Design and Implementation of an Embedded Spatial Measurement System for Indoor Exploration and Navigation

Taiwo Rabiu

April 12th. 2022

Device Overview

Features

The purpose of this device is to create a 3-Dimentional mapping of its surroundings. This can be used in many applications ranging from self-driving cars, face identification services and also with camera assistance. The microcontroller used in this device is the MSP432E401Y with a stepper motor ULN2003 and ToF sensor VL53L1X.

- 360-degree rotation while capturing data
- 3-Dimentional geographical mapping of surroundings (based on data collected)
- 256KB of memory
- 12 bit ADC
- Cost: $200
- 48Mhz clock speed
- Operating Voltage (3.3V to 5V)
- Uses English language
- Uses I2C and UART serial communication
- 115200 Baud rate

General Descriptions

The system works by controlling a stepper motor from the microcontroller (figure 2) to rotate 360-degrees and mounting a Time-of-Flight sensor (or LIDAR) on the motor. The microcontroller then uses the ToF sensor to take 8 measurements of distance (in millimeters) every 45 degrees. These are then calculated into x, y, and z coordinates on a 3-dimentional scale and is mapped to 3-D space. These 8 measurements can be taken as many times as convenient by the user before the final visual is desired.

Once the data and all the measurements are done, it is sent to the PC (figure 1) through UART (assuming an I2C connection between them has been established) and then using Python, the PC calculates and creates 3D visual of the data collected [2].

Block Diagram

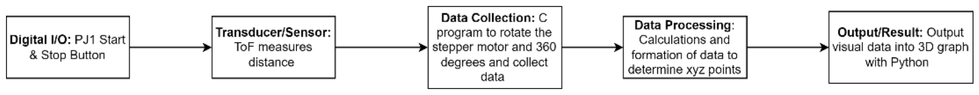

Digital I/O: PJ1 Start & Stop Button
Transducer/Sensor: ToF measures distance
Data Collection: C program to rotate the stepper motor and 360 degrees and collect data
Data Processing: Calculations and formation of data to determine xyz points
Output/Result: Output visual data into 3D graph with Python

Figure 1: Block Diagram Overview

Picture

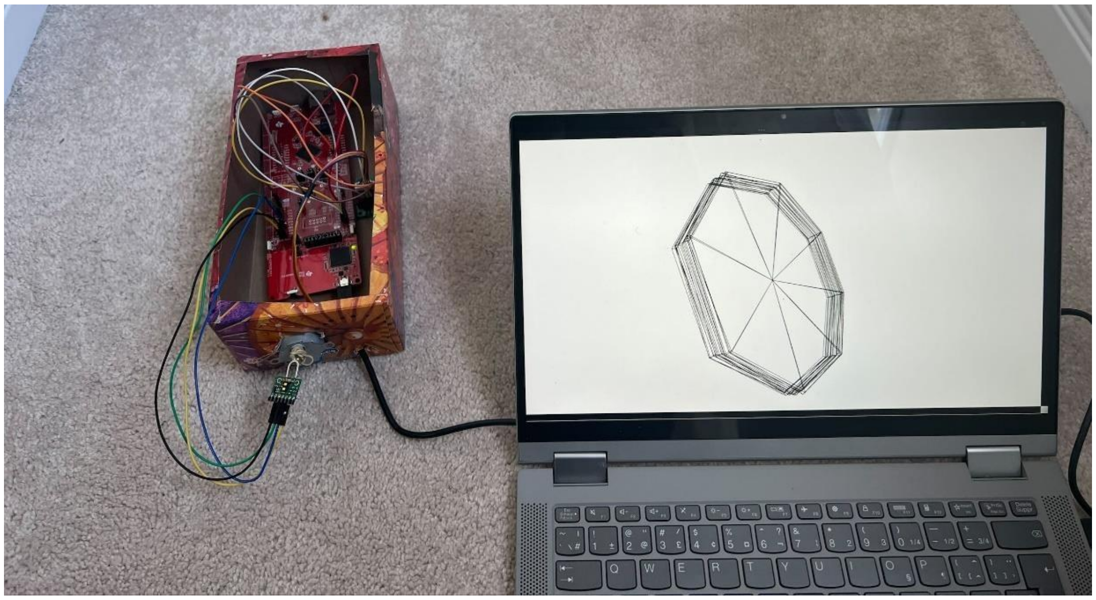


Figure 2: Entire device put together

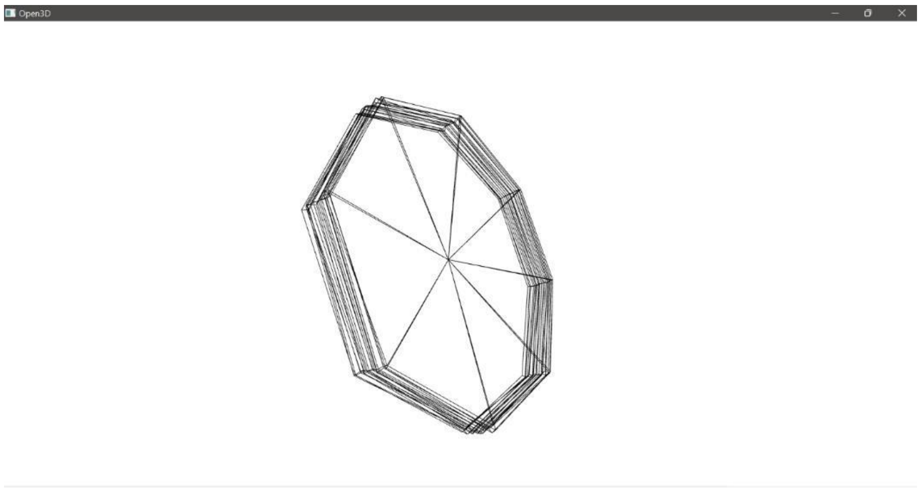


Figure 3: PC Display

Device Characteristics Table

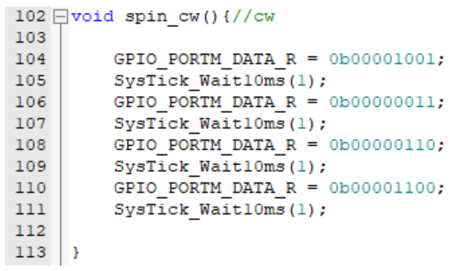
Microcontroller’s Port M pins 0-3 is connected to the stepper motor IN 1-4 respectively, and there is a supply of 5V and negative supply of GND. The ToF sensor should also have a supply of 5V and negative supply of GND with SDA to PB3 and SCL to PB2.

Characteristic	Method of implementation
Programming Language	C code and Python
Pins Used	PM 0-3 (Motor), PB2, PB3 (SCL SDA), 5V, GND
Bus Speed	48 MHz
Serial Part	UART
Communication Speed	115200 Baud rate
Special Libraries	VL53L1X ultra lite driver User Manual [3]
External circuitry	No/On board button PJ1 is used as control

Detailed Description

The Time-of-Flight sensor works by measuring how long it takes for emitted pulses of light to reach the nearest object and be reflected to the detector. It emits pulses of infrared laser light, the light/signal is then reflected back and is collected by the ToF receiver. The delay between the emitted and received periodic signal is measured. I2C communication is established between the microcontroller and the ToF. This is done by firstly connecting the ToF (VL53L1X) to the microcontroller (MSP432E401Y) as seen in the circuit schematic (figure 12). Port B pin 2 and 3 are configured as I2C pins for SDA and SCL (respectively) to establish the I2C connection with the ToF sensor.

The stepper motor works by energizing coils in a logical order to control attraction between magnets making the motor rotate. The motor spins used in this device takes a full step approach. This is done with the microcontroller using PM 0-3. After the connections PM0 – IN1, PM1 – IN2 etc., are made (figure 12) the microcontroller can “spin” the stepper motor shaft by delivering an output of logic HI into the 4 coils (IN 1-3) in a logical order making the magnets move and causing the outer shaft to spin 360 degrees (figure 4). Note a wait time of 10ms is used between each transition to give time for the magnets to move, without this the motor would just appear to vibrate.



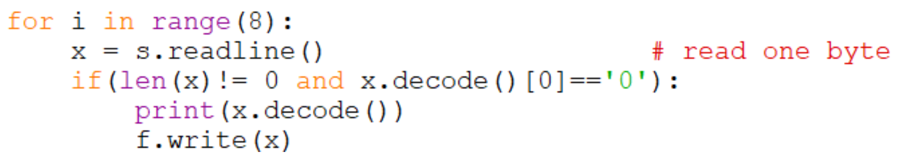
```c
void spin_cw(){//cw

    GPIO_PORTM_DATA_R = 0b00001001;
    SysTick_Wait10ms(1);
    GPIO_PORTM_DATA_R = 0b00000011;
    SysTick_Wait10ms(1);
    GPIO_PORTM_DATA_R = 0b00000110;
    SysTick_Wait10ms(1);
    GPIO_PORTM_DATA_R = 0b00001100;
    SysTick_Wait10ms(1);

}
```

Figure 4: Counter clockwise operation of the stepper motor (full wave)

The code starts by initializing all processes to be used i.e. I2C, UART, interrupts and input and output pins PM 0-3, PB2 and PB3. Note the ToF sensor requires a small amount of time before it can begin to take measurements. Once the program on the microcontroller starts it effective waits for a button push (using interrupts) to perform the next set of actions (figure 13). Once the button press has been detected the motor is spinned 45 degrees then pauses to take a measurement using the ToF. After the measurement from the ToF sensor is done the same step is repeated where the motor spins 45 degrees then halt to take a total of 8 measurements. After each measurement is taken the microcontroller reads it then sends its as an output to the computer using UART where it can be acknowledged and collected for use in the next step.

Visualization

The computer used was a Lenovo IdeaPad 16 GB RAM , intel core i5 chip on Windows 11. The Program that was used is Python 3.8.9 (with IDLE 3.8.9). The libraries that are used in python were “serial”, “math”, “NumPy” and “open3D”. “Serial” was used to create a connection between the PC port and the microcontroller to send data using UART communication. “Math” was used mainly for mathematic calculations mostly using sin, cos, and pi. “NumPy” was used to organize the points/data collected to form a 3D point cloud. Finally, “open3D” was used to visualize and show the connections of the points with lines. After the data (in distance) is collected it is then stored inside a text file for easy access later on (Figure 5).



```python
for i in range(8):
    x = s.readline()                    # read one byte
    if(len(x)!= 0 and x.decode()[0]=='0'):
        print(x.decode())
        f.write(x)
```

Figure 5

After the data is stored in the text file, it is opened again but in “read only” mode to read and calculate the points for the next step. The method of calculation that was used in this stage was to multiply the distance measured by cos of the angle it was collected and the sin of the same angle. For example, let a measurement at x=0 and angle of 90 degree be 50 mm. The point that will be generated will be 0, 50*cos(π/2), 50*sin(π/2) (Figure 6). This would be done for every measurement from 45 to 360 degrees. Note the "math" library in python uses radians instead of degrees. Once all the point values are calculated they are then written to another text file. This text file consists of only x, y, and z points.

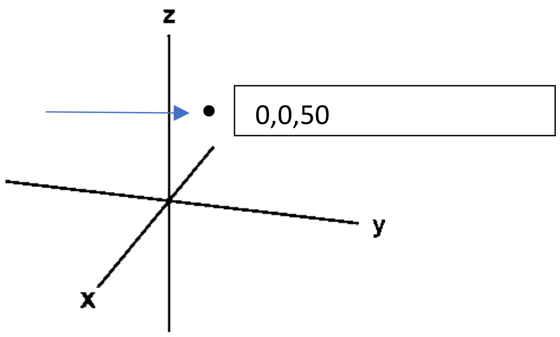
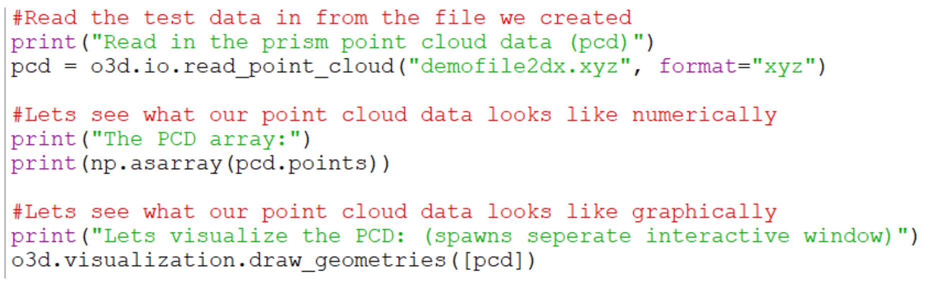


Figure 6

Finally, the x, y, z points from the file are then open in read mode to graph and visualize them. Using open 3D, the data that was calculated is then read and then graphed as points (Figure 7).



```
#Read the test data in from the file we created
print("Read in the prism point cloud data (pcd)")
pcd = o3d.io.read_point_cloud("demofile2dx.xyz", format="xyz")

#Lets see what our point cloud data looks like numerically
print("The PCD array:")
print(np.asarray(pcd.points))

#Lets see what our point cloud data looks like graphically
print("Lets visualize the PCD: (spawns seperate interactive window)")
o3d.visualization.draw_geometries([pcd])
```

Figure 7

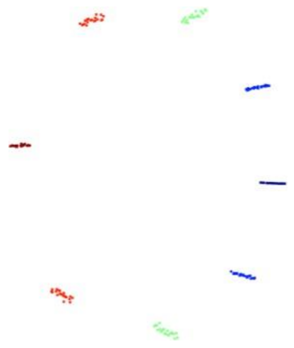


Figure 8: point cloud using 20 slices

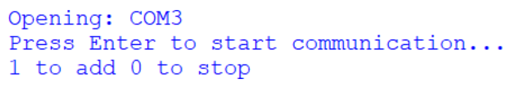
A point cloud is formed as seen in figure 8. Lastly lines are connected through each point of slices to form a space mapping as seen earlier in figure 3.

Application Example with Expected Output and Brief Users Guide

Once the program has began running (ToF has booted, and all ports initialized). It will wait for a signal before beginning to steer the motor and measure with the ToF. This is done with the use of the onboard button PJ1. After the button is pushed the motor spins with the ToF and at every 45 degrees it stops to take a distance measurement with the ToF sensor. Note that the Python program will ask if user is ready to begin by pressing enter (figure 9). Then the following key commands "1" and "0" are used if the user wants to input another slice (8 distance measurements) or to stop to view output. For instance, to map a hallway, the steps required are:

1. Load the program onto the microcontroller and run it (wait for ToF to boot)
2. Open Python and press Enter
3. Press "1" to add a slice
4. Press PJ1 to begin collecting the slice
5. Hold the product steady until full rotation is complete
6. Take a step forward
7. Go to step 3 to add another slice or "0" to stop and view end output

After these steps have been taken a pop up like figure 8 plotting measurements should be seen and to see the lines connected close the first pop up.

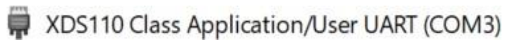


```
Opening: COM3
Press Enter to start communication...
1 to add 0 to stop
```

Figure 9:

Setting up the device

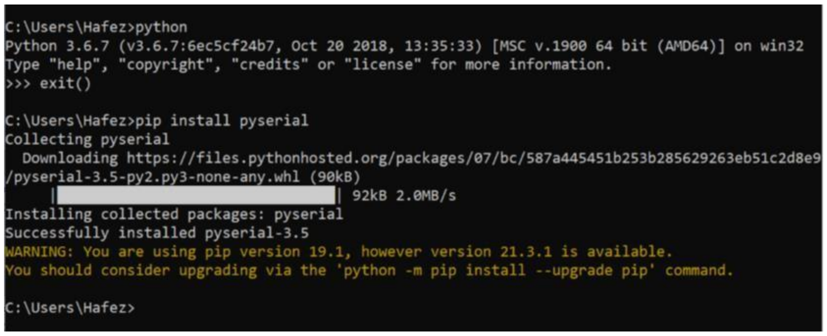
In order to set up the device the user would first need set up the device as in the shown in the circuit schematic in figure 10 connecting the stepper motor and ToF to the microcontroller. Once this is done build the code in Keil then flash it to the microcontroller. Next to set up the communication protocol first plug in the microcontroller then go to settings, then device manager. Scroll down "Ports (COM and LPT)" (figure 10). Find the port corresponding to XDS110 Class Application/User UART (COM #) where # is a digit. Take note of this for later as in this case the COM port is 3 but may be different depending on the user.



XDS110 Class Application/User UART (COM3)

Figure 10

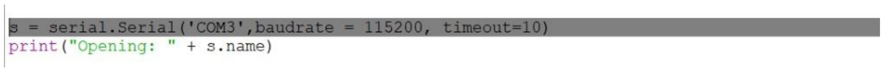
Next ensure you have Python installed with a version 3.9 or less (3.10 and higher does not work with open 3d). This could be done in the home page of command prompts by typing "python" and pressing enter. Once Python is verified return to the home page of command prompt then enter "pip install pyserial" (figure 11) and "pip install open3d"to begin to install the pyserial and open module. After this process is completed open the python file.



```
C:\Users\Hafez>python
Python 3.6.7 (v3.6.7:6ec5cf24b7, Oct 20 2018, 13:35:33) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> exit()

C:\Users\Hafez>pip install pyserial
Collecting pyserial
  Downloading https://files.pythonhosted.org/packages/07/bc/587a445451b253b285629263eb51c2d8e9
/pyserial-3.5-py2.py3-none-any.whl (90kB)
    |████████████████████████████████| 92kB 2.0MB/s
Installing collected packages: pyserial
Successfully installed pyserial-3.5
WARNING: You are using pip version 19.1, however version 21.3.1 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.

C:\Users\Hafez>
```

Figure 11

Inside the python file in line 17 ensure that the COM number is the same as found above and the baud rate is a value of 115200 in order to properly collect the data from the microcontroller.



```
s = serial.Serial('COM3',baudrate = 115200, timeout=10)
print("Opening: " + s.name)
```

Lastly, note that the xyz plane is defined with respect to the measurements as in (figure 6). Axis y and z are used for the vertical slices and x is used for the movement forward which is manually incorporated in the python file [1].

Limitations

1. Summarize any limitations of the microcontroller floating point capability and use of trigonometric function.

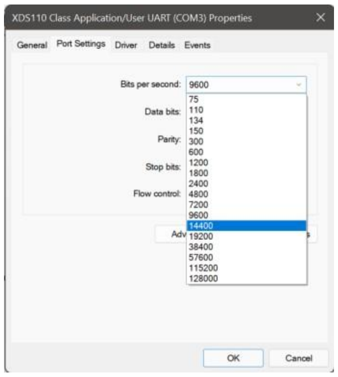
The Floating-Point Unit (FPU) on the microcontroller provides conversions between fixed points and floating-point data formats. It provides 32-bit instructions for single precision (C floats) data processing operations [4]. Python received the distance values as integers. Trigonometric functions were implemented into the Python file by importing the math Python library import math. Performing trigonometry by using the sine and cosine functions from the math library did not cause any limitation since casting interview as a float is possible in Python.

2. Calculate your maximum quantization error for each of the ToF module.

The maximum quantization error is the rang error seen in the datasheet from 20mm to 25mm.

3. What is the maximum standard serial communication rate you can implement with the PC? How did you verify?

The maximum standard serial communication rate that could be implement with the PC is 128000 bits per second. This was verified firstly by ensuring both devices (PC and microcontroller) are capable of running at such speed then testing the code to verify the same output was given as before.



Bits per second: 9600

4. What were the communication method(s) and speed used between the microcontroller and the ToF modules?

The communication method and speed used between the microcontroller and ToF modules was I2C to establish the communication between the two modules and UART to send data between the two modules. The speed used was 100kbps.

5. Reviewing the entire system, which element is the primary limitation on speed? How did you test this?

The element with the primary limitation on speed would be PSYDIV function controlling the clock/bus speed. Testing this was done by changing the value of PSYDIV also changing the clock speed to a lower value (slower speed) and it was noted that the speed of the entire system was reduced by timing the motor and observing the motor speed.

Works Cited

[1] “2021-2022 2DX3 2DX4 PROJECT SPECIFICATION – OBSERVE, REASON, ACT:

SPATIAL MAPPING USING TIME-OF-FLIGHT ” Project specifications for 2DX3, Department of Engineering, McMaster University, Winter, 2022.

[2] “Week 7 – Communication Protocols” class notes for 2DX3, Department of Engineering, McMaster University, Winter, 2022.

[3] “A guide to using the VL53L1X ultra lite driver UM2510 User manual,” 2021. [Online]. Available: https://www.st.com/resource/en/user_manual/dm00562924-a-guide-to-using-the-vl53l1x-ultra-lite-driver-stmicroelectronics.pdf

[4] “MSP432E401Y SimpleLinkTM Ethernet Microcontroller 1 Device Overview,” Texas Instruments 2017.

Circuit Schematic

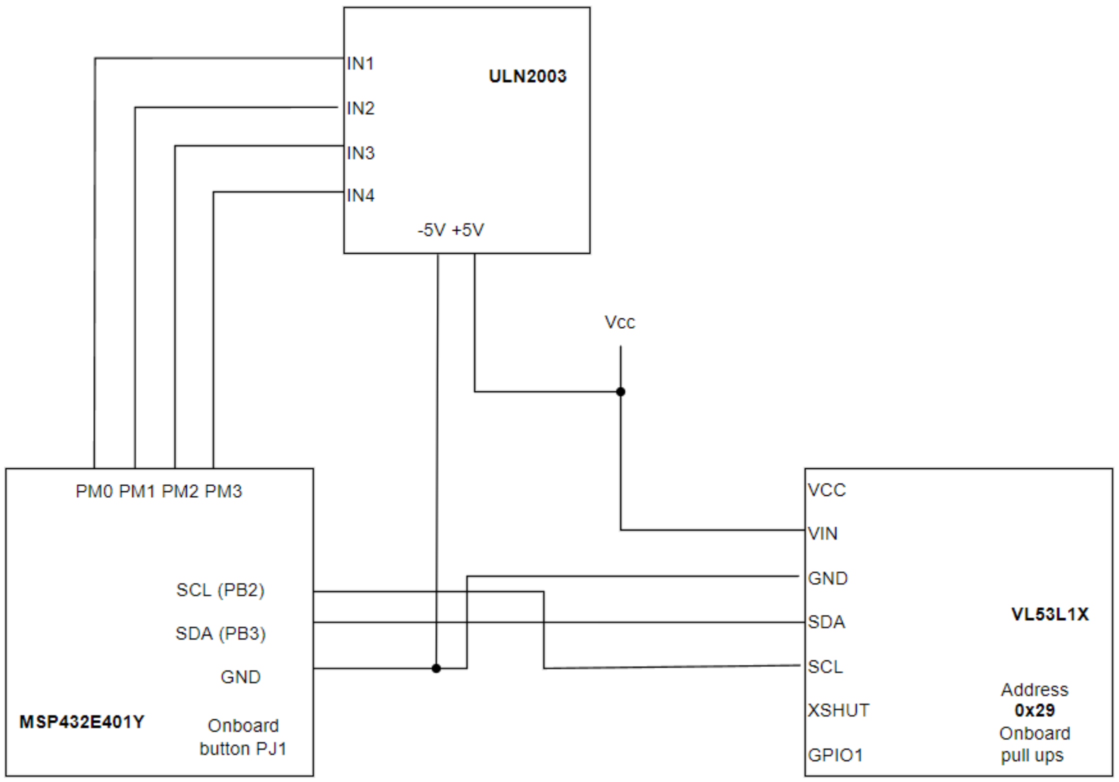


Figure 12: Circuit Schematic

Programming Logic Flowchart

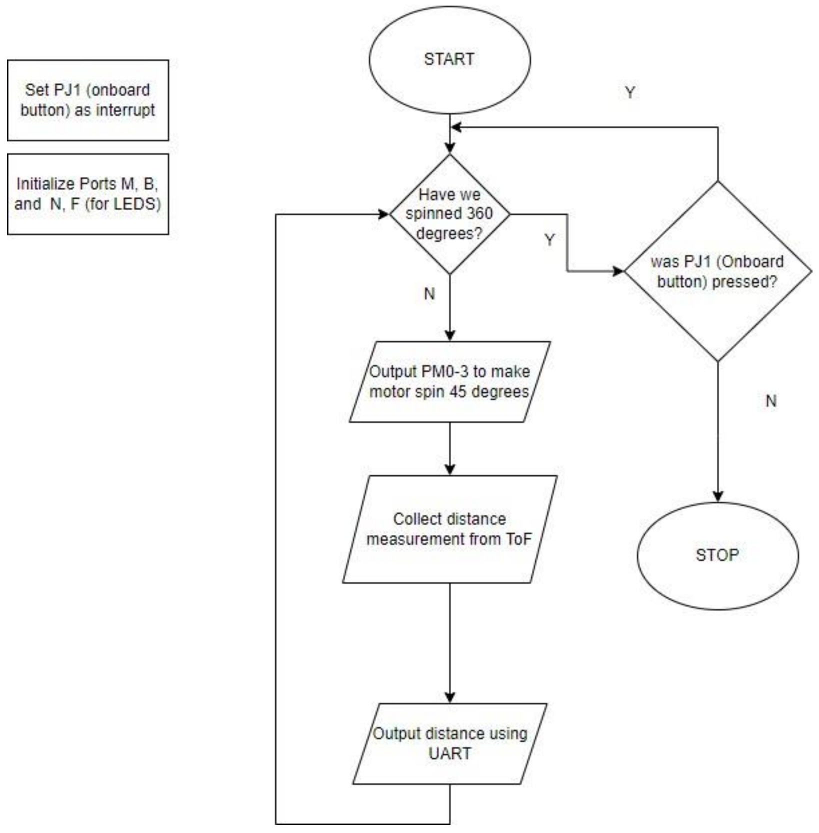

Set PJ1 (onboard button) as interrupt
Initialize Ports M, B, and N, F (for LEDS)
START
Have we spinned 360 degrees?
Y
N
Y
was PJ1 (Onboard button) pressed?
N
Output PM0-3 to make motor spin 45 degrees
Collect distance measurement from ToF
Output distance using UART
STOP

Figure 13: Flow Chart for microcontroller program in Keil

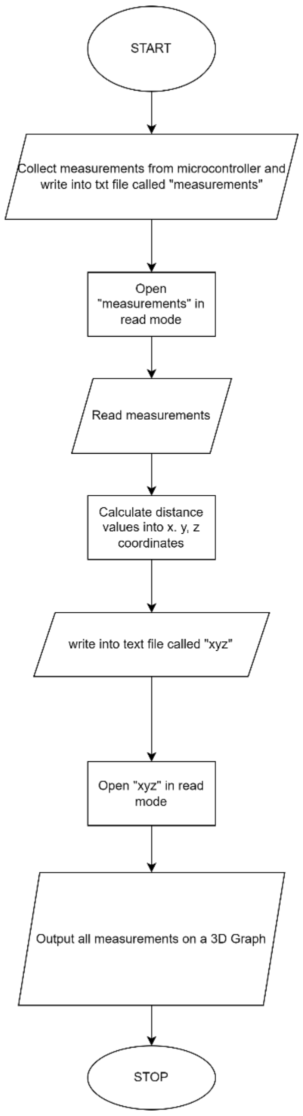

START
Collect measurements from microcontroller and write into txt file called "measurements"
Open "measurements" in read mode
Read measurements
Calculate distance values into x. y, z coordinates
write into text file called "xyz"
Open "xyz" in read mode
Output all measurements on a 3D Graph
STOP

Figure 14: Flowchart for Python source code