COMPENG 2DX3 – Microprocessor Systems Project

Final Report

OM PATEL (patelo7, 400378073)

Date of Submission: April 15, 2024

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

Table of Contents

Device Overview

Features

- VL53L1X Time-of-Flight (ToF) sensor
    - Size: 4.9x2.5x1.56 mm
    - Emitter: 940 nm invisible laser (Class 1)
    - Fast and accurate ranging up to 4 meters
        - Frequency up to 50 Hz
    - I^{2}C interface communication with microcontroller up to 400 kHz
- ULN2003 driver board → 28BYJ-48 Stepper Motor
    - 4 input pins controlling the motor stepping
    - 64:1 gear ratio
    - Step angle is 0.18 degrees, 2048 steps total, for a full 360-degree rotation
- MSP432E401Y microcontroller
    - Clock speed of 60 MHz
    - 32-bit registers
    - USB to communicate with computer and for power supply
        - Data transfer to computer using 8N1 UART, 115200 baud rate
- Python script on computer for data analysis
    - Saves distance measurements from ToF sensor
    - Uses data points to visualize using Open3D python library
- Device takes scans of a room using MSP432E401Y microcontroller + ToF sensor and outputs a discretized visualization using python

General Description

The device is designed for capturing spatial information within indoor environments. It combines the capabilities of the VL53L1X Time-of-Flight (ToF) sensor (mounted to the 28BYJ-48 Stepper Motor) and the MSP432E401Y microcontroller to create discretized room scans and visualize them using Python.

The VL53L1X Time-of-Flight (ToF) measures 4.9 mm x 2.5 mm x 1.56 mm. The ToF sensor emits a 940 nm invisible laser (Class 1). Its ranging capability extends up to 4 meters, making it ideal for room-scale applications. With a maximum sampling rate of 50 Hz, the ToF sensor captures real-time distance data efficiently. Communication with the microcontroller occurs via the I2C interface, supporting speeds of up to 400 kHz.

The MSP432E401Y microcontroller links the peripherals to collect and transmit data. Operating at a clock speed of 60 MHz, it handles data processing and control utilizing 32-bit registers. The microcontroller is powered through a micro-USB connected to a computer; this serves a double purpose as the Python script accesses the UART port linked to the microcontroller. For communication with the computer, the device employs 8N1 UART at a baud rate of 115200. The Python script runs on the computer and interfaces with the device. It saves distance measurements obtained from the ToF sensor. Using the Open3D Python library, the script visualizes the collected data points, creating a discretized representation of the scanned room. An assumption made is that each xy-slice is exactly 10 cm apart, no matter how much the user actually moves.

Block Diagram

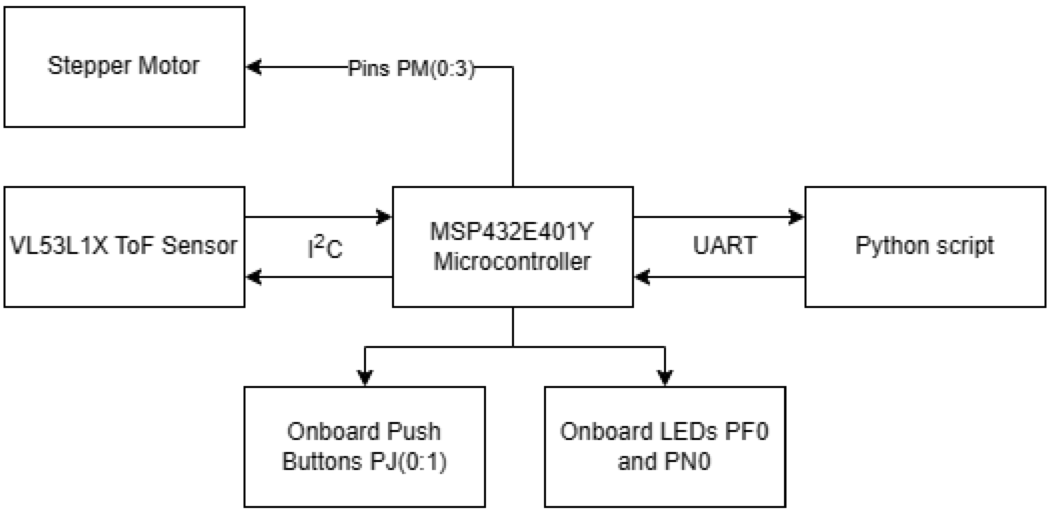


Figure 1. Block diagram outlining device structure.

Device Characteristics Table

Table 1. Device characteristics table listing associated values

Property	Value
Max distance (Dark ambient lighting)	360 cm
Max distance (Light ambient lighting)	73 cm
Distance Measurements	5 samples per 11.25 degrees
Bus speed (Microcontroller)	60 MHz
Pins used	PM(0:3) for motor, PJ(0:1) buttons, PB(2:3) I^{2}C ports, 3.3V, 5V, GND, PN0 and PF0 for LEDs
Pyserial communication port	Manually entered by user
Displacement per xy slice assumption	10 cm
Baud rate	115200

Detailed Description

Distance Measurement

Distance measurements are handled through the ToF sensor. The component operates through infrared technology, emitting photons via the emitter and receiving the same photons via the receiver. The distance is calculated based on the time it takes for the receiver to catch the emitted photons; the equation being: $d = 0.5 \cdot C \cdot ToF$ where d is the calculated distance in millimeters, C is the speed of light, ToF is the measured photon flight time. With the calculated distance (among other information the ToF sensor uses like sensor state), the I2C communication protocol transmits the data back and forth between the microcontroller and the ToF sensor. The data being transmitted is 16 data bits. Due to the ranging limitations of this sensor (360 cm long range mode), the optimal locations to spatial scan are small-medium rooms under dark ambient lighting conditions. The max distance for dark lighting is greater than the light due to how the photons get affected from external lighting.

With the 28BYJ-48 stepper motor, the ToF is mounted to the rotating component using a 3D-printed piece fastening it to place. For each 11.25-degree rotation, the stepper pauses to allow for stable scans from the ToF sensor. The sensor takes 5 distance measurements of the same position in the same xy-slice for every 11.25-degree rotation; for each successful distance scan, LED4 (PF0) flashes. This results in 32 distance measurements which are broken down into x and y components using trigonometric functions. The theta value increments by $\frac{\pi}{16}$ per 11.25-degree rotation before resetting to 0 after a complete circle. This process is initiated by pressing onboard button PJ0 upon the first 360-degree scan (after confirming the Python connection via UART). When the first 360-degree scan is complete, each subsequent run automatically increases the z-axis displacement by 10 cm (internally by Python) and is controlled using PJ1. To completely terminate the scanning and data collecting process, button PJ0 is pressed. After finishing the main objective, the microcontroller defaults to a duty-cycle producing a 2s-period LED flash (LED2 PN0).

Visualization

Throughout the scanning process, each distance measurement component (x and y) and transmitted immediately from the microcontroller to the computer for Python to read and store from UART. The Python script waits for a non-empty UART message which dictates what happens next in the data-acquisition process. If the incoming UART message reads 't\r' after decoding, then PJ0 was pressed to terminate the scanning process, therefore the overall while-loop is broken. If the incoming UART message reads 'sk\r' after decoding, then PJ1 was pressed while the motor was in the process of rotating (had not completed a full cycle); this causes the remainder of the data points in the 32 xy pair per slice to become 0. If the incoming UART message reads anything otherwise, it is counted as a valid distance measurement and saves the data to a ".xyz" file. When 32 xy pair data points are collected, the z component increments by 100 mm automatically. The user then repositions forward to continue collecting points, or to terminate the while-loop.

After the termination of the data-collecting while-loop, the newly written ".xyz" file is read using the Open3D library. The first objective is to create a point cloud datatype and use it to display the data points as an array using NumPy. The next loop gives each vertex a unique number. Next, each data point per xy-slice is connected to the next point to create a closed shape. This is repeated for each slice. After all slices are closed shapes, each of the data points within a slice are connected to the data point in the same "position" in the next slice. This forms a closed 3D shape which, in optimal conditions, resembles the room in which the distance measurements were taken (Figure 2 and Figure 3). The Open3D visualization function is used and generates a 3D representation of the data points taken and analyzed.

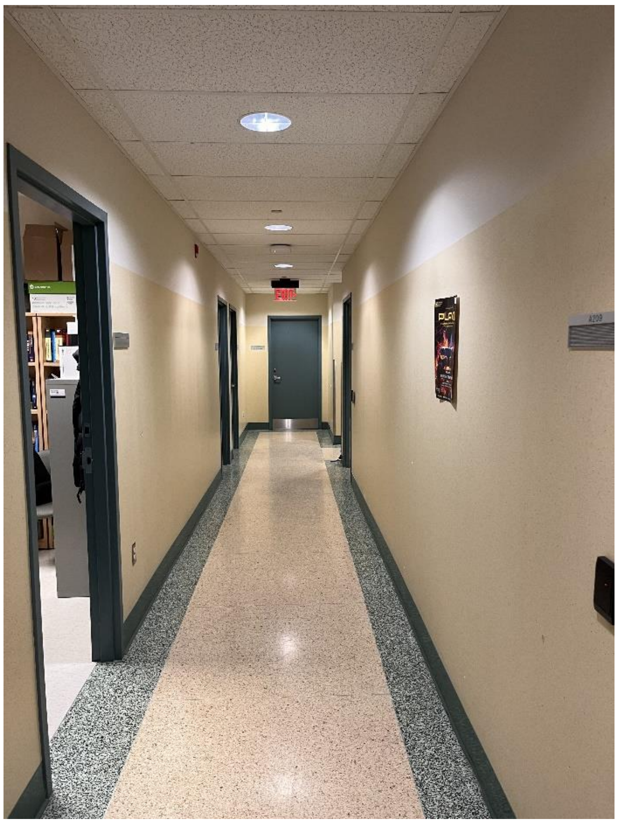


Figure 2. Picture of assigned hallway to be measured.

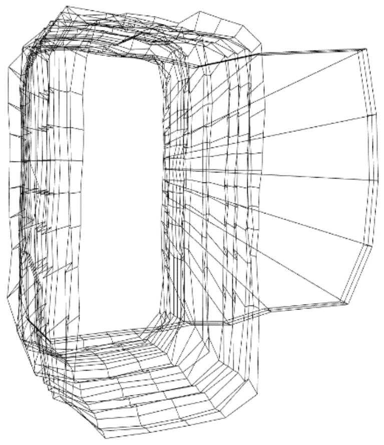
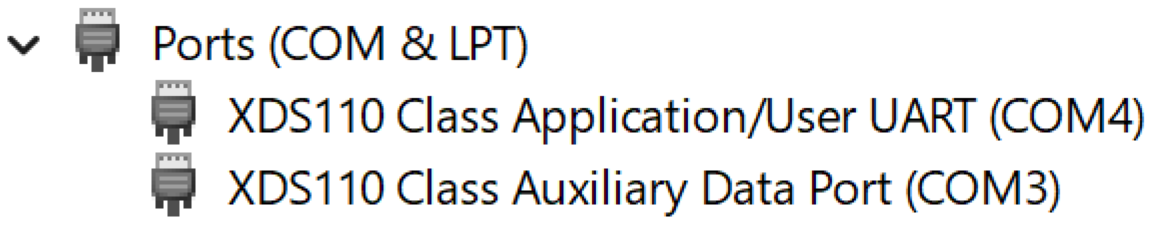


Figure 3. 3D representation of the assigned hallway, with some measurements of the corner at the end.

Instructions

All wiring connections are setup according to the circuit schematic (Figure 5). Within the Python script, a change may be required depending on the individual UART port accessed. At line 16, the 'COM4' section should be changed to correspond to the computer's UART port (Figure 4). In this case, 'COM4' is correctly set to the computer's UART port.



Ports (COM & LPT)
XDS110 Class Application/User UART (COM4)
XDS110 Class Auxiliary Data Port (COM3)

Figure 4. UART port shown using device manager on Windows 11.

The following refers to the steps taken to produce Figure 3.

1. Flash project code to microcontroller, reset to ensure proper functionality.
2. Run the Python script.
3. Press 'Enter' on the Python script, the IDE should read confirmation messages from the microcontroller.
4. Press PJ0 to begin the first scanning process.
5. Wait for the first 360-degree scan to complete (32 measurements). LED4 (PF0) should blink to indicate each distance measurement successfully completed.

6. In the case of stopping the current scan to return to the home position without completing a full circle, press PJ1.
7. Move forward by approximately 10 cm or press PJ0 to end the scanning process.
8. Press PJ1 to initiate subsequent scan.
9. Repeat steps 5-8 until ready to terminate the scanning process.
10. Once scanning is terminated by PJ0, the Python script should also terminate the while-loop, and the 3D visualization of the scanned location should auto-generate.

The expected output is shown in Figure 2 (the hallway found in ITB second floor, around rooms A210 to A214). The actual output is shown in Figure 3. Due to the simplicity of the assigned location, it is almost impossible to differentiate between the right wall from the left without having the corner turn at the end, showcasing the long hallway and the ToF sensor attempting to handle the out-of-range error. This could be solved by stabilizing the device (as the scans were made while holding the device, leading to some wobbly lines), and by taking more measurement points (increasing from 32 measurements to 64 or beyond). There is also a discrepancy with the z-displacements as the actual steps taken exceeded the preset 10 cm change, shrinking the length of the hallway the device is trying to represent.

Limitations

As mentioned above, the number of measurements taken, lack of stability, and overstepping the z-displacements can lead to unwanted discrepancies when comparing the expected output with the actual, measured, output. Remaining still for upwards of 1024 measurements would lead to a higher resolution. Displacing exactly 10 cm per circular scan would further match the expected output as well. And using a chair or other static object to hold the device while scanning would reduce any noise from the readings. But these changes are incredibly slow or agitating for a person to deal with. By introducing automation tools such as a simple moving robot could be a potential solution to these issues.

Further limitations include:

- ToF Distance Sensor Quantization Error
    - The sensor uses a 16-bit ADC which loses resolution when dealing with more precise distance values. The max quantization error is calculated as $3600 \div 2^{16}$ which is equivalent to 0.055 mm.
- Serial Communication with PC
    - The baud rate used for the device is 115,200. The actual maximum is dictated by the 16 MHz clock speed of the secondary clock. Since the UART protocol checks the incoming bit 16 times, the theoretical baud rate limit is 1,000,000.
- ToF Distance Sensor Communication
    - The I2C communication protocol is set to 100 kbps.
- Floating Point Capabilities
    - The microcontroller has a 32-bit ARM processor which single-precision floats are stored with 1 sign bit, 8 bits before a decimal point, and 23 bits after the decimal point.
    - Trigonometric functions (sine and cosine); ex. sin(pi) = 1.22e-16, which is effectively zero, and this slight discrepancy is a negligible factor.

Circuit Schematic

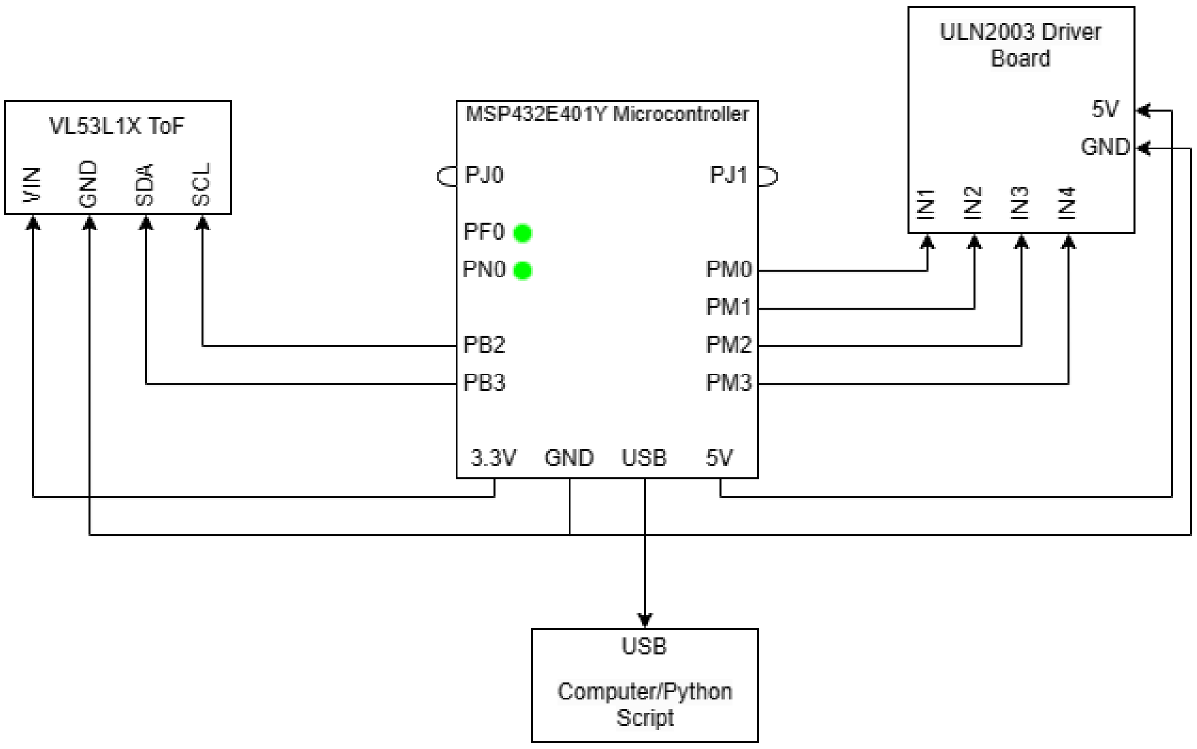


Figure 5. Circuit schematic depicting the different components of the device and their connections.

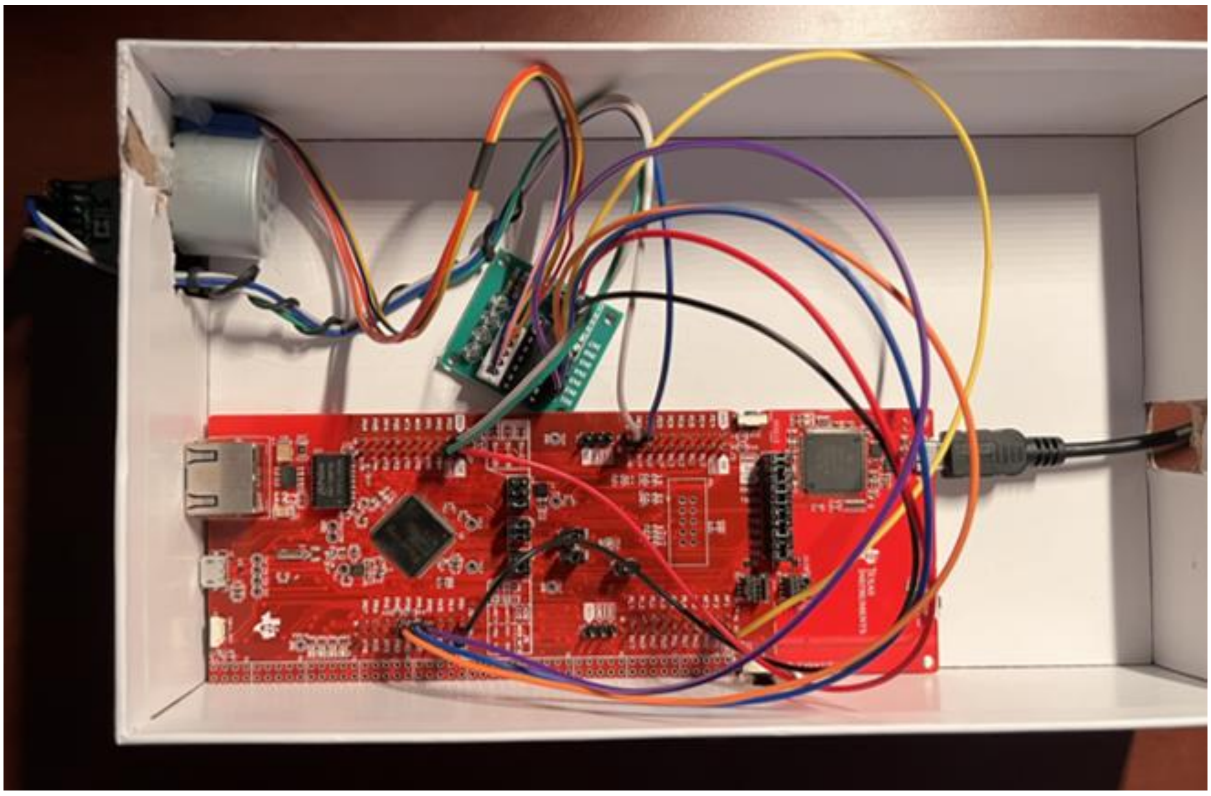


Figure 6. Physical circuit adhering to the circuit schematic.

Programming Logic Flowchart

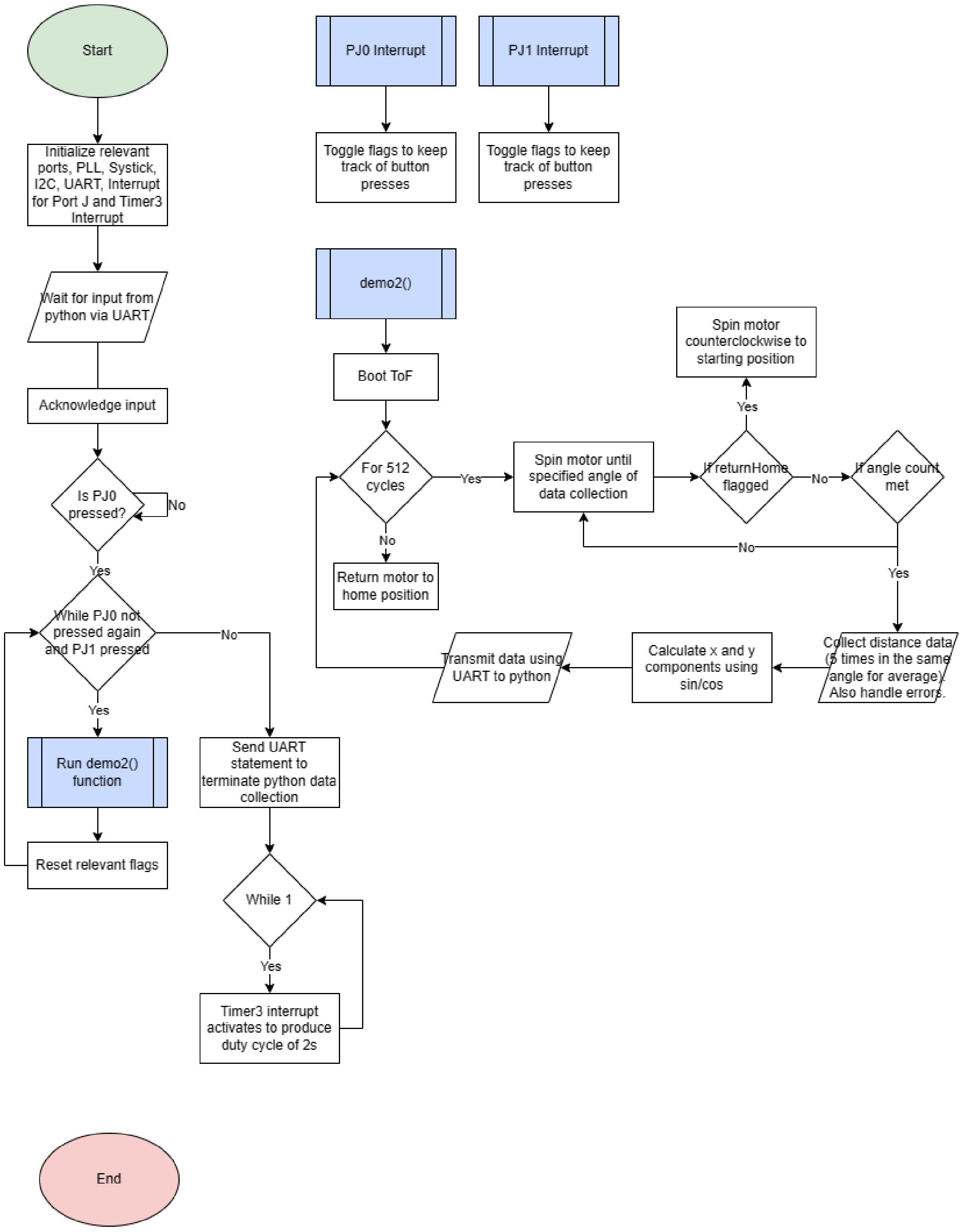

Start
Initialize relevant ports, PLL, Systick, I2C, UART, Interrupt for Port J and Timer3 Interrupt
Wait for input from python via UART
Acknowledge input
Is PJ0 pressed?
No
Yes
While PJ0 not pressed again and PJ1 pressed
No
Yes
Run demo2() function
Reset relevant flags
Send UART statement to terminate python data collection
While 1
Yes
Timer3 interrupt activates to produce duty cycle of 2s
End
PJ0 Interrupt
Toggle flags to keep track of button presses
PJ1 Interrupt
Toggle flags to keep track of button presses
demo2()
Boot ToF
For 512 cycles
Yes
No
Return motor to home position
Spin motor until specified angle of data collection
If returnHome flagged
Yes
Spin motor counterclockwise to starting position
No
If angle count met
No
Yes
Collect distance data (5 times in the same angle for average). Also handle errors.
Calculate x and y components using sin/cos
Transmit data using UART to python

Figure 7. Program flow on the microcontroller side, notice how "End" is not connected since the program remains in a dormant state after completing main function.

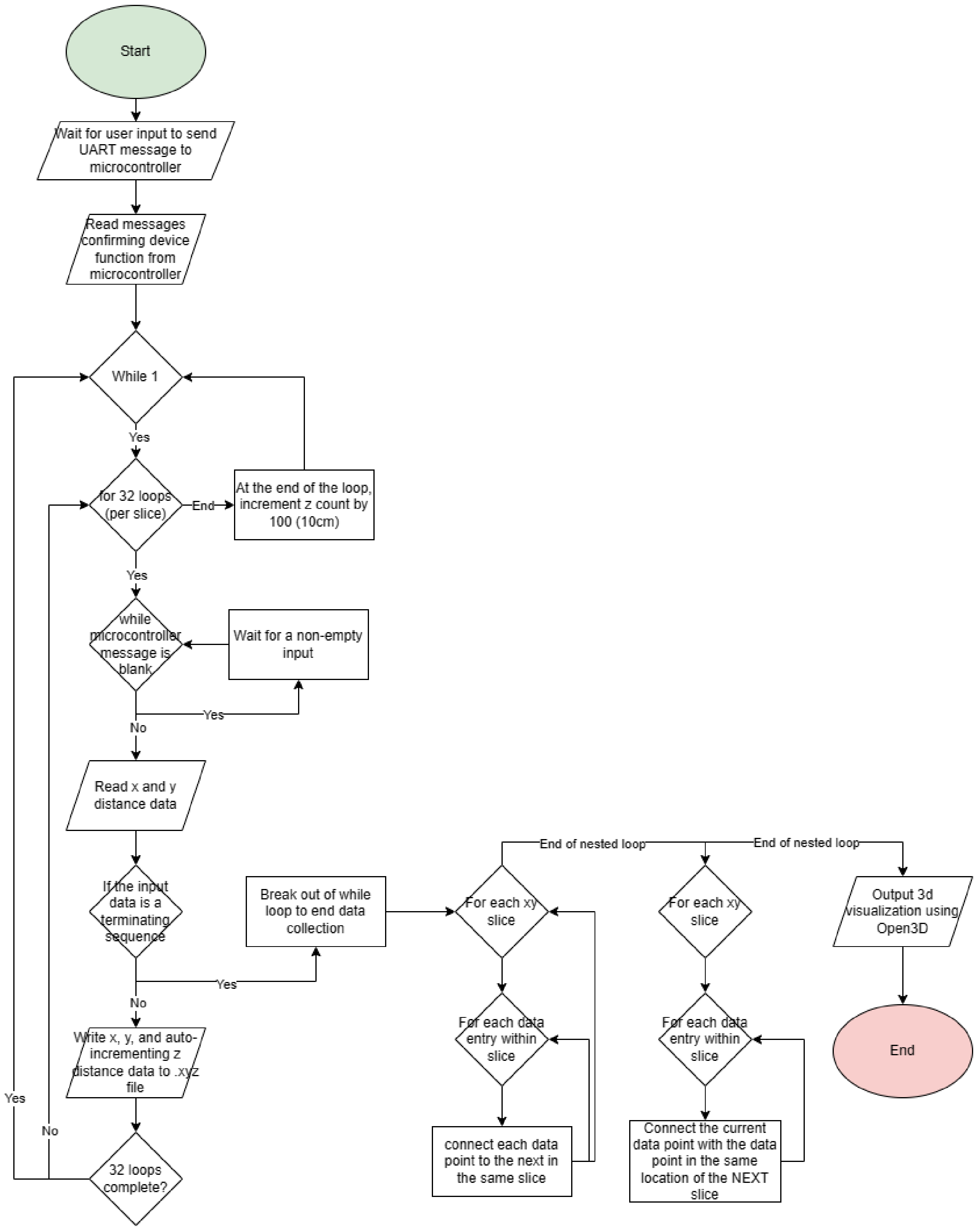

Start
Wait for user input to send UART message to microcontroller
Read messages confirming device function from microcontroller
While 1
Yes
for 32 loops (per slice)
End
At the end of the loop, increment z count by 100 (10cm)
Yes
while microcontroller message is blank
Wait for a non-empty input
Yes
No
Read x and y distance data
If the input data is a terminating sequence
Yes
Break out of while loop to end data collection
No
Write x, y, and auto-incrementing z distance data to .xyz file
32 loops complete?
Yes
No
For each xy slice
For each data entry within slice
connect each data point to the next in the same slice
End of nested loop
For each xy slice
For each data entry within slice
Connect the current data point with the data point in the same location of the NEXT slice
End of nested loop
Output 3d visualization using Open3D
End

Figure 8. Program flow on the python/computer side.