COMPENG 2DX3
Final Project Report

Manan Dua – L01 – 400519918 – duam4

Instructors: Dr. Thomas Doyle, Dr. Yaser M. Haddara, Dr. Shahrukh Athar

Date: April 9, 2025

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 **[Manan Dua, duam4, 400519918]**

1. Device Description: High-Accuracy Mapping Lidar (HAML)

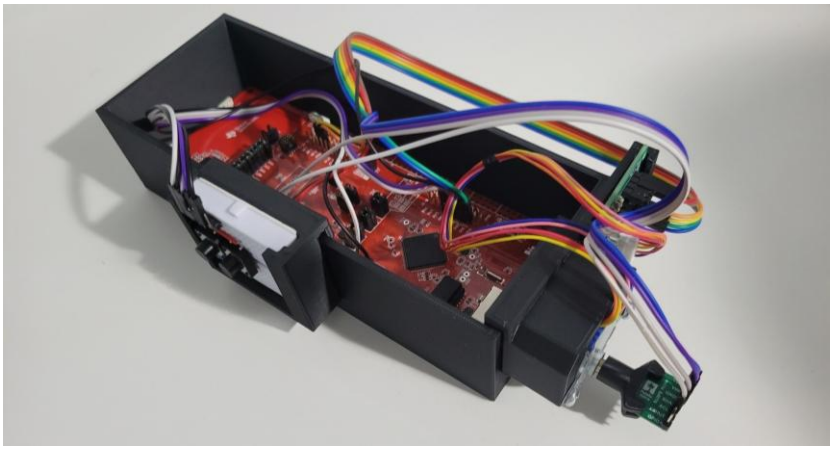

Figure 1: Fully Assembled Lidar Sensor

1.1 Features

Core

- MSP432E401Y microcontroller with programmable GPIO pins
- Operating with a clock frequency of 26 MHz
- Requires 17kB of memory for the program

User Interface

- Single push-button interface for device operation
- Additional button for homing function

Measurement Capabilities

- 360° room measurement capability
- VL53L1X Time-of-Flight (ToF) sensor takes distance readings every 5.625°
- Maximum measurable distance: 4 meters

Motor Control

- Utilizes the 28BYJ-48 Stepper Motor in Full-Step mode for maximum torque
- Controlled via ULN2003 motor driver board

Power

- The device runs on 5V DC from the onboard micro-USB port
- The VL53L1X operates at 3.3V provided from the microcontroller

Connectivity

- I²C integration between the microcontroller and Time-of-Flight sensor
- UART communication between the microcontroller and computer
- Communication through the micro-USB port on the device
- Communicates with a Python script on the receiver device
- Baud rate: 115200 bps

Indicators

- Status LEDs indicatin0.19g:
  - Data measurement in progress
  - UART communication status
  - Failed measurement indication

Data Processing & Visualization

- Point cloud conversion using Python 3.12
- 3D mapping and visualization using Open3D in Python

Cost

- Total cost of parts needed is around $105
  - MSP432E401Y - $75
  - VL53L1X – $15
  - 28BYJ-48 and ULN2003 - $10
  - 3D printed parts - $5

1.2 General Description

The High-Accuracy Mapping Lidar (HAML) is a device that is capable of mapping out a 3D model of indoor areas within 4 meters of the device. The device runs on the MSP432E401Y microcontroller, which runs on the Arm® Cortex®-M4F Processor Core with Floating-Point Unit. The device utilizes the VL53L1X Time of Flight sensor, the 28BYJ-48 Stepper Motor, and the ULN2003 motor driver board. The devices are connected using a custom 3D printed chassis. The MSP432E401Y serves as the leader in this circuit, whereas the VL53L1X is the follower. The device takes measurements by reading distance and range status values from the time-of-flight sensor to ensure a valid measurement was obtained. The device then rotates the stepper motor, which the time-of-flight sensor is connected to, by 5.625° and takes a new measurement, taking 64 measurements per run. This process is repeated until the device has made a full 360° rotation. Signals that are obtained by the VL53L1X are processed and converted to a digital signal using the onboard analog-to-digital converter (ADC). This data is then sent to the microcontroller using the I^2C communication protocol. The microcontroller saves this data into an array, where it can be later transmitted to a computer. To transmit this data to a computer, the user must initiate a Python program which sends a signal to the Microcontroller using UART, signalling it to transmit the data, which the microcontroller does though UART in 8-bit packets at a baud rate of 115200 bits per second. The Python program collects this data and manipulates it to form a cartesian plane in an XYZ file to plot using Open3D. This is done using trigonometric functions to determine the X, Y, and Z coordinates at which each measurement would be. This data is then processed in Open3D to plot and connect each of the coordinates to form a 3D plane of the measured area.

1.3 Block Diagram

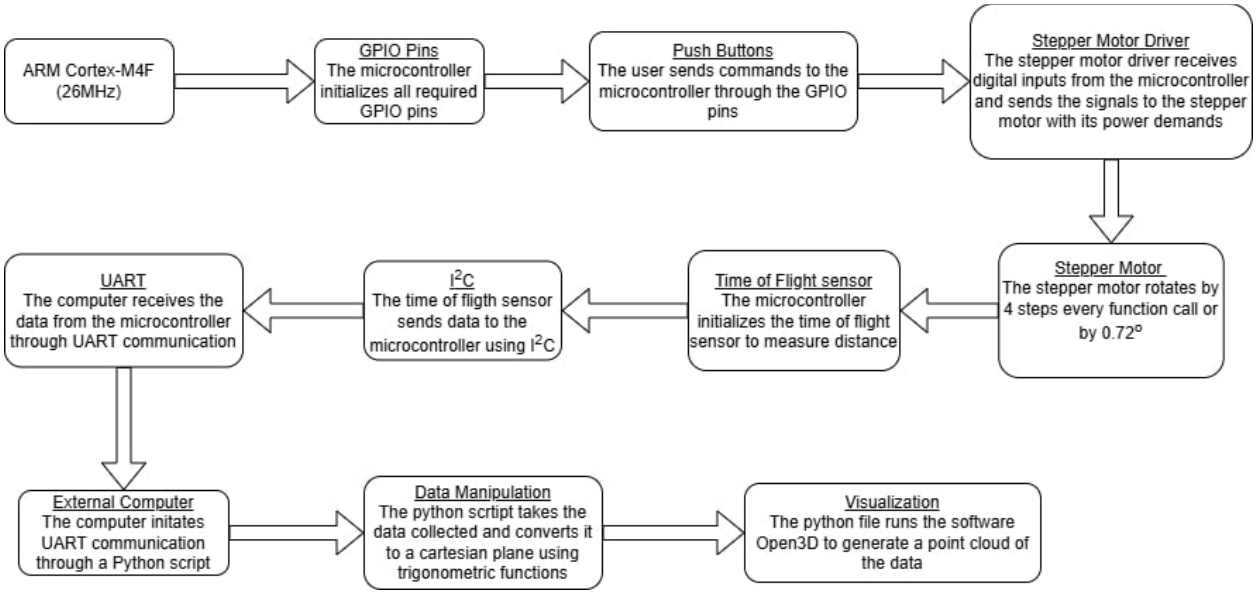

Figure 2: Block Diagram showing data flow

2. Device Characteristics Table

Table 1: Device Characteristic Table

Characteristic	**Detail**
Microcontroller bus speed	26MHz
SCL (ToF pin)	Mono-direction pin on the time-of-flight sensor used to synchronize the clock between the microcontroller and the time-of-flight sensor for data transfer. Connected to **PB2** on the microcontroller.
SDA (ToF pin)	Bi-directional data pin on the time-of-flight sensor to send and receive data. Connected to **PB3** on the microcontroller.
Vin (ToF power)	3.3V
GND (ToF power)	GND
IN1 (Motor driver input)	Connected to **PH0** on the microcontroller
IN2 (Motor driver input)	Connected to **PH1** on the microcontroller
IN3 (Motor driver input)	Connected to **PH2** on the microcontroller
IN4 (Motor driver input)	Connected to **PH3** on the microcontroller
V^+ (Motor driver power)	5V
V^- (Motor driver power)	GND
Push Button 1	External active low momentary push button connected to **PM0**. Used to initiate scan.
Push Button 2	External active low momentary push button connected to **PM1**. Used to return motor to home location.
Power LED	Onboard LED D0 connected to **PN1**
UART Indication LED	Onboard LED D1 connected to **PN0**
Failed Measurement LED	Onboard LED D2 connected to **PF4**
UART Baud rate	115200 bps
Python Version	3.12.0
Open3D Version	0.19.0
Serial port used	COM6

3. Detailed Description

3.1 Distance Measurement

This device uses the VL53L1 time-of-flight sensor to obtain distance measurements. The VL53L1X is a high-precision sensor, capable of measuring distances up to 4 meters with millimeter accuracy. It operates based on the ToF principle, where an infrared laser pulse is emitted, and the time taken for the reflected light to return to the sensor is measured. This process allows the sensor to determine the distance to the target accurately.

The VL53L1X features a Vertical-Cavity Surface-Emitting Laser (VCSEL) that emits an infrared laser beam at a wavelength of 940 nm. This wavelength is invisible to the human eye and is designed to minimize interference from ambient light sources.

After the emitted laser pulse strikes a target, it reflects toward the sensor. The VL53L1X uses a Single Photon Avalanche Diode (SPAD) array to detect the returning photons. SPADs are highly sensitive photodetectors capable of capturing extremely weak light signals, making them ideal for precise ToF measurements.

The sensor's internal processing unit measures the time interval between the emission and detection of the laser pulse. Since the speed of light is a known constant (approximately 3.0×10^8 m/s), the distance to the target can be calculated using the formula:

$$Distance = \frac{Photon\ Travel\ Time}{2} * speed\ of\ light$$

$$Example:\ Distance = \frac{20ns}{2} * \left(3 * 10^8 \frac{m}{s}\right) = 3m = 3000mm$$

The division by two accounts for the round-trip nature of the measurement, as the laser pulse travels to the target and back to the sensor. The distance the sensor returns is in the units of millimetres. The ToF sensor is connected to the microcontroller through an I2C interface for data transmission. The main C code, uploaded to the microcontroller via Keil software, includes API commands that facilitate communication with the ToF sensor. During startup, the microcontroller configures the necessary ports for button inputs, the stepper motor driver, onboard LEDs, and the I2C communication required for the TOF sensor. To begin the scan, the ToF sensor is pointing down at -90°. The program then enters a polling system to monitor whether the button has been pressed. Once this condition is met, execution proceeds to the spin() function. This function advances the stepper motor by four steps per call and triggers a scan using the printTOF() function every 32 steps, corresponding to an interval of 5.625°. Since a full 360° rotation requires 2048 steps, the system performs 64 measurements (360° ÷ 5.625° = 64) by stopping and scanning at every 32-step interval (2048 ÷ 64 = 32). This is managed by checking the step count using a modulo operation. Once the system completes 2048 steps, it calls the spin() function again, but with a parameter in indicate spinning in the opposite direction, which reverses the motor to its initial position at -90°. The system is programmed to take measurements in 300mm intervals, meaning each slice the device captures must be 300mm from the previous scan in the +X direction.

These 64 distance measurements form a radial scan around the sensor's starting position. Since each measurement has a known angle and distance, we use basic trigonometry to convert polar coordinates into Y and Z displacements relative to the sensor's origin. This allows the system to calculate not just the absolute distances to surrounding objects but also their spatial positions, effectively mapping displacement in two dimensions. By combining angle, distance, and step count, the project visualizes how far and in what direction objects are located from the sensor.

```
void printTOF(){
   uint16_t Distance;
   uint8_t dataReady;
  uint8_t RangeStatus;
  while (dataReady == 0){
     status = VL53L1X_CheckForDataReady(dev, &dataReady);
         VL53L1_WaitMs(dev, 5);
   }
   dataReady = 0;

   //read the data values from ToF sensor
   status = VL53L1X_GetRangeStatus(dev, &RangeStatus);
   int i = 0;
   for(i = 0; i < 5; i++){
     if(RangeStatus==0) break;
     status = VL53L1X_ClearInterrupt(dev); //clear interrupt has to be called to enable next interrupt
     status = VL53L1X_GetRangeStatus(dev, &RangeStatus);
     FlashLED2(1);
   }
   if (i == 5){
     if(runs >= 1){ //if a data point on the previous run exists
       temp[var] = data_array[var*runs];
     }
     else if(var >= 1){ //if a prevoid data point exists
       temp[var] = temp[var-1];
     }
     else{ // if very first data point
       status = VL53L1X_GetDistance(dev, &Distance);         //The Measured Distance value
       status = VL53L1X_ClearInterrupt(dev); //clear interrupt has to be called to enable next interrupt
       temp[var] = Distance;
     }
     var++;
     return;
   }
   status = VL53L1X_GetDistance(dev, &Distance);         //The Measured Distance value
   status = VL53L1X_ClearInterrupt(dev); //clear interrupt has to be called to enable next interrupt
   temp[var] = Distance;
   var++;
}
```

Figure 3: Code of the printTOF() function in c

When the printTOF() function is called, it obtains the distance and range status of the ToF sensor. The program will ensure the values obtained by the ToF sensor are valid before saving them. If they are not valid, it will check up to 5 times. If it is still invalid, it will handle the failed reading by doing the following:

- If a previous run exists, set the current data point to that of the previous run
- If a previous value in the same run exists, set it to that value
- If no previous run exists and no value before this data point, use the value obtained

The device is also equipped with a homing function that, when pressed while the device is taking a scan, will discard all the measurements found during that scan and return the sensor to its original starting position. This is done by saving the measurement to a temporary array, and only when the device has finished the full scan will it save that data to the main data array, otherwise, it will be overwritten by the next scan.

```
if(total_rot >= 360 || total_rot <= -360){ // if a full scan has been complete
  pwr = 0;
  direc ^= 1;
  GPIO_PORTN_DATA_R &= ~(0b00000010); //indicate no scan is taking place
  while(total_rot!=0){
    spin(direc, 0); // return the sensor to its original starting position
  }
  direc ^= 1;
  for(int i = data_points*runs, j = 0; j < data_points; i++, j++){
    data_array[i] = temp[j]; // save all data points from the temp array into data_array
  }
  runs++; // increase the number of runs
  var = 0;
}
```

Figure 4: Code of the homing function of the device

3.2 Visualization

The next part is the visualization of the data that was collected, which is done through a Python script, O3DCreateData.py, at the end of the data collection. To complete the needed data manipulation, the python script imports numpy, open3D, serial, and math libraries. The python file can be executed anytime when the user is finished colling data, without the need for human interaction with the device or microcontroller. Before running the Python script, the user must ensure that the communication port the Python file communicates with matches that of the UART communication port of the microcontroller. This can be confirmed by opening the device manager on a Windows machine and going to the "Ports" tab and looking for the device called "UART," and ensuring this communication port is the same as the port in the Python script. In our case, the code is designed to communicate with COM6.

When the Python script is executed, the microcontroller first transmits the number of data points per run and the total number of runs. This allows the script to determine the expected data size when receiving values via UART and to automatically compute the trigonometric values needed to convert the values into a cartesian plane, regardless of the user-defined settings for data points per run and the number of runs completed. Using trigonometric functions, it was found that the Z-coordinate will use the cosine of the angle the ToF sensor is measuring at, and the Y-coordinate will use the sine of the angle the ToF sensor is measuring at. The X-coordinate only increments by 300mm after every run, meaning the sensor must be moved 300mm in the +X direction to get an accurate result. The ToF sensor starts the measurement pointing downward, so the starting angle of the sensor is -90º. This value is then used to calculate the Y and Z values and is incremented after every calculation by a factor of 360/(number of data points). This angle gets reset after all the values of every run are converted to a cartesian plane.

```
s.write('s'.encode())
data_array = []
x = s.readline()
data_points = int(x.decode())
print("Data points: " + str(data_points))

x = s.readline()
runs = int(x.decode())
print("Runs: " + str(runs))


for i in range(data_points*runs):
    x = s.readline()
    data_array.append(int(x.decode()))
```

Figure 5: Python code obtaining data through UART

```
x = []
y = []
z = []
for j in range(runs):
    deg = float(-90)
    increment = float(360/data_points)
    for i in range(data_points):
        z.append(data_array[i+j*data_points]*math.cos((math.pi/180)*deg))
        y.append(data_array[i+j*data_points]*math.sin((math.pi/180)*deg))
        x.append(j*300)
        deg+=increment
```

Figure 6: Python code transforming data into a cartesian plane

Once data collection is complete and the values are converted into Cartesian coordinates, the Python script automatically launches the Open3D open-source software to generate a 3D visualization of the point cloud. The software first plots the raw point cloud, followed by a second visualization where the points are connected with lines to form a structured 3D model of the scanned location.

4. Application Note, Instructions, and Expected Output

4.1 Application

This LiDAR-based spatial scanning system is adaptable, with numerous applications, as it can be used across many industries for planning, analysis, and automation. Its ability to produce correct environmental scans makes it suitable for use in construction, renovation, security, robotics, and automation. By scanning indoor spaces such as rooms, corridors, stairs, and lifts, a system allows the identification of structural defects, the estimation of repairs required, and pre-designing the improvement. The above information can thereafter be utilized by engineers and architects to optimize the use of space, improve building design, and enhance safety levels.

Apart from interior mapping, this system can also be integrated into various applications that require basic environmental scanning but do not require high-depth perception. In the automotive industry, for example, LiDAR technology is utilized for parking assistance and obstacle detection, enabling vehicles to navigate through tight spaces safely. Similarly, in security systems, LiDAR-based sensors can scan environments for intrusions, triggering alarms or locking devices based on pre-defined parameters.

This technology also applies to robotics, as the navigation systems navigate using spatial data. Robots used in warehouses, for example, can use the same kind of scanning technology to drive over obstacles, go along mapped routes, and optimize motion in small areas. In home automation, this system might be utilized to assist with automation, like adaptive lighting or climate control as a matter of room occupancy and furniture arrangement.

While the system has broad applications, its optimal application remains indoor scanning due to its limited range and susceptibility to ambient light. However, with the development of sensor technology and software processing, future iterations of this system could enhance outdoor capabilities and expand its application in more complex environments. Despite its limitations, the system is a low-cost, effective, and versatile tool for a range of spatial analysis applications, offering value in commercial and research environments.

4.2 Instructions

The following instructions guide the user through setting up the device to scan a designated area. These steps assume that the user has all the necessary hardware, including the microcontroller, motor, motor driver, time-of-flight sensor, jumper cables, and other required components. Additionally, the user must ensure that the appropriate software is installed, including Python 3.12, Open3D version 0.19, the numpy, math, and serial libraries, and Keil software to upload the file **duam4 – 2DX3 Final Project Code.c** to the MSP microcontroller.

1) The first step is to configure the circuit wiring, referring to both the Device Characteristics Table and the Circuit Schematic provided below for guidance. The final setup should resemble the images shown.

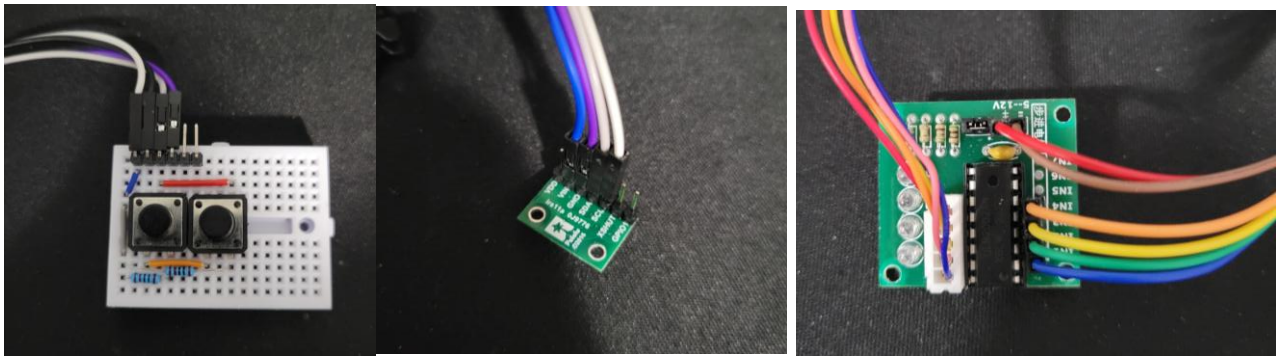

Figure 7: Images of the connections to all the components

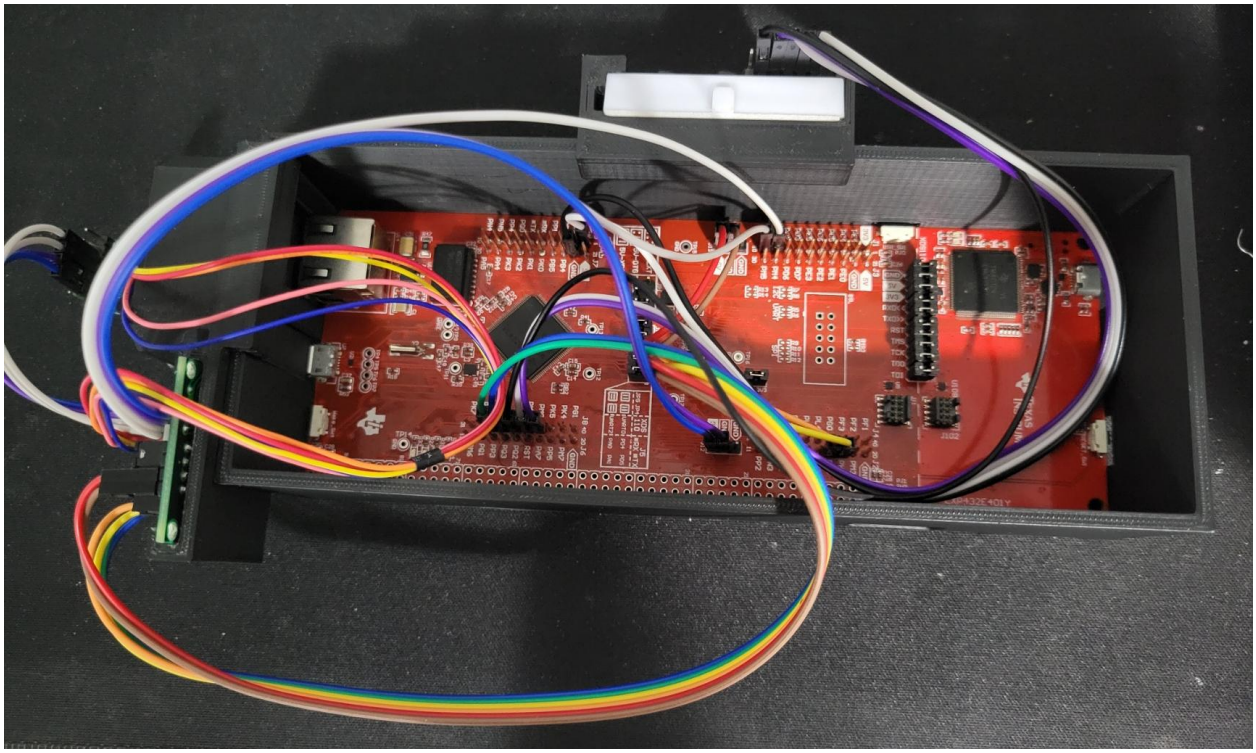

Figure 8: Image showing the connection of all devices in the device casing

2) After completing the circuit connections, the next step is to place the microcontroller and its components inside an enclosure. In this example, a custom 3D-printed chassis is used, but a cardboard box can also serve as an alternative. Secure the microcontroller inside the enclosure and mount the stepper motor externally using screws or glue. Attach the time-of-flight (ToF) sensor to the stepper motor shaft, ensuring it is properly aligned and unobstructed. The ToF sensor should be pointing down to begin the measurement. Additionally, position the buttons so they remain easily accessible.

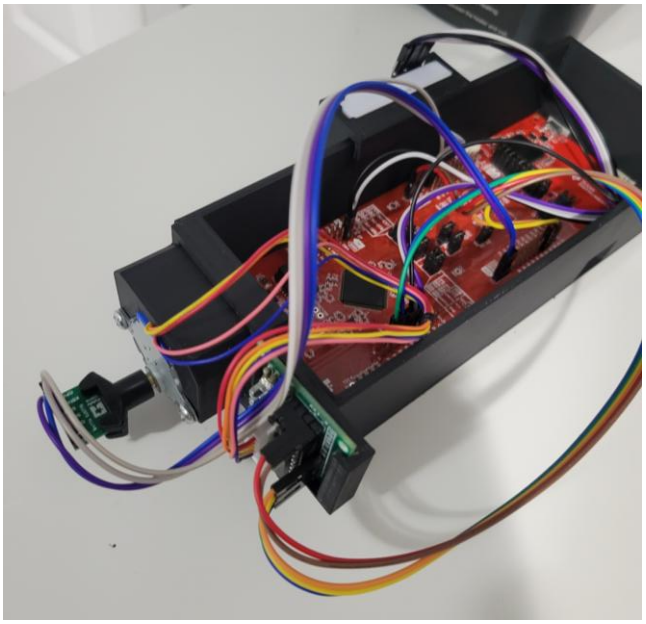

Figure 9: Image of the device in its enclosure with the initial sensor position

3) Your system should now be properly set up, and you can connect the microcontroller to your PC. Once powered on, the microcontroller's power LED will turn on. Next, open the Device Manager on your Windows computer to identify the COM port to which the device is connected under the "Ports" tab. It should appear under the name "XDS110 Class Application/User UART (COM6)," though the COM number may vary. Be sure to take note of the assigned COM port, as it will be needed later.
4) Next, upload the Keil code by opening the project and clicking the Translate, Build, and Load buttons in the top-left corner. Once the code has been successfully uploaded, press the reset button on the microcontroller to finalize the setup.
5) Next, update the Python script with the appropriate COM port at the top of the program on line 15.
6) Return to the hardware and place the device in the desired scanning location. Press the first button to start the scan, then wait for the device to complete a full rotation and automatically return to its home position. The device uses the YZ-plane for vertical distance slices and the X-plane for displacement.
7) After the scan is complete, move the device 300mm along the +X-axis and press the first button again to start the next scan.
8) If the user wishes to redo a scan at any point, simply press the second button during the scan. This will return the motor to the home position and discard any values recorded in that run.
9) Once all values have been collected, run the Python script **O3DCreateData.py** to retrieve data from the microcontroller via UART. The processing time will vary depending on the number of completed runs. Do not terminate the program until it has finished running. After the data is collected, the program will generate a 3D graph displaying only the point clouds, followed by a fully simulated 3D model with connected lines.

4.3 Expected Output

After going through the instructions and getting the device working, you can proceed to test it at various locations. For this example, the system was tested at location I, which is the second floor of the John Hodgins Engineering Building (JHE). The expected output of the scan based on this hallway would be similar to how the hall looks, which can be seen below:

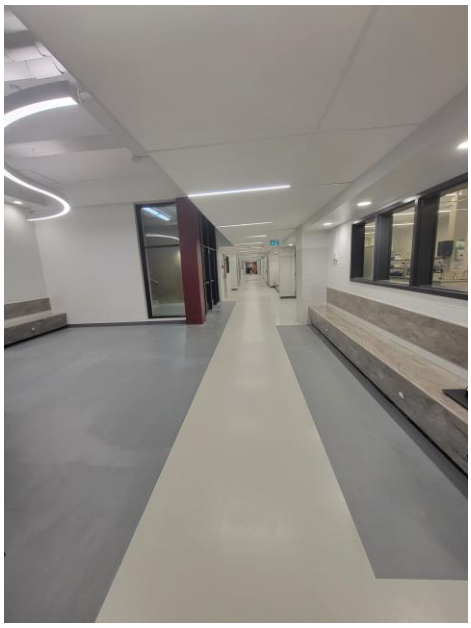

Figure 10: Image of the scanned hallway

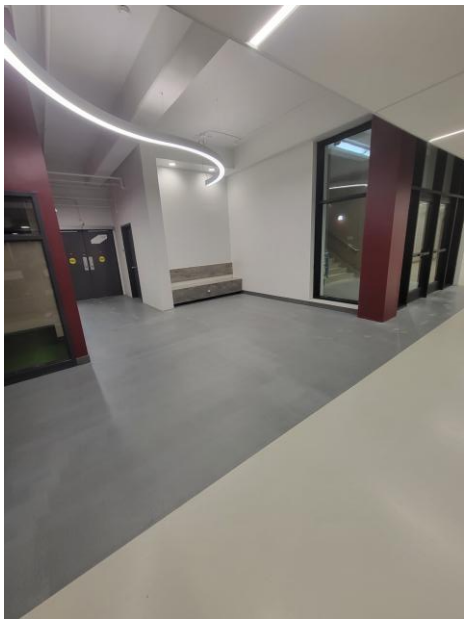

Figure 11: image of unscannable portion of the hallway

At the start of the scan area on the left-hand side, there is a long hallway exceeding 4 meters in length. Since this distance is larger than the maximum range covered by the TOF sensor, the system cannot record the reflected light in time. Therefore, it uses its error-handling capability to fill the gap in the data. Thus, the system successfully generates an output without gaps by utilizing values from the past scans to complete the unknown area.

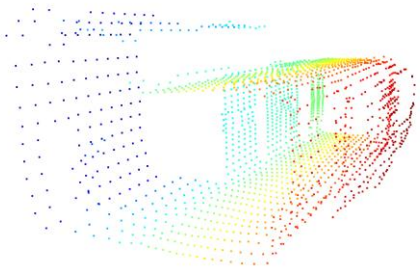

Figure 12: Open 3D Data cloud of the points scanned

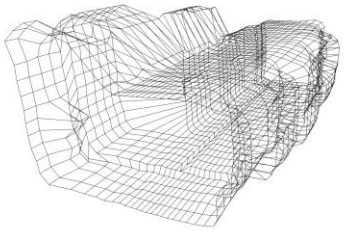

Figure 13: Open3D Data cloud with connected points

As shown above, the scan output closely matches the expected result based on the image of the hallway. It's important to note that the device was positioned near the center of the hallway to ensure even distribution of data points. However, some minor inaccuracies were observed, particularly in areas with bright lighting, uneven surfaces, or transparent materials such as glass.

5. Limitations

The MSP432E401Y microcontroller uses an ARM Cortex-M4F core, which includes a Floating-Point Unit (FPU). It features 32 single-precision 32-bit registers for floating-point operations. However, performing trigonometric calculations directly on the microcontroller is challenging as it requires specialized libraries. Additionally, it is limited in speed and precision, especially when performing repeated or complex floating-point calculations, such as those involving trigonometric functions. The C code has 32-bit precision when computing floating numbers, whereas Python has 64-bit precision, making it more accurate. To optimize performance, these computations are instead handled in Python, using the processing power of a PC.

The microcontroller's program stores data in an integer array of size 2048, allowing for 32 scans with 64 data points per scan. While this provides a substantial storage capacity, it may not be suitable for scanning very large areas. Simply increasing the array size would demand more memory, potentially impacting the microcontroller's performance. Additionally, the device has a fixed memory limit of 1024KB, which constrains the amount of data that can be stored.

The time-of-flight (ToF) sensor features a built-in ADC (Analog-to-Digital Converter) that converts analog signals into digital data, allowing the microcontroller to process the measurements. However, the accuracy of this conversion is limited by the ADC's resolution, which results in a quantization error of approximately 1 mm, matching the sensor's resolution.

$$Maximum\ Quantization\ Error\ =\ Resolution\ =\ 1mm$$

Additionally, the serial communication rate was set to 115200 bps, as this speed was tested multiple times with the microcontroller and verified using RealTerm to ensure the output matched the expected baud rate. However, the maximum baud rate a PC can support is 128000 bps, which can be confirmed by checking the port settings in Device Manager under the device properties.

I^2C was used to communicate between the ToF sensor and the microcontroller, which was limited to a clock speed of 100kbps.

Another limitation is the device's bus speed, which affects how quickly the program executes and collects data. In this case, the bus frequency is set to 26 MHz, though the microcontroller is capable of running at speeds up to 120 MHz.

$$Bus\ Frequency = \frac{\left(\frac{\frac{fXTAL}{Q+1}}{N+1}\right)*\left(MINT + \left(\frac{MFRAC}{1{,}024}\right)\right)}{PSYDIV+1} = \frac{\left(\frac{\frac{25000000}{0+1}}{24+1}\right)*\left(104 + \left(\frac{0}{1{,}024}\right)\right)}{3+1} = 26MHz$$

The primary limitation on the system's speed was the stepper motor's rotation rate. When the delay between steps was too short, the motor would occasionally fail to rotate altogether. At higher speeds, the system also produced incorrect range status values. However, by manually testing different delays and observing the serial output, a stable and reliable step delay was found to be around 2ms, which produced consistent accurate scans. Additionally, the ToF sensor had its limitations. Its maximum range restricted performance in larger or more complex environments, and it often struggled to produce accurate readings on surfaces like glass or those with uneven textures. This can be seen in the example provided in the expected results. However, from manual tests, it was found that the sensor tends to read these values more accurately when closer to the object. However, this may not always be a good solution if the area is very large, and the device needs to be placed in the center of the room to obtain all the details of the area.

6. Circuit Schematic

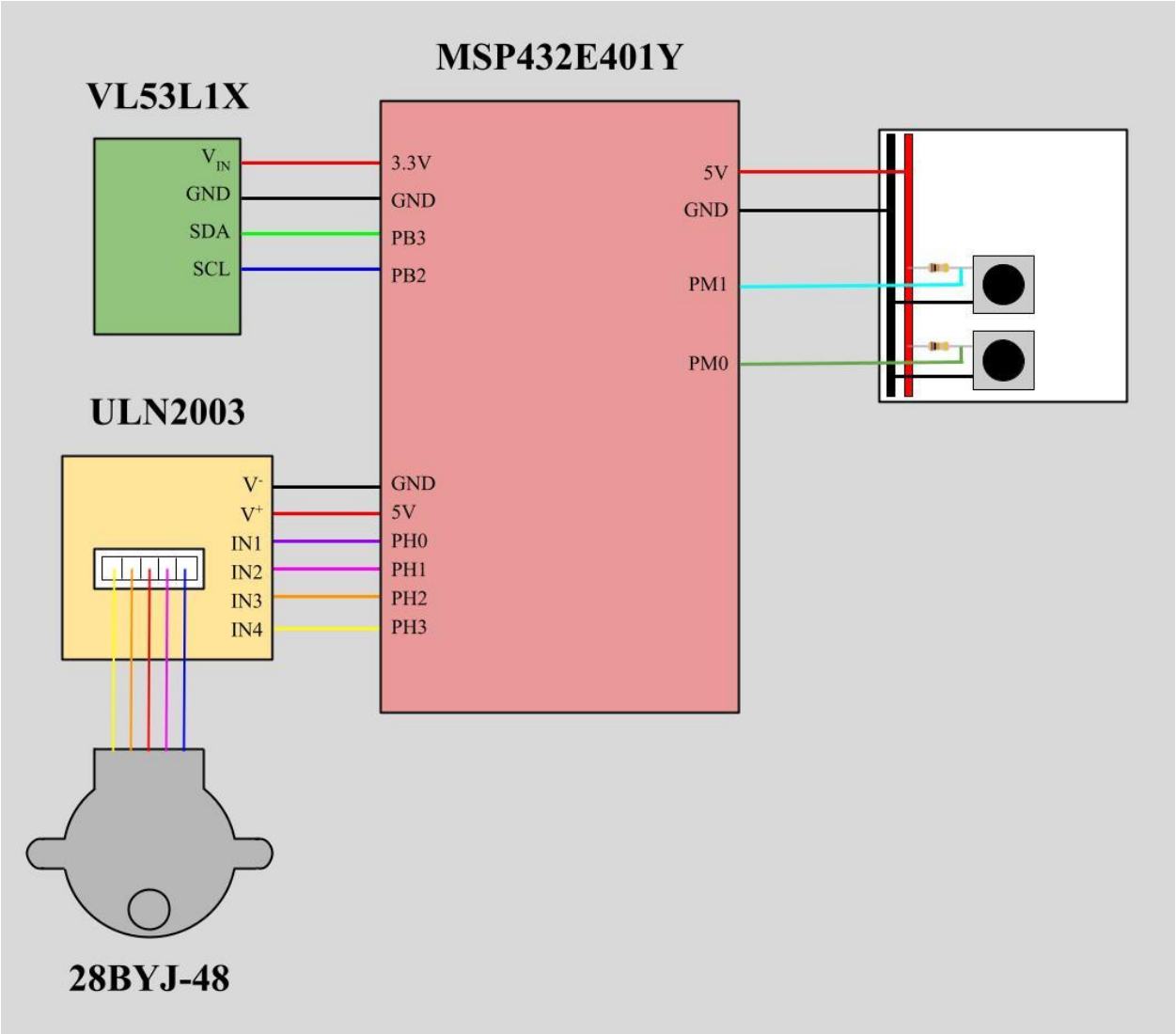

Figure 14: Circuit Schematic

7. Programming Logic Flowcharts

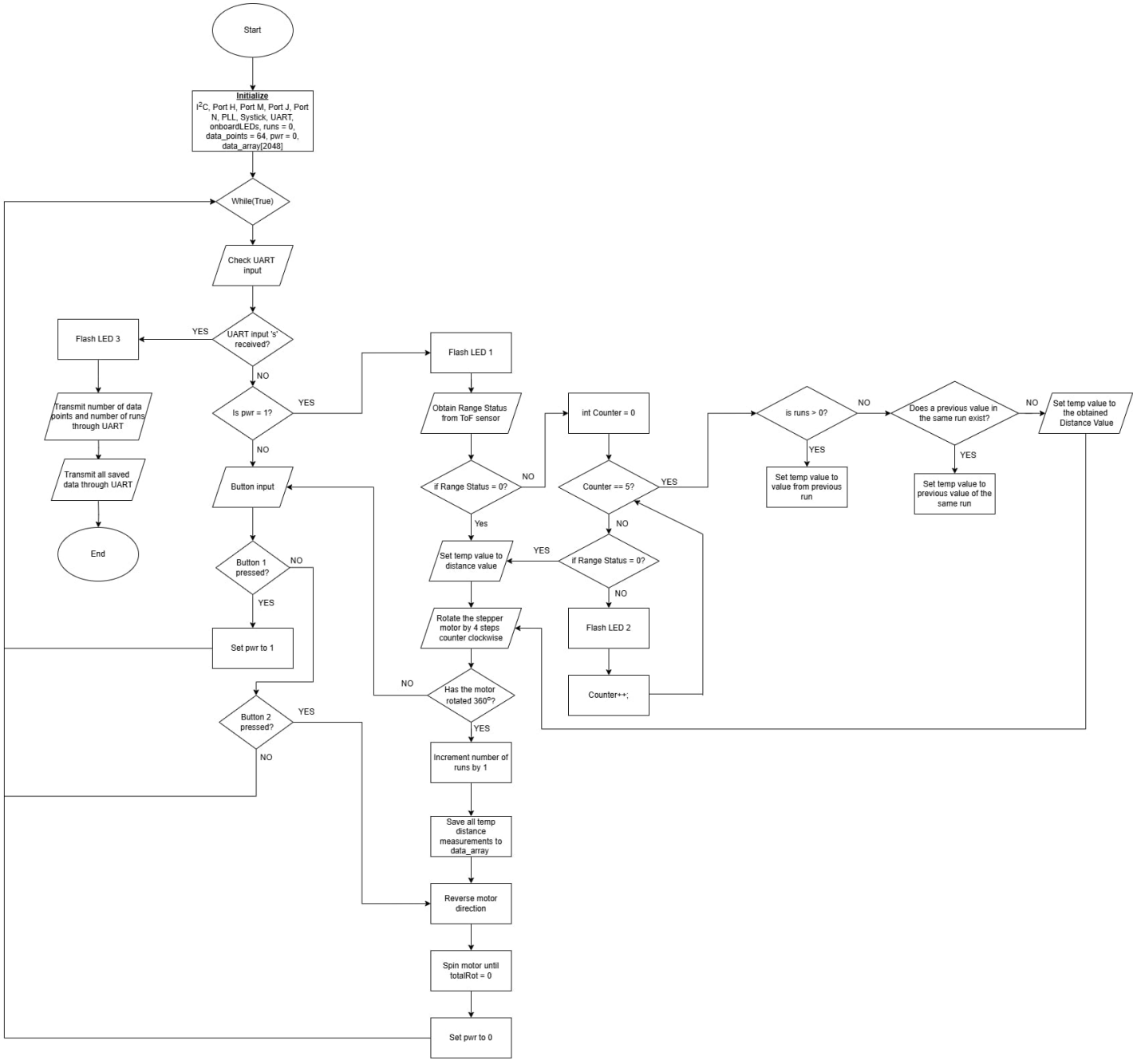

Figure 15: Flowchart of the Microcontroller code

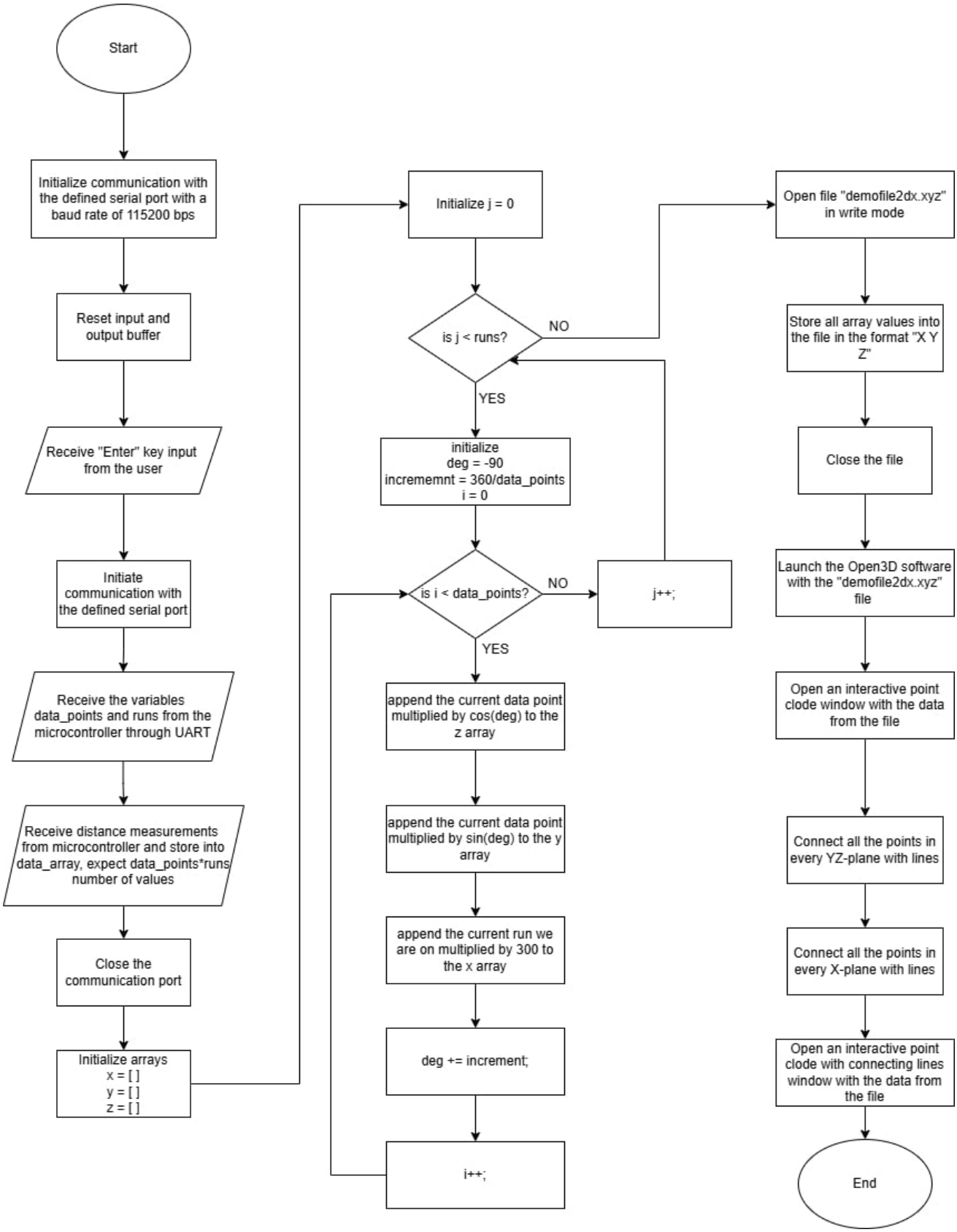

Figure 16: Flowchart of the Python code