COMPENG 2DX3: Microprocessor Systems Project

Final Project
Due Date: April 9th, 2025

NAME: GOBIND KAILEY
STUDENT NUMBER: 400522877

INSTRUCTORS: DR. YASER M. HADDARA, DR. SHAHRUKH ATHAR, DR. THOMAS DOYLE

Device Overview

Features

System Communication Protocols

The components communicate using the following protocols:

- **I2C Serial Communication:** Used between the TOF sensor and microcontroller, with data transmission at 100 kbps, programmed in C.
- **UART Serial Communication:** Used between the microcontroller and a PC, with a baud rate of 115200 bps, programmed in Python.
- **3D Visualization:** Implemented using Python 3.11.9, with Pyserial for data handling and Open3D for rendering.

Texas Instruments MSP432E401Y Microcontroller [1]

Its specifications include:

- **Microcontroller Processor:** ARM Cortex-M4
- **Max Bus Speed:** 120MHz
- **Voltage:** Recommended DC operating range: 2.97V - 3.63V (Standard: 3.3V)
- **Cost:** CAD 72.55
- **Communication**: Supports I2C and UART for serial communication
- **Memory**: 1024 KB Flash, 6 KB EEPROM, 256 KB SRAM
- **Analog to Digital Conversion:** 12-bit SAR ADC

Time-of-Flight Sensor (VL53L1X - Pololu) [2]

The VL53L1X sensor is used for distance measurements and is integrated with the microcontroller. Its specifications include:

- **Max distance measurable:** 4 meters, configurable (long, medium, and short ranges)
- **Operating Voltage**: 2.6V - 5.5V (VDD: 2.6V - 3.5V when Vin is disconnected)
- **Communication**: I2C protocol, operating at a maximum frequency of 400 kHz
- **Cost**: CAD 21.40 (Electronic Components)
- **Ranging Frequency**: Up to 50 Hz

Stepper Motor and Driver Board (28BYJ-48 with ULN2003)

The ToF sensor is mounted on the unipolar stepper motor, allowing for 360-degree scanning along a single plane. Its specifications include:

- **Gear Ratio:** 64:1, with two center-tapped coils driving an internal gear, which in turn moves an outer gear
- **Full-Step Mode:** 2048 steps per revolution of the internal gear, corresponding to 512 steps per revolution of the outer gear
- **Control:** Operated via the ULN2003 driver board, which requires six connections:
    - Four wires for coil control
    - One for power
    - One for ground
    - Operating Voltage – would be 5V or 12V depending on the project requirements (5V in our case).

General Description

This project aims to generate a 3D model of an indoor location using a time-of-flight (ToF) sensor as a cost-effective alternative to a more expensive LIDAR sensor. The system is built with a microcontroller (MSP432E401Y), a stepper motor (28BYJ-48), and a ToF sensor (VL53L1X), which measures distances by emitting infrared light and detecting reflections. The measured distance is provided in millimeters and is used to calculate the x-y values to reconstruct spatial data.

The ToF sensor captures distance measurements by emitting a laser pulse and calculating the time taken for the light to return after reflecting off an object. This measurement is inherently digital, eliminating the need for separate analog-to-digital conversion (ADC). The sensor operates at a 12-bit resolution and transmits data via I2C communication to the microcontroller. To capture a 360-degree scan, the sensor is mounted on the stepper motor using a 3D-printed mount. The motor rotates in discrete steps of 11.25 degrees, collecting 32 distance points per full rotation. The sensor's movement along the z-axis is incremented manually in the code to extend the scan across multiple layers, enabling full 3D space mapping.

Once the ToF sensor collects a measurement, the data is transmitted via I2C to the MSP432E401Y microcontroller. The microcontroller then relays this data to a PC through UART serial communication using COM3 with a baud rate of 115200 bps. The PC-side program, written in Python 3.11.9, reads this serial data using PySerial and processes it in real-time. Each scan generates a point cloud, where each point consists of x, y, and z coordinates. These points are stored in a .xyz file, which serves as the data source for visualization.

To create a 3D representation of the scanned area, the PC software processes the collected .xyz data using Open3D, a Python library for 3D point cloud visualization. Each recorded data point is positioned in a Cartesian coordinate system. The distance from the sensor determines the radial position, the rotation angle of the stepper motor sets the coordinate, and the z-axis is incremented manually in the code, which helps create the 3D visualization.

Block diagram

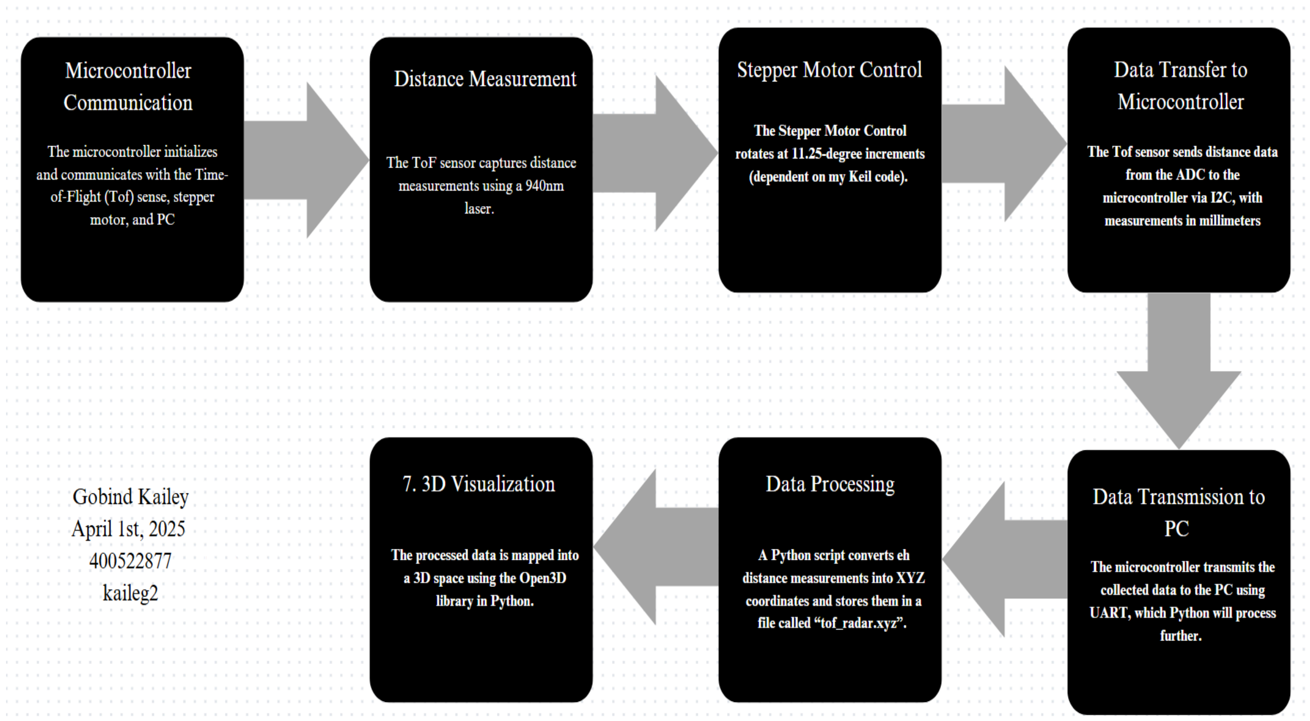

Microcontroller Communication
The microcontroller initializes and communicates with the Time-of-Flight (Tof) sense, stepper motor, and PC
Distance Measurement
The ToF sensor captures distance measurements using a 940nm laser.
Stepper Motor Control
The Stepper Motor Control rotates at 11.25-degree increments (dependent on my Keil code).
Data Transfer to Microcontroller
The Tof sensor sends distance data from the ADC to the microcontroller via I2C, with measurements in millimeters
Data Transmission to PC
The microcontroller transmits the collected data to the PC using UART, which Python will process further.
Data Processing
A Python script converts eh distance measurements into XYZ coordinates and stores them in a file called "tof_radar.xyz".
7. 3D Visualization
The processed data is mapped into a 3D space using the Open3D library in Python.
Gobind Kailey
April 1st, 2025
400522877
kaileg2

Figure 1: Block diagram: Data flow chart

Device Characteristics Table

Microcontroller	
Bus Clock Speed	10 MHz
Measurement Status	Onboard LED D1 (PN1)
UART Tx	Onboard LED D2 (PN0)
Additional Status (Toggle Motor)	Onboard LED D3 (PF4)
Onboard Pushbuttons	PJ0, PJ1
Baud Rate, I2C Transmission Rate	115200 bps baud, 167k bits/s I2C
ULN2003 Stepper Motor Driver	
Supply Voltage	5V pin
Ground	GND pin
IN1 – IN4 Pin	PL0 – PL3 pins
VL53L1X ToF Sensor	
Supply Voltage	3.3V pin
Ground	GND pin
SCL pin	PB2 pin
SDA Pin	PB3 pin
Software and Tools	
Microcontroller Code	Keil uVision (programmed in C)
Python Version	3.11.9
Python Tool for 3D Visualization	Open3D
Python Tool for Serial Communication	Pyserial
Port	COM3

Table 1: Device Characteristic Table

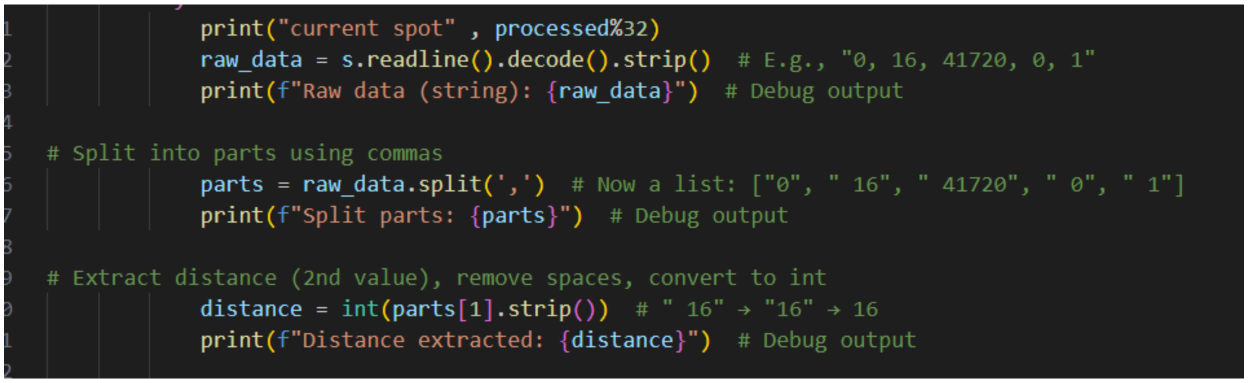
Detailed Description

Distance Measurement: Acquisition

The ToF sensor measures distance by emitting an inferred light (940nm) pulse towards a specific direction, and it calculates the time it takes to reflect off an object and return to the sensor. The distance is then calculated through the equation $distance\ = \frac{time * C}{2}$. In this equation, the C represents the speed of light, and the reason we divide by 2 is that we are traveling there and back. The ToF measures distance in millimeters (mm). The Tof sensor transmits measurement data to the microcontroller using the I2C, which allows efficient two-wire serial communication between devices. The serial data line (SDA) is the line used to transmit data between the sensor and the microcontroller. The serial clock line (SCL) provides a clock signal that synchronizes the data transmission between the sensor and the microcontroller. The number of measurements I took depended on the angle. Furthermore, in this project, I programmed my stepper motor to rotate clockwise and whenever it hit 11.25 degrees it would read a measurement and transfer it to my PC using UART. As a result, per-rotation I had 32 distance measurements read and processed. My onboard LEDs (PN1 and PN0) would blink simultaneously at every 11.25-degree approach because I would receive and transmit data at approximately the same time. To mitigate the problem caused by wires getting in the way of the sensor, I implemented my code to spin 360 degrees clockwise while retrieving 32 data points, and then spin counter-clockwise for 360 degrees retrieving no data points and untangling wires.

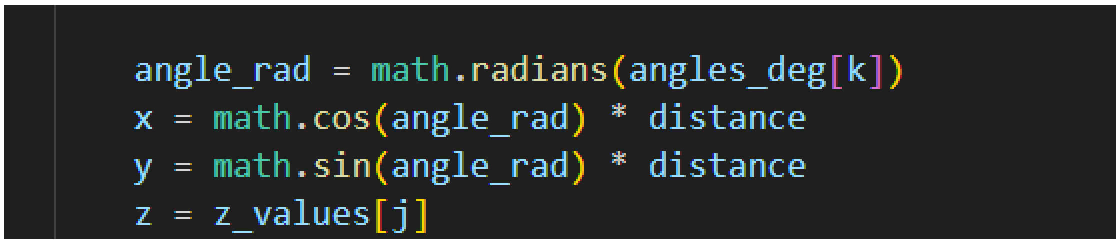
Distance Measurement: Data processing

For this project, through I2C we first received a time value as previously mentioned, and then it got converted to $distance\ = \frac{time * C}{2}$. At every 11.25 degrees, the distance value was fed into my Python code through URAT. I received a string, which held (RangeStatus, Distance, SignalRate, ambientRate, and SpadNum). I used to strip and split to manipulate this string and create an array for retrieving the index 1 value (distance). After retrieving that value I used $(X\ =\ cos(angle)\ *\ distance)$ and $(Y\ =\ sin(angle)\ *\ distance)$ to get the x and y values. For the Z value, I pre-set an array (figure 4) that would be iterated each time 32 measurements were taken. The figures below showcase the code used to modify the distance value and then convert it to the x-y-z plane.



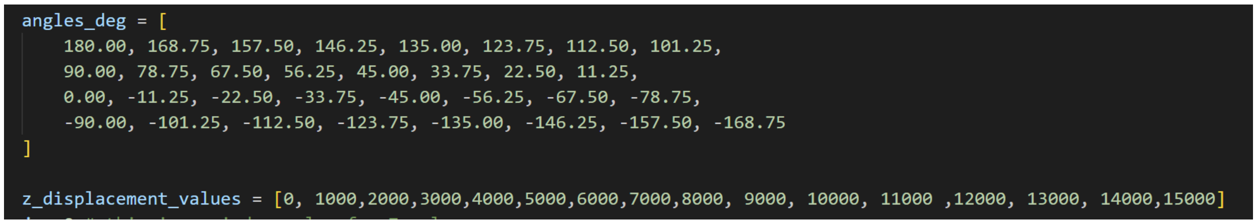
```
            print("current spot" , processed%32)
            raw_data = s.readline().decode().strip()  # E.g., "0, 16, 41720, 0, 1"
            print(f"Raw data (string): {raw_data}")  # Debug output

    # Split into parts using commas
            parts = raw_data.split(',')  # Now a list: ["0", " 16", " 41720", " 0", " 1"]
            print(f"Split parts: {parts}")  # Debug output

    # Extract distance (2nd value), remove spaces, convert to int
            distance = int(parts[1].strip())  # " 16" → "16" → 16
            print(f"Distance extracted: {distance}")  # Debug output
```

Figure 2: Showcasing how data is manipulated to receive distance



```
    angle_rad = math.radians(angles_deg[k])
    x = math.cos(angle_rad) * distance
    y = math.sin(angle_rad) * distance
    z = z_values[j]
```

Figure 3: Retrieving/ setting values of X, Y, Z



```
angles_deg = [
    180.00, 168.75, 157.50, 146.25, 135.00, 123.75, 112.50, 101.25,
    90.00, 78.75, 67.50, 56.25, 45.00, 33.75, 22.50, 11.25,
    0.00, -11.25, -22.50, -33.75, -45.00, -56.25, -67.50, -78.75,
    -90.00, -101.25, -112.50, -123.75, -135.00, -146.25, -157.50, -168.75
]

z_displacement_values = [0, 1000,2000,3000,4000,5000,6000,7000,8000, 9000, 10000, 11000 ,12000, 13000, 14000,15000]
```

Figure 4: Angles and Z_displacement values preset

Distance Measurement: Formula

The formula used is $distance = \frac{time * C}{2}$ to measure the distance in millimeters. An example of converting from distance to x-y-z values is as follows. Let's say I got a distance value of 226 mm. To obtain my x-y values, I would use:

$$X = cos(angle) * distance$$
$$= cos(\pi) * 226 = -226mm$$

$$Y = sin(angle) * distance$$
$$= sin(\pi) * 226 = 181.06\, mm$$

We can set Z = 100 mm

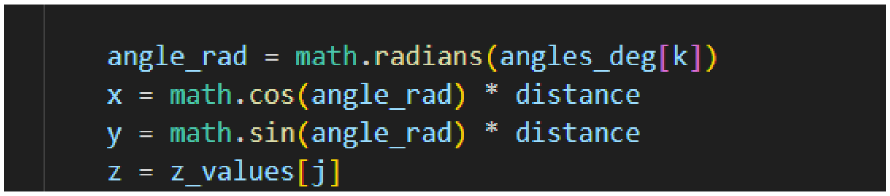
Overall this format produces an x-y-z coordinate plane when more points are taken.

Distance Measurement: Implemented Displacement

Displacement played a paramount role in this project, as without it, we would not have calculated any x-y values for a 3D location. Setting a displacement enables us to create better visuals and understand the dynamics of a specific room. To implement this, I created an array of Z values, through my loop I would loop through for 32 steps to complete one rotation. Once complete, I would increment the array index, enabling us to step forward in real life, in accordance with the distance measurement. This can also be noticed from the array in Figure 4.

Visualization

To obtain a 3D visual model, I used Python as my main coding language. Within Python, I used the Numpy Library to convert the point cloud data into a NumPy Array, which aided with efficient processing. I used the PySerial Library to communicate with a microcontroller via serial UART connection, resulting in an exchange of measurements captured by the ToF sensor. I used the Open3D library for reading, processing, and visualizing data, producing an interactive 3D environment from the scanned location. I also used the Math library for basic math calculations using cos and sin, which can also be seen in the figure below.



```
angle_rad = math.radians(angles_deg[k])
x = math.cos(angle_rad) * distance
y = math.sin(angle_rad) * distance
z = z_values[j]
```

Figure 3

Application Note, Instructions, and Expected Output

Instructions:

1. Installation Requirements for Windows Only

Install version 3.11.9 from the Python Website

- **Install PySerial Library**
  - Open CMD in Start
  - Locate the desired directory
  - Type “pip install pyserial” and press enter
- **Install Open3D**
  - Open CMD in Start
  - Locate the desired directory
  - Type “pip install open3D” and press enter

Python Installation and Verification:

1. Open CMD in Start
2. Type “python” and press enter
3. If installed correctly it will display the version number, (make sure you have installed Python correctly before proceeding).

Keil uVisison Installation and Verificaiton:

1. **Connect microcontroller to PC**
2. **Confirm target setting are correctly set:**
   - ARM Compiler: “Use default compiler version 6”
   - Go into debug and Use: Find “CMSIS-DAP Debugger” in the drop-down menu
   - Go into settings: under CMSIS-DAP - JTAG/SW Adapter set it to XDS110
   - Finally under C/C++ (AC6) changed optimization to -O0
3. **Translate build and load the C code into the microcontroller**

2. Wiring microcontroller wires to the desired components

Using the device characteristic table or the circuit schematic in Figure 16, Configure your code and connect the stepper motor and ToF sensor to the microcontroller and your PC. In this design, there is no need to have an external pushbutton as PJ0 and PJ1 which are onboard and configured as input.

3. Setup Python code for URAT communication and other measurements

Find your specific port under the device manager and follow Figure 5 to locate the port number. (COM3 in my case) and changed the first parameter in Figure 6 to that specific port.

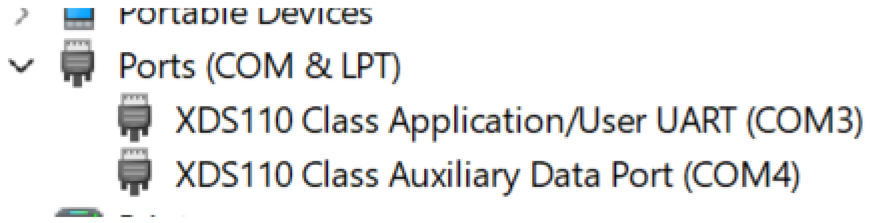

Portable Devices
Ports (COM & LPT)
XDS110 Class Application/User UART (COM3)
XDS110 Class Auxiliary Data Port (COM4)

Figure 5: Finding ports in Device Manager

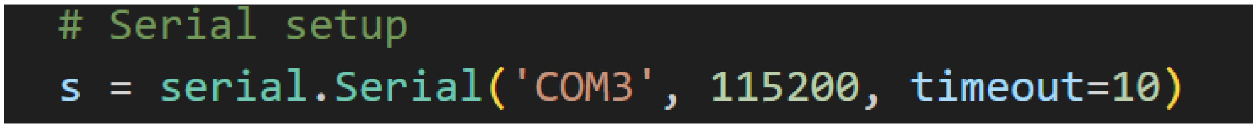

```
# Serial setup
s = serial.Serial('COM3', 115200, timeout=10)
```

Figure 6: Serial Setup

Depending on the number of horizontal distance scans you want to complete, you can change the "layers" parameter value layers%32. In our case 96 would correspond to 3 layers.

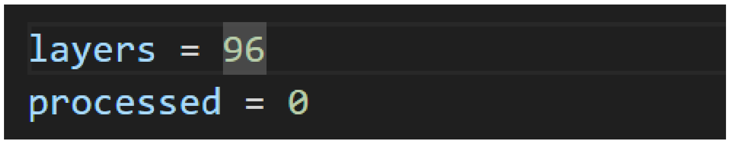
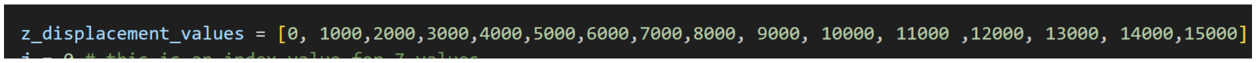

```
layers = 96
processed = 0
```

Figure 7: Showcasing layers variables that can be set

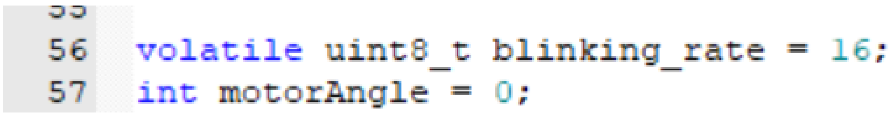
In my code, I have set my Z-displacement to increment 1000mm, and this value could be modified to obtain a shorter or greater displacement depending on the requirements of the room. At present, we can notice 16 indexes, but this value could be increased manually to account for more steps of displacement.



```
z_displacement_values = [0, 1000,2000,3000,4000,5000,6000,7000,8000, 9000, 10000, 11000 ,12000, 13000, 14000,15000]
```

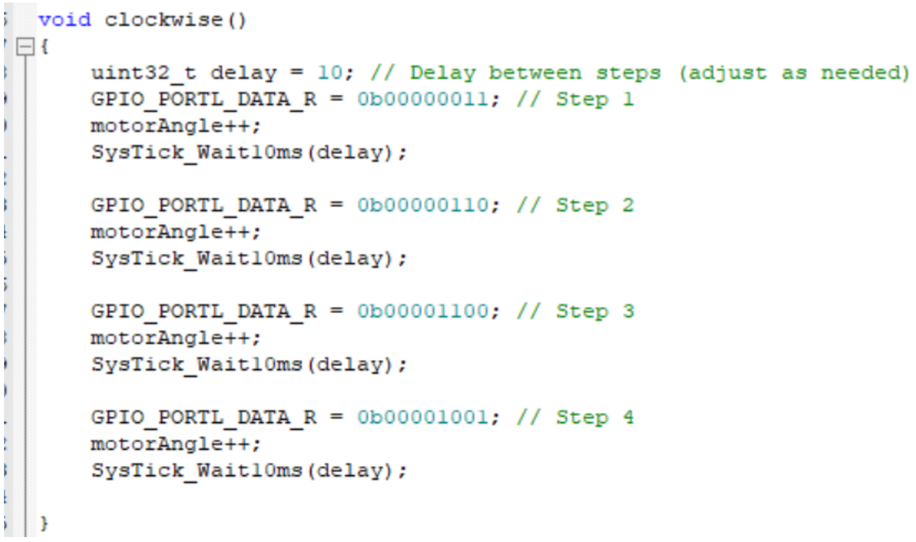
Figure 8: Z values

As described above we can calculate the X-Y values, while setting the Z-values for displacement. Currently, my code is hard coded for 32 data points for 1 rotation at 11.25 degrees each. This can be easily modified by changing the variable called “blinking_rate” in Keil to 64 if you want to transfer data every 45 degrees. To calculate the blinking_rate value you would do 360/desired value, for finding 45 degrees, we would be 360/45 = 8. And because I am incrementing my “motorAngle” by 4 (Figure 10), which represents the steps we are on out of 2048, we would do 2048/4 = 512. Finally, we would do 512/8 = 64 steps to achieve an angle of 45 degrees.



```
volatile uint8_t blinking_rate = 16;
int motorAngle = 0;
```

Figure 9: Blinking rate and motorAngle initialization



```
void clockwise()
{
    uint32_t delay = 10; // Delay between steps (adjust as needed)
    GPIO_PORTL_DATA_R = 0b00000011; // Step 1
    motorAngle++;
    SysTick_Wait10ms(delay);

    GPIO_PORTL_DATA_R = 0b00000110; // Step 2
    motorAngle++;
    SysTick_Wait10ms(delay);

    GPIO_PORTL_DATA_R = 0b00001100; // Step 3
    motorAngle++;
    SysTick_Wait10ms(delay);

    GPIO_PORTL_DATA_R = 0b00001001; // Step 4
    motorAngle++;
    SysTick_Wait10ms(delay);

}
```

Figure 10: Clockwise function used to spin motor

4. Run the microcontroller and Python file

1. Translate, build, and load the C program onto the microcontroller through Keil.
2. Run the Python file
3. Press enter on the Python terminal to start receiving data.
4. Hit reset on the microcontroller
5. Toggle the PJ0 button to start rotating the stepper motor and receive data every 11.25 degrees.

6. Press PJ1 if you would like to keep the motor running but stop receiving data.

5. Visulaize the Scan

After you have received a specific amount of scans, your Python code will open an external file (assuming no errors), which displays rainbow dots on your screen, that correlate to the room you just scanned, and if you close that out, you will see a better 3D visualization that represents the area you just scanned.

Expected Output Scan

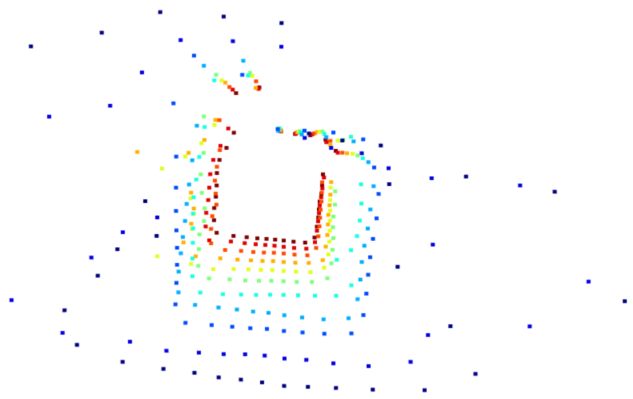
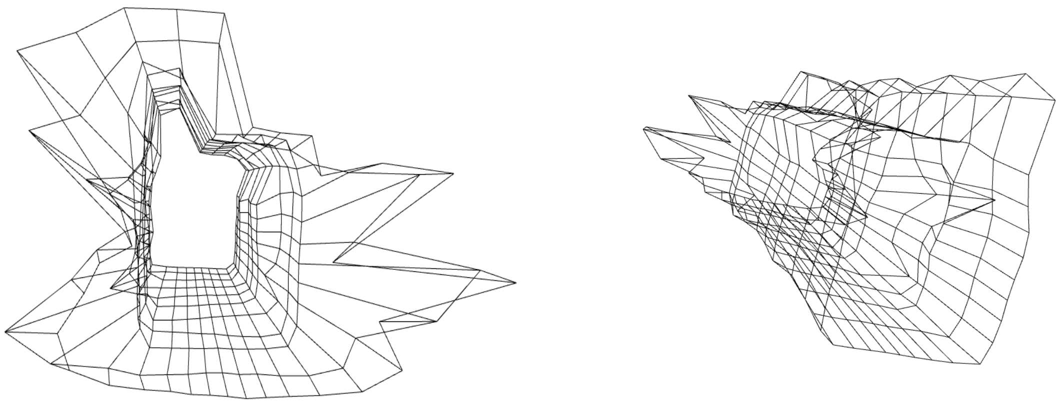


Figure 11: Preview of 3D visualization



Figures 12 and 13: Data Visualization

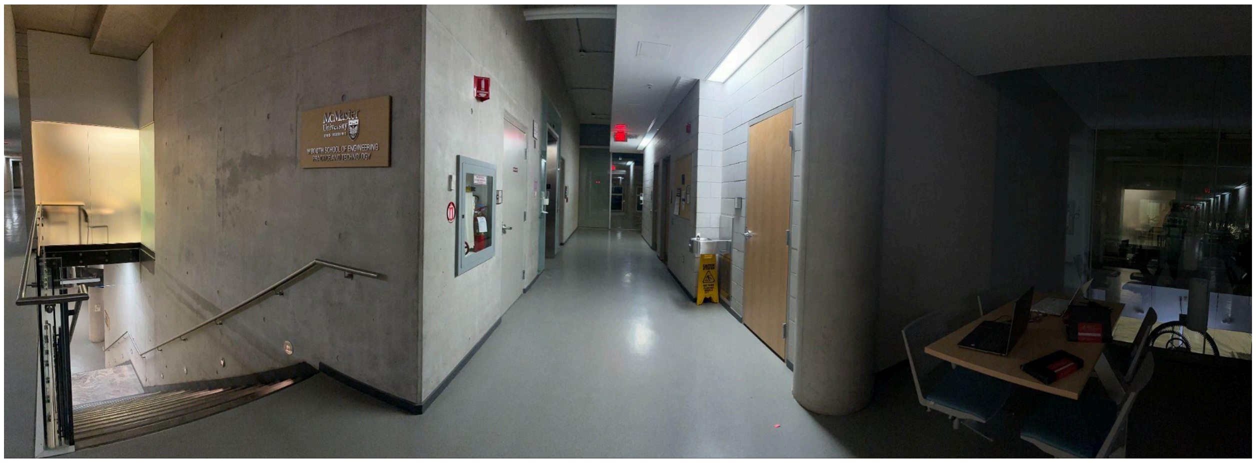


Figure 14: Live Photo of specific assigned location

As you can see the live photo and the data visualization resemble one another as expected. To prove that they are similar we can point out some points; you can notice in Figure 14 that the ceiling goes higher on the left side, we can also notice the right side wall is wider at first, and then it gets a block smaller (near the water fountain), we can also see the elevator on the left side. From Figures 12 and 13, this can also be apparent. All of these points can be noticed in all images and proves I have successfully implemented this project.

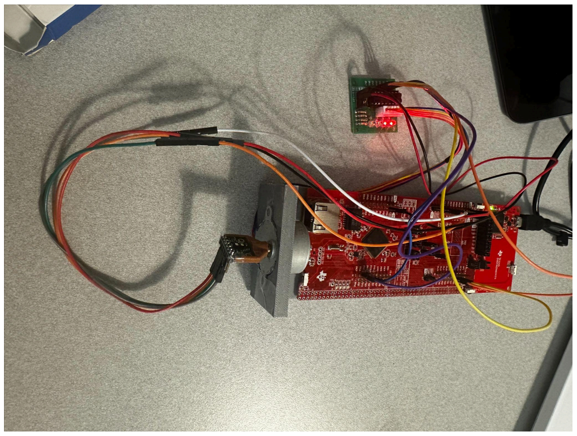


Figure 15: Live Photo of Component Connections

Limitations

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

   The microcontroller has limitations relating to the floating point, the floating point unit (FPU) only supports single-precision calculations, which means that it is a 32-bit floating point arithmetic efficiently. This is not ideal in all cases because we may have calculations that require higher precision like (double-precision) 64-bit operations in hardware. Furthermore, this would mean that software needs to do this, which leads to slower performance. While using floating-point arithmetic, it is hardware accelerated, conversely trigonometric functions (sin, cos, tan) would be relatively slow, as they are computed using software-based math libraries. When both limitations combine (single precision and trig calculations) this can result in reduced accuracy and slower performance, especially in applications that can require recursion or precise floating point trigonometric computations.

2. **Calculate your maximum quantization error for the ToF module.**

   Max quantization error represents the max difference between the actual analog value and its quantized digital value. The Max quantization value is also known to be equal to the resolution, and the datasheet mentions it to equal 1mm[2]. Although this MQE can be suitable for numerous applications, it can also be a limitation for precision-critical tasks where even small discrepancies can lead to negative outcomes.

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

   The maximum standard serial communications rate that can be implemented is 128,000 bps. This can be verified at the properties of the COM port in the device manager and found that the maximum baud rate that was supported was 128,000 bps. This was also verified through testing and checking which baud rate can be used.

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

The communication method used between the microcontroller and ToF models was an inter-integrated circuit (I2C) protocol. There are multiple I2C speed modes you can use; standard mode: 100 KHz, Fast mode: 400 KHz, for this project I used 167K bits/s, and that was implemented by changing the value of MTPR value in Keil to correspond to the 10MHz bus speed.

5. **Reviewing the entire system, which elements are the primary limitation on speed? How did you test this?**

Through a review of the entire system, some of the elements that are the primary limiting factor of the system speed are the time required to acquire distance data from the ToF, and the time the stepper motor is required to rotate to the next position. The ToF sensor has a programmable timing budget ranging from 20 ms to 1000 ms. In long-distance mode, this budget is set to 140 ms, meaning each measurement takes 140 ms. These delays are significantly greater than the time required for serial communication, making them a dominant constraint on system speed. To test this, I looked at the quantity and the time required for data to be serially transmitted.

Circuit Schematic

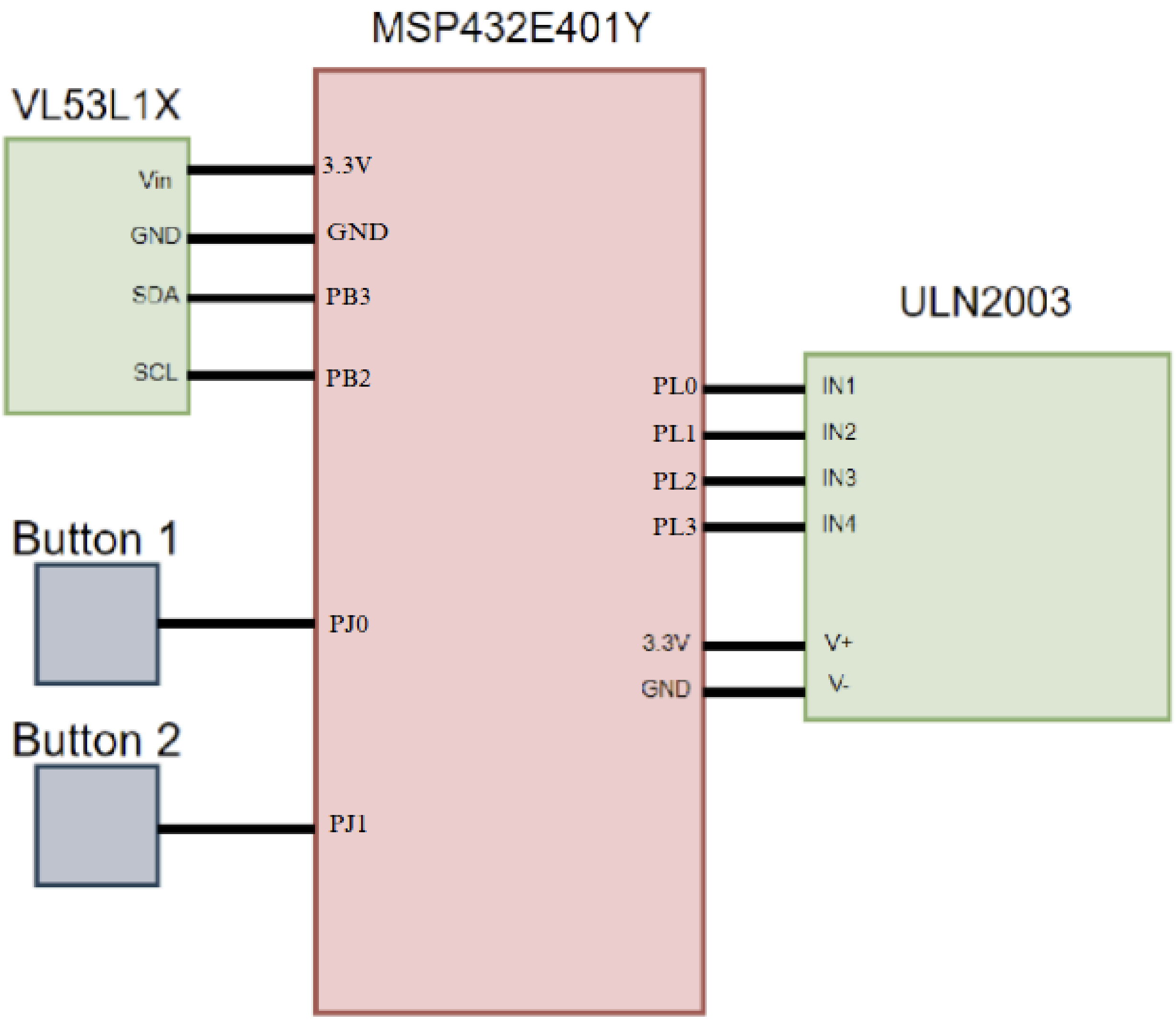
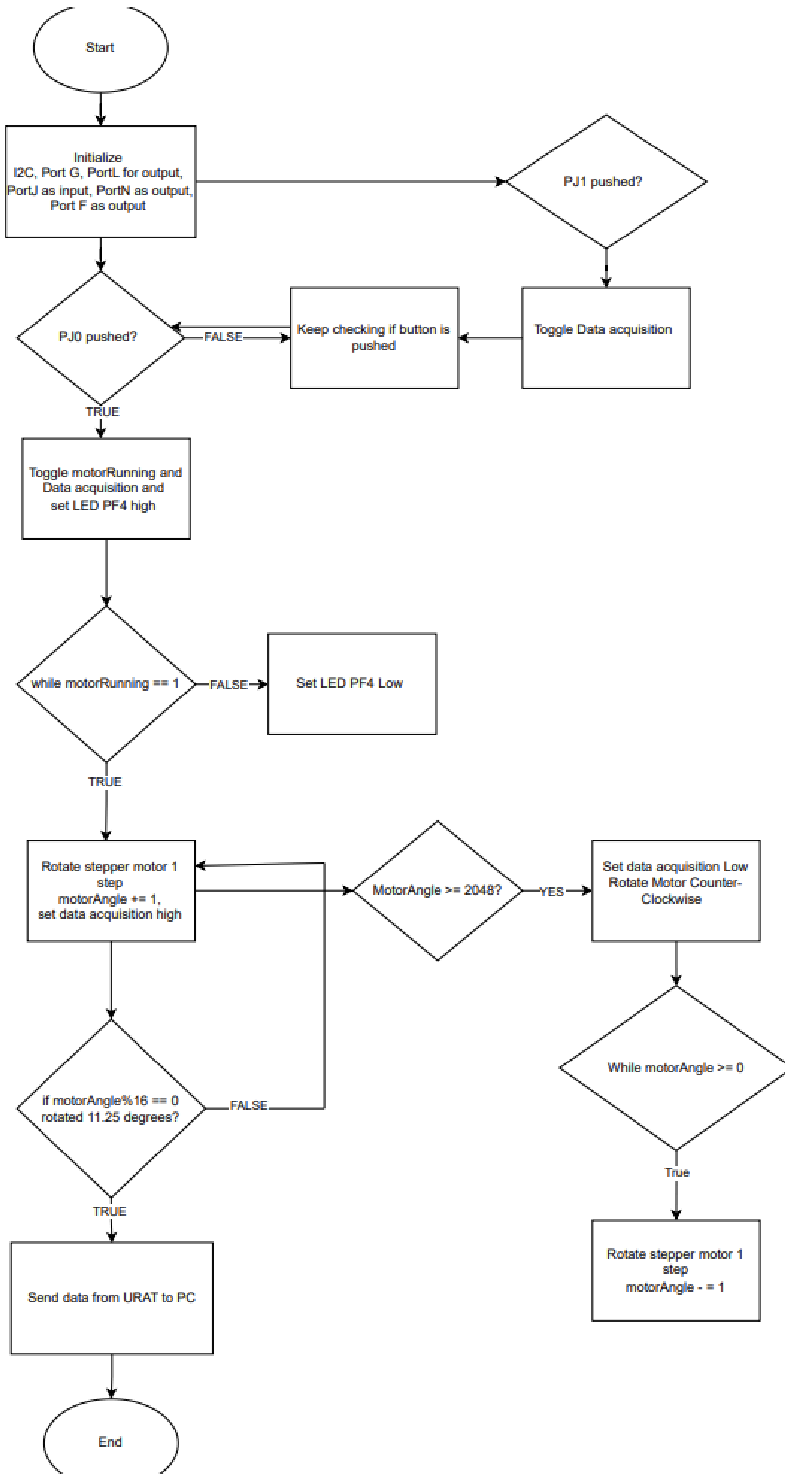


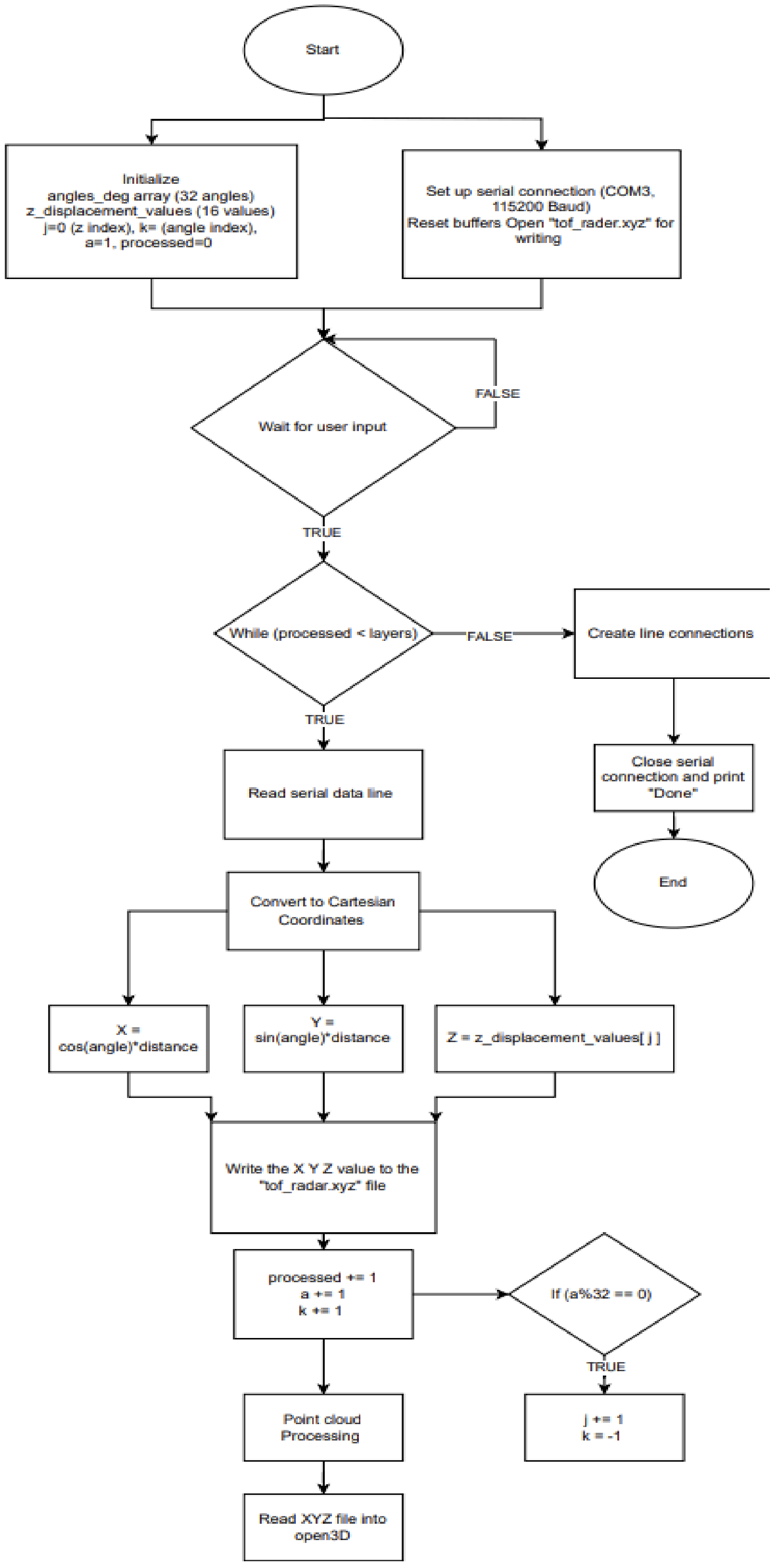
Figure 16: Circuit Schematic of System Design

Programming Logic Flowchart(s)

Keil Code:


Start
Initialize
I2C, Port G, PortL for output,
PortJ as input, PortN as output,
Port F as output
PJ1 pushed?
PJ0 pushed?
FALSE
Keep checking if button is pushed
Toggle Data acquisition
TRUE
Toggle motorRunning and Data acquisition and set LED PF4 high
while motorRunning == 1
FALSE
Set LED PF4 Low
TRUE
Rotate stepper motor 1 step
motorAngle += 1,
set data acquisition high
MotorAngle >= 2048?
YES
Set data acquisition Low
Rotate Motor Counter-Clockwise
While motorAngle >= 0
True
Rotate stepper motor 1 step
motorAngle - = 1
if motorAngle%16 == 0
rotated 11.25 degrees?
FALSE
TRUE
Send data from URAT to PC
End

Python Code Flow-Chart:


Start
Initialize
angles_deg array (32 angles)
z_displacement_values (16 values)
j=0 (z index), k= (angle index),
a=1, processed=0
Set up serial connection (COM3, 115200 Baud)
Reset buffers Open "tof_rader.xyz" for writing
Wait for user input
FALSE
TRUE
While (processed < layers)
FALSE
Create line connections
TRUE
Read serial data line
Close serial connection and print "Done"
End
Convert to Cartesian Coordinates
X = cos(angle)*distance
Y = sin(angle)*distance
Z = z_displacement_values[j]
Write the X Y Z value to the "tof_radar.xyz" file
processed += 1
a += 1
k += 1
If (a%32 == 0)
TRUE
j += 1
k = -1
Point cloud Processing
Read XYZ file into open3D

References

[1] “MSP432E401Y data sheet, product information and support | TI.com.” https://www.ti.com/product/MSP432E401Y?utm_source=google&utm_medium=cpc&utm_campaign=epd-null-null-gpn_en_rsa-cpc-pf-google-ww_en_cons&utm_content=msp432e401y&ds_k=%7B_dssearchterm%7D&DCM=yes&gad_source=1&gclid=CjwKCAjwktO_BhBrEiwAV70jXnKq3XzYrnhTygnFrWxsxAzjgeJbvJA92NNN-5IeLwnXDWnhWyeL6hoCgtgQAvD_BwE&gclsrc=aw.ds#tech-docs

[2] “Pololu - VL53L1X Time-of-Flight Distance Sensor Carrier with Voltage Regulator, 400cm Max.” https://www.pololu.com/product/3415/specs