2DX: Microprocessor Systems Final Project

Instructors: Drs. Boursalie, Doyle, and Haddara

Daniel Loi – loid1 – 400315666– 2DX3 – Monday Evening – L07

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 **[Daniel Loi, loid1,400315666]**

Device Overview

Features:

- MSP432E401Y Microcontroller LaunchPad™ Development Kit
  - 2 System clocks
    - PIOSC frequency: 16 MHz
    - MOSC frequency: 25 MHz
  - Cortex M4F Microprocessor
  - 2 User switches and 1 reset switch
  - Used as the leader for ToF and UART communicator for visualization
- VL53L1X Time-of-Flight Sensor
  - Operating Voltage: 3.3V
  - I2C Communication Protocol
  - Used to take measurements up to a maximum range of 4m
- MOT-28BYJ48 Stepper Motor
  - Operating Voltage: 5V
  - Used to rotate the ToF sensor for a 360-degree measurement
- 1 External Push Button
  - Use for stopping/starting the measurement for the system
  - Operating Voltage: 3.3V
- Open-source Python code and visualization
  - Open3D python library for visualization of measurements
  - pyserial for UART communications at 115200 baud rate

General Description:

The MSP432E401Y is the main component of this system, it allows for I2C communication with peripherals and UART communication with the PC. When the microcontroller is powered on, the microcontroller immediately configures all the input and output ports used in this system. Then it starts the configuration of the UART program and the I2C communication protocol for the ToF sensor. Using an interrupt-based program, the system will not start taking measurements until the on-board button PJ1 is pressed which lets the sensor immediately take a measurement at the horizontal then starts to rotate the stepper motor at an angle depending on how many points the user wants to have. At each distance measurements the microcontroller receives the data from the distance measurement using I2C and transfers that data to the PC using UART to prepare with the visualization.

The external push button uses an active low system which means that when its idle it has a voltage of 3.3V and goes to 0V when the button is pressed. The button has a connection with the microcontroller at port PM1. The button is programmed to have a higher priority than the on-board button to prioritize the need to stop, otherwise the system will continue to run normally.

The MOT-28BYJ48 stepper motor is used to have more precision with angles allowing for more accurate mapping of the area that is being measured. The connection of the motor uses PH[0:3] connected to IN[1:4] respectively and operates with a voltage of 5V. After completing a complete cycle,

the motor will rotate back 360 degrees to untangle the wires that the ToF sensor uses while it is mounted on the motor.

The VL53L1X Time-of-Flight sensor uses LIDAR technology to obtain the measurements by measuring the time it takes for a photon to hit the target and reflect onto the sensor. This sensor is mounted onto the stepper motor where it moves along with the angle that the motor rotates through to stop and take a measurement each time.

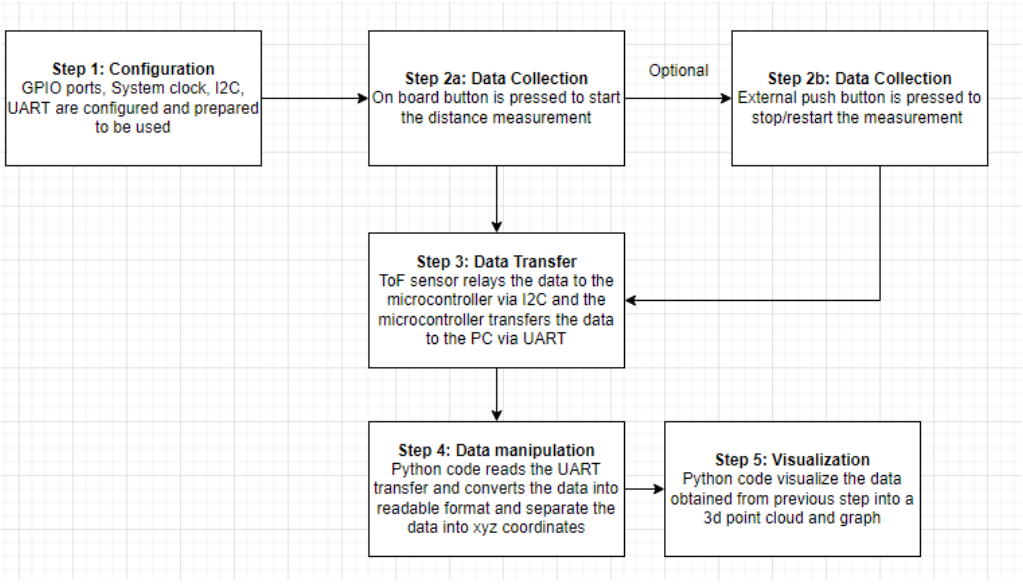

Figure 1 Block Diagram

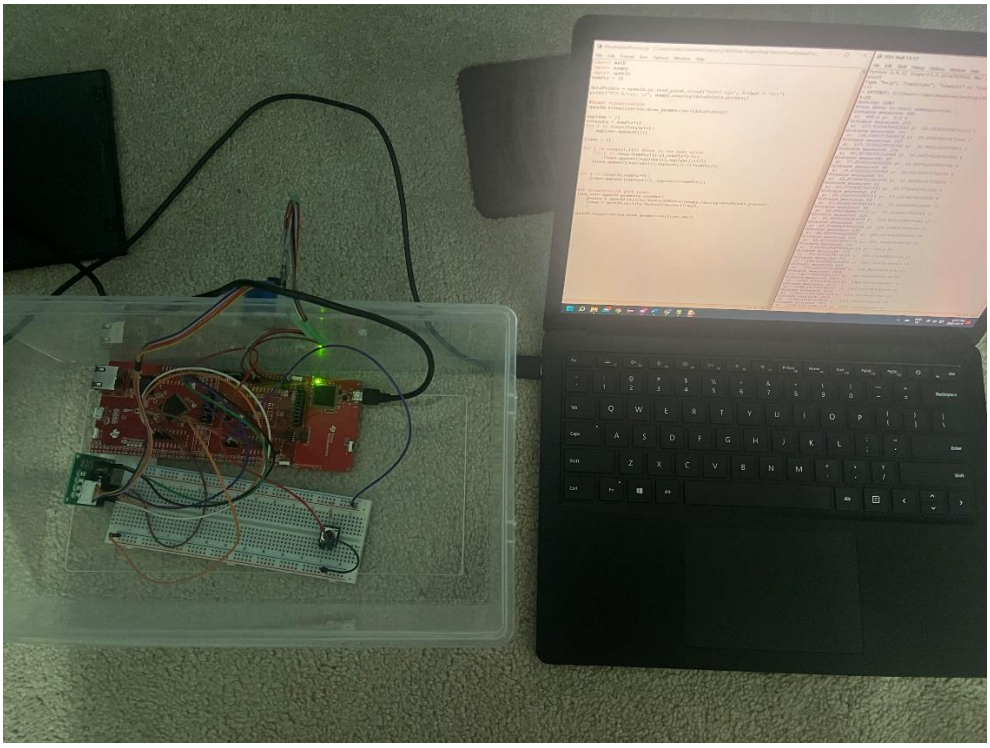

Figure 2 Personal setup

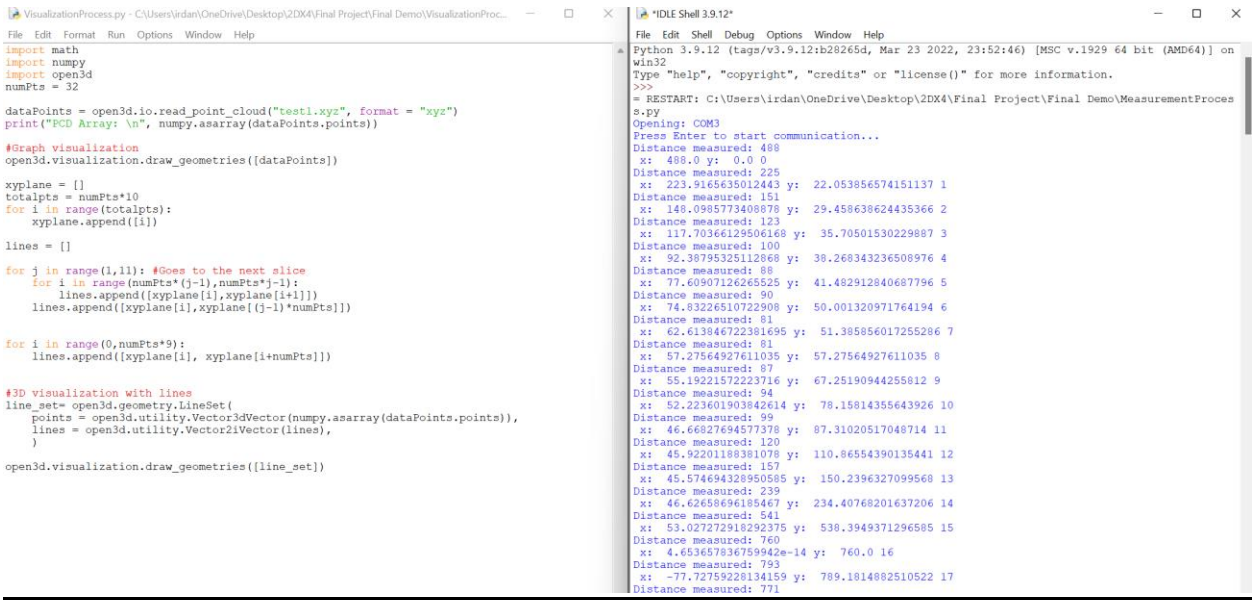

Figure 3 System output

Device Characteristic Table

Key Parts	Description
Microcontroller	Bus Frequency: 480 MHz Assigned System Clock Speed: 80 MHz Status LED: PF4
Stepper Motor	Operating Voltage: 5V Pins: Port H [0:3] = IN[1:4] Connection to GND required Time delay per step: 10 ms
VL53L1X Time-of-Flight sensor	Operating Voltage: 3.3V (2.6V - 5.5V) Connection to GND required Pins: SDA = PB3 SCL = PB2
External Push Button	Operating Voltage: 3.3V Connection to GND required Pins: PM1
Python UART and visualization	Micro USB A/B connection UART Baud Rate: 115200 bps Python libraries: Open3D, pyserial, math, numpy

Table 1 Characteristic Table

Detailed Description

Distance Measurement:

To start the measurement process, the peripherals and wiring must be connected beforehand, the wiring for the system can be found in the circuit schematic page. Then the microcontroller needs to be restarted once the code is loaded into the microcontroller from the Keil project. Afterwards the configuration for the ports, sensor activation and communication protocols will start. The on-board LEDs D1 and D2 will be active and will turn off once the configuration is over.

After the configuration step is finished, the python module should be run with the python IDLE application that comes along with the python installed. Once the python module is run, the python code will send a signal to the microcontroller to enable the UART communication through the XDS110 Class Application/User UART port that is found through the device manager found in the PC search bar. To find what COM the PC uses it is necessary to go to the Ports section and find the port mentioned earlier only when the microcontroller is connected onto the PC.

After the UART connection is established, all that is left to do is to start the measurement program. To start the program the on-board button PJ1 located on the right side of the board is required to be pressed. The sensor will take a measurement immediately and depending on how many points the user wants to have mapped per cycle will have the motor rotate with the angle equally split apart from each other. After completing an entire cycle, the motor will rotate backwards to its initial position to prevent and of the ToF sensors' wires from getting tangled up and interfering with the measurement of another plane. To calculate the distance, the sensor uses the following formula:

$$Distance = \frac{Photon\ travel\ time}{2} * Speed\ of\ light$$

Each data point is immediately processed and transferred to the PC via UART. The data points are then processed into its x and y coordinate to prepare for the visualization process. The z axis will have to be a manual input and require a person to physically move the system in the z axis to measure different planes. The x and y components can be calculated by:

$$x = Distance * \cos(\theta)$$

$$y = Distance * \sin(\theta)$$

The process of the distance measurement can also be found in figure x in the flowchart section for more details on the microcontroller and python algorithm.

Visualization:

This system is using a windows operating system laptop and python version 3.9.12. The python libraries and function used in this system includes: Open3d, pySerial, math and numpy. Further details about the PC specifications can be found in the figures below.

System Manufacturer	Microsoft Corporation
System Model	Surface Laptop 3
System Type	x64-based PC
System SKU	Surface_Laptop_3_1867:1868
Processor	Intel(R) Core(TM) i5-1035G7 CPU @ 1.20GHz, 1498 Mhz, 4 Core(s), 8 Logical Pr...

Figure 4 Physical Specifications

Installed Physical Memory (RAM)	8.00 GB
Total Physical Memory	7.60 GB

Figure 5 Memory Specification

As mentioned earlier, the UART data transfer happens at 115200 bps and the COM port needed for this code is found under the device manager ports with the name of XDS110 Class Application/User UART, the port number can be found next to this port name. It is necessary to have the correct port named in the python code or else the visualization will not work. After getting the data for the distance, the x and y coordinates are calculated and stored into an array and will continue to do so until all the points are measured. The z axis is an assumed value set by the user where they will have to physically move the system forward however far they want. The total amount of points to captured can be calculated by:

$$Total\ points = Points\ per\ plane * Number\ of\ planes$$

The formatting used to store this data is a xyz file. This file will only contain 3 columns, the first column is the x values, second column is the y values and the last column is the z column. This format is necessary to interact with the Open3D library. The figure below is an example of how this is formatted.

```
0.000000 180.000000 0.000000
-34.921168 175.560565 0.000000
-68.500334 165.374436 0.000000
-100.002642 149.664530 0.000000
-125.865007 125.865007 0.000000
-148.833061 99.447072 0.000000
-165.374436 68.500334 0.000000
-174.579780 34.726077 0.000000
-178.000000 0.000000 0.000000
```

Figure 6 XYZ file formatting

After formatting the file, the visualization process will start. To being with, a point cloud visual is created using the data in the xyz file which will provide a new window with the resulting visual. A point cloud image is a view of all the data points with their respective coordinates in an empty space. This helps the user get a general idea of where the measurements were taken with and get a better understanding of the axis portrayed in the visualization. An example of the point cloud graph can be found in figure x.

The next step in the visualization process is to get the lines connected to each point in the plane and connect each plane to each other. To achieve this, the program will first need to assign values to each point and link them together through an array where the current point will be incremented to connect to the point ahead in the file. For connecting the planes together, there will be a loop that links

the index of the point together with the point in the next plane by going to the current index added with the number of points the user wants in the plane. The program to link the points together can be found in figure 5.

```
xyplane = []
totalpts = numPts*10
for i in range(totalpts):
    xyplane.append([i])

lines = []

for j in range(1,11): #Goes to the next slice
    for i in range(numPts*(j-1),numPts*j-1):
        lines.append([xyplane[i],xyplane[i+1]])
    lines.append([xyplane[i],xyplane[(j-1)*numPts]])


for i in range(0,numPts*9):
    lines.append([xyplane[i], xyplane[i+numPts]])
```

Figure 7 Creating lines for visualization

With the connections created, to visualize it in Open3D, the visualization.draw_geometry method is used with the line_set which was created using geometry.LineSet method with the parameters being the data points and the indexes of what datapoint to connect to. With this the expected space that is recreated can be found in the figure below with the hole at the bottom representing the sensor location

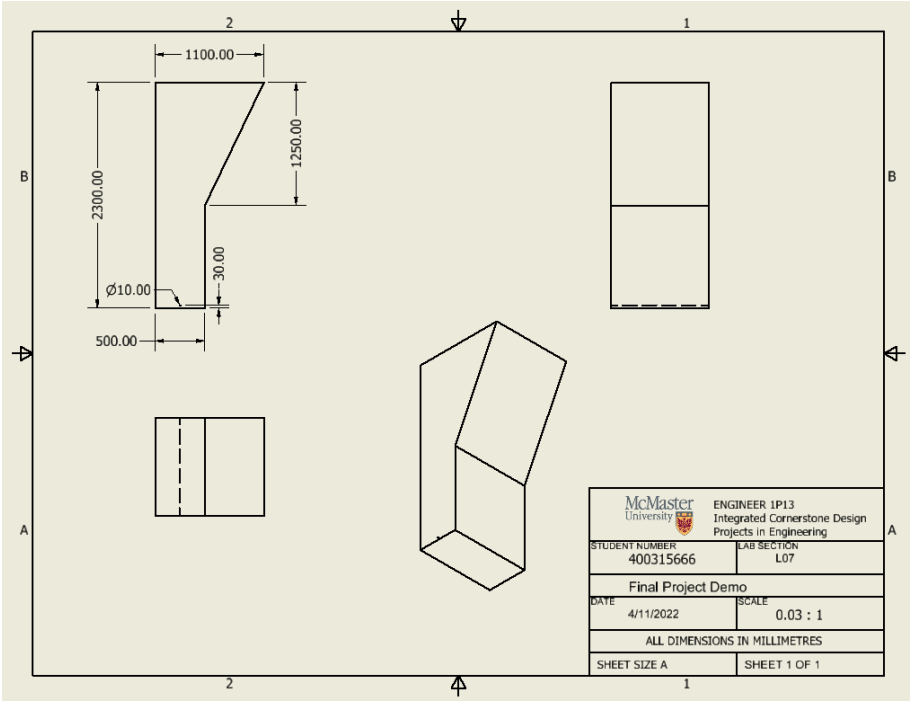

Figure 8 Rough Sketch of environment

Application Example

1. Find an environment for the system to start measuring. Can not be wider or taller than 4m from the sensor
2. Connect the Micro USB into the top of the micro controller which is the port opposite to the ethernet connector.
3. Open Keil with the project, adjust any of the clock speed and point number parameter as needed and when once finished hit the translate, build, and load button on the top left of the application.

Figure 9 Loading Code into the microcontroller

4. Once the code is loaded onto the microcontroller, hit the reset button next to the Micro USB connected to the board and wait for the LEDs to stop flashing when the reset button is hit
5. Open and run the python code named “MeasurementProcess” included via the python IDLE program for the respective version under 3.10
6. Adjust the COM# in the code for the user specific device. This can be found in device manager and under ports. The proper port number has the title of XDS110 Class Application/User UART

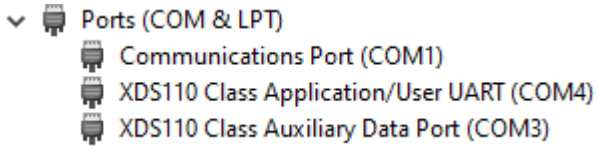

Figure 10 COM#

7. Once the program is run, it will prompt to press enter to start the program and once enter is pressed it is necessary to also press PJ1 on the microcontroller to start the measurement of the point
8. The program will take measurements with the counter at the right to count the amount of data points that has been measured. The motor will rotate at an angle that is equally separated from each other and the angle depends on how many data points the user desires.

```
Opening: COM3
Press Enter to start communication...
Distance measured: 488
 x:  488.0 y:  0.0 0
Distance measured: 225
 x:  223.9165635012443 y:  22.053856574151137 1
Distance measured: 151
 x:  148.0985773408878 y:  29.458638624435366 2
Distance measured: 123
 x:  117.70366129506168 y:  35.70501530229887 3
Distance measured: 100
 x:  92.38795325112868 y:  38.268343236508976 4
Distance measured: 88
 x:  77.60907126265525 y:  41.482912840687796 5
Distance measured: 90
 x:  74.83226510722908 y:  50.001320971764194 6
```

Figure 11 Expected measurement output

9. Once the data for the plane has been measured, the motor will reset back to its initial position, when that happens, move the system forward to the desired amount. In this case the amount moved forwards is 10 cm.
10. Repeat step 7-8 until all the points are measured and the COM closes
11. To visualize this data, open the python file “VisualizationProcess” with the same IDLE program mentioned in step 5 and run the module. The expected visualization point cloud visual should be first and to get a more detailed visualization, close the window of the point cloud visual. The expected results can be found in the figures below

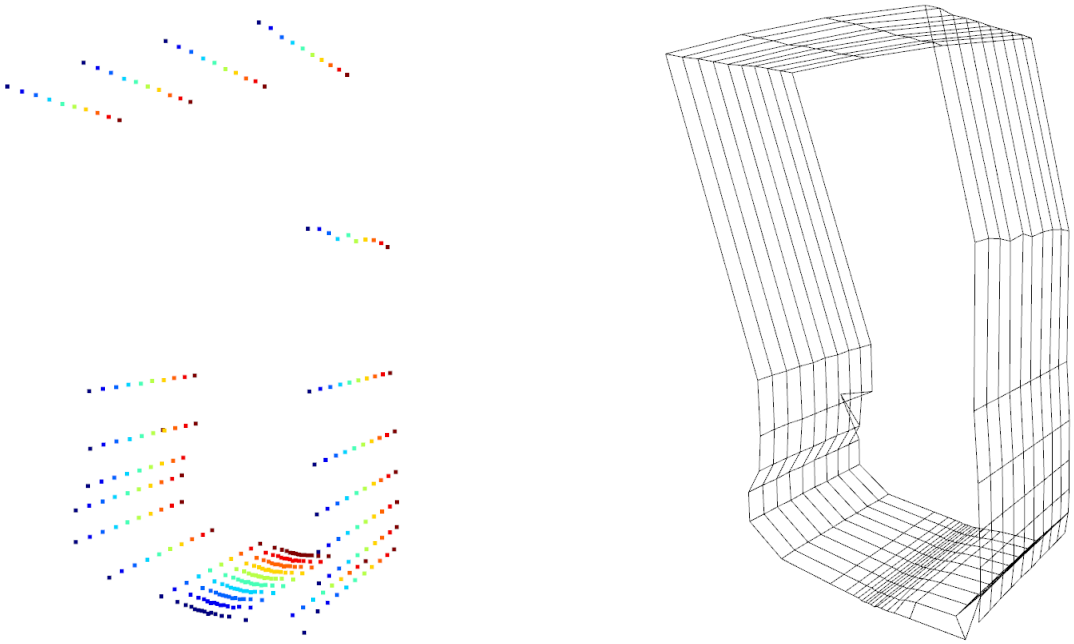

Figure 12 Point Cloud and Final Visual

Here the planes can be clearly visualized with the x axis being the horizontal and y being the vertical plane. The z axis represents the distance that the user has moved the system forwards.

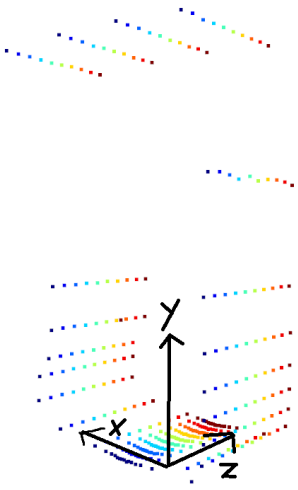

Figure 13 Axis of visualization

User Guide

This guide is to allow users to install the proper software and circuitry used in this system. The steps shown here based off the software that was personally used and can be substituted for some alternatives.

1. Keil Installation: https://www2.keil.com/mdk5
    a. This software is used to directly interact with the Microcontroller. The general installation procedure is simple, but the most important part is when the Pack Installer is prompted.
    b. When the Pack Installer menu shows up in the left side in the “Devices” tab search for the microcontroller this system uses and install the pack

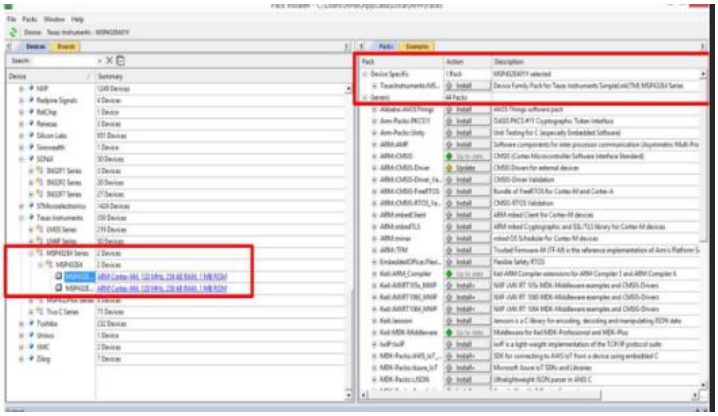

Figure 14 Keil Installation

2. Python Installation https://www.python.org/downloads/
    a. Use the link above to download python. This system should be compatible for any version before 3.10 and after 3.3. Once the desired version is selected at the bottom there will be downloads for the corresponding operating systems. Download the version that uses your operating system
    b. After the installation is done, run the program and follow the instructions prompted and add the python application to the path
    c. When python is finished installing it is necessary to do a restart on your computer. Afterwards open command prompt by searching it up in the search bar and type in the following commands to download the libraries used in the code.
        i. pip install pyserial
        ii. pip install open3d
3. Physical Hardware
    a. Make sure that all the hardware is present and connect the female-to-female wires to the corresponding ports shown in the circuit schematic
    b. Mount the motor using a method of your choice, a simple solution is using adhesive like hot glue or tape

Limitations

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

The limitations shown in the microcontroller is the accuracy of the data provided because the microcontroller will only send whole number measurements to the UART code which will create errors regarding the precision of the system. The floating-point unit in the processor is also another limitation because the binary representation of any decimal can go on for very long or is repetitive which causes the accuracy of decimal point to lower. This means the trigonometric functions that can very easily produce these repeating decimals will have lower overall accuracy when used in calculation along with other infinite decimal numbers. Another factor is also the wiring, the movement of wires with the motor can also cause the measurements to be less accurate which is not good for the system.

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

$$\text{Quantization Error} = \frac{V_{FS}}{\mathbf{2^8}} = \frac{3.3V}{2^8} = 0.0129V$$

The quantization error uses 8 bits because the I2C communicates through 8 bit packages so the system is dependent on the I2C communication protocol rather than the microprocessor.

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

The maximum baud rate for the UART communication was 115200 bps because the limiting factor was with the microcontroller. This can be verified with the data sheet of the microcontroller.

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

The ToF sensor uses I2C communication protocols and the relationship between the microcontroller and ToF is that the microcontroller is the leader and the ToF is the follower. According to the datasheet of the ToF sensor, the maximum speed for I2C communication is up to 400kbits/s.

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

The main limitation of this system is the stepper motor since it operates using delays to trigger steps to turn. Since it uses the delays, going too fast will lock the motor into place because the motor is receiving the inputs too fast. It also depends on the system clock speed, my system uses 80 MHz but it is possible to go up to 120MHz and change the delay to the very minimum for the motor to operate. This was tested by changing the clock speed to 120MHz and making a delay of around 5 milliseconds based off the clock speed. This change gave noticeable results with how fast the motor moves but creates a draw back of how much heat was created which melted the adhesive of the mounting system.

Circuit Schematic

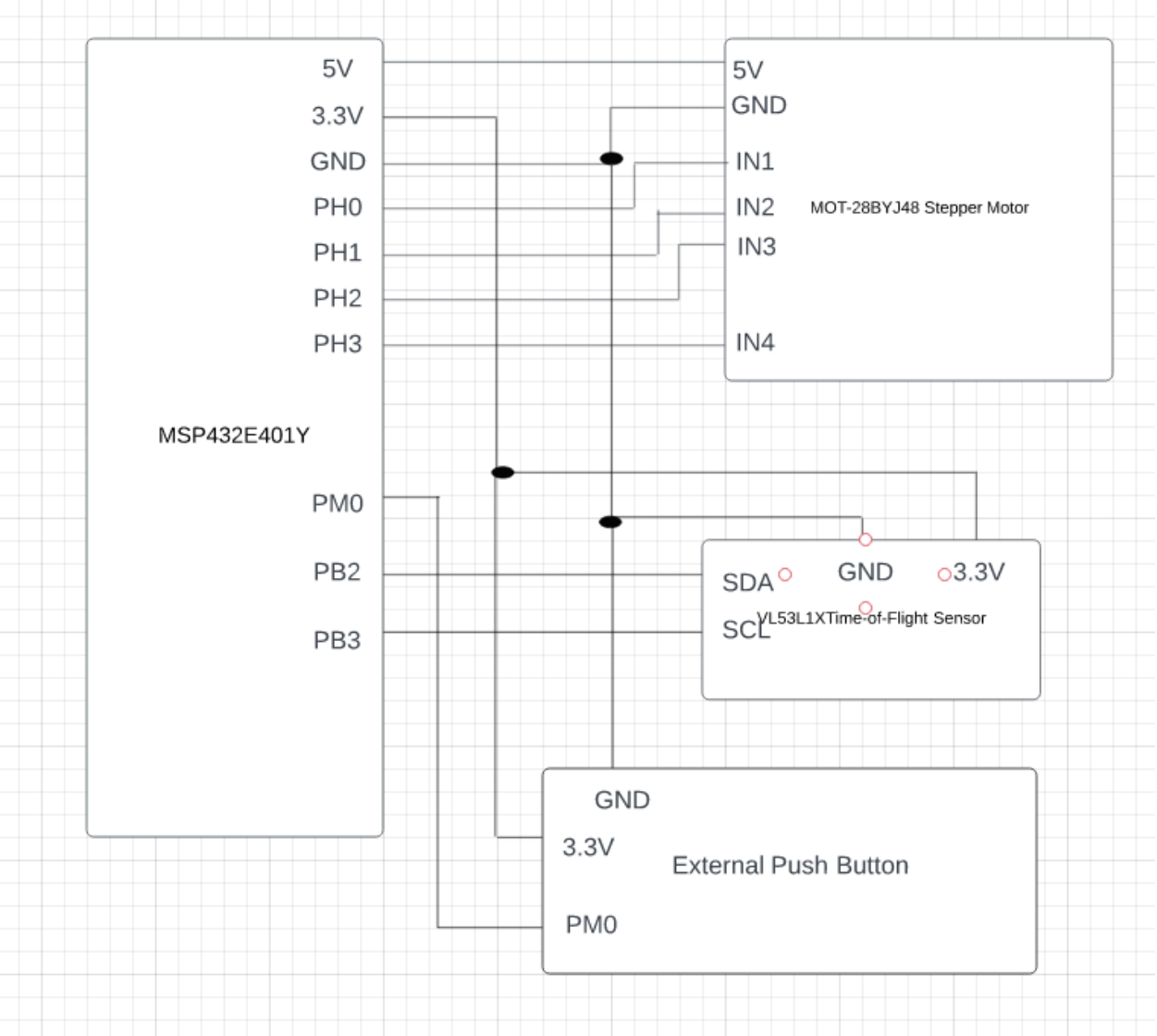

Figure 15 Circuit Schematic

Flowcharts

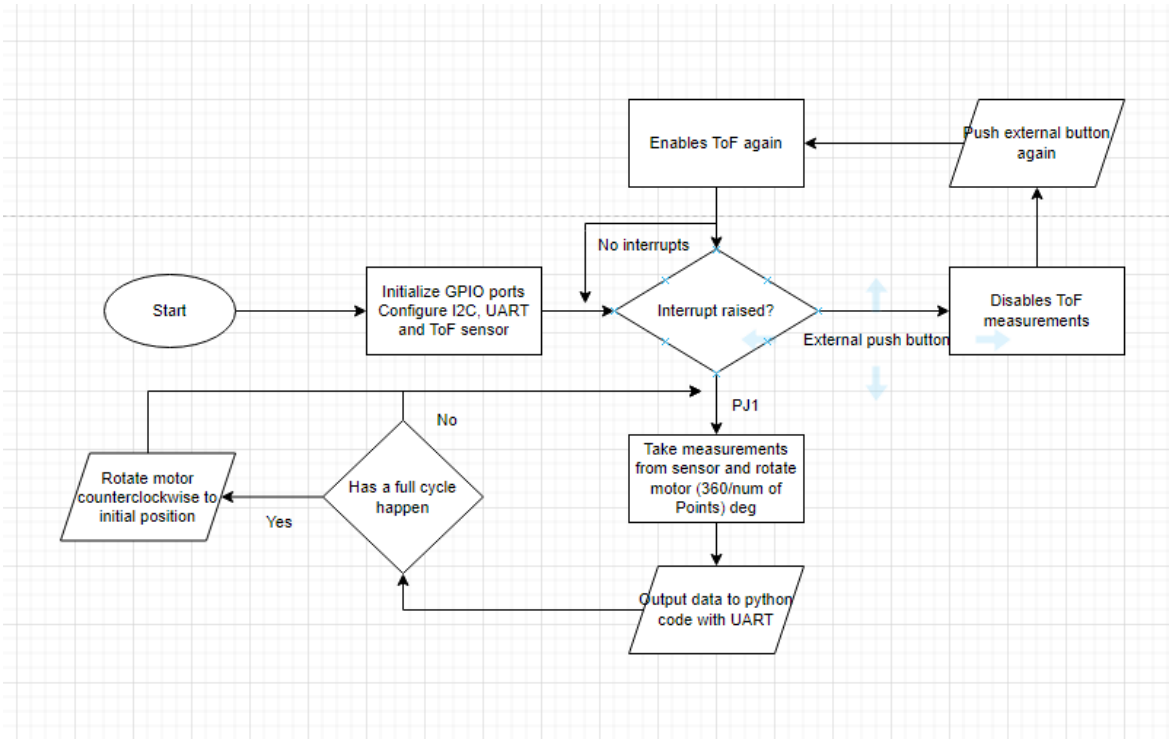

Figure 16 Flowchart for Microcontroller

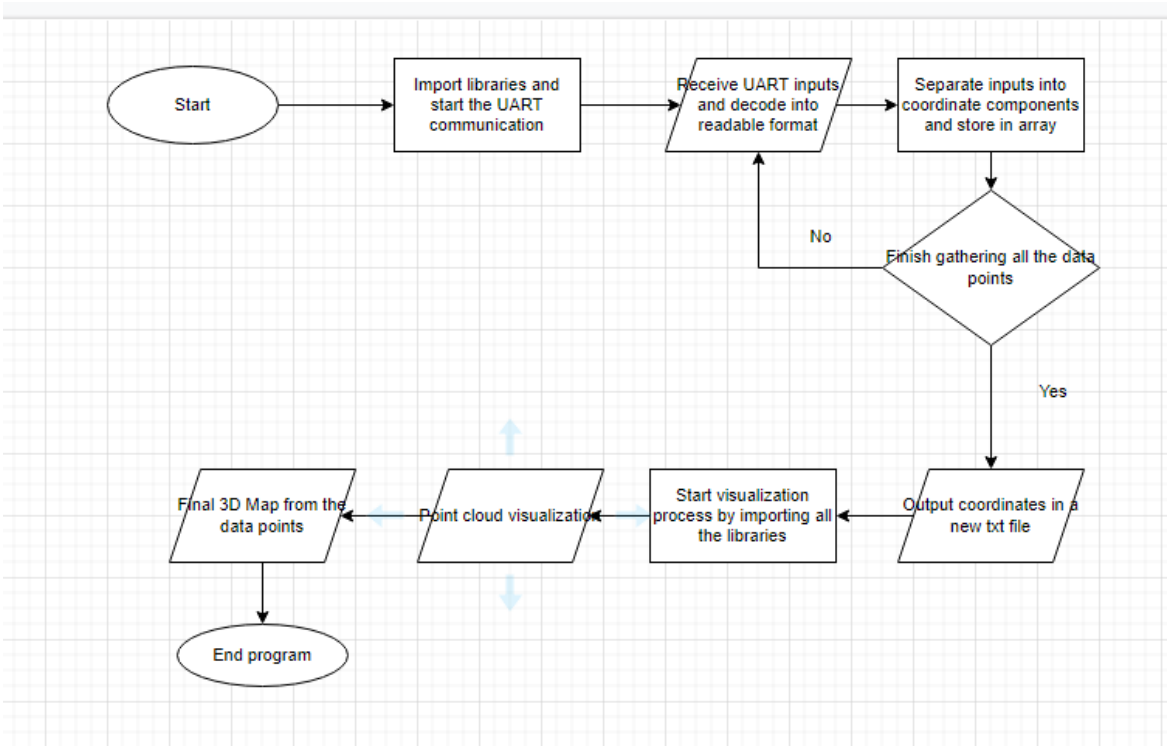

Figure 17 Flowchart for Python program