2DX: Microprocessor Systems Final Project

Instructors: Drs. Boursalie, Doyle, and Haddara

Christy Joseph-Anton – josepc11 – 400325365 – 2DX3
– Friday Afternoon – L05

Contents

Device Overview

Features

- Embedded Spatial Measurement System
- 48 MHz Bus Speed
- Operating Voltage of 4.75-5.25V for MCU
- 1024 KB of Flash memory
- 2 12-bit ADC Channels, each up to 2 Msps
- Approximately $92.97+taxes for all components
- Uses C for MCU, and Python for PC
- ToF sensor communicates with MCU using I2C, MCU communicates with PC using UART
- Baud rate of 115200 bps
- Full 360-degree planar mapping
- Fixed displacement samples along orthogonal axis of plane
- Measurements transmitted to PC via USB in real-time
- Generates interactive 3D model of each scan
- Data acquisition start/stop button
- Microcontroller communicates directly with included Python program

General Description

The 2DX3 Final Project is an all-in-one spatial mapping embedded system that uses a time-of-flight sensor. It can scan an area with 360-degree planes along an orthogonal axis and display a 3D graphical representation on a PC running a Python program. It also stores XYZ coordinates in a separate file that may be used for further analysis.

This system encompasses a microcontroller unit (MCU) based on the Cortex M4, a stepper motor, and a ToF sensor. The system also needs a PC running prerequisite software. The MCU is responsible for processing data, as well as communicating with the ToF sensor and PC via I2C and UART, respectively. It also provides power to the various components mentioned above. The ToF sensor is responsible for data acquisition. It is mounted on the stepper motor, allowing it to capture data from a 360-degree plane. Finally, the PC is used to receive distance measurements, and store/display them for the user, as well as provide power to the MCU.

The VL53L1X measures distance using LIDAR. It emits a pulse of light with a wavelength of 940nm. The light will then hit the nearest object and return to the sensor. The component measures how long it takes for the emitted pulse of light to reach the object and return to the sensor. Using the time and speed of this pulse, distance can be obtained. It then uses I2C to transmit the distance data in millimeters to the MCU.

The included Python program waits for the user’s input, then allows the MCU and ToF sensor to boot. Once this process is complete, an onboard push button is used to start/stop data acquisition. After all planes have been scanned along the orthogonal axis, the included Python script will calculate the correct XYZ coordinates and save them to a file. This file is then used to generate a 3D visualization of the scan.

The following components are used:

- MSP-EXP432E401Y Microcontroller
- 28BYJ-48 Stepper Motor and ULN2003 Driver
- VL53L1X ToF Sensor
- Jumper wires
- PC running Python 3.6-3.9

Block Diagrams

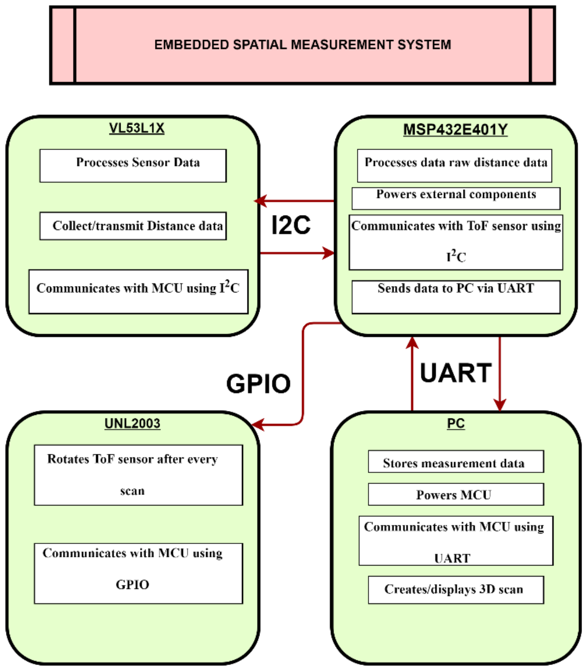

EMBEDDED SPATIAL MEASUREMENT SYSTEM
VL53L1X
Processes Sensor Data
Collect/transmit Distance data
Communicates with MCU using I2C
I2C
MSP432E401Y
Processes data raw distance data
Powers external components
Communicates with ToF sensor using I2C
Sends data to PC via UART
GPIO
UART
UNL2003
Rotates ToF sensor after every scan
Communicates with MCU using GPIO
PC
Stores measurement data
Powers MCU
Communicates with MCU using UART
Creates/displays 3D scan

Figure 1: Block Diagram 1

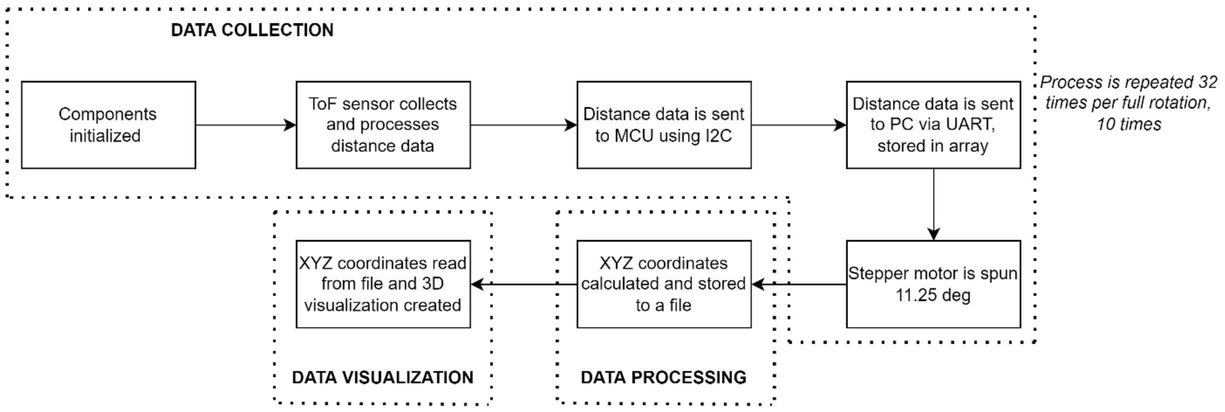

DATA COLLECTION
Components initialized
ToF sensor collects and processes distance data
Distance data is sent to MCU using I2C
Distance data is sent to PC via UART, stored in array
Process is repeated 32 times per full rotation, 10 times
Stepper motor is spun 11.25 deg
XYZ coordinates calculated and stored to a file
XYZ coordinates read from file and 3D visualization created
DATA VISUALIZATION
DATA PROCESSING

Figure 2: Block Diagram 2

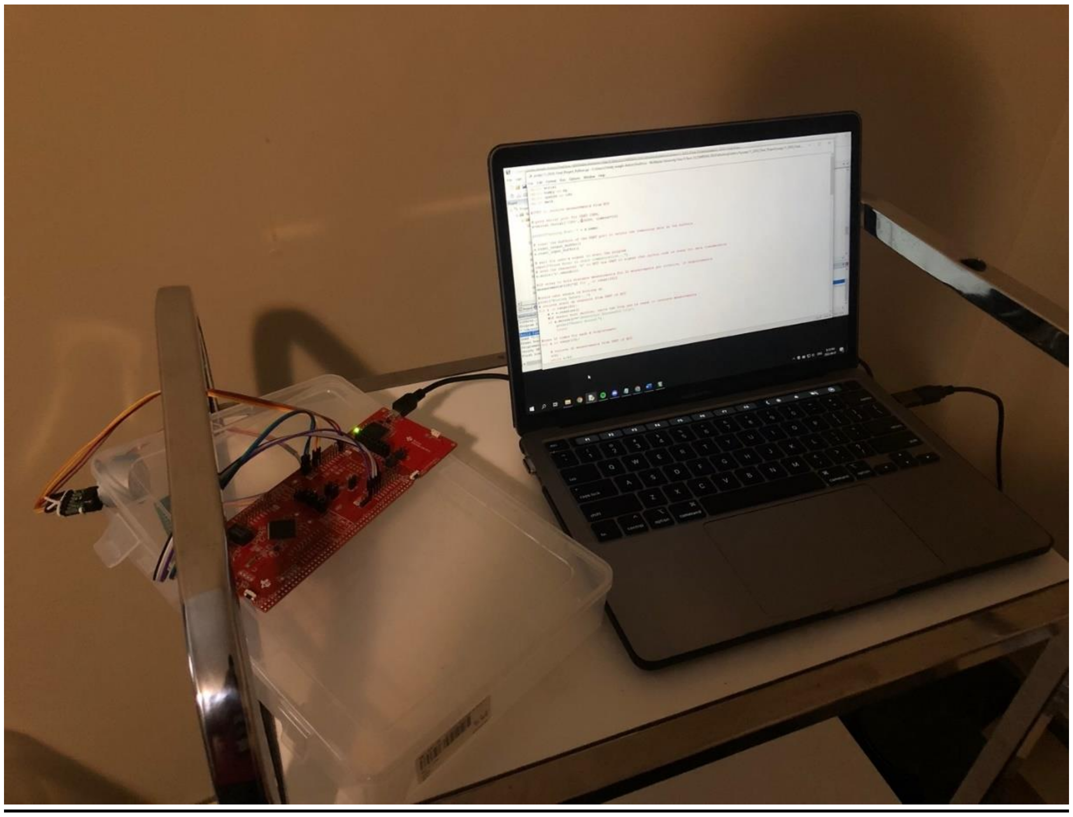
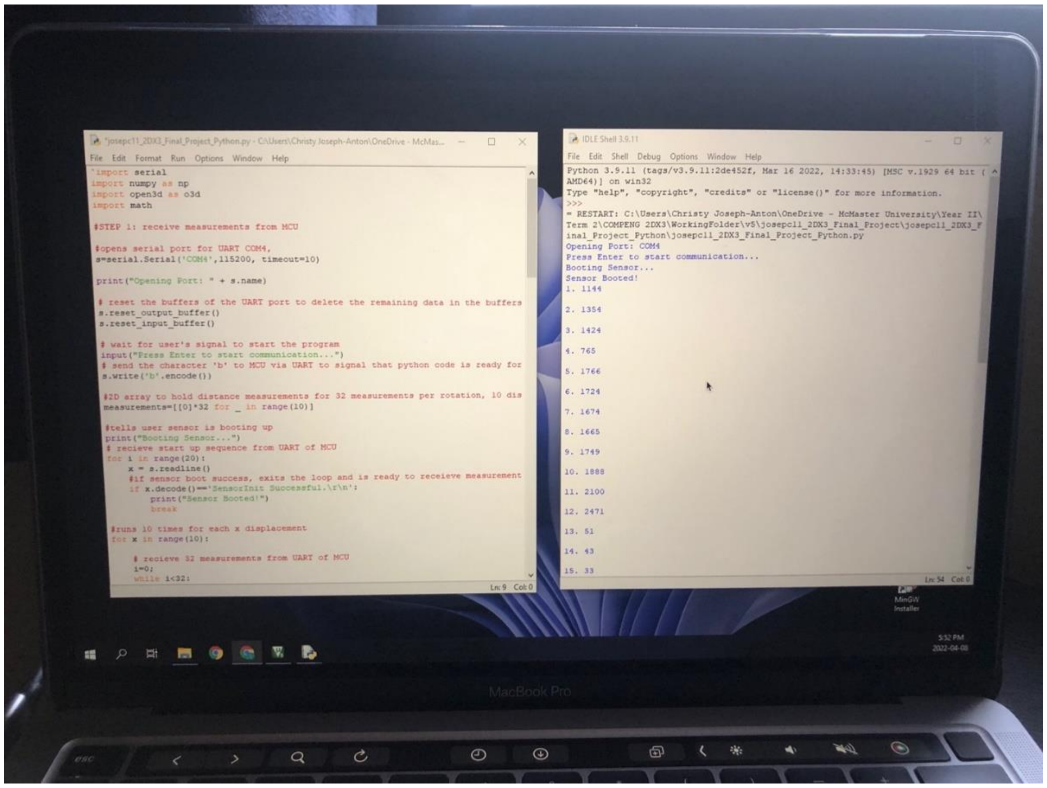


Figure 3: Picture of Entire System



*josepc11_2DX3_Final_Project_Python.py - C:\Users\Christy Joseph-Anton\OneDrive - McMas...
File Edit Format Run Options Window Help
Ln: 9 Col: 0
IDLE Shell 3.9.11
File Edit Shell Debug Options Window Help
Ln: 54 Col: 0
MinGW Installer
5:52 PM
2022-04-08
MacBook Pro

```
import serial
import numpy as np
import open3d as o3d
import math

#STEP 1: receive measurements from MCU

#opens serial port for UART COM4,
s=serial.Serial('COM4',115200, timeout=10)

print("Opening Port: " + s.name)

# reset the buffers of the UART port to delete the remaining data in the buffers
s.reset_output_buffer()
s.reset_input_buffer()

# wait for user's signal to start the program
input("Press Enter to start communication...")
# send the character 'b' to MCU via UART to signal that python code is ready for
s.write('b'.encode())

#2D array to hold distance measurements for 32 measurements per rotation, 10 dis
measurements=[[0]*32 for _ in range(10)]

#tells user sensor is booting up
print("Booting Sensor...")
# recieve start up sequence from UART of MCU
for i in range(20):
    x = s.readline()
    #if sensor boot success, exits the loop and is ready to receieve measurement
    if x.decode()=='SensorInit Successful.\r\n':
        print("Sensor Booted!")
        break

#runs 10 times for each x displacement
for x in range(10):

    # recieve 32 measurements from UART of MCU
    i=0;
    while i<32:
```

```
Python 3.9.11 (tags/v3.9.11:2de452f, Mar 16 2022, 14:33:45) [MSC v.1929 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\Christy Joseph-Anton\OneDrive - McMaster University\Year II\
Term 2\COMPENG 2DX3\WorkingFolder\v5\josepc11_2DX3_Final_Project\josepc11_2DX3_F
inal_Project_Python\josepc11_2DX3_Final_Project_Python.py
Opening Port: COM4
Press Enter to start communication...
Booting Sensor...
Sensor Booted!
1. 1144

2. 1354

3. 1424

4. 765

5. 1766

6. 1724

7. 1674

8. 1665

9. 1749

10. 1888

11. 2100

12. 2471

13. 51

14. 43

15. 33
```

Figure 4: Picture of PC Screen

Device Characteristics

MSP-EXP432E401Y		ULN2003		VL53L1X		PC	
Feature	Detail	Device Pin	MCU Pin	Device Pin	MCU Pin	Feature	Detail
Bus Speed	48 MHz	VDD	5V	VDD	3.3V	Operating system	Windows 10
Serial Port	COM4	GND	GND	GND	GND	Program	Python 3.6-3.9
Baud Rate	115200 bps	IN1	PL0	SDA	PB3	Special Libraries	Open3D Numpy Math Serial
		IN2	PL1				
		IN3	PL2				
		IN4	PL3				

Detailed Description

Distance Measurement

A graphical representation of the section below can be found under the "Programming Logic Flow Chart" section.

To boot the device, see the "Brief Users Guide" section. This process includes setting up the python code, booting the ToF sensor, and enabling ranging. Once the device is booted, the device will stay in the idle state. The state of the device can only be changed by the onboard push button, PJ1. This functionality is driven by an interrupt. This implementation was chosen over polling because it requires less CPU power, and results in faster start/stop operations. Once the button is pushed, the device will get distance data from the ToF sensor via I2C.

Distance capture is done by the VL53L1X ToF sensor. It emits a pulse of light with a wavelength 940nm that then reflects off a near object. It then determines how long that pulse of light takes to come back to the sensor. Using the speed of light, and the time taken for the light to return, distance can be found. The following equation can be used to determine distance.

$$distance = \frac{2c}{t}, \text{ where c} \approx 3.0 * 10^8 m/s$$

Note, the time measured by the sensor is the time for the light to travel from the sensor and back, therefore the time must be divided by two to find the distance of the object relative to the sensor.

The VL53L1X ToF sensor accomplishes the above distance measurement using Analog-to-Digital Conversion (ADC). This process involves the following components:

1. Transducer:
   Converts the nonelectrical source to an electrical signal, this is done by the physical sensor.
2. Signal Conditioning:

Prepares the electrical signal for digital conversion, this will perform operations such as level shifting, and amplification to get the signal within the range of the ADC input values.

3. Analog to Digital Conversion:
   Converts continuous electrical signal to a discrete digital value. This process will most likely include quantization error. See "Limitations" section for more details.
4. Digital output:
   Raw digital ADC value is converted into distance value in millimetres.

Once the VL53L1X ToF sensor has found a distance, it will send it to the MCU via I2C. Distance from the sensor is a 16-bit value. It is then stored in a variable 'Distance' in the MCUs onboard memory.

The ToF sensor will also send its Range Status. This tells the MCU whether the data is valid. If the data was not valid, the device would pause at a given angle until valid data is received. However, this functionality was commented out due to unpredictability of where the device will be used, as stronger ambient lighting conditions could cease functionality. This option can be uncommented if needed.

Next the distance data sent to the Python program running on the PC via UART. This is done by writing the data to the UART buffer. Note, since UART is configured to send 8-bit data at a time, two frames are needed to send the 16-bit distance data. Next, the motor is spun 11.25 deg in the clockwise direction, and then flashing the onboard LED. This process is repeated 32 times. Also, the state of the device is checked every loop. So, if the GPIO interrupt is triggered, the device will go into the idle state until the button is pressed again. Once it is completed, the motor is reset to its original position by rotating 360 deg in the counter clockwise direction. The python program will then inform the user to move 20cm forward. Once, the user has moved, they must press the onboard push button again to repeat the process. This process is repeated 10 times.

The python program will be started and initialize as per the first 6 steps in the Python Flowchart in section "Programming Logic Flow Chart". It will then receive the data from the MCU via UART and store them in a 2D array. See flowchart for more details. Next, the program will calculate the XYZ coordinates for each measurement value. This is done using the following formulas. For detailed information about of the coordinates are appended to the file see the Python Flowchart mentioned above.

i=0 ; # of displacements, increments each loop, up to 10

distance=measurements[i][anglesNum] ; where anglesNum is # of angles, starts at 0, increments each loop, up to 32

angle=0 ; angle of stepper motor, increments by 11.25deg each loop, up to 360 deg

$$\boldsymbol{X = 200 * i}$$

$$\boldsymbol{Y = distance * \cos(angle)}$$

$$\boldsymbol{Z = distance * \cos(angle)}$$

The above formulas are used to append the correct XYZ coordinates to a file. These were chosen based on the coordinate system used for this design. See "Axis Definition" for more information. Using trigonometry, the distance data from the ToF sensor and the angle of the stepper motor can be used to determine these coordinates. The program will append 320 coordinates, each separated by a new line. The data is stored in an .xyz file.

Visualization

The 3D visualization is performed by a Python library called Open3D. The libraries' functions were used to visualize the data. The data set used for visualization is the .xyz file created in the previous section.

Visualization of a hallway was tested using 2020 MacBook Pro running Windows 10 Pro using virtualization software. The laptop possesses an Intel Core i5-1038MG7, Intel Iris Plus Graphics, 16GB of DDR4 RAM, and a 500GB SSD. The virtualization software used was Parallels Desktop 17 for Mac 17.1.1.

Python Code was run using the built in IDLE IDE included with Python 3.9.11 64 bit. The latest available versions of the libraries Open3D, NumPy, math, and serial were installed onto the system.

The .xyz file that stores the coordinates is opened and used to make a point cloud within the open3D library. To do this, the 'io.read_point_cloud' function is used. The .xyz file is passed as an argument to the function. To verify this, the newly created array is printed to the console to be verified. Next, each vertex is given a unique number. This will be applied to all 320 points. Next lines are created between each of the vertices in each plane. This was done by creating a line between two adjacent vertices in the plane. This is repeated 32 times per plane. In the last step in the visualization process, the 10 planes are connected using lines. Each of the vertices on plane 1 is connected to the corresponding vertices on vertex 2. This is repeated for all planes. The lines are then mapped to the 3D coordinate vertices. Finally, the 3D visualization is displayed using the 'visualization.draw_geometries()' function included in the open3D library, using the newly created lineset as its argument. A graphical explanation of the above process can be found in the Python Flow chart in the "Programming Logic Flow Chart" section.

To test the device, a hallway was scanned. The lights were also dimmed to help the sensor's performance.

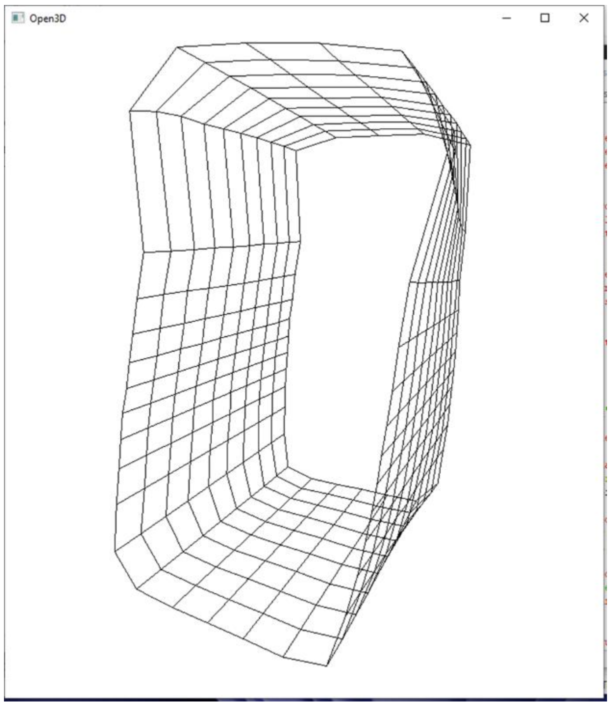


Figure 5: 3D scan of hallway

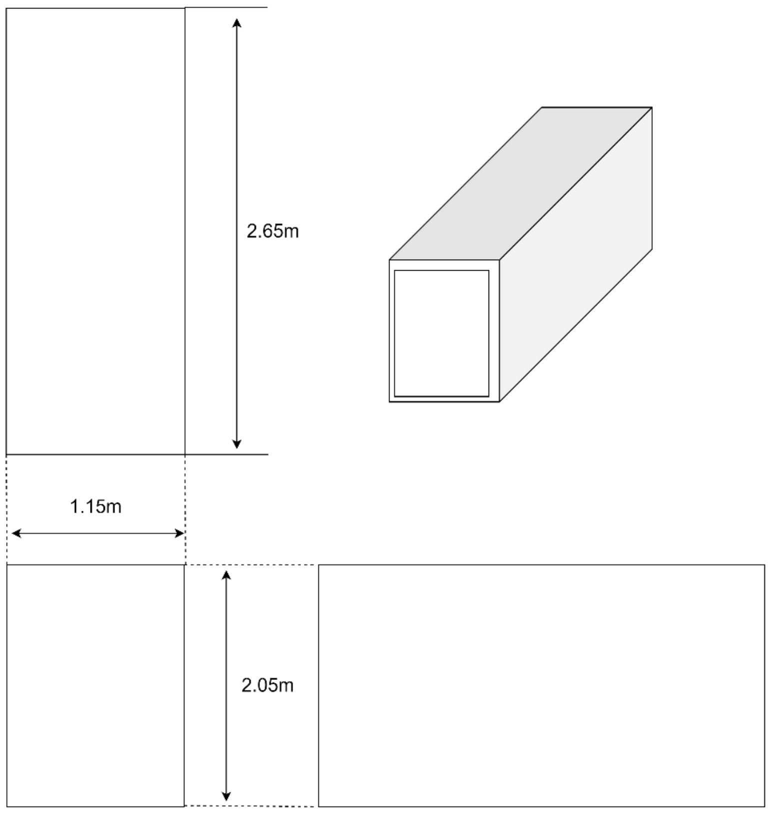
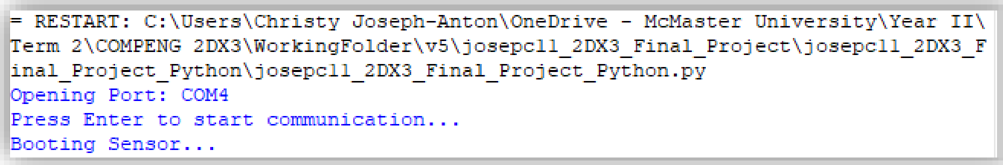


Figure 6: rough dimensioned sketch of hallway

Application Example with Excepted Output

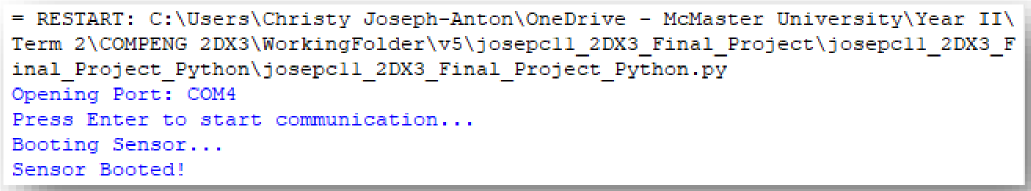
This next section will outline how to use the embedded device.

1. Hold system above waist level, in a hallway, or other room to be scanned. The system should not be too close to the ground to ensure whole room is scanned.
2. Press the reset switch on the microcontroller. *
3. Open included Python Code on a PC running prerequisite software (see User's Guide) *
4. Press 'Enter' on when Python Code is ready. This tells the MCU the python code is running and begins the Sensor initialization process. The following message should appear:



```
= RESTART: C:\Users\Christy Joseph-Anton\OneDrive - McMaster University\Year II\
Term 2\COMPENG 2DX3\WorkingFolder\v5\josepcl1_2DX3_Final_Project\josepcl1_2DX3_F
inal_Project_Python\josepcl1_2DX3_Final_Project_Python.py
Opening Port: COM4
Press Enter to start communication...
Booting Sensor...
```

5. Once the sensor is booted, the following message will appear:



```
= RESTART: C:\Users\Christy Joseph-Anton\OneDrive - McMaster University\Year II\
Term 2\COMPENG 2DX3\WorkingFolder\v5\josepcl1_2DX3_Final_Project\josepcl1_2DX3_F
inal_Project_Python\josepcl1_2DX3_Final_Project_Python.py
Opening Port: COM4
Press Enter to start communication...
Booting Sensor...
Sensor Booted!
```

6. The system is now ready for data collection. Press the onboard push button (PJ1) to start a planar scan. The data will be displayed in real time in the Python Console. **
7. Once 1 360-degree scan is complete, move the system 20cm in the orthogonal plane.
8. Repeats steps 6 and 7 nine more times, for a total of 10 360-degree scans
9. Once data collection is complete, the python program will calculate coordinates automatically, and display a 3D visualization of the room that was scanned. This will open in another window.

*order of steps 2 and 3 do not matter

**data acquisition can be started/stopped at anytime. The device will not respond to button presses while it is resetting the sensor to its original position after each 360-degree scan. This is to ensure the sensor is always at the correct starting position.

Below is a scan of a hallway. This is an example of the expected output of the system.

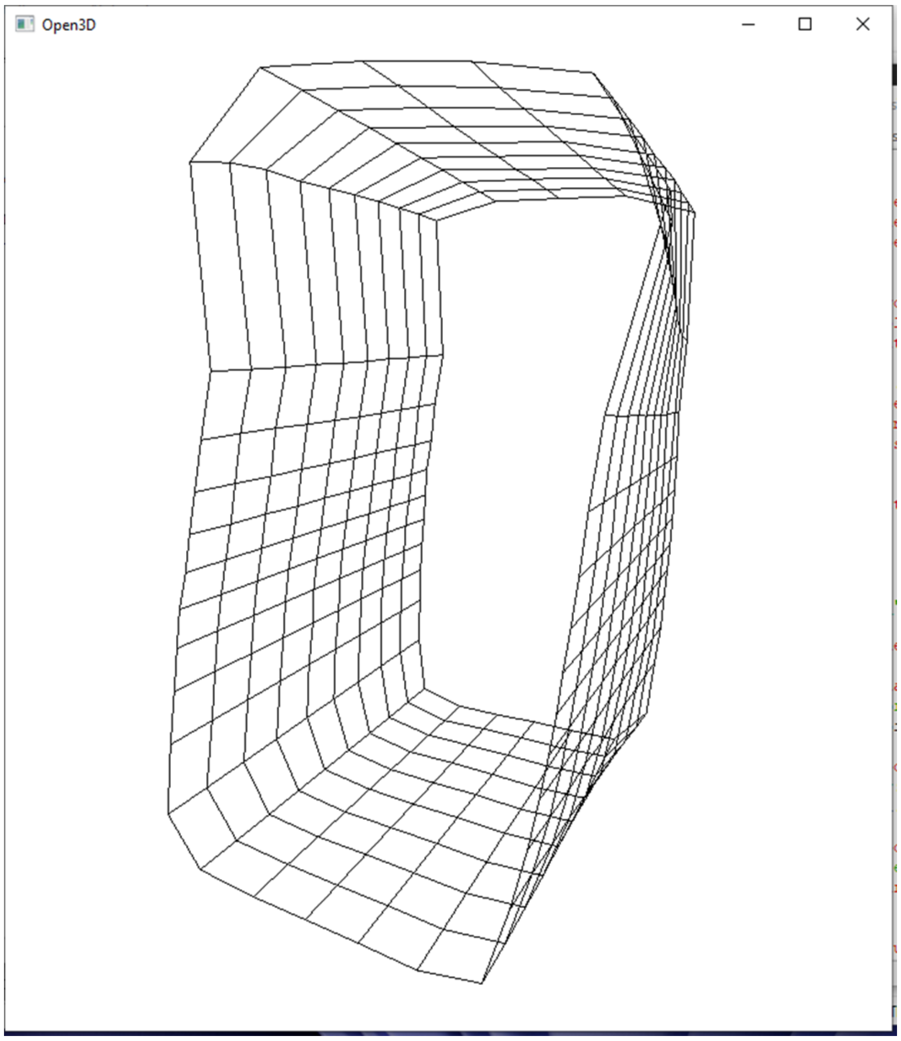
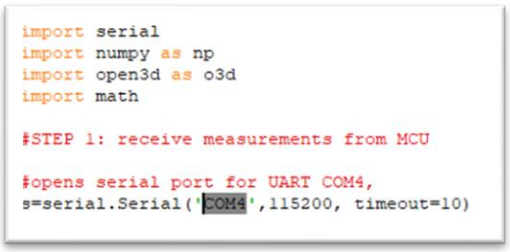


Figure 7: Expected Hallway Output

Brief Users Guide

Main Guide

1. Assemble circuit as shown in Circuit Schematic below (*Figure 8*).
2. Attach ToF sensor to stepper motor securely, ensure wires will not get tangled during operation
3. Mount stepper motor securely to box or another similar item. Ensure sensor is pointing upwards.
4. Flash C code to microcontroller via micro-USB.
5. Reset microcontroller CPU using onboard reset switch
6. Open included Python code with an editor (Ex. Python, VS Code)
7. Change the value ‘COM4’ to user specific COM Port. For further instructions on how to find user specific COM Port settings, refer to “Finding COM Port Settings Guide” below.



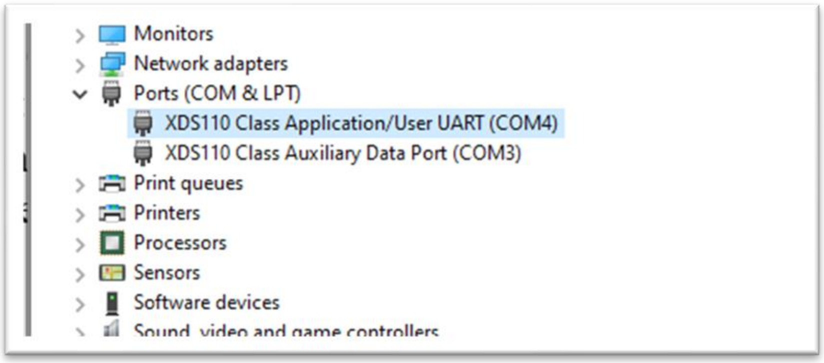
```
import serial
import numpy as np
import open3d as o3d
import math

#STEP 1: receive measurements from MCU

#opens serial port for UART COM4,
s=serial.Serial('COM4',115200, timeout=10)
```

The baud rate can also be changed here. However, additional parameters must be changed in the MCUs C code. Lines 17/18 in the uart.c file correspond to changing the MCU's baud rate.

8. Run the Python Code
9. The system is now setup for data collection and ready to be used. Refer to "Application Example with Excepted Output" to operate the system.

Finding COM Port Settings Guide

1. Ensure the microcontroller is connected
2. Right click on the windows logo in the bottom left.
3. Select Device manager
4. Expand the field 'Ports'
5. Beside the field 'XD110 Class Application/User UART', in brackets will be the specific COM Port for your setup.


Monitors
Network adapters
Ports (COM & LPT)
XDS110 Class Application/User UART (COM4)
XDS110 Class Auxiliary Data Port (COM3)
Print queues
Printers
Processors
Sensors
Software devices
Sound, video and game controllers

In this example, the Port is COM4.

Axis Definition

For this device, the axis is defined as follows. This explanation assumes the device is being held in front of the user, sensor initially pointing upwards, and the scan will be taken moving forward. The positive X direction is pointing away from the user, The positive Y axis is pointing upwards, and finally the positive Z axis is pointing to the left. Each measurement is taken on the YZ plane, and displacement is along the positive X axis.

Limitations

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

The MSP432E401Y Microcontroller has a Floating- Point Unit (FPU) that supports single precision add, subtract, multiply, divide, multiply-and-accumulate, square root operations. It can also perform conversions between fixed/floating point numbers. It can support floating point numbers up to 32 bits in precision. To perform trigonometric calculations, the c library 'math' can be used. However, the required calculations are done on the PC for this system due to increased precision, code readability and ease of importing libraries into Python.

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

The maximum quantization can be founding using the following equation:

$$error = \frac{d_{FS}}{2^n}, n = \#bits, d_{FS} = \text{full scale distance}$$

The sensor has a minimum distance of 0mm and a maximum of 4000mm, therefore Dfs is 4000mm. The distance measurement is represented as a 16bit number, therefore n=16:

$$error = \frac{d_{FS}}{2^n}$$
$$error = \frac{4000mm}{2^{16}}$$
$$error = \frac{4000mm}{2^{16}}$$
$$error = 0.061035\ mm$$

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 can be implemented with the PC is of 115200 bps. This was verified by looking at the maximum standard speeds found in the program RealTerm under the ports heading. RealTerm lists the standard port speeds.

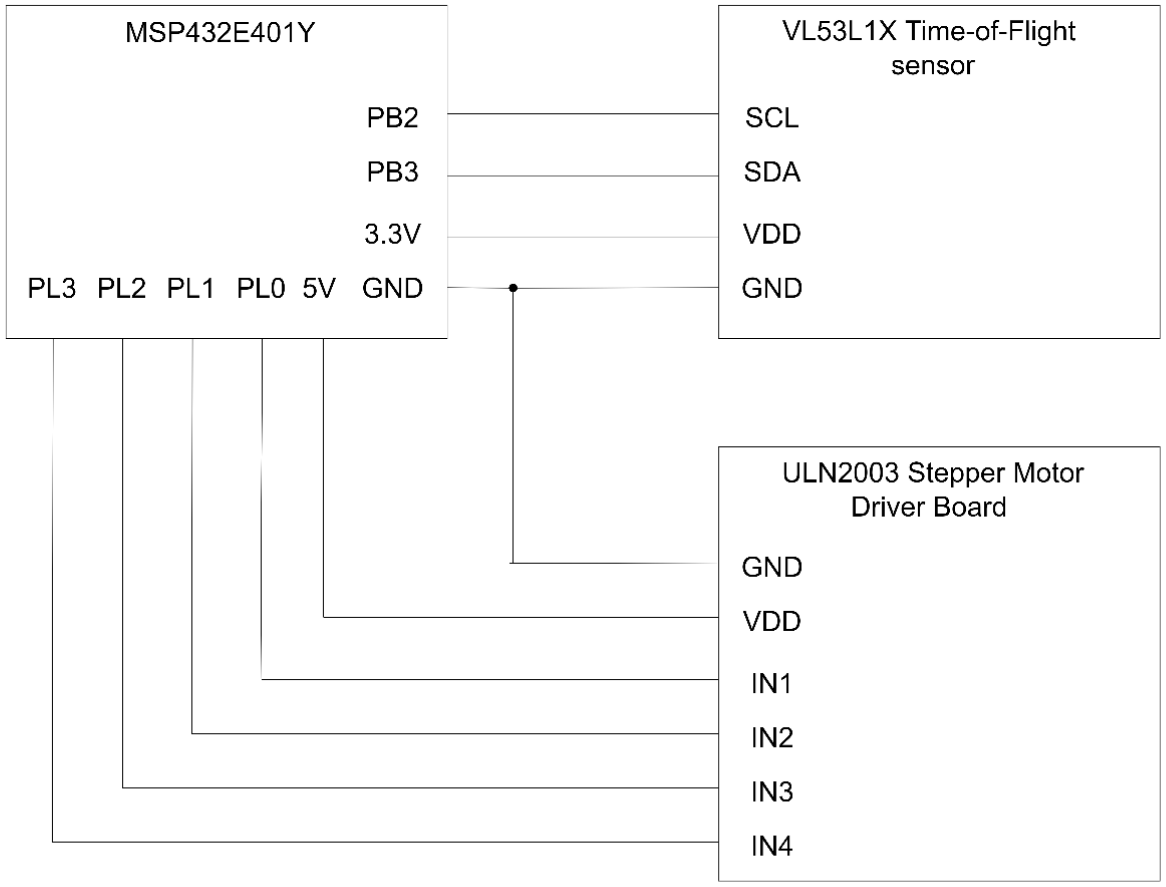
4. **What were the communication method(s) and speed used between the microcontroller and the ToF modules?**

Communication between the ToF modules and the microcontroller was done using I2C serial communication. The speed of clock is set to 100kbps.

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

The primary limit on speed in this system is the ToF sensor's ranging capabilities. While trying to increase the speed of entire system, some delays were removed. However, the ToF sensor would return no relevant data, as it did not have enough time to get a new reading. It is apparent the ToF needs a minimum amount of time to function properly. Speeding up the stepper motor's operation did not cause any problems like the ToF sensor.

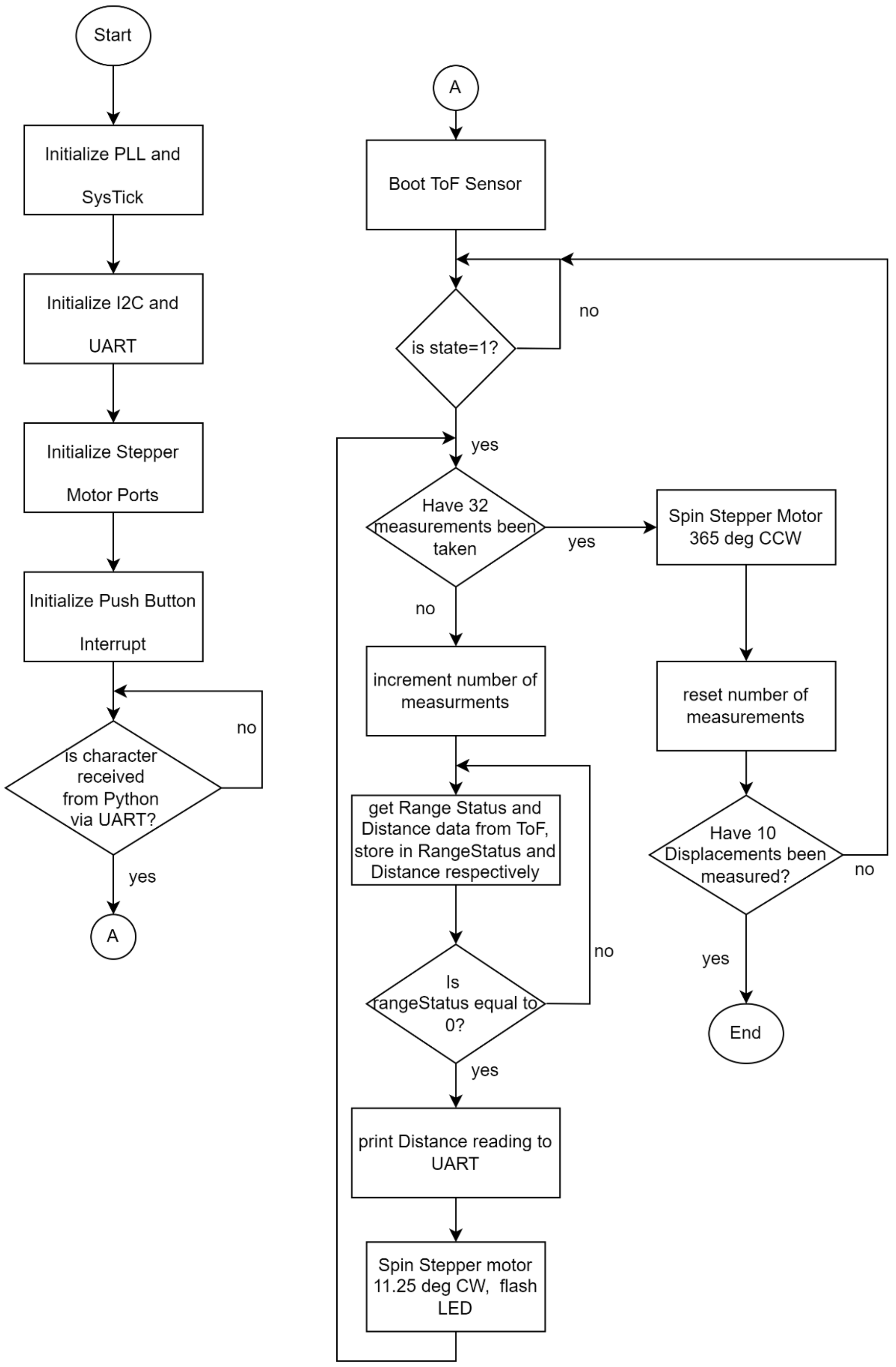
Circuit Schematic



The stepper motor attaches to the ULN2003 driver board

Figure 8: Circuit Schematic

Programming Logic Flow Chart



Start
Initialize PLL and SysTick
Initialize I2C and UART
Initialize Stepper Motor Ports
Initialize Push Button Interrupt
is character received from Python via UART?
no
yes
A
A
Boot ToF Sensor
is state=1?
no
yes
Have 32 measurements been taken
yes
no
Spin Stepper Motor 365 deg CCW
increment number of measurments
reset number of measurements
get Range Status and Distance data from ToF, store in RangeStatus and Distance respectively
Have 10 Displacements been measured?
no
Is rangeStatus equal to 0?
no
yes
yes
End
print Distance reading to UART
Spin Stepper motor 11.25 deg CW, flash LED

Figure 9: MCU Code Flowchart

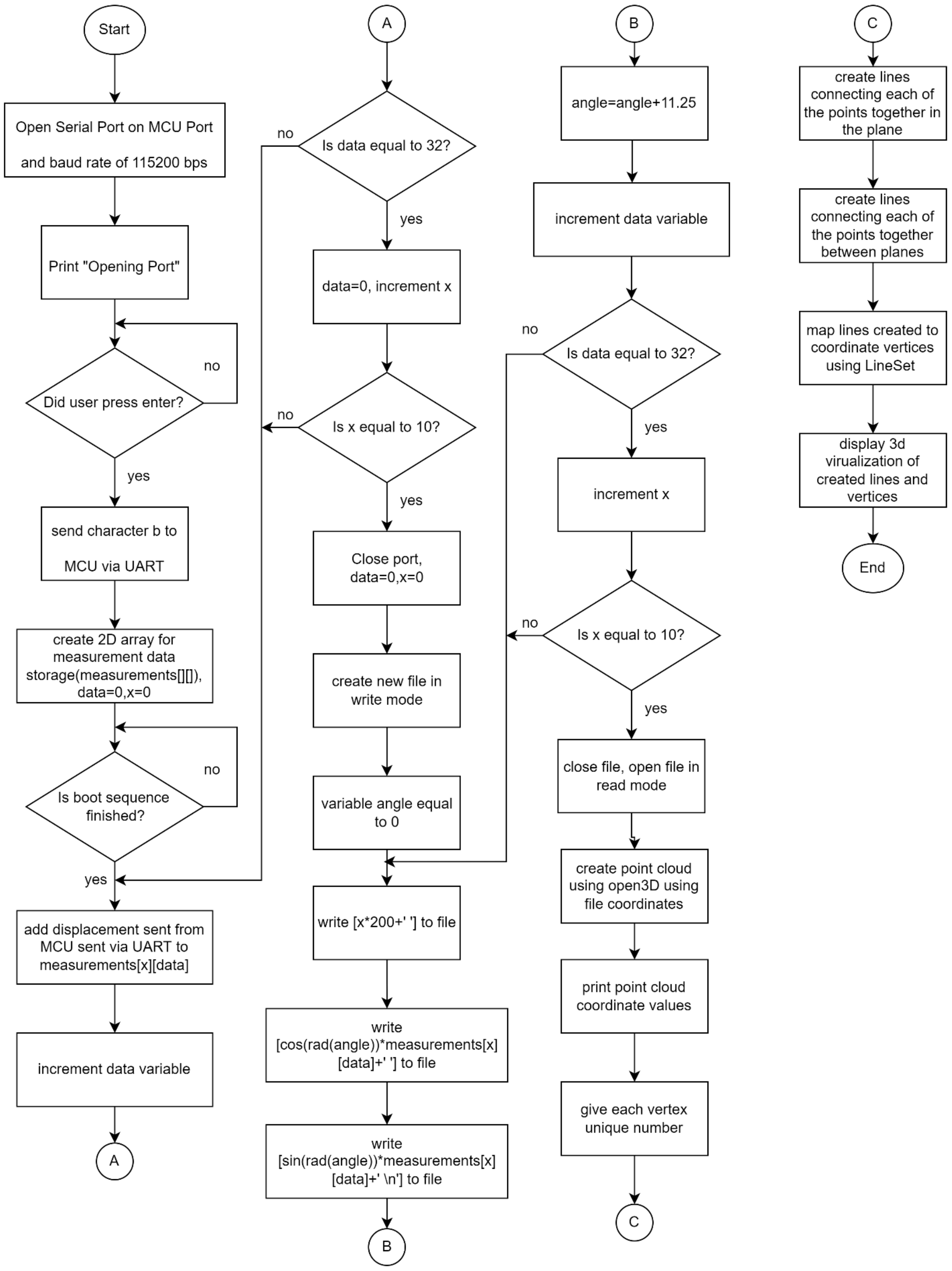

Start
Open Serial Port on MCU Port and baud rate of 115200 bps
Print "Opening Port"
Did user press enter?
no
yes
send character b to MCU via UART
create 2D array for measurement data storage(measurements[][]), data=0,x=0
Is boot sequence finished?
no
yes
add displacement sent from MCU sent via UART to measurements[x][data]
increment data variable
A
A
Is data equal to 32?
no
yes
data=0, increment x
Is x equal to 10?
no
yes
Close port, data=0,x=0
create new file in write mode
variable angle equal to 0
write [x*200+' '] to file
write [cos(rad(angle))*measurements[x][data]+' '] to file
write [sin(rad(angle))*measurements[x][data]+' \n'] to file
B
B
angle=angle+11.25
increment data variable
Is data equal to 32?
no
yes
increment x
Is x equal to 10?
no
yes
close file, open file in read mode
create point cloud using open3D using file coordinates
print point cloud coordinate values
give each vertex unique number
C
C
create lines connecting each of the points together in the plane
create lines connecting each of the points together between planes
map lines created to coordinate vertices using LineSet
display 3d virualization of created lines and vertices
End

Figure 10: Python Code Flowchart

References

“2021_2022_2DX3_2DX4_Project_Specification”, Project Specification for COMPENG 2DX3, Department of Electrical and Computer Engineering, McMaster University, Winter, 2022.

“Computer Engineering 2DX3 & 2DX4 2021-2022 Laboratory Manual”, Laboratory Manual for COMPENG 2DX3, Department of Electrical and Computer Engineering, McMaster University, Winter, 2022.

“Week 2- Transduction and analog signal acquisition”, Lecture Notes for COMPENG 2DX3, Department of Electrical and Computer Engineering, McMaster University, Winter, 2022.

“Week 5- Interrupts (Studio and Lab: 7, Final Project Suggested Milestones 5-1)”, Lecture Notes for COMPENG 2DX3, Department of Electrical and Computer Engineering, McMaster University, Winter, 2022.

“Week 6- Implementing Signal Transfer Functions & Calibration (Studio and Lab: 8, Final Project Suggested Milestones 5-3)”, Lecture Notes for COMPENG 2DX3, Department of Electrical and Computer Engineering, McMaster University, Winter, 2022.

“Week 8- System Integration, Organization, and Abstract Data Types”, Lecture Notes for COMPENG 2DX3, Department of Electrical and Computer Engineering, McMaster University, Winter, 2022.

“Week 9- Data Visualization”, Lecture Notes for COMPENG 2DX3, Department of Electrical and Computer Engineering, McMaster University, Winter, 2022.

“Mot-28byj48 stepper motor w/ ULN2003 Driver,” *Electromechanical :: Motors :: Stepper Motors :: Mini Stepper Motors :: MOT-28BYJ48 Stepper Motor w/ ULN2003 Driver*. [Online]. Available: https://abra-electronics.com/electromechanical/motors/stepper-motors/mini-stepper-motors/mot-28byj48-stepper-motor-w-uln2003-driver.html. [Accessed: 08-Apr-2022].

“MSP-EXP432E401Y,” *DigiKey*. [Online]. Available: https://www.digikey.ca/en/products/detail/texas-instruments/MSP-EXP432E401Y/8106010. [Accessed: 08-Apr-2022].

“Pololu - VL53L1X time-of-flight distance sensor carrier with voltage regulator,” *Pololu Robotics & Electronics*. [Online]. Available: https://www.pololu.com/product/3415. [Accessed: 08-Apr-2022].