around the jar's cap (40mm radius) meaning that it isn't rotating on the x-axis directly as defined in figure 3. Which could cause these issues with the results.

Meaning if a more reasonable mounting of the ToF would visualize more accurately than the visualization shown in Figure 4.

5. Limitations:

1) Given that all the trigonometric calculations (with floats) are done on the computer the system has no limits on that. However, even if these were to be implemented in the microcontroller before it is sent to the computer then the floating point unit that the microcontroller has would be able to deal with that as well. However, the FPU only deals with 32-bit and not 64-bit.
2) The maximum quantization error is 12 V for the microcontroller, 16V for the IMU and 8V for the ToF (This is easily calculated using, Max Quantization Error = resolution/2, where resolution is the Vmax-Vmin/number of intervals).
3) The maximum standard serial communication with the PC is a baud rate of 115200 (signal per second) this is verified by running the code while having the realterm program open (instead of python) and setting the baud rate to 230400 signal per second. As seen in figure 5, for most the run the data was distorted and it was unclear what is being communicated to the PC from the microcontroller, while at the end the data starts to get recognized when the baud rate is set back to 115200 (signal per second).

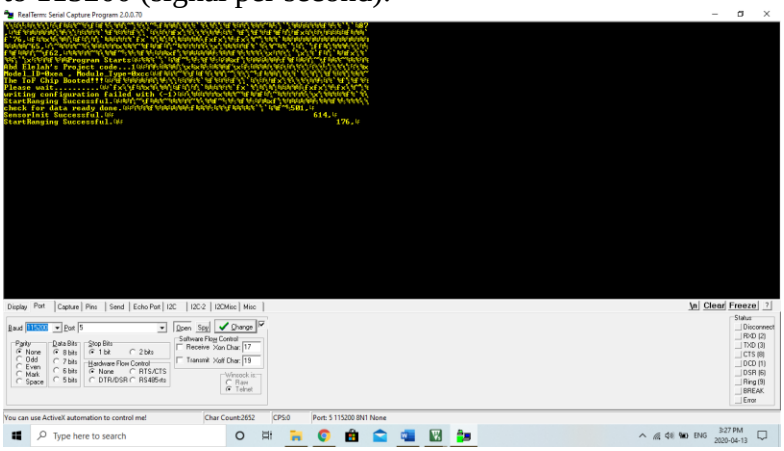


Figure 5, showing the communication between both devices with 230400 vs 115200 signal per second (in order of usage)

4) The communication method used by the microcontroller and the ToF sensor is a Master-Slave (respectively) I^2C serial bit by bit communication. The communication between the two devices is up to 400 KHz, as this is the maximum speed for the I^2C for the ToF sensor.
5) Reviewing the whole system, the element that effects the speed the most is the bus speed of the microcontroller. This can be tested by simply setting the bus speed at a lower value than what it currently is. From running the same code, it is evident that the speed of the bus has the greatest effect on the speed of the entire system.
6)The IMU has a typical sampling rate of around 32 KHz as max, the Nyquist rate in this case then is 64 KHz (2 times the sampling rate). When this frequency is exceeded antialiasing occurs and data signals get distorted. While the sampling rate could also be at 8KHz and can not exceed 16 KHz frequency due to antialiasing occurring.

6. Circuit Schematic:

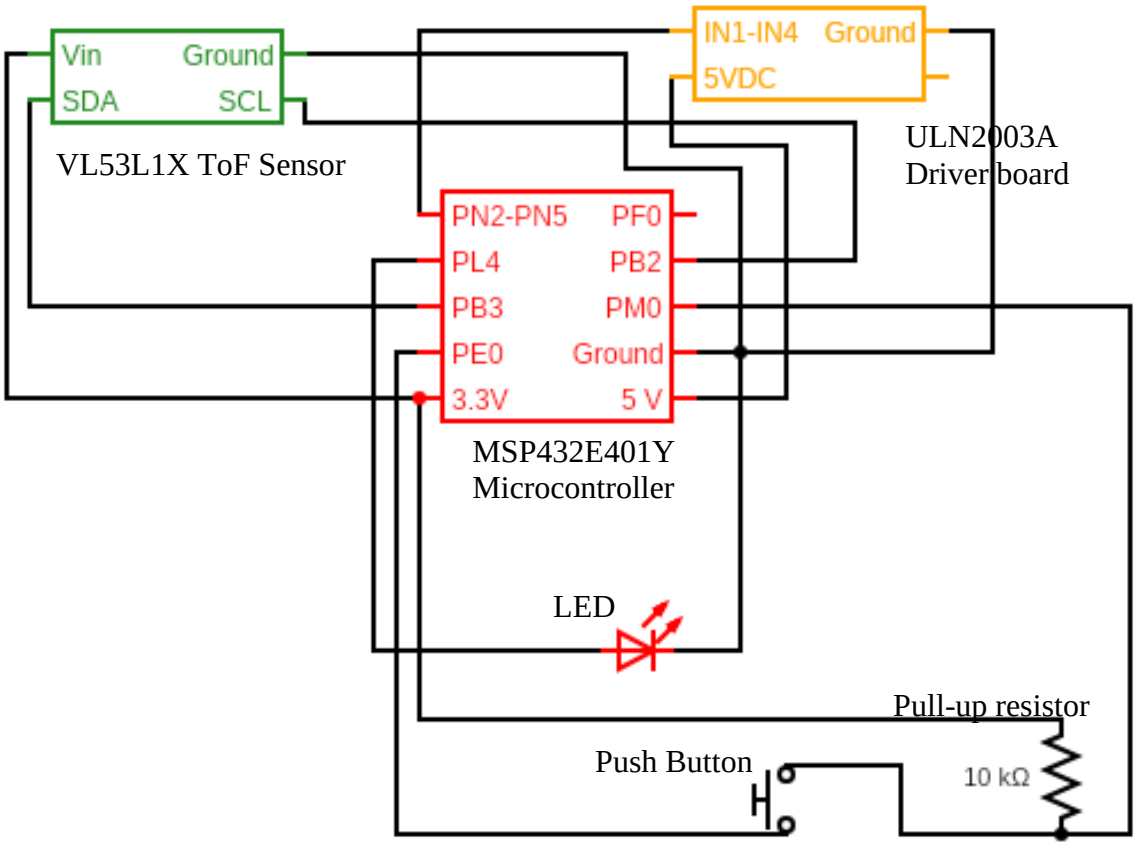


Figure 6. The circuit schematic of the whole system

8. Programming Flowcharts:

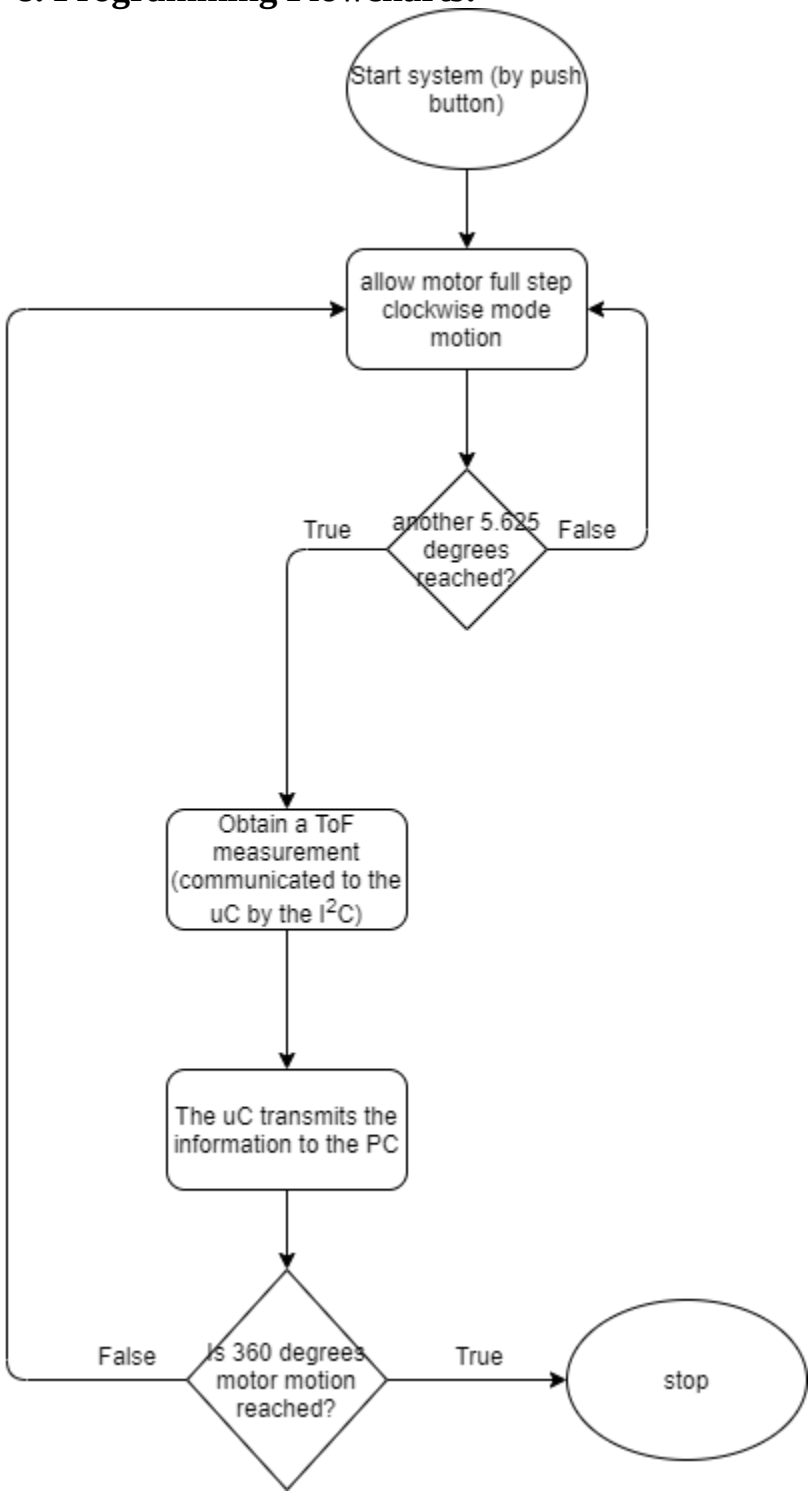


Figure 7. Flow chart explaining the Distance measurement program

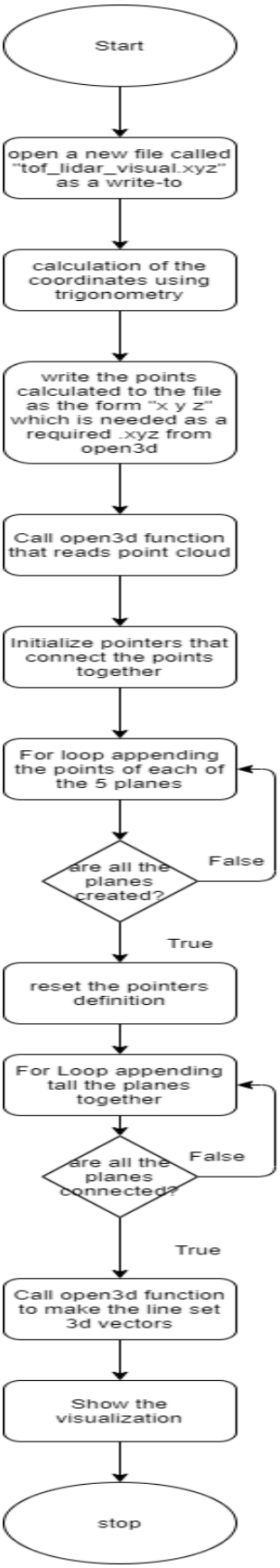


Figure 8. Flowchart showing the visualization programming process