

2DX3: Microprocessor Systems

Final Project

Instructor: Drs. Boursalie, Doyle and Haddara

Kyle Stamp – stampk1 – 400291081 – COMPENG 2DX3 – Monday
Afternoon – L01

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 **[Kyle Stamp, stampk1, 400291081]**

Device Overview

Features

- VL53L1X time-of-flight sensor
- MSP432E401Y microcontroller
- 28BYJ-48 stepper motor
- External push button control
- 2x 10k Ω resistors
- 24 MHz bus speed
- 5V stepper motor operating voltage
- 3.3V time-of-flight sensor operating voltage
- I2C serial communication between time-of-flight sensor and microcontroller
- I2C in Fast Mode at 400 kbps
- UART serial communication between microcontroller and PC
- 115.2 kHz baud rate
- C microcontroller language
- Python User Interface
- Open3D 3D Visualization
- Total Cost: ~\$100

General Description

This system maps a space by taking distance measurements at varying angles on the xy axis to create slices, and the user moves along the z-axis. The output is a 3D rendering of the space. The system contains a microcontroller, stepper motor, time-of-flight sensor, push button and two 10k Ω resistors. The microcontroller controls the flow of the system. It is responsible for controlling the stepper motor, obtaining data from the time-of-flight sensor through I2C, and sending data to the PC for the user using UART. The stepper motor has the time-of-flight sensor attached to its shaft. It rotates on 22.5-degree intervals for a 360-degree rotation, with the time-of-flight sensor taking a distance measurement at the end of each interval. This is done by the sensor sending out pulses of light at 50Hz, measuring the time it takes to return. The obtained distance is in millimetres¹. The UART data is saved in Python, converted to cartesian coordinates using trigonometric functions and written to a file. This file is read, and the data is plotted as a point cloud and as a 3D rendering of the space.

Block Diagram

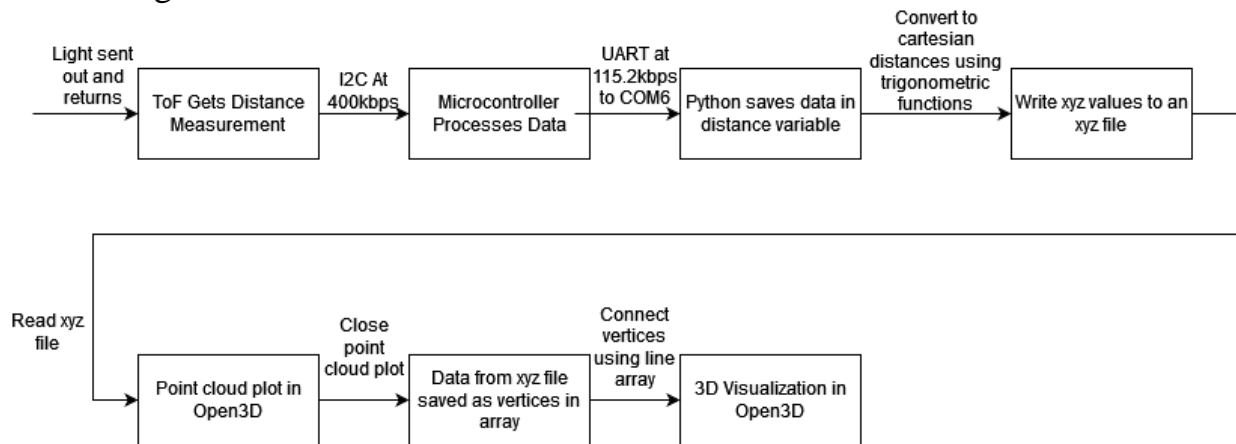


Figure 1: Block Diagram

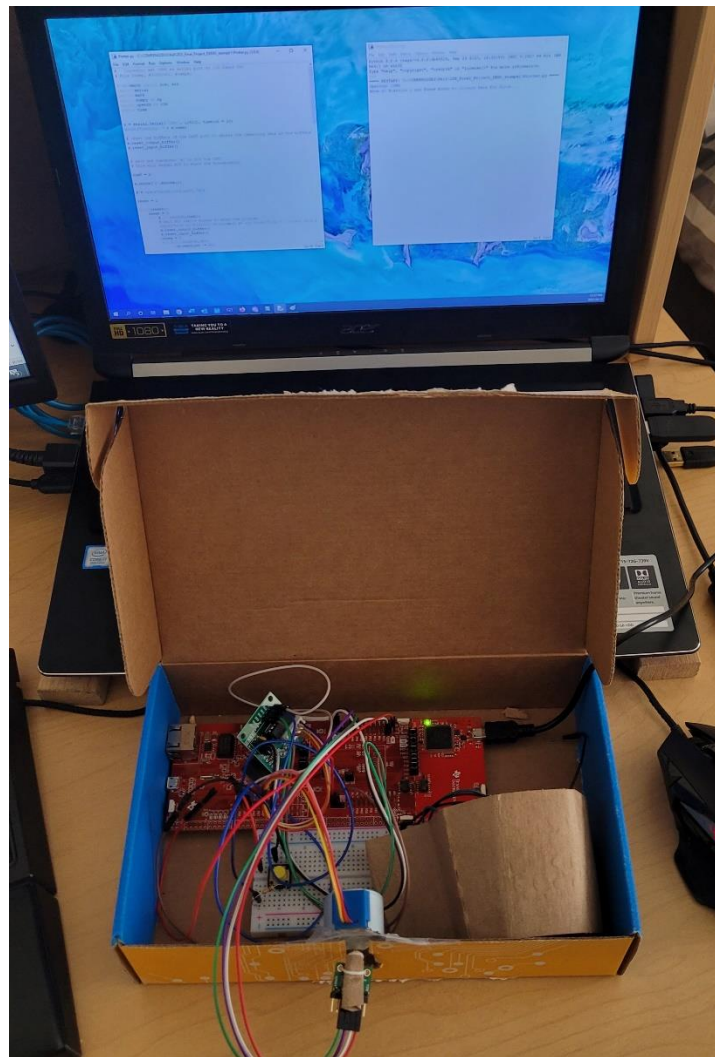


Figure 2: View of Whole System

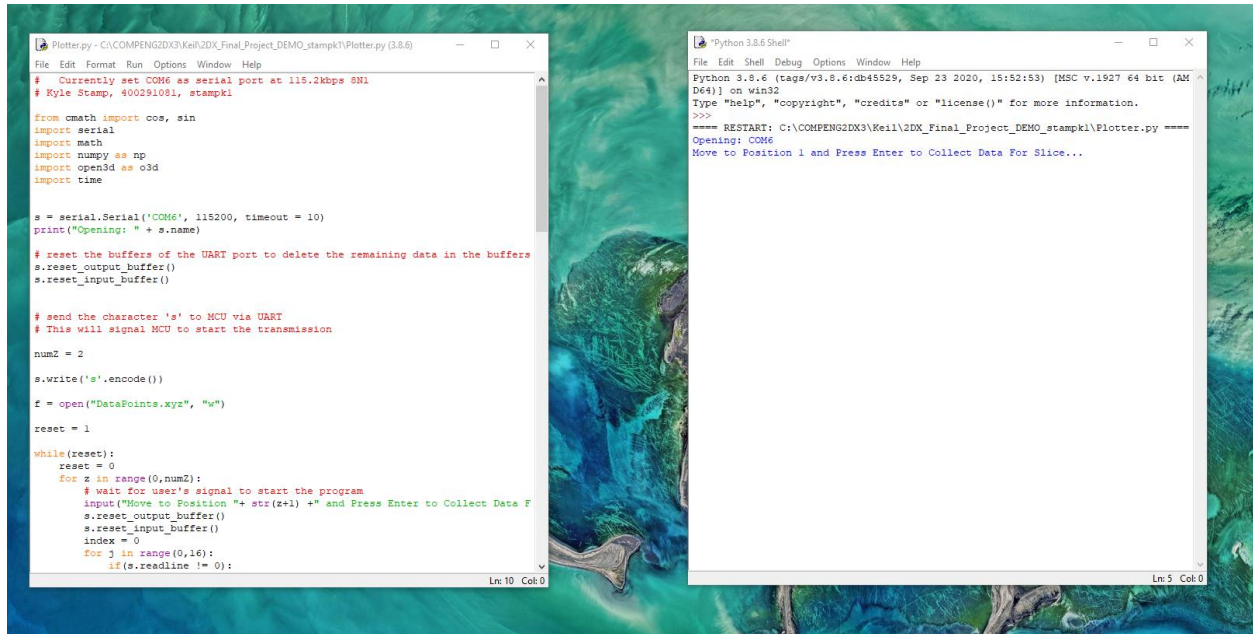


Figure 3: PC Display

Device Characteristics Table

Table 1: Device Characteristics

Stepper Motor Ports	IN1:4 to PH0:3
Push Button Port	PM0
Bus Speed	24 MHz
Serial Port	COM6
Baud Rate	115.2 kbps
I2C Rate	400 kbps
Python Libraries	Serial Numpy Open3D cmath
I2C Ports	SCL to PB2 SDA to PB3

Detailed Description

Distance Measurement

The whole system schematic can be seen in Figure 7. At the start of the microcontroller program, all required ports are initialized by setting the clock, direction, enabling, and setting any alternate functions. The time-of-flight sensor is setup using the ToF API functions. In this setup, the sensor amplifies, filters and level shift the ADC voltage to calibrate the sensor. The amplification allows the ADC to take advantage of the full range of digitization. Filtering removes noise to improve the accuracy of the digitized value. Level shifting shifts the voltage sine wave up or down to prevent clipping to retain the signals shape¹. 4 onboard LEDs flash to signal to the user when the device is finished setting up each stage, for a total of 3 times. Once the sensor is finished booting, the program enters an infinite loop that waits for the user to press the external push button. Upon being pressed, a zero value is sent to the microcontroller which sets a flag to prevent continuous execution due to a held button. As well, the sensor is set to start ranging and the program waits until the sensor is ready to take data using the SysTick_Wait function in a while loop. With a button press, the microcontroller controls the stepper motor, making it rotate for 360 degrees on 22.5-degree intervals. This is done by setting a bit corresponding to one of four coils in the stepper motor. By setting one at a time with a slight delay in between, the stepper motor can rotate one step. Once the motor has rotated the 22.5-degree interval, the time-of-flight sensor takes a distance measurement. This is done by sending out a pulse of light and measuring the time it takes to return to the sensor. It has a maximum distance of 4m in long distance mode, at the expense of some loss of accuracy due to light levels. This data is transmitted serially to the microcontroller via I2C in fast mode at a maximum speed of 400kbps. The data is then sent to the PC serially using UART communication, with a baud rate of 115.2kbps. The PC uses pyserial to connect with the microcontroller in Python, obtaining the data through COM6.

If at any point during the data collection process the external push button is pressed, the stepper motor stops rotating, and the program returns to the checking for the start button press. This check is polled each time the motor moves one step.

The input of the system is the external push button and keyboard on the PC. The push button enables the motor and sensor, allowing the data to be acquired, as well as stopping and resetting the system. The PC keyboard interacts with the Python shell to begin accepting data through UART. The outputs of the system are the onboard LEDs, the stepper motor, and the plots produced from Python. The onboard LEDs are a visual indicator that certain processes, such as the sensor booting, have finished to signal to the user that the device is ready for use. The stepper motor is an external device controlled by the microcontroller, which also provides a visual indication of the system performing correctly, as well as being essential to the system's function in collecting proper data. The Python plots in Open3D are the main outputs for this system. They are the end goal of the project and show the system has performed correctly. The logic for this system can be seen in Figures 8 and 9.

Visualization

To visualize the 3D space, Python 3.8 was used on a Windows 10 laptop with an Intel i7-8570H CPU and 16GB of DDR4 RAM. Python uses the pyserial library to communicate with the microcontroller, cmath for trigonometric conversion from radial to cartesian coordinates and numpy and open3d to create a point cloud and 3D rendering of the space. The entire Python program logic can be seen in Figure 8.

The distance data obtained from UART communication with the microcontroller is decoded and casted to an integer to convert the bytes to a value representing the radial distance in millimeters. The x and y axes are defined as the horizontal and vertical axis, with the z-axis being depth of the 3D space. So, each slice of data is on the xy axis, and these slices are created along the z axis. The collection process is looped 16 times to obtain 16 data points for a given z distance. This provides a good resolution for the final 3D visualization while keeping a low runtime. If the number of data points per slice were higher, the 3D visualization would be more accurate, but each 360-degree rotation of the stepper motor would take considerably more time, thus greatly increasing the runtime of the whole system. Depending on the number of desired slices stored in a variable called numZ, this loop for the data points is repeated numZ times. Within the inner data point loop, each time a measurement is sent, the value is converted to cartesian coordinates using the following formulae:

$$x = (((dist)) * \cos((j * 22.5) * \pi/180)) \quad (1)$$

where x is the x-axis distance in mm, dist is the radial distance in mm, j is a unitless counter to count each interval of 22.5 degrees.

$$y = (((dist)) * \sin((j * 22.5) * \pi/180)) \quad (2)$$

where y is the y-axis distance in mm, dist is the radial distance in mm, j is a unitless counter to count each interval of 22.5 degrees.

The z-axis is incremented by 1m at the end of the 360-degree motor rotation, at which point the user should move forward one meter. The real component of these 3 values is written to an xyz file called "DataPoints.xyz", discarding the imaginary portion. If at any point during the data collection process the user presses the push button again, a reset flag will be set. This closes and opens the file again to delete all data in the file. As well, the program will start from the beginning of the slice loop, essentially resetting the whole collection process. The python interface will begin prompting the user to press enter for the first position again. Finally, to visualize the obtained data, each line of the xyz file is read and plotted as a point cloud using Open3D. For the 3D shape, a vertex is defined for each data point ranging from 0 to the number of data points. These values are stored in an array. This array is traversed, creating a line

connecting each subsequent vertex for each slice on the xy plane and storing it in a line array. As well, each slice is traversed to create connections to the next adjacent vertex on the z-axis. Each xy slice is 16 points. For two slices, for example, point 2 will connect a line to point 1, point 3 and point 18, which is the second point in the next slice. Once all points have been connected correctly, a 3D rendering of the created shape is displayed using Open3D and the Python program ends. An example output 3D visualization of a hallway can be seen in Figure 4.

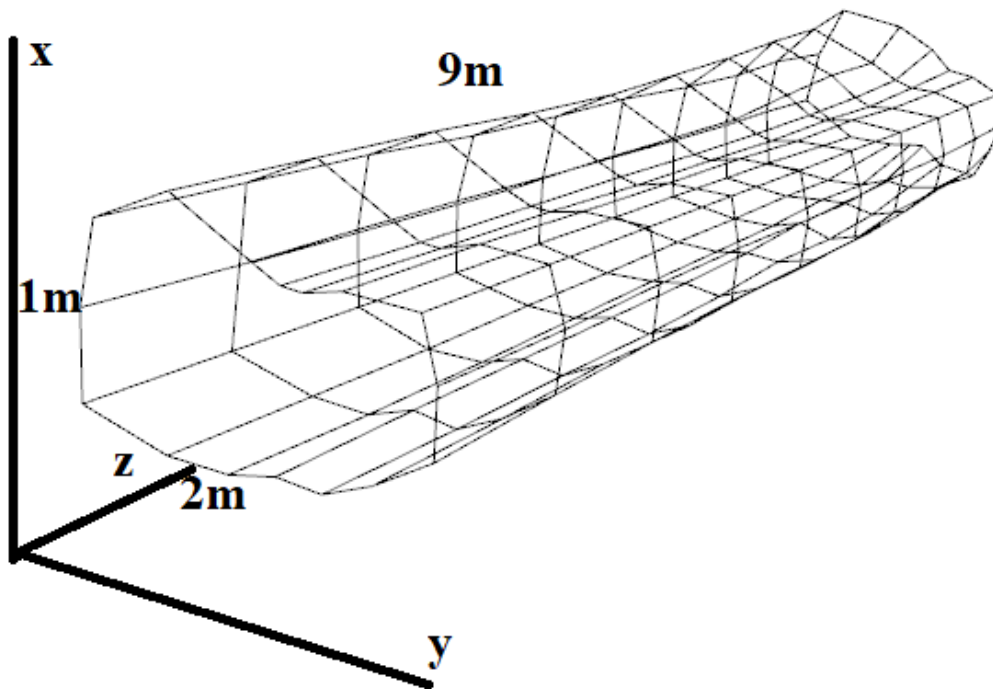


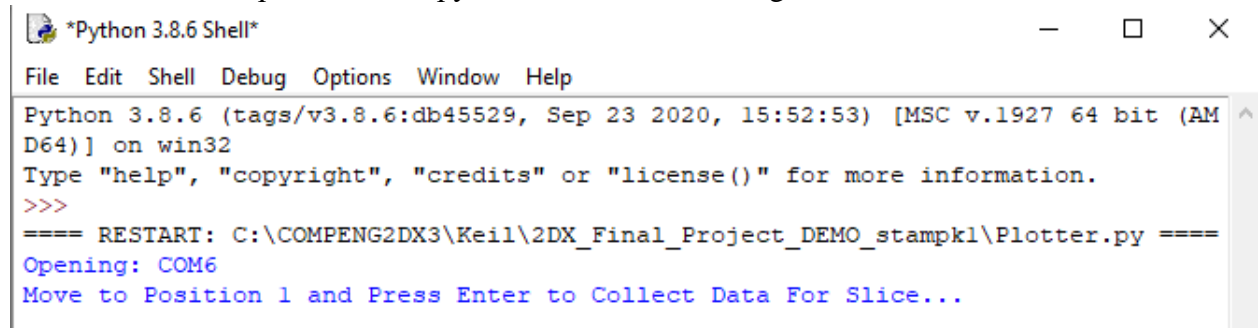
Figure 4: Hallway 3D Visualization

The 3D visualization in Figure 4 is rotated on its side for easier viewing.

Application Example with Expected Output

For this device, the x axis is defined as the width, the y-axis is defined as the height and the z-axis is defined as the depth.

1. Once the reset button on the microcontroller is pressed, the program begins
2. The time-of-flight sensor begins setup
3. The onboard LEDs all flash together 3 times to signal when each phase of the setup process has finished
4. When the push button is pressed, the sensor starts ranging and the program waits until the data is ready to be acquired
5. The function “spin” is called, which contains a for loop that executes 512 times to make the stepper motor rotate 360-degrees
6. Within the loop, a check is performed every 32 iterations of the loop to flash the 2nd onboard LED and take a distance measurement from the sensor using I2C
7. The distance value is sent serially via UART through COM6 to Python on the PC with pyserial.
8. In python, COM6 is opened with a 115.2kbps baud rate using pyserial
9. A file called “DataPoints.xyz” is opened in write mode to erase any data if the file already existed
10. A flag called reset is created and set to 1
11. A loop is entered that iterates for the number of specified slices of data called numZ, representing the z-axis value, which runs 10 times
12. The reset flag is set to 0 and the prompt “Move to position # and press enter to collect data for slice...” is printed to the python shell, as seen in Figure 5.



```
*Python 3.8.6 Shell*
File Edit Shell Debug Options Window Help
Python 3.8.6 (tags/v3.8.6:db45529, Sep 23 2020, 15:52:53) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
==== RESTART: C:\COMPENG2DX3\Keil\2DX_Final_Project_DEMO_stampk1\Plotter.py ====
Opening: COM6
Move to Position 1 and Press Enter to Collect Data For Slice...
```

Figure 5: Python Shell

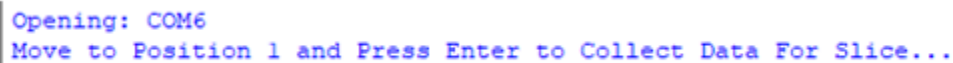
13. When the enter key is pressed, a for loop iterates 16 times
14. In the for loop, the distance value sent over UART is decoded, cast to an integer, and is saved in a variable called “dist”
15. A check is performed to check if dist is set to the value 4294967295, as this value is out of range of the possible distance measurement range.
16. If the value is obtained, the file is closed and reopened to reset all data in the file, the reset flag is set to 1, the message “Stopping...” is printed to the shell and the inner for loop is exited

17. If dist is some other value, it is printed to the shell, a variable x is set to the x-axis component of the distance using Equation #, a variable y is set to the y-axis component of the distance using Equation #, and the x, y, and z values are printed to the file
18. If the reset flag is 1, the z-axis for loop is exited and the program goes back to step 11
19. The z-axis value increments by 1 to represent 1 metre and the steps 12 through 18 are repeated
20. Port M0 is checked for a button press
21. If a button press occurs, a -1 is sent via UART to the PC, which results in the value 4294967295, and the program exits the for loop
22. If no button press occurs during the spin function, the for loop continues iterating the same process 512 times
23. Outside the for loop, the 2nd onboard LED is flashed and one more measurement is taken for the 0-degree angle
24. The distance measurement is sent serially via UART through COM6 to Python on the PC with pyserial.
25. The function reversespin is called to spin 360-degrees counterclockwise to untangle the time-of-flight sensor's wiring
26. The program goes back to checking for a button press and the flags for a button press and stopping are reset
27. Once data for all slices has been obtained, the python program opens a point cloud, with 10 sets of 16 coloured dots, representing each data point
28. Closing the point cloud plot opens a 3d shape, with 10 slices, each with 16 vertices

User's Guide

1. Install Python 3.8 from official Python website
2. Once installed, go to advanced system settings, environment variables, and edit path by adding a path to the python installation location and the python script's location. This can be done by opening the install location and copying the path at the top of the file explorer window. Save these changes.
3. Go in command prompt and type "python" to check if these changes worked. The output should be "Python 3.8".
4. Install the pyserial package in the command prompt using the command "pip install pyserial"
5. Install the numpy package in the command prompt using the command "pip install numpy"
6. Install the Open3D package in the command prompt using the command "pip install open3d"
7. The command prompt may need to be opened and closed between steps 3 to 6 if the install does not work
8. Open the Plotter.py file
9. Plug microcontroller into a USB port on PC

10. Open device manager and check if port is COM6
11. Alternatively, use the command “Python -m serial.tools.list_ports -v” in command prompt to see all ports in use
12. Keep changing ports until the microcontroller is in COM6
13. If COM6 is not an available port on the PC, change COM6 in the line “s = serial.Serial('COM6', 115200, timeout = 10)” to whichever port the microcontroller is plugged into
14. Set baud rate in this same line to 115.2kbps
15. Change numZ variable to desired length of space to map in metres
16. Place or hold device at one end of desired 3d space
17. Press reset button on microcontroller
18. Wait for all 4 onboard LEDs to flash together 3 times so the device can properly boot up
19. Press Enter key on keyboard in Python when prompted to begin waiting for data, then quickly after press external push button. This prompt can be seen in Figure 6.



```
Opening: COM6
Move to Position 1 and Press Enter to Collect Data For Slice...
```

Figure 6: User Prompt in Python Shell

20. Wait for stepper motor to spin fully clockwise and counterclockwise
21. Move 1m forward and press the Enter key again to prompt in Python, quickly after pressing the external push button
22. Repeat process for the previously specified times in the variable numZ
23. A point cloud plot will open on screen
24. Press the red x on the point cloud plot window to open the 3d mapping of the room

Limitations

- 1) The microcontroller and the trigonometric functions used in python use floating point arithmetic². This form of calculation has less accuracy than the use of fractions. Due to the nature of decimals, if the result has infinite decimals in floating point form, the number of decimals will need to end at some point. Thus, when this occurs, the number is rounded meaning the result is an approximation and not an exact value. Depending on how little digits the value gets rounded to, the result will be less and less accurate. The more digits kept, the more computation power and time is needed³.
- 2) The max quantization error is equal to the ADC resolution. The ToF operates at 3.3V and in long distance mode can take measurements up to 4m, meaning 12 bits are stored¹.

$$\Delta = \frac{V_{FS}}{2^m}$$

$$\Delta = \frac{3.3}{2^{12}}$$

$$\Delta = 0.00081$$

- 3) The maximum standard serial communication rate is 115.2kbps. This rate is the current rate used in the UART communication. To test for the max, the next highest rate was used, being 128kbps. Upon using this rate in Python, data was not being transmitted from the microcontroller to the PC but does transmit using 115.2kbps.
- 4) To communicate between the time-of-flight sensor modules and the microcontroller, I2C was used. The sensor communicates in fast mode at 400kbps¹.
- 5) Reviewing the whole system, the primary limitation on speed is the time-of-flight sensor. The stepper motor was set step with a very low delay of 8ms per step, meaning rotating did not take a considerable amount of time, even when rotating fully counterclockwise to untangle wires. The time-of-flight sensor requires time to bootup when the device starts up, a delay to prepare for data acquisition, and a delay when transmitting and receiving data to and from the microcontroller. This delay is very noticeable when collecting data, as the motor takes considerable time to pause and wait for the data collection from the sensor. When trying to optimize this performance by removing or lowering these delays, the UART began transmitting zeroes or random values, indicating the time-of-flight sensor is incapable of performing any faster. This is especially important depending on the resolution of the 3D mapping. If there are lots of data points per slice, for example over 20, the program will take a very long time to create the 3D rendering. This means there is a tradeoff between resolution and speed of the program, being limited by the time-of-flight sensor.

Circuit Schematic

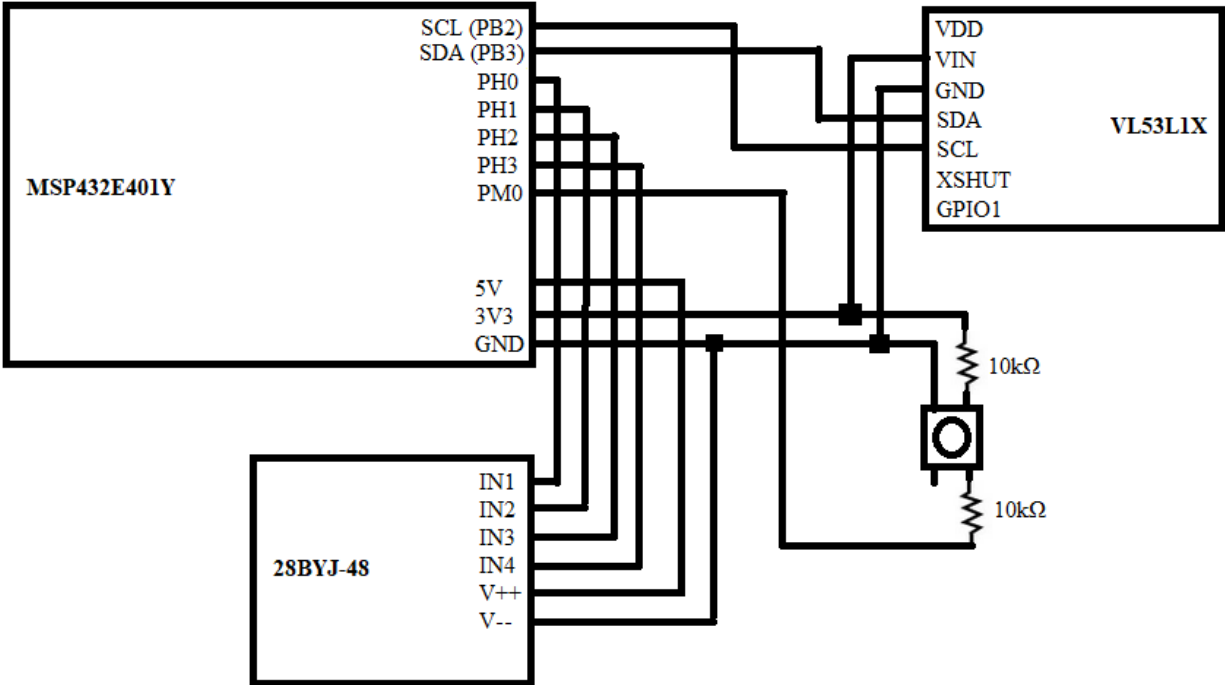


Figure 7: System Circuit Schematic

Programming Logic Flowcharts

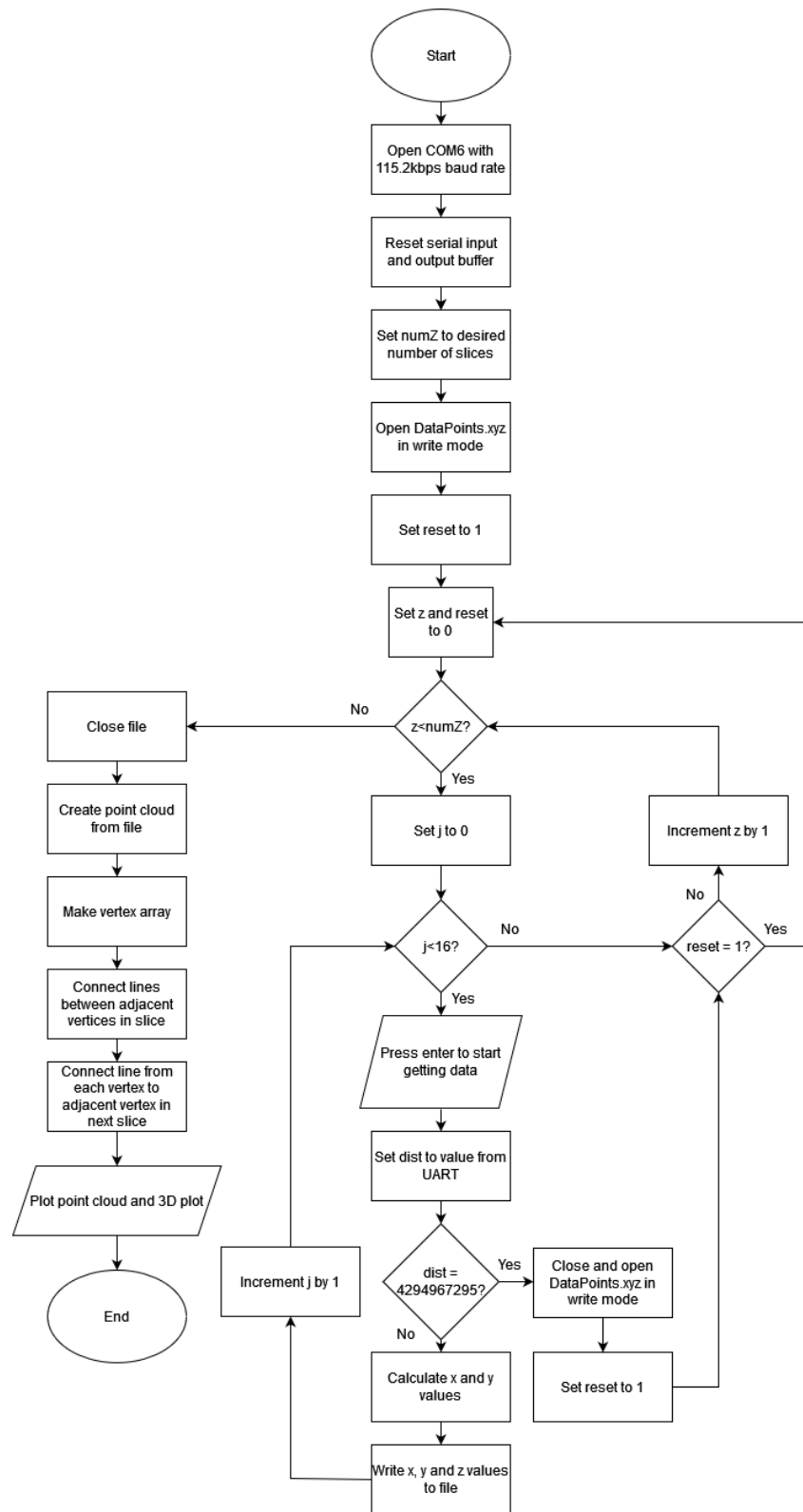


Figure 8: Python Logic Flowchart

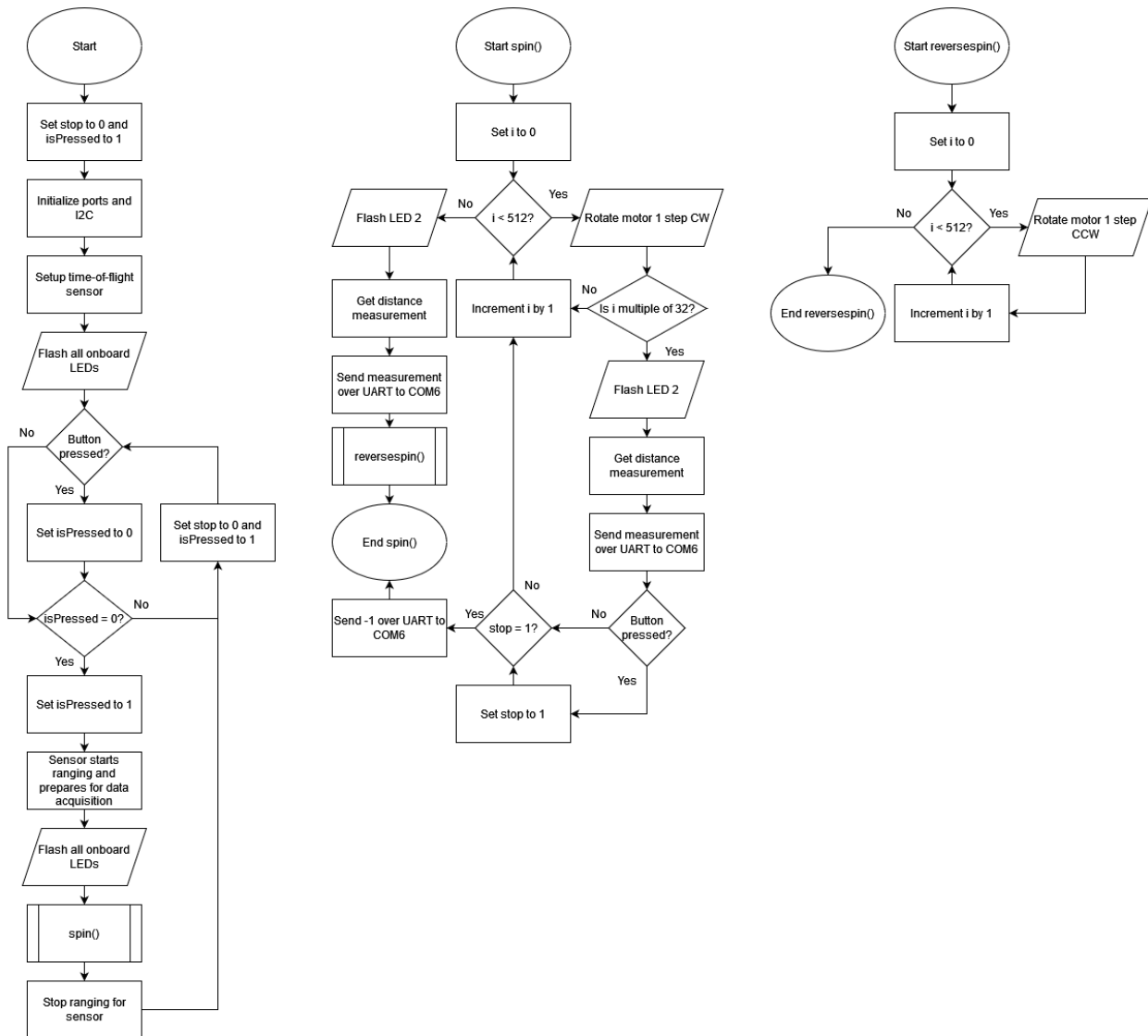


Figure 9: Microcontroller Logic Flowchart

References

- [1] “VL53L1X” Project and Related for COMPENG 2DX3, Department of Engineering, McMaster University, Winter 2022. [Accessed: 10-Apr-2022].
- [2] “MSP432E4 SimpleLink Microcontrollers - Technical Reference Manual”, Microcontroller Resources for COMPENG 2DX3, Department of Engineering, McMaster University, Winter 2022. [Accessed: 10-Apr-2022].
- [3] “Floating Point Arithmetic: Issues and Limitations”, *Python*. [Online]. Available: <https://docs.python.org/3/tutorial/floatingpoint.html>. [Accessed: 10-Apr-2022].