COMPENG 2DX3
Microprocessor System Project
Final Project Report

Deyontae Patterson – patted9 – 400480946

Submitted by Deyontae Jaden Patterson, patted9, 400480946

Table of Contents

Table of Figures/Tables

Device Overview

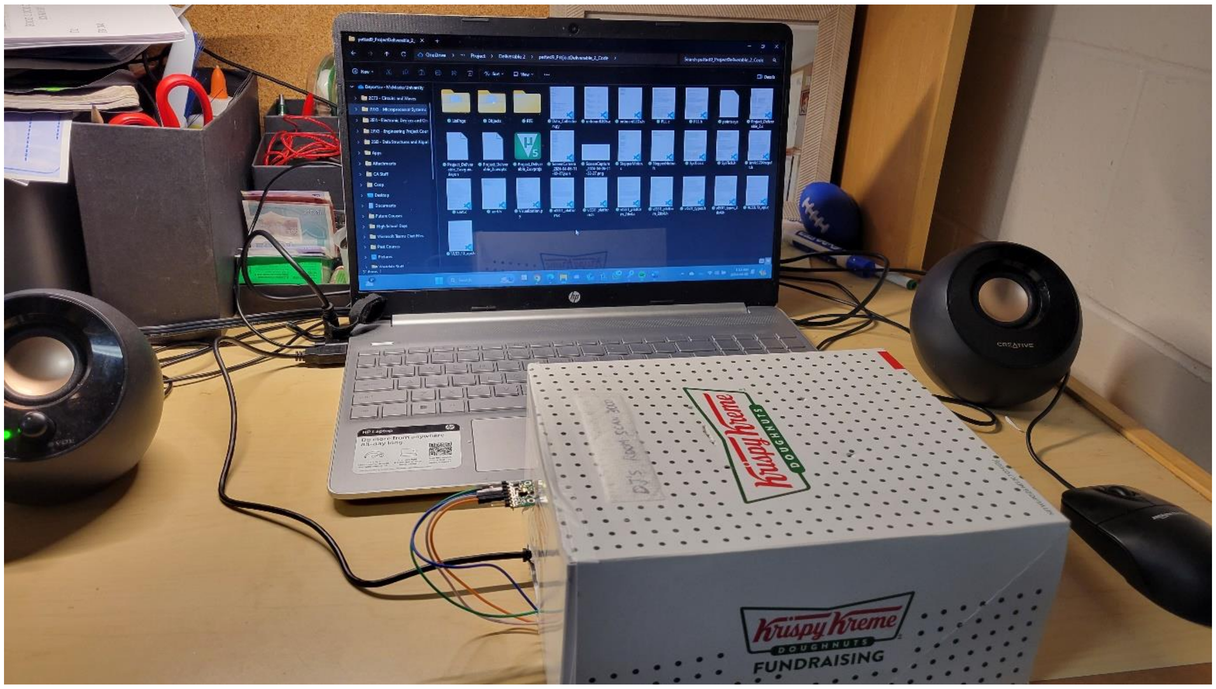


Figure 1: Intended EASY Setup of Device

Features/Specs

This device takes Vertical scans of its environment and recreates the findings in 3D visualization software. The device was built using:

- MSP432E401Y Microcontroller LaunchPad™ Development Kit
  - ARM Cortex-M4 Central Processing Unit
  - Bus Speed of 12 MHz (default of 120 MHz)
  - Operating Voltages of 2.5V to 5V
- VL53L1X Time-of-Flight Sensor
  - Able to record distance up to 4 meters away
  - Operating Voltage of 3.3V
  - Up to 50 Hz ranging frequency
  - I^2C connection
- ULN2003A-PCB Stepper Motor

    - Operating Voltage of 5V
    - 22.5-degree Step Angle
- 1 External Push Button
    - Starts/Stops measurement runs.
- Python Code
    - Pyserial used for UART communication with Microcontroller at 115200 Baud Rate
    - Open3D used for 3D visualization of measurements

General Description

The device's main function is to produce 3D recreations of empty physical spaces such as Hallways and mainly empty rooms. The following is an overview of the process for its general functionality:

The system's design may be compared best to a classic Finite State Machine, where the microcontroller stays in the idle state of repeatedly flashing PN1 until the expected stimuli is received; this stimuli is pushing the external push button on the breadboard. The Push Button initiates the scanning process by sending a LO signal to the input pin PM0 (Active Low configuration). From here, the microcontroller sends signals to outputs PH0 to PH3, causing the Stepper Motor to rotate as at speed of about 0.025 rad s^{-1} for 22.5 degrees.

At this point, the ToF Sensor records its distance from the closest object and sends this data to the microcontroller via I^2C connection. The microcontroller reads and send data to Pyserial via UART connection. Python interprets this data by converting to x, y and z coordinates by use of Pythagorean theorem, then writes it to a file called "points.xyz".

Once data is collected, the Stepper Motor will continue this turning at 22.5 degree intervals until a full revolution. After which, the stepper motor will automatically begin the "ReturnHome()" function by rotating anticlockwise until reaching home, so as to avoid tangling of the wires. Note that if the push button is pressed, while in the measurement collecting state, this will also initiate the ReturnHome() function.

This "Step-Measure-Return" motion continues for a specified number of YZ-slices. By the end, the user is able to open a 3D recreation of the scanned space in Python's Open3D feature.

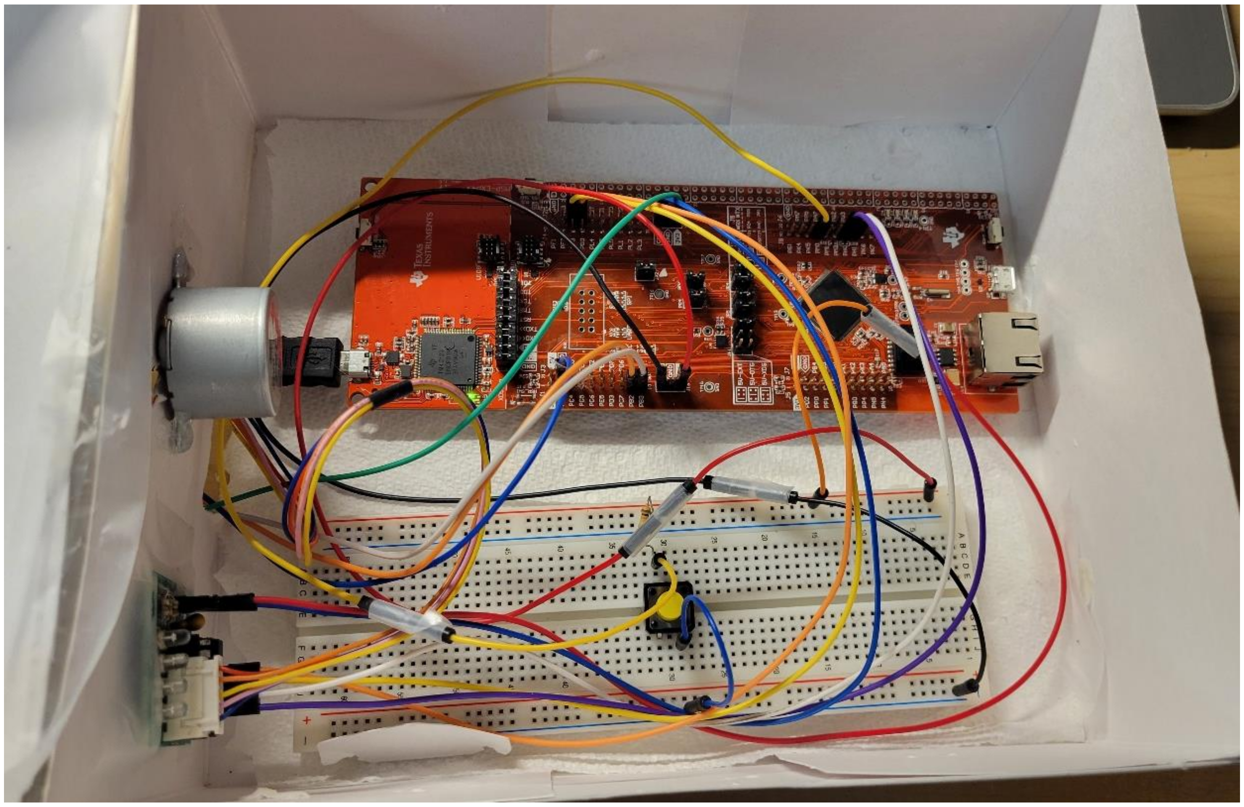
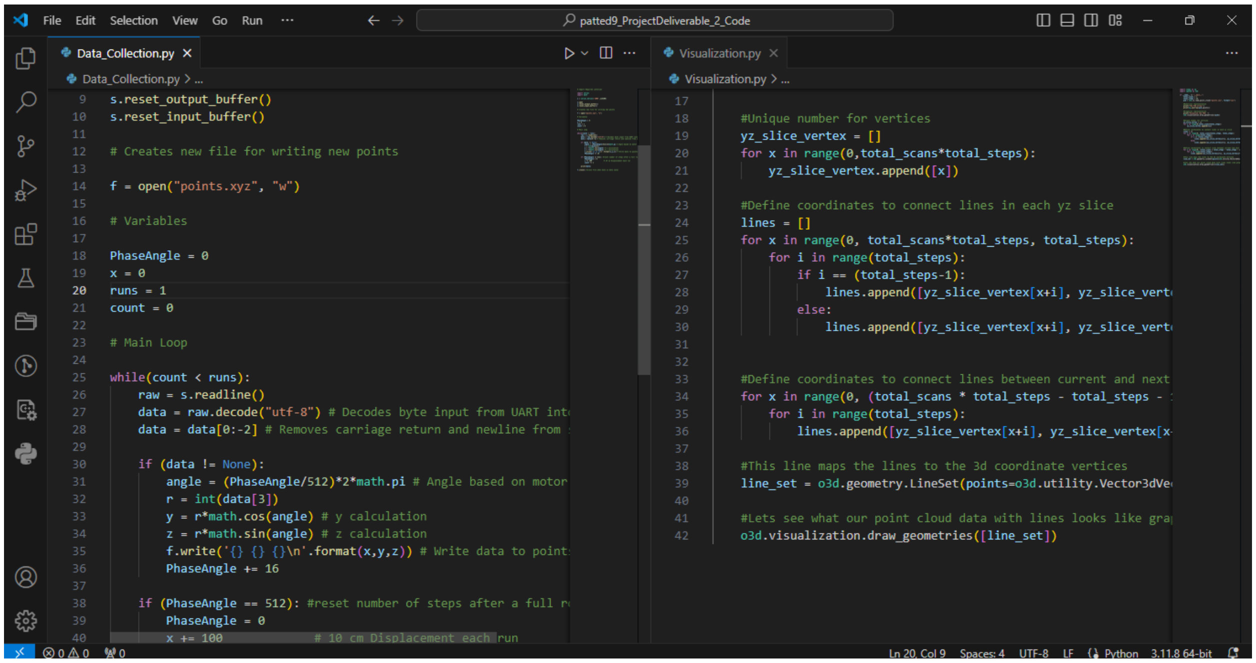


Figure 2: Inner Circuitry of Device

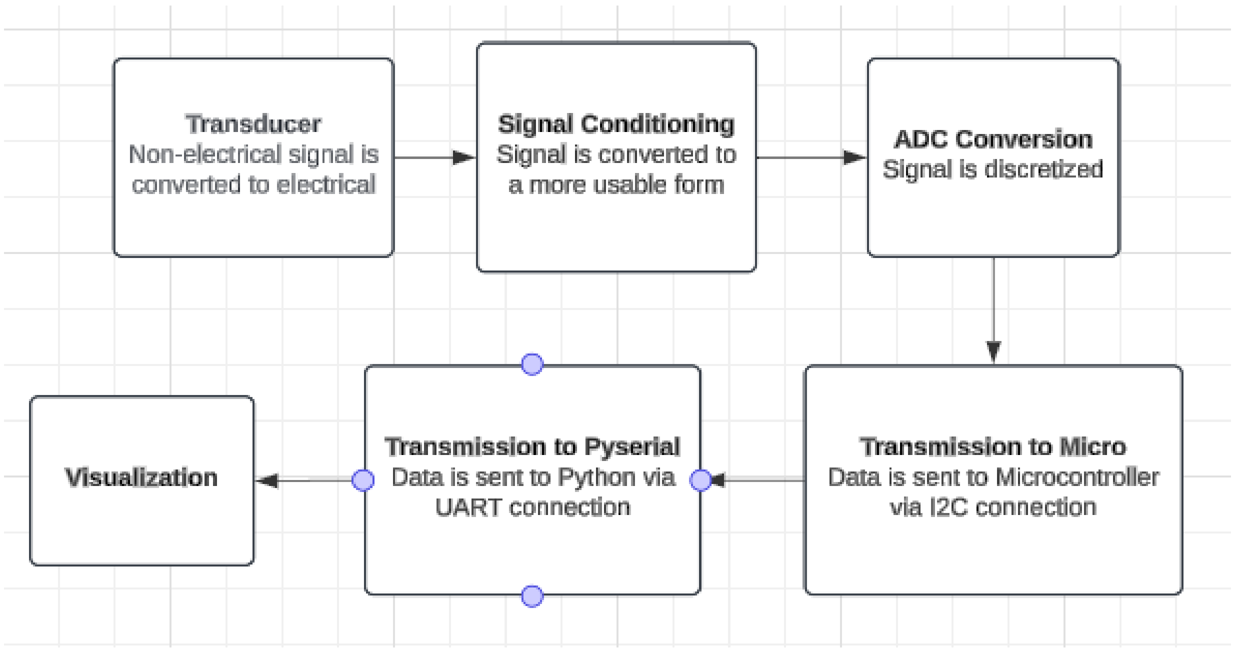


```python
s.reset_output_buffer()
s.reset_input_buffer()

# Creates new file for writing new points

f = open("points.xyz", "w")

# Variables

PhaseAngle = 0
x = 0
runs = 1
count = 0

# Main Loop

while(count < runs):
    raw = s.readline()
    data = raw.decode("utf-8") # Decodes byte input from UART int
    data = data[0:-2] # Removes carriage return and newline from

    if (data != None):
        angle = (PhaseAngle/512)*2*math.pi # Angle based on motor
        r = int(data[3])
        y = r*math.cos(angle) # y calculation
        z = r*math.sin(angle) # z calculation
        f.write('{} {} {}\n'.format(x,y,z)) # Write data to point
        PhaseAngle += 16

    if (PhaseAngle == 512): #reset number of steps after a full r
        PhaseAngle = 0
        x += 100            # 10 cm Displacement each run
```

```python

    #Unique number for vertices
    yz_slice_vertex = []
    for x in range(0,total_scans*total_steps):
        yz_slice_vertex.append([x])

    #Define coordinates to connect lines in each yz slice
    lines = []
    for x in range(0, total_scans*total_steps, total_steps):
        for i in range(total_steps):
            if i == (total_steps-1):
                lines.append([yz_slice_vertex[x+i], yz_slice_vert
            else:
                lines.append([yz_slice_vertex[x+i], yz_slice_vert


    #Define coordinates to connect lines between current and next
    for x in range(0, (total_scans * total_steps - total_steps -
        for i in range(total_steps):
            lines.append([yz_slice_vertex[x+i], yz_slice_vertex[x-

    #This line maps the lines to the 3d coordinate vertices
    line_set = o3d.geometry.LineSet(points=o3d.utility.Vector3dVe

    #Lets see what our point cloud data with lines looks like gra
    o3d.visualization.draw_geometries([line_set])
```

Figure 3: Screenshot of PC Screen During Operation

Block Diagram



Transducer
Non-electrical signal is converted to electrical
Signal Conditioning
Signal is converted to a more usable form
ADC Conversion
Signal is discretized
Transmission to Micro
Data is sent to Microcontroller via I2C connection
Transmission to Pyserial
Data is sent to Python via UART connection
Visualization

Figure 4: Block Diagram of System Operations

Device Characteristics Table

Table 1: Description of Components in Device

Main Components	**Description**
Microcontroller	• Bus Frequency → 12 MHz • Measurement LED → PF4 • Error LED → PN1
ToF Sensor	• Operating Voltage → 3.3V; Connected to Ground • Pins: SCL → PB2; SDA → PB3
Stepper Motor	• Operating Voltage → 5V; Connected to Ground • Pins: PH0 to PH3 set to IN0 to IN3
Push Button	• Operating Voltage → 5V; Connected to Ground • Pins: PM0
Python 3.8	• Imported Libraries: Pyserial, Open3D, Math, NumPy • UART Connection: 115200 Baud Rate

Detailed Description

Distance Measurements

The main component for measuring distance is ToF, which does this by sending out a light signal, then waits for it to return to reflect into the sensor. It records the time taken for the signal to return with the known speed of light to find the distance travelled.

$Distance(m) = \frac{Speed\ of\ Light\ (m/s)}{time\ (s) \times 2}$, since the time recorded would include the time for reaching the distance and back, the final distance is divided by 2 to give just the time for the signal to reach the measured distance. The ToF then acts as a transducer (converting non-electrical signals to electric), Conditioning Circuit (tweaking the signal so it can be converted to digital more easily) and ADC Converter (converting the continuous Analog signals to discretized digital ones) before outputting the measured distance to the microcontroller.

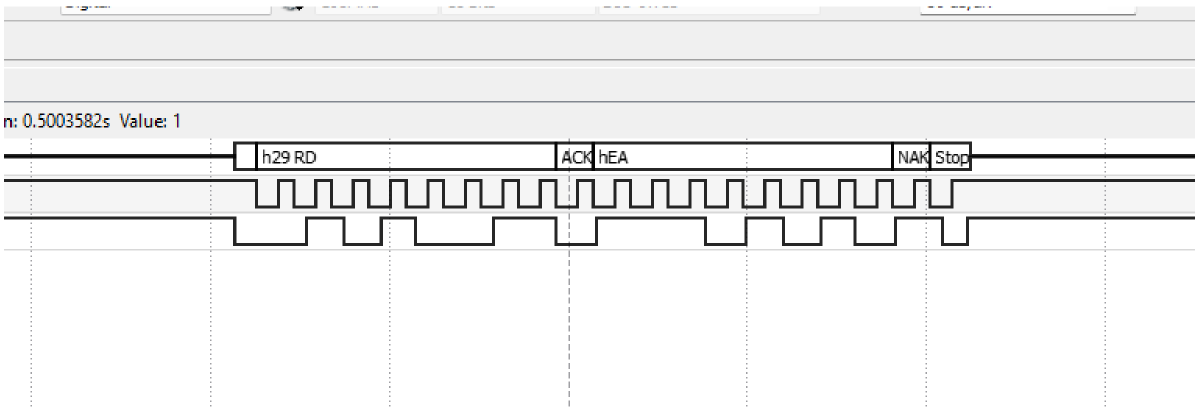


Figure 5: Past Example of I2C Connection between ToF Sensor and Microcontroller

Data_Collection.py opens a UART serial communications with the Microcontroller in COM4 at baud rate 115200 bps and 8-N-1 format. The Python code is responsible for receiving this data and storing this data, but first, the data must be converted to format usable by the Open3D application. The conversion process is a simple calculation of the y and z-coordinates by use of the equations: $y = r \times cos(\theta)$ and $z = r \times sin(\theta)$, where r is the distance measured and θ is the phase angle at that time (calculations are possible courtesy of the Python's Math Library). The x-coordinate remains constant for this portion.

After The x-y-z coordinates are appended to a list and the stepper motor moves another 22.5 degrees for another measurement of x-y-z (a total of 16 measurements per slice). Finally, depending on how far the user wishes to measure down this space, the device can take multiple slices, each 10 cm apart (here, the x-coordinate is incremented 100 mm each slice). Once done, the final list is written to a file called "points.xyz" for Visualization.py to interpret.

Visualization

Once Data_Collection.py is finished running, the User runs "Visualization.py" for representing this data is a 3D space (courtesy of numpy and open3d libraries). In this file, the points in each slice are connected to create an enclosed shape. Open3D reads the x-y-z coordinates as point clouds to be plotted all around a center. It does the same treatment for the next y-z slice and connects all of the corresponding point clouds between each with 100 mm offset between each slice. Finally, Open3D reads this and represents it in an automatic popup window for open viewing.

Application

User Installation Guide

The following are list of setup instructions for the User to configure the device to their PC and be ready for regular use:

1. Keil Installation
    a. In the Pack Installer, install the pack for the Microcontroller your system uses.
2. Python 3.8 Installation
    a. Use “pip install pyserial” and “pip install open3d” in the integrated terminal
3. Visual Studio Code (recommended for editing code).
4. RealTerm (Recommended for error detection in UART).

Hardware

1. Microcontroller
2. Stepper Motor
3. Motor Driver
4. Time of Flight Sensor
5. 10k Ohm resistor
6. Male-to-Male Wires
7. Female-to-Female Wires

Instructions

1. Find a location to perform a scan on and ensure it is not directly in line with a light source (preferred if all lights are off), a window or any space that is more than 4 meters away.

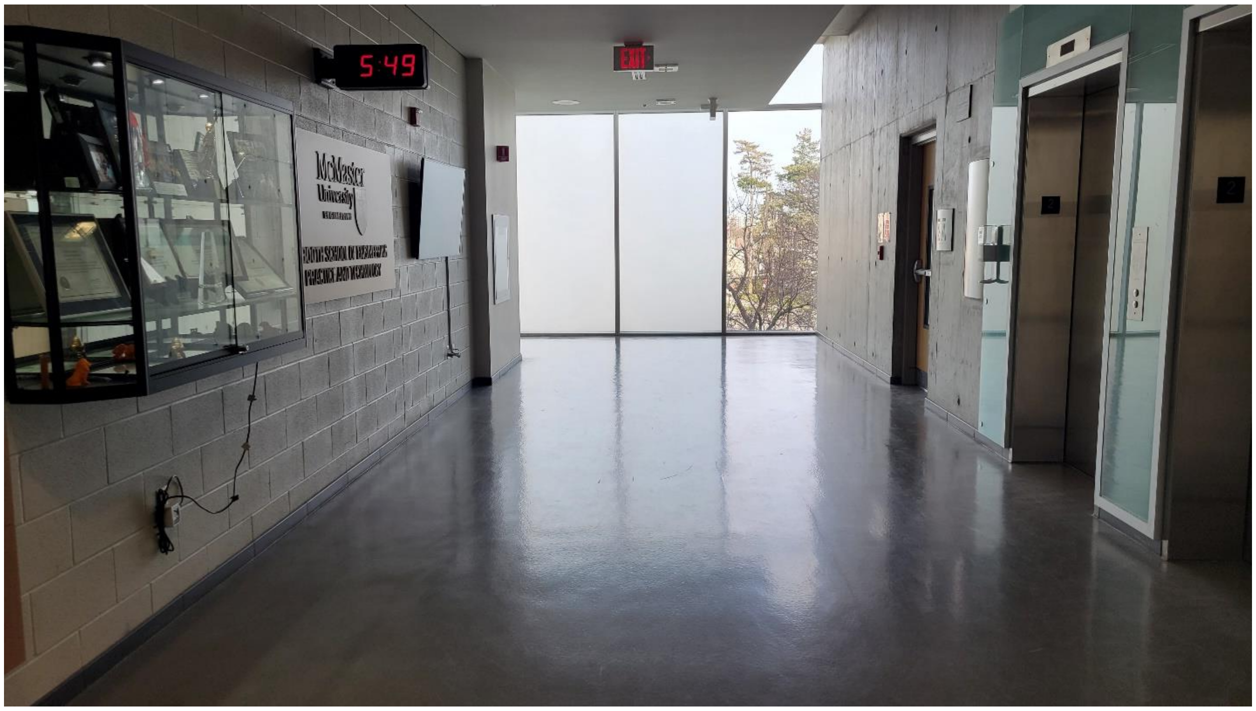


Figure 6: Hallway used for Example Scan

2. Place the Device in the center of the space.

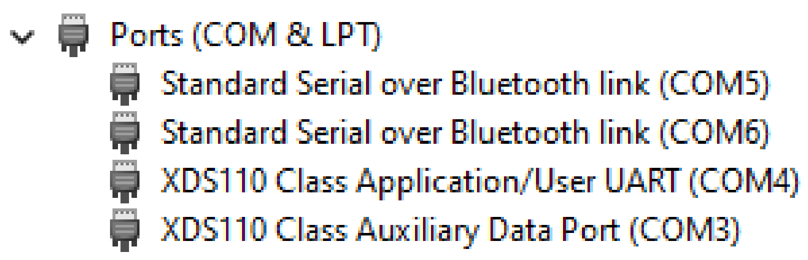
3. Ensure all inner circuitry is still intact (see Figure 2).

4. Connect the Microcontroller's Micro USB (on the same side as the reset button) to your PC through the hole in the front.

5. Open the Keil Project, Data_Collection.py and Visualization.py and change the following for varying the number of YZ-slices:
    a. Project_Deliverable_2.c:
        i. line 194 → "while(runs < **[NUMBER OF SLICES]**)"
        ii. line 215 → "runs = **[NUMBER OF SLICES]**"
    b. Data_Collection.py, line 20 → "runs = **[NUMBER OF SLICES]**"

c. Visualization, line 5 → "total_scans = **[NUMBER OF SLICES]**"

6. Select the correct Communication Port for UART connection:
   a. Open your PC's Device Manager application.
   b. Search for "Ports (COM & LPT)" and press the drop-down arrow. Take note of your unique number in "XDS110 Class Application/User UART (COM#)"



Ports (COM & LPT)
Standard Serial over Bluetooth link (COM5)
Standard Serial over Bluetooth link (COM6)
XDS110 Class Application/User UART (COM4)
XDS110 Class Auxiliary Data Port (COM3)

Figure 7: User-specific Port #

   c. Change Data_Collection.py, line 6 → "s = serial.Serial('COM**[NUMBER]**',115200)"

7. Save changes to both Python files and save changes to C code. Use the "Translate", "Build" and "Download" options in Keil to load new instructions to microcontroller

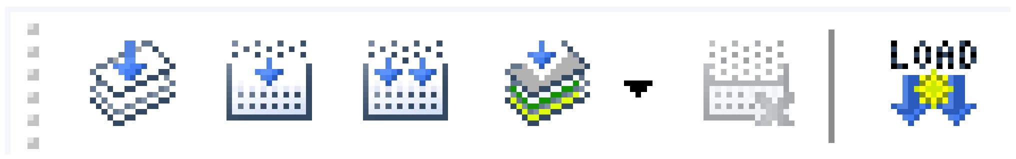
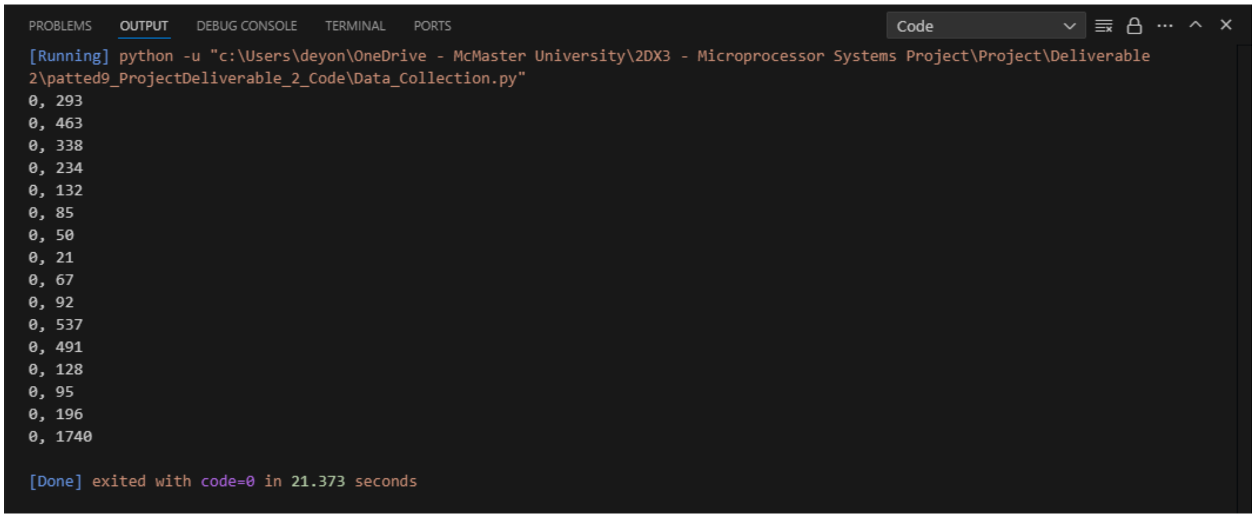


Figure 8: Buttons for Flashing new instructions to Microcontroller

8. Finally, press the reset button at the top of the microcontroller. If done correctly, the Microcontroller's onboard LEDs should all flash in sequence, then after the Microcontroller should be in the idle state with the LED at PN1 flashing about every 2 seconds.

9. Run Data_Collection.py and quickly press the external push button to activate the stepper motor (if too much time passes, the Python code will timeout and Data_Collection.py will have to be run again). Hold the Microcontroller still until the ToF Sensor completes a full rotation.

10. Once the rotation is complete, the stepper will automatically begin rotating anticlockwise until reaching the starting position; for this period, no measurements are being taken.
    a. If the User is planning on performing multiple scans, take this opportunity to push the device up by 10 cm in a straight line. Once the stepper is back in the starting position, it will automatically begin rotating clockwise again for the next slice's scans. Repeat this steps until all slices are complete.
    b. If the User is scanning a single slice, move on to Step 9.



```
[Running] python -u "c:\Users\deyon\OneDrive - McMaster University\2DX3 - Microprocessor Systems Project\Project\Deliverable 2\patted9_ProjectDeliverable_2_Code\Data_Collection.py"
0, 293
0, 463
0, 338
0, 234
0, 132
0, 85
0, 50
0, 21
0, 67
0, 92
0, 537
0, 491
0, 128
0, 95
0, 196
0, 1740

[Done] exited with code=0 in 21.373 seconds
```

Figure 9: Expected Terminal output of Data_Collection.py

11. Once this entire process is complete, run Visualization.py and wait for the Open3D popup window to open. Initially, 3D diagram of an array of rainbow-colored point clouds show
12. appear. Close this popup and wait for the second popup to open.

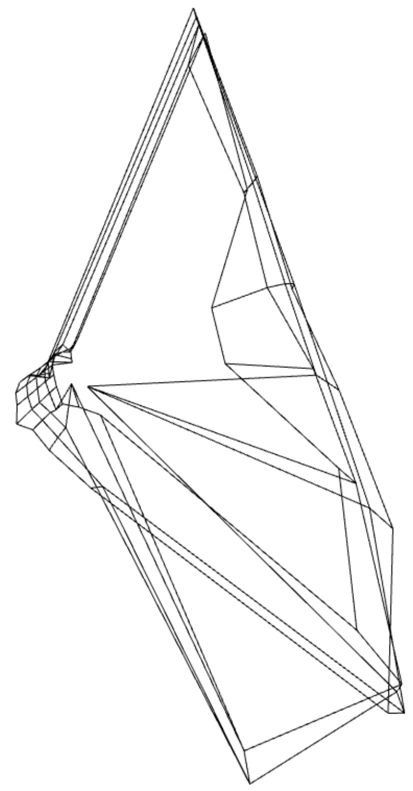


Figure 10: 3D Scan of Hallway

Limitations

1. The Microcontroller, while able to make very accurate calculations is still limited to 32-bits for representing its values. This limitation may affect the results of trigonometric calculations involving very small numbers.

2. Quantization Errors for ToF Modules:

$$Quantization\ Error = \frac{V_{FS}}{2^m}$$

$$ToF\ Error = \frac{3.3V}{2^8} = 12.89\ mV$$

3. The maximum UART communication speed for the microcontroller is 115200 bps. This is shown in the microcontroller’s datasheet, and tests with higher speeds result in errors in communication. These errors can be seen in an application such as RealTerm.

4. The means of communication between the ToF sensor and the Microcontroller was I2C at a standard rate of 100,000 bps.

5. The primary limiting factor to speed is the angular velocity of the stepper motor. While it is true that speed can be increased by reducing the delays within steps, decreasing these delays too much may result in the stepper getting instructions too quickly to execute them, stopping the stepper from rotating at all. The stepper also limits the speed when retuning to the home position.

Circuit Schematic

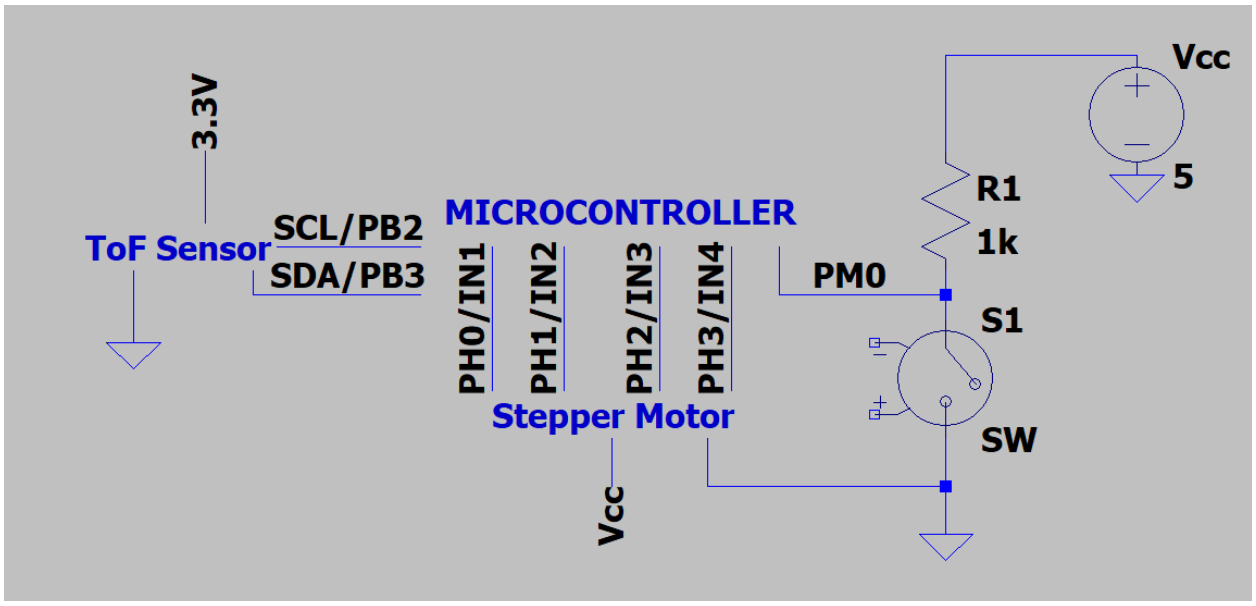


Figure 11: High Level Circuit Schematic

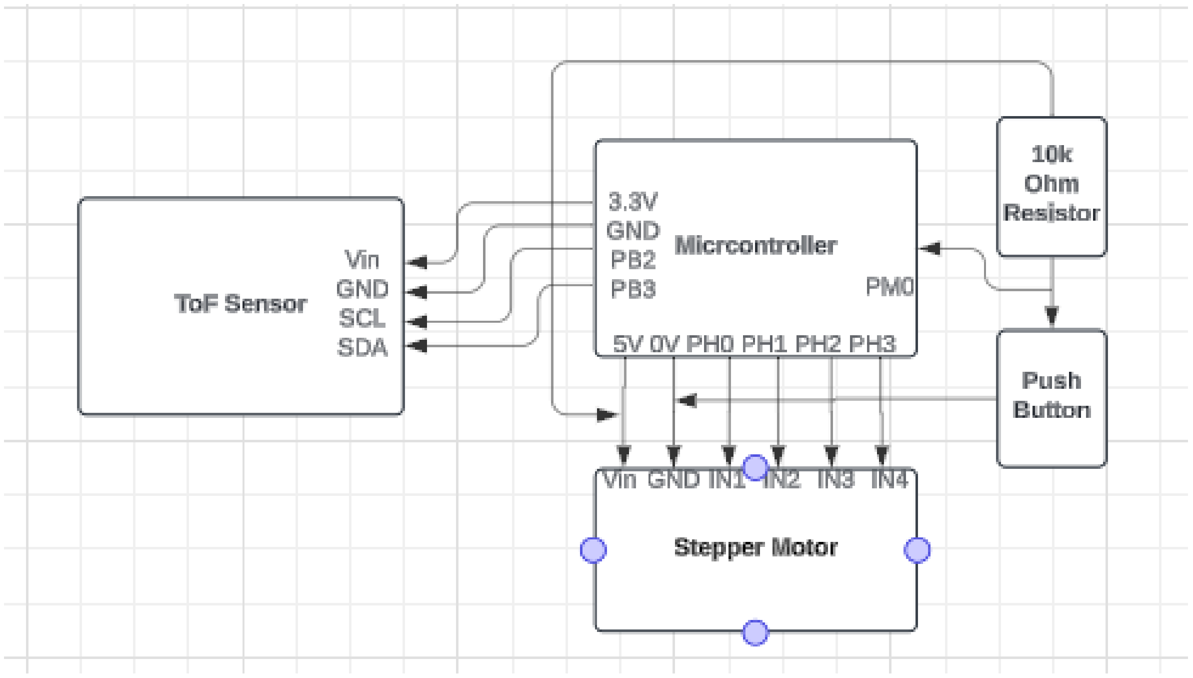


Figure 12: Low-Level Circuit Schematic

Program Logic Flowchart

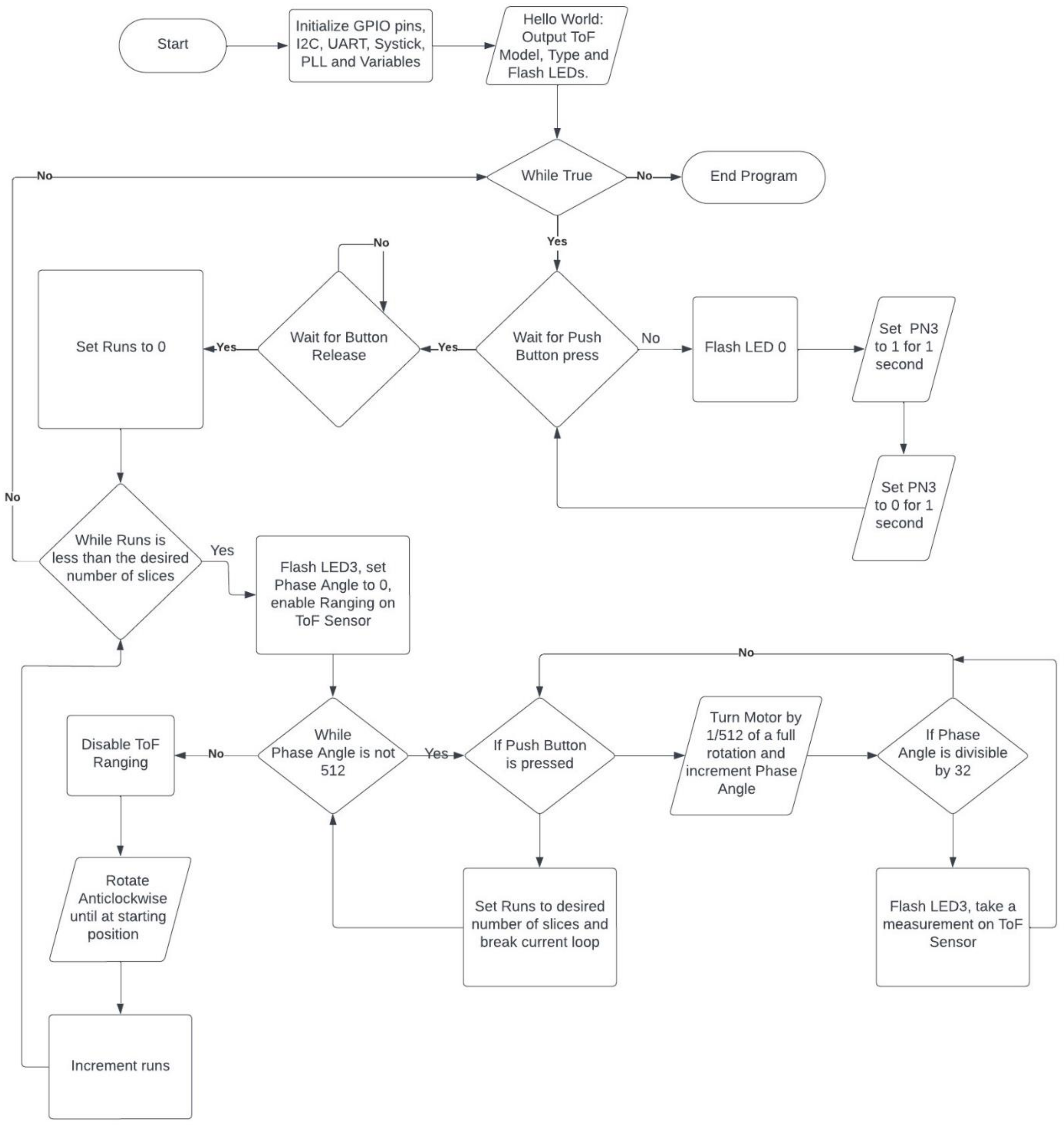

Start
Initialize GPIO pins, I2C, UART, Systick, PLL and Variables
Hello World: Output ToF Model, Type and Flash LEDs.
While True
No
End Program
Yes
No
Wait for Push Button press
Yes
Wait for Button Release
No
Yes
Set Runs to 0
No
Flash LED 0
Set PN3 to 1 for 1 second
Set PN3 to 0 for 1 second
No
While Runs is less than the desired number of slices
Yes
Flash LED3, set Phase Angle to 0, enable Ranging on ToF Sensor
While Phase Angle is not 512
No
Disable ToF Ranging
Rotate Anticlockwise until at starting position
Increment runs
Yes
If Push Button is pressed
Set Runs to desired number of slices and break current loop
Turn Motor by 1/512 of a full rotation and increment Phase Angle
If Phase Angle is divisible by 32
No
Flash LED3, take a measurement on ToF Sensor

Figure 13: Flowchart of C Code