2DX: Microprocessor Systems
Final Project

Instructors: Drs. Boursalie, Doyle, and Haddara

Yasmin Souza – souzay – 400325538 – COMPENG 2DX3 – Wednesday Afternoon – L03

Table of Contents

Device Overview

Features:

- VL53L1X Time-of-Flight sensor
    - Operating voltage: 3.3V
    - Measuring range: Maximum of 4m
- 28BY J-48 Stepper Motor
- ULN2003 Stepper motor Driver
    - Operating voltage: 5V
- SimpleLink MSP432E401Y microcontroller
    - Bus speed: 40 MHz
    - Clock Speed: 120 MHz
    - 2 12-Bit SAR-based ADC modules
    - ADC sampling rate: up to 2 million samples per second
    - Programmed in C
    - 1 MB of Flash Memory with 4-Bank Configuration
    - 256KB of SRAM with Single-Cycle Access
- Visualization programmed in Python using open3D library
- Serial Communication
    - I2C: Used between ToF sensor and microcontroller
    - UART: Used between PC and microcontroller
    - Baud rate: 115200 bps
- Total Cost: Approximately $220
    - Including breadboard, microcontroller, USB, ToF sensor, stepper motor, and wires / cables

General Description:

This device is an embedded spatial measurement system consisting of a microcontroller, time-of-flight (ToF) sensor, stepper motor, and external push button. The ToF sensor is attached to the stepper motor to take a 360° measurement of the surrounding distance within a single geometric plane. To take a distance measurement, the ToF sensor emits a laser pulse. It then measures the time it takes for the laser to hit the target and reflect back on itself. By multiplying this with the speed of light and diving by two, the ToF calculates the distance to the target and returns this value in millimetres. The device can be carried through an area such as a hallway to take multiple of these measurements at regular distances along the orthogonal axis. From here, the device uses serial communication to send the data to a personal computer. Once this data is received by a PC through UART serial communication, Python code is used to create a 3D visualization of the data gathered. This device can be used to generate a graphical representation of a hallway or small room with a width of up to 8m. The output should resemble the hallway layout closely. Some discrepancies may arise from the sensor's wires interfering with the sensor.

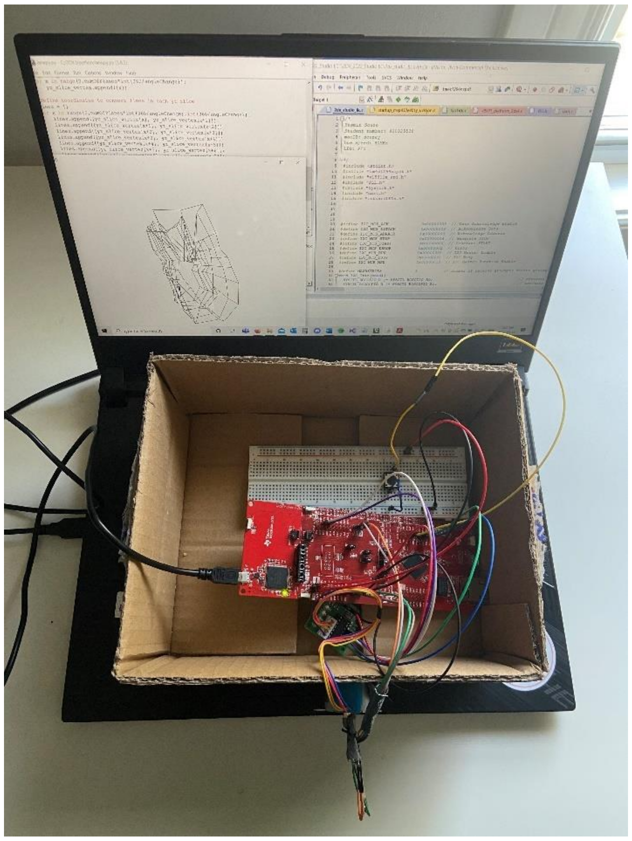
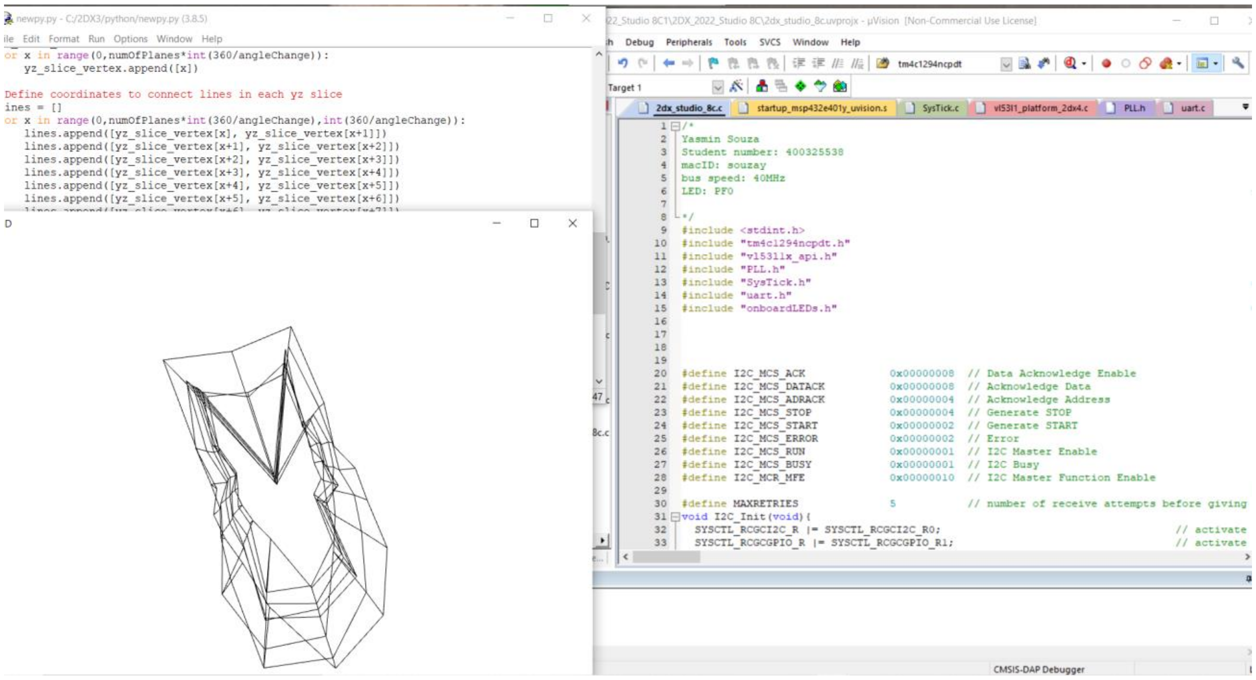


Figure 1.1: Picture of the entire system



newpy.py - C:/2DX3/python/newpy.py (3.8.5)
ile Edit Format Run Options Window Help

```
or x in range(0,numOfPlanes*int(360/angleChange)):
   yz_slice_vertex.append([x])

Define coordinates to connect lines in each yz slice
ines = []
or x in range(0,numOfPlanes*int(360/angleChange),int(360/angleChange)):
   lines.append([yz_slice_vertex[x], yz_slice_vertex[x+1]])
   lines.append([yz_slice_vertex[x+1], yz_slice_vertex[x+2]])
   lines.append([yz_slice_vertex[x+2], yz_slice_vertex[x+3]])
   lines.append([yz_slice_vertex[x+3], yz_slice_vertex[x+4]])
   lines.append([yz_slice_vertex[x+4], yz_slice_vertex[x+5]])
   lines.append([yz_slice_vertex[x+5], yz_slice_vertex[x+6]])
```

22_Studio 8C1\2DX_2022_Studio 8C\2dx_studio_8c.uvprojx - µVision [Non-Commercial Use License]
h Debug Peripherals Tools SVCS Window Help
tm4c1294ncpdt
Target 1
2dx_studio_8c.c
startup_msp432e401y_uvision.s
SysTick.c
vl53l1_platform_2dx4.c
PLL.h
uart.c
CMSIS-DAP Debugger

```
/*
Yasmin Souza
Student number: 400325538
macID: souzay
bus speed: 40MHz
LED: PF0

*/
#include <stdint.h>
#include "tm4c1294ncpdt.h"
#include "vl53l1x_api.h"
#include "PLL.h"
#include "SysTick.h"
#include "uart.h"
#include "onboardLEDs.h"




#define I2C_MCS_ACK             0x00000008  // Data Acknowledge Enable
#define I2C_MCS_DATACK          0x00000008  // Acknowledge Data
#define I2C_MCS_ADRACK          0x00000004  // Acknowledge Address
#define I2C_MCS_STOP            0x00000004  // Generate STOP
#define I2C_MCS_START           0x00000002  // Generate START
#define I2C_MCS_ERROR           0x00000002  // Error
#define I2C_MCS_RUN             0x00000001  // I2C Master Enable
#define I2C_MCS_BUSY            0x00000001  // I2C Busy
#define I2C_MCR_MFE             0x00000010  // I2C Master Function Enable

#define MAXRETRIES              5           // number of receive attempts before giving
void I2C_Init(void){
  SYSCTL_RCGCI2C_R |= SYSCTL_RCGCI2C_R0;                          // activate
  SYSCTL_RCGCGPIO_R |= SYSCTL_RCGCGPIO_R1;                        // activate
```

Figure 1.2: Screenshot of PC display when the program is finished running

Block Diagram:

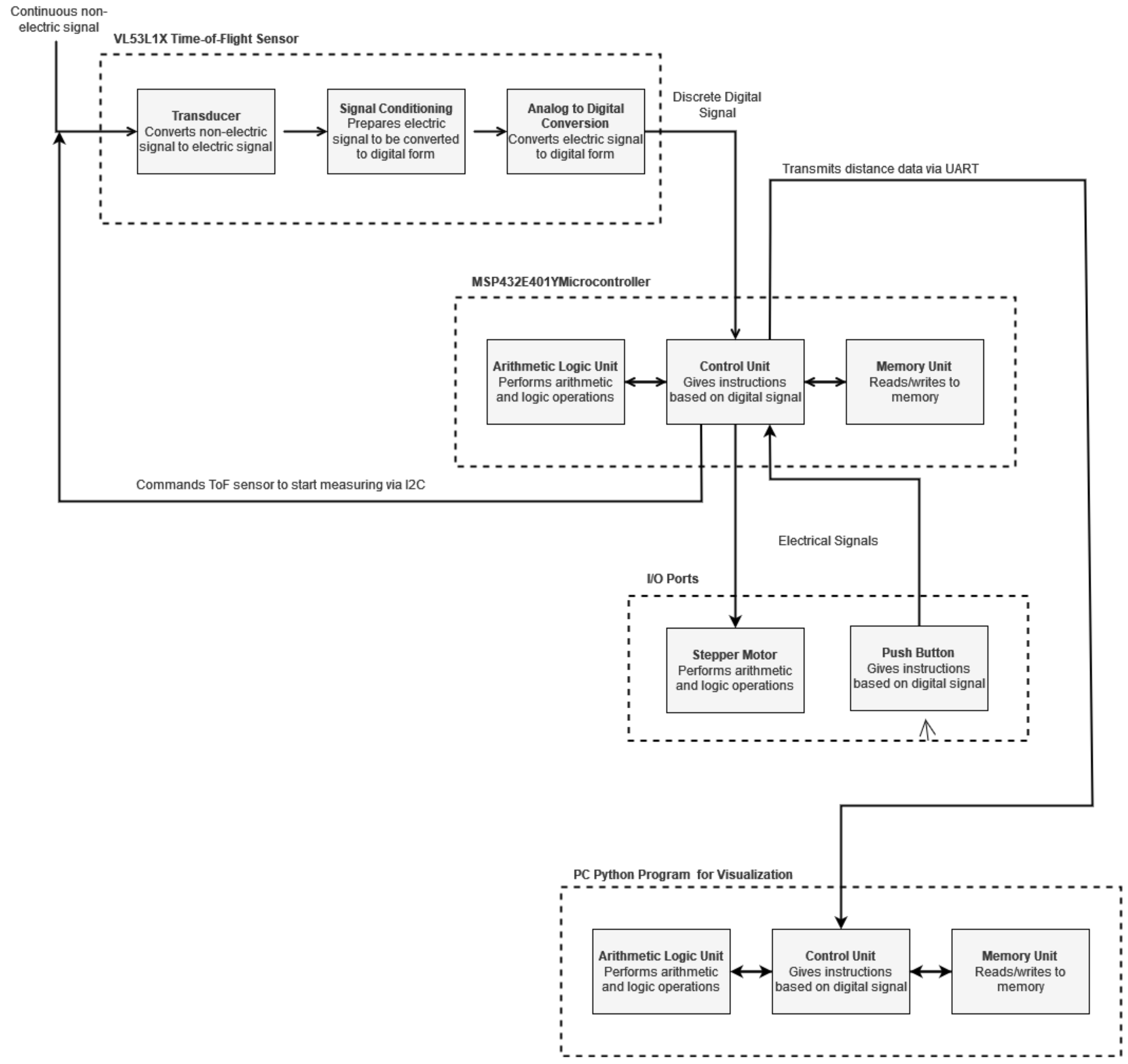

Continuous non-electric signal
VL53L1X Time-of-Flight Sensor
Transducer
Converts non-electric signal to electric signal
Signal Conditioning
Prepares electric signal to be converted to digital form
Analog to Digital Conversion
Converts electric signal to digital form
Discrete Digital Signal
Transmits distance data via UART
MSP432E401YMicrocontroller
Arithmetic Logic Unit
Performs arithmetic and logic operations
Control Unit
Gives instructions based on digital signal
Memory Unit
Reads/writes to memory
Commands ToF sensor to start measuring via I2C
Electrical Signals
I/O Ports
Stepper Motor
Performs arithmetic and logic operations
Push Button
Gives instructions based on digital signal
PC Python Program for Visualization
Arithmetic Logic Unit
Performs arithmetic and logic operations
Control Unit
Gives instructions based on digital signal
Memory Unit
Reads/writes to memory

Figure 1.3: Block Diagram of Spatial Mapping System

Device Characteristic Tables

Table 2.1: Pin Setup of MSP432E401Y

Pin Name	**Description**
GPIO B2	SCL via I2C
GPIO B3	SDA via I2C
GPIO [H0:H3]	Stepper Motor Output using PWM
GPIO M0	Input for Button Press
5V	Vin for Stepper Motor
3V3	Vin for ToF and push button
GND	GND for ToF, Stepper Motor, and push button

Table 2.2: Communication Parameters

Parameter	Value
UART Bus Speed	40MHz
UART Baud Rate	115200bps
Serial Port	COM3

Detailed Device Description

The computer used in this application was an Asus ROG Strix G15 using an Intel Core i7-10750H CPU, with 16GB of RAM. It is running a 64-bit operating system, Windows 10 Home Version 20H2. The programming for this system was done in Keil uVision5 coded in C to acquire the distance measurement and send it to the PC as well as Python 3.8 to visualize the data.

Distance Measurement:

To commence data acquisition, the user must press the push button. The C code for the microcontroller continuously polls for a button press and calls a function to rotate the motor once the button press occurs. This function causes the motor to spin 365° in the clockwise direction. To accomplish this rotation, the ULN2003 Stepper motor Driver energizes the coils of the stepper motor in a logical order according to the code. At each 22.5° interval, the motor pauses its movement, turns on the onboard LED PF0, and calls the measure function. The measure function sends a signal to the time-of-flight sensor to begin taking a measurement of the distance via I2C communication.

The VL53L1X time-of-flight sensor measures its distance to the target with the use of LIDAR (light detection and ranging). This method begins with the time-of-flight sensor emitting a pulse of infrared light. The target reflects the light back to the sensor where it is collected by the time-of-flight receiver. The sensor then measures the time delay between the emitted and received signals and calculates the distance to the target in millimetres using Equation 3.1 below.

$$Measured\ Distance = \frac{photon\ travel\ time}{2} * speed\ of\ light$$

Equation 3.1: Measuring Distance from Time Delay

Once a full spin of the motor is complete with measurements taken at every 22.5°, the motor spins another 365° in the counter-clockwise direction without taking measurements to prevent the wires from getting tangled. The spin function is then called seven more times where this process is repeated for the user to take multiple samples of data, walking a fixed amount of 30cm down the hallway between each scan.

The time-of-flight sensor communicates the ranging measurements to the microcontroller through the I2C interface. From here, the microcontroller sends the measurements to the PC by UART communication. The UART serial communication port used to communicate this data is COM3 with a baud rate of 115200 bps.

Visualization:

The python program for visualization uses four different libraries: PySerial, math, Open3D and NumPy. PySerial gives us access to the serial port COM3 for communication where we receive the distance measurements taken by the time-of-flight sensor. The math library is then used in order to transform these distances into XYZ coordinates where x is taken to be the length of the hallway, y is taken to be the height, and z is taken to be the width. The formulas outlined below in Equation 3.2 were used for this calculation. J is iterated 16 times for each sample to get the angle, theta, for each of the 16 measurements per rotation. X initialized to be 0.3, which is the distance in meters the user should walk between taking each sample. Once the angle is acquired, the XYZ coordinates could be easily calculated through basic trigonometric functions with the constant x displacement value and distance measurement. The XYZ coordinates are then written to a new file called "demofile2dx.xyz".

$$theta = (2 - (j * angleChange)/180) * math.pi - math.pi$$
$$x = x$$
$$y = distance * math.\sin(theta)$$
$$z = distance * math.\sin(theta)$$

Equation 3.2: Formulas used for calculating the XYZ coordinates

Now that the coordinates have been written to the new file, Python's Open3D library can be used to generate a point cloud of the coordinates. After these points are plotted, each vertex is given a unique number. Then, coordinates are defined to connect the lines in each YZ slice, as well as coordinates to connect lines between the current and next YZ slice. Finally, this program uses the open3D library to recreate a three-dimensional representation of the area.

Application Example

For the coordinate system of this application, the x-axis is defined as the length of the hallway (parallel to the path in which the user moves the device), the y-axis is the height, and the z-axis is the width of the hallway

Steps:

1. Follow the system schematic diagram in Figure 7.1 to make the appropriate connections between the devices.
2. Connect the USB-A cable plug in to an available USB port on the PC and plug the Micro-B plug to the XDS-110 USB port located at the top of the MSP-EXP432E401Y microcontroller.
3. Verify that the red LED on the left side of the XDS-110 is on.
4. Verify that COM3 has been assigned to the device.
5. Begin running the Python program. (Note: Should receive "Waiting for data…" message as the microcontroller has not sent data to the PC yet)
6. Position the system so that the motor is pointing down the hallway, the computer is stable, and it is held at approximately one third to half the height of the hallway.
7. Press the push button to begin data acquisition.
8. Once the system has taken 16 measurements and the motor begins rotating backwards, move 30cm down the hallway.
9. Repeat step 8 for seven more samples.
10. Upon completion, a 3D rendering of the hallway chosen should be outputted from the Python code.

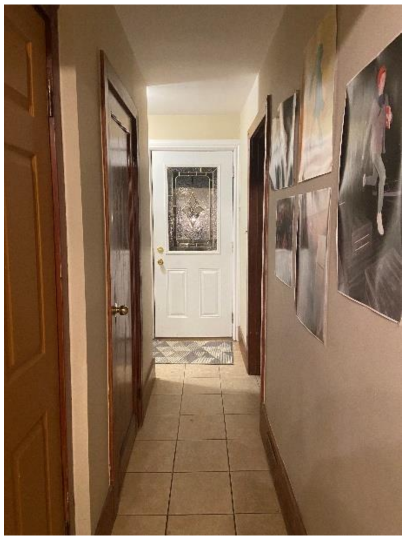


Figure 4.1: Hallway for reconstruction

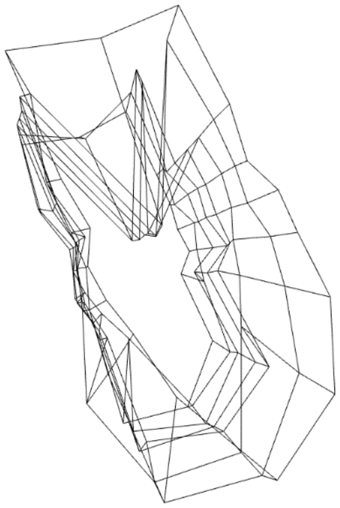


Figure 4.2: Graphical representation from program

<u>Note for Application Example:</u> As can be observed in Figure 4.2, in the graphical reconstruction, there appears to be a spike on the ceiling of the hallway that is not actually there. This was caused by the wires from the time-of-flight sensor interfering with the sensor during rotation, yielding some inaccuracies in the results. To maximize the accuracy of the system, the user should try to keep the wires away from the sensor when it is taking a measurement.

User Guide

For the coordinate system of this device, the x-axis is defined as the length of the hallway (parallel to the path in which the user moves the device), the y-axis is the height, and the z-axis is the width of the hallway

Steps:

1. The push button, 28BY J-48 stepper motor, and the VL53L1X time-of-flight sensor must be connected to the MSP432E401Y microcontroller.
2. Follow the system schematic diagram in Figure 7.1 to make the appropriate connections.
3. Connect the USB-A cable plug in to an available USB port on the PC and plug the Micro-B plug to the XDS-110 USB port located at the top of the MSP-EXP432E401Y microcontroller.
4. Verify that the red LED on the left side of the XDS-110 is on.
5. The bus speed is set to 40 MHz. To modify this on the Keil uVisions5 project, change the PSYSDIV value in the PLL.h file. The bus speed will be equivalent to 480MHz/(PSYSDIV+1).
6. If the user decides to change the bus speed, the user must also adjust the SysTick_Wait10ms function in the SysTick.c file. The delay will equal the bus speed multiplied by the value in SysTick_Wait.
7. Verify that COM3 has been assigned to the device and that the baud rate is 115200 bps.

Limitations

Limitations of the microcontroller floating point

the MSP432E401Y microcontroller has a Floating-Point Unit (FPU) that supports mathematical and arithmetic calculations with both signed and unsigned integers. However, the FPU allows only single-precision floating point for 32-bit CPUs, whereas double-precision floating points can be applied for 64-bit CPUS. This decreases the precision of our system.

Maximum quantization error the time-of-flight module:

Max quantization error for ToF module = 400mm/ 2^16

Max quantization error for ToF module = 0.061035

Max quantization error for IMU module = 32g/ 2^16

Max quantization error for IMU module= 0.00479

Maximum standard serial communication rate with the PC:

The maximum standard serial communication that can be implemented on the Asus ROG Strix G15 is 128000 bps. This can be verified under the device manager.

Communication method and speed used between the microcontroller and the ToF modules:

To communicate between the microcontroller and ToF modules, I2C serial communication protocol was used. They communicated with each other with a 40KHz frequency.

Primary limitation on speed:

Through a review of the entire system, it is evident that the time-of-flight sensor is the primary limitation on speed. This is mainly because it takes a long time to get ready to start ranging, as well as the time it takes to start and stop between every scan. It must also take 16 scans 8 times, which is very time consuming considering these factors.

Circuit Schematic

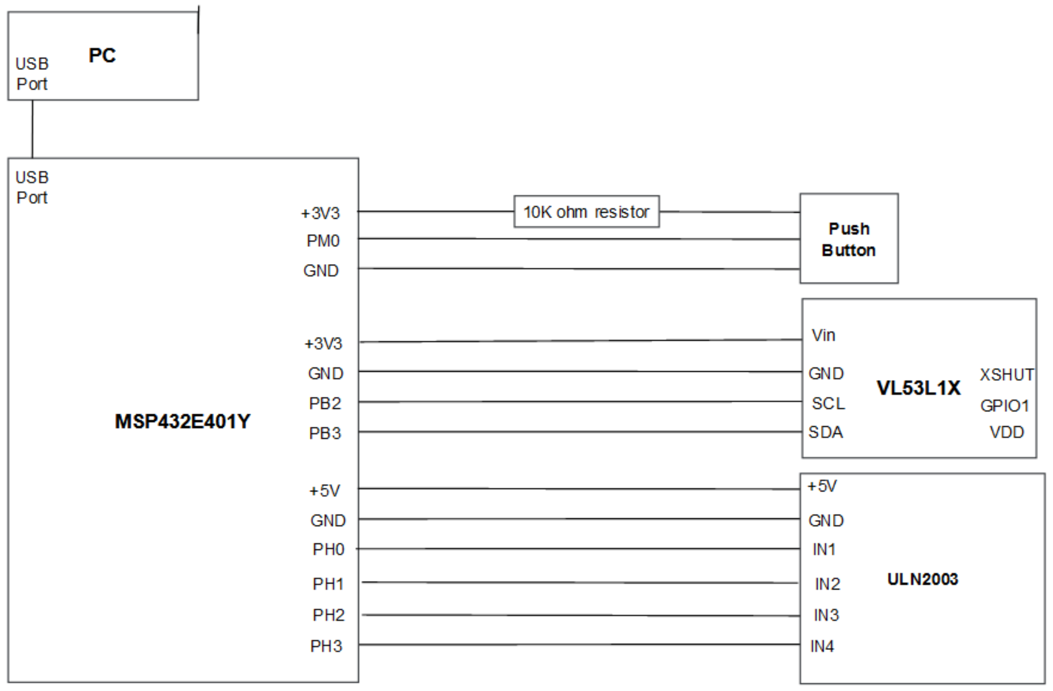


Figure 7.1: Wiring Diagram for Spatial Mapping System

Programming Logic Flowcharts

Keil Program Logic Flowchart

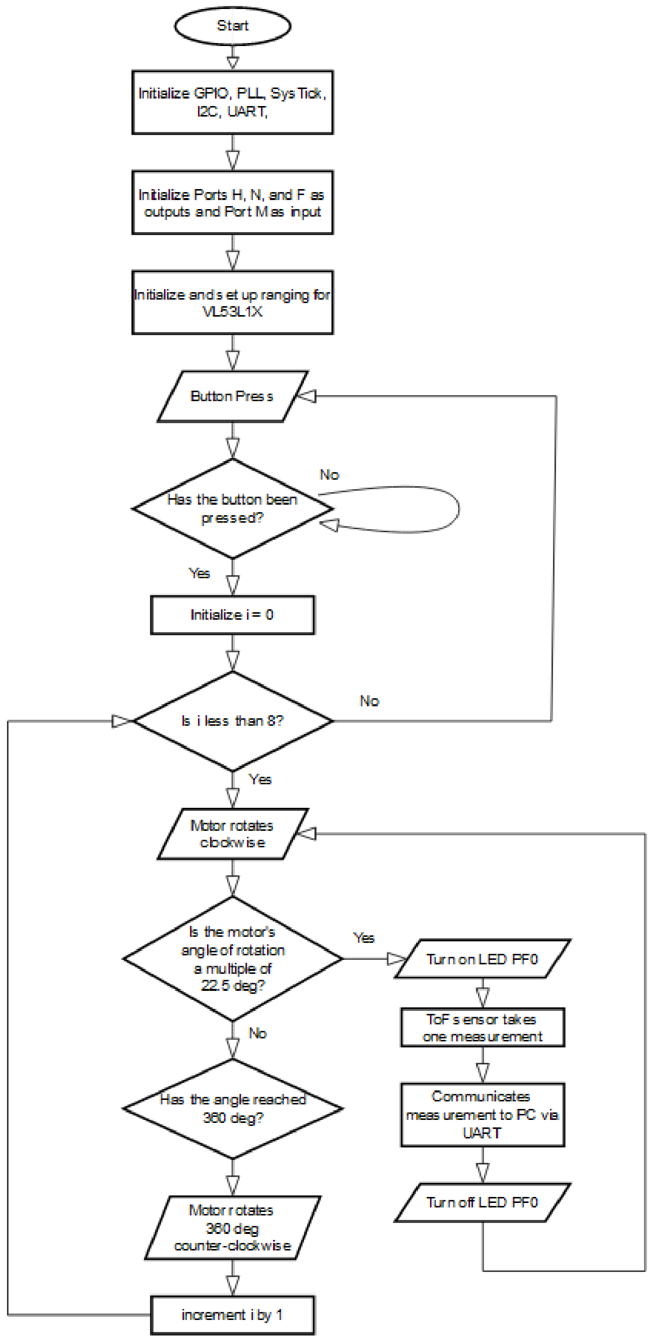

Start
Initialize GPIO, PLL, SysTick, I2C, UART,
Initialize Ports H, N, and F as outputs and Port M as input
Initialize and set up ranging for VL53L1X
Button Press
Has the button been pressed?
No
Yes
Initialize i = 0
Is i less than 8?
No
Yes
Motor rotates clockwise
Is the motor's angle of rotation a multiple of 22.5 deg?
Yes
Turn on LED PF0
ToF sensor takes one measurement
Communicates measurement to PC via UART
Turn off LED PF0
No
Has the angle reached 360 deg?
Motor rotates 360 deg counter-clockwise
increment i by 1

Figure 8.1: Flow Chart for Keil Program

Python Program Logic Flowchart

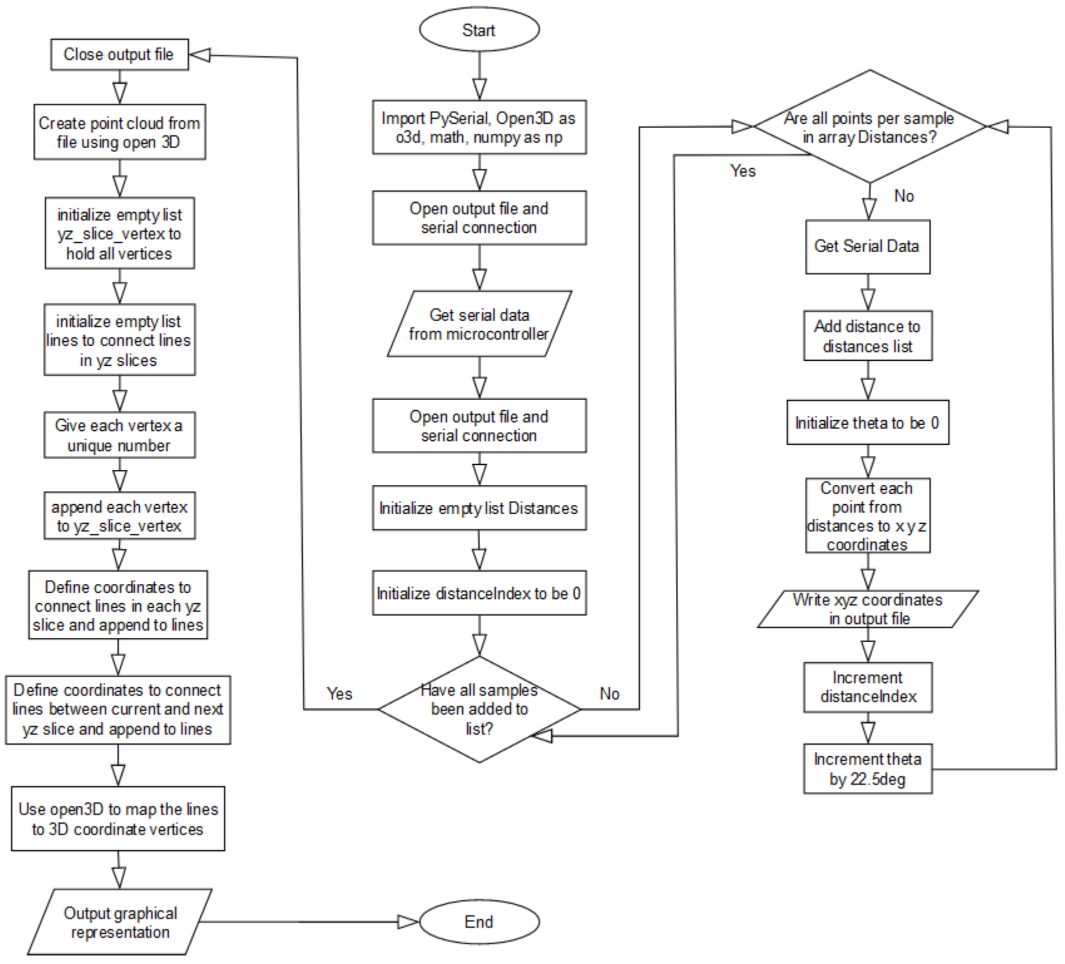

Start
Import PySerial, Open3D as o3d, math, numpy as np
Open output file and serial connection
Get serial data from microcontroller
Open output file and serial connection
Initialize empty list Distances
Initialize distanceIndex to be 0
Have all samples been added to list?
Yes
No
Are all points per sample in array Distances?
Yes
No
Get Serial Data
Add distance to distances list
Initialize theta to be 0
Convert each point from distances to x y z coordinates
Write xyz coordinates in output file
Increment distanceIndex
Increment theta by 22.5deg
Close output file
Create point cloud from file using open 3D
initialize empty list yz_slice_vertex to hold all vertices
initialize empty list lines to connect lines in yz slices
Give each vertex a unique number
append each vertex to yz_slice_vertex
Define coordinates to connect lines in each yz slice and append to lines
Define coordinates to connect lines between current and next yz slice and append to lines
Use open3D to map the lines to 3D coordinate vertices
Output graphical representation
End

Figure 8.2: Flow Chart for Python Program