2DX4: Microprocessor Systems Project
Final Project

Instructors: Dr. Bruce, Dr. Haddara, Dr. Hranilovic, Dr. Shirani

Joshua Chan – L04 – chanj94 1- 400194033

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 Joshua Chan, chanj94, 400194033.

Design Questions Google Drive Folder: https://drive.google.com/open?id=1v2NJ8DC48C-ZskJVYYaaX7QUmdfT8Dk8

Design Question 1: https://drive.google.com/open?id=1uAb8fLT9K_buh40-sILw6gi1kMqMOqMV
Design Question 2: https://drive.google.com/open?id=1u8YaXH7eIQyjpFLpPWnwNl3bD8E9AyAJ
Design Question 3: https://drive.google.com/open?id=1u14r8_7yklSKxd4JQX23B5l1z4LxLH_a

Contents

1 Device Overview

1.1 Features

- Data acquisition device used to map indoor environments
- Can be used as a component of other systems (e.g., robotics navigation, autonomous drone, layout mapping, etc.)
- Generates a 3D model of a room
- Fully integrated compact module
    - Microcontroller with a 120-MHz Arm® Cortex® -M4F Processor Core
        - Bus/Eclock Speed: 48 MHz
        - Operating Voltage: 3.3-5V
        - Simplex Serial Communication
            - Baud Rate: 115200 bits/s
        - Memory Resources:
            - 1024KBof Flash Memory
            - 256KB of SRAM
            - 6KB EEPROM
    - VL53L1X with a range of 4m
    - ULN2003 Stepper Motor providing field of view of 360°
- Programmed in C and Python
- Low cost (~$100)

1.2 General Description

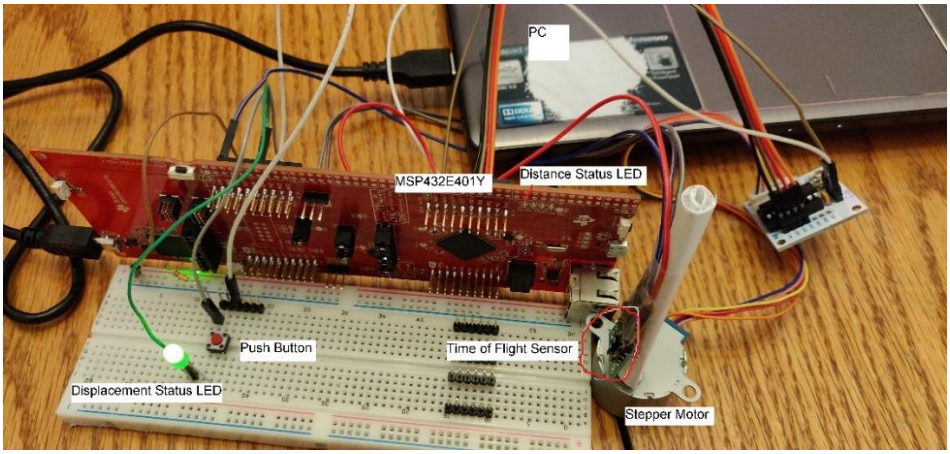

Figure 1: Complete Device

The project is an embedded system used to map indoor environments. It measures distances at 45° increments per revolution (8 points of measurements/revolution). Distance is measured using the VL53L1X Time-of-Flight sensor. The sensor is mounted on and rotated by the ULN2003A-PCB Stepper Motor. The processes are controlled using the SimpleLink™ Ethernet MSP432E401Y MCU LaunchPad™ Development Kit Board. A push button starts the program to take the distance measurements. The program follows a polling design. Each distance measurement is indicated by a blinking onboard LED. The status of the device is indicated by an

external LED. The data is sent from the device using Serial Simplex Communication. The Python processes the measurements to be visualized as a 3D model.

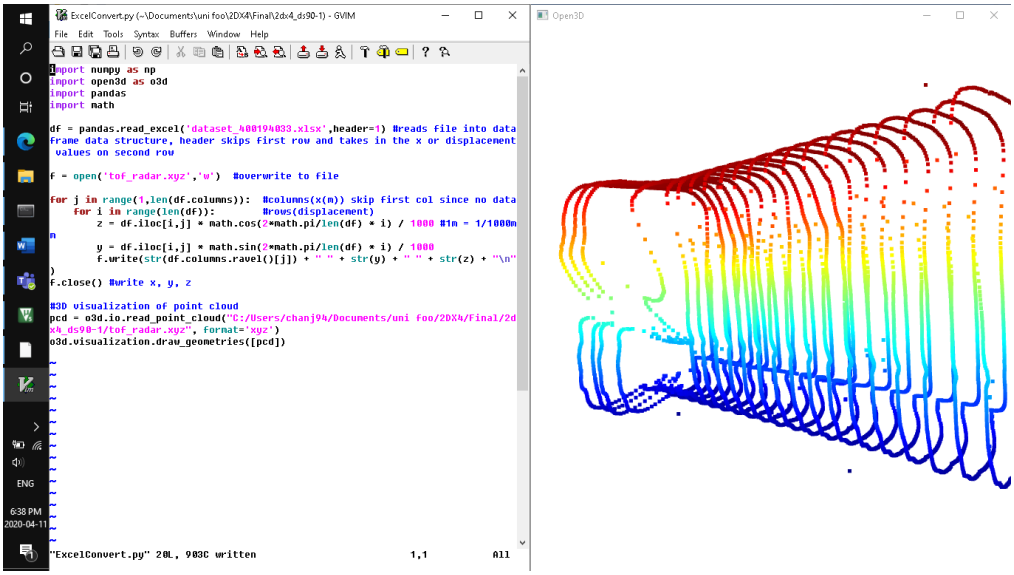

Figure 2: 3D Visualization of a hallway

1.3 Block Diagram (Data flow graph)

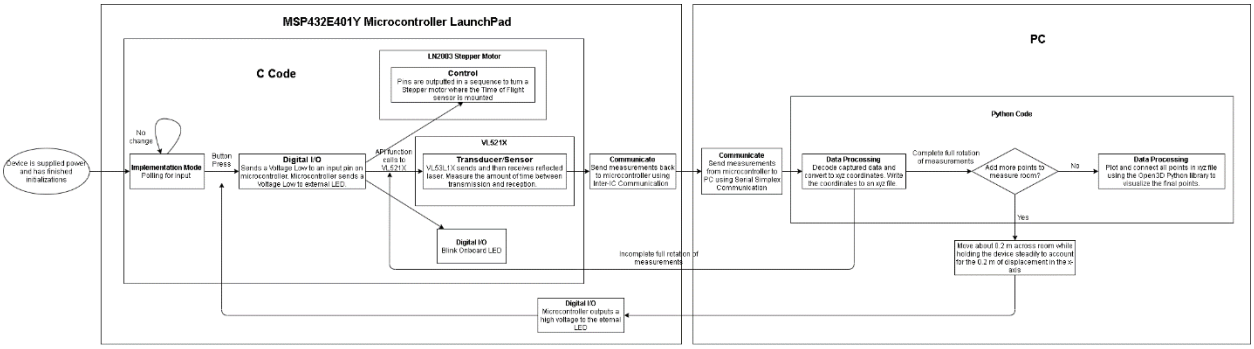

Figure 3: Device Data Flow chart

2 Device Characteristics

Component	**Description**
Microcontroller	
	Pins
Displacement Status LED	PL4
Distance Status LED	PF0
Push Button	PE0
Stepper Motor	PK[0-3]
ToF Sensor	PB2 to SCL, PB3 to SDA
	Quantity
Bus Speed	48 MHz
Serial Port	COM4

Communication Speed	115200 bits/s

3 Detailed Description

3.1 Distance Measurement

The distance measurement starts with a momentary switch. The momentary switch connects a pull-up resistor to ground when pressed. The pull up resistor (10k) is connected in series with the VCC (3.3V). An input pin (PE0) on the microcontroller (MSP432E401Y) is connected to the same node where the switch and the resistor are connected. PE0 reads a logic high when the button is not pressed. Please refer to **Figure 6: Circuit Schematic**.

The microcontroller and python program 1 should already be running when a distance measurement is taken. The microcontroller runs the C Code which initializes the PLL, SysTick, the onboard LEDs, the I2C, the UART, and ports. The VL53L1X is then booted, initialized with default settings, and ranging is enabled. The C code then uses the polling method by entering an infinite while loop. The port L4 is configured to output a logic high. This output turns on an external LED which signals that the microcontroller is ready for data acquisition. The program will keep looping and checking for whether Port E0 is given an input of a logic low.

When the momentary switch is pressed. A logic low is sent to Port E0 which is read by the microcontroller. The program enters a for loop for a count of eight times for eight measurements in a revolution with an increment of 45°. API function calls in the order VL53L1X_CheckForDataReady, VL53L1X_GetRangeStatus, and VL53L1X_GetDistance are called to prepare the VL53L1X for data acquisition, to get the range status, and to get the distance.

After the API function calls are sent to the VL53L1X, the VL53L1X emits pulses of infrared laser light which are then reflected off the nearest object and back to the detector on the VL53L1X. The VL53L1X measures the amount of time it takes for the emission and reception and calculates the distance in millimeters. The VL53L1X can only measure up to 4m. If VL53L1X is somehow unable to take a proper measurement, eg) the distance is too far, the ranging status will not return a 0. If it able to take a proper measurement, it will return a 0. This value is what is returned from the function call VL53L1X_GetRangeStatus. The distance is returned from the function call VL53L1X_GetDistance. The values are returned to the microcontroller through I2C Communication.

Then, a function is called to rotate the stepper motor. The stepper motor is configured on output ports K[0-3] on the microcontroller. The outputs on the ports are cycled in the sequence: 0b1001,0b1100,0b0110,0b0011 where 64 subsequences turn the stepper motor 11.25°. Therefore, 64 * 4 = 256 subsequences are required to turn the stepper motor 45°. The stepper motor is what turns the VL53L1X.

The measurement data is then sent using the Universal asynchronous receiver-transmitter on the microcontroller using Serial Simplex Communication to serial port COM4 on the PC. The for loop resumes. After that, the while loop resumes polling.

A python program is used to capture each individual measurement data. The python program must be already running before taking the measurement. The serial Python library is used. The serial port COM4 is assigned a baud rate of 115200 bits/s receive the data and is assigned to a variable. A file name "tof_radar.xyz" is overwritten (or created if non-existing) to contain the points of the measured data in Cartesian coordinates. Counter variables are created to count the angle and count the displacement. The program enters an infinite while loop. The data is read and decoded. If the data read receives a Range Status of 0 meaning that it was a successful measurement, the measured range is then assigned to a variable. The angle counter and the range variable represent the radius and angle in a polar coordinate. The following formulas are used to convert from polar coordinates to Cartesian coordinates and in meters:

$y = r * \sin(\theta) / 1000$

$z = r * \cos(\theta) / 1000$

eg) if the measured range was 1599 mm, and angle it was measured at was 135°,

$z = 1599\ mm * \cos(135°) / 1000mm$

$= -1.1307m$

The point is then written to the .xyz file where the displacement counter is used as the x coordinate. The correct format: *x y z*. Each point is on a new line.

The angle counter is then incremented 45°. A conditional is checked. If the counter is at 360° or a full revolution, reset the counter back to 0°. The displacement is then incremented 10mm or 1cm since it is assumed that each measurement will be taken with a displacement of 1 cm apart. The program resumes looping and waits for the next output from the microcontroller.

Two separate programs are required to be running. Refer to **Figure 7: Programming Logic Flowchart.**

3.2 Displacement Measurement

The IMU sensor would have measured displacement by using the Digital Motion Processor (DMP). The DMP ac quires data from accelerometers, gyroscopes, magnetometers, and additional 3rd party sensors, and processes the data. The resulting data can then be read from the DMP's registers. The microcontroller can access the data from the IMU sensor using I2C communication. The measurement would take place after the VL53L1X takes its measurement.

Displacement was implemented in the project by moving about 1cm across the room from the initial starting point while steadily holding the device. The displacement per measurement was kept track of in the Python program.

3.3 Visualization

The computer used was the Lenovo IdeaPad Z580. It is running Windows 10. The processor is Intel® Core™ i7-3520M CPU @ 2.90GHz. The RAM is 10GB. It is a 64-bit Operating System, x64-based processor.

Anaconda was used to run the python programs. The Python version is 3.7.6 and the pip version 20.0.2.

Once a satisfactory amount of points has been measured, another python program is run to create the 3D visualization of the points. The points are stored in an .xyz file name “tof_radar.xyz”. The program imports the numpy and Open3D python library. The “tof_radar.xyz” file is read, and the points are assigned to a variable as a cluster of points using the function: o3d.io.read_point_cloud(*Filename*,format='xyz'). These cluster of points is referred to as a point cloud. The points can be connected together using the function: o3d.geometry.LineSet(…). The finished result should be a window displaying points modelling the measured room. The Open3D library is then used to draw and visualize a 3D model of the points by using the function: o3d.visualization.draw_geometries([*pcdvar*]). The points are plotted on an x,y,z axis. Refer to Python Program 2 in **Figure 7: Programming Logic Flowchart.**

4 Application Example

1) Connect the microcontroller to the PC via USB cable.
2) Allow the microcontroller to finish initializing parameters which is indicated when the external/Displacement Status LED turns on.
3) Run the Python program,”Final.py” to allow the serial port COM4 of the PC to receive data via serial simplex communication.
4) Hold the device steadily. Ensure that the axle of the stepper motor is parallel to the floor so that the axis of rotation is pointing away.

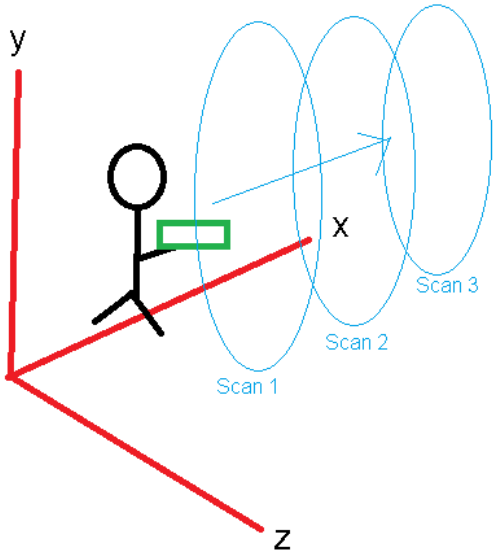

Figure 4: Visualizing x,y,z axes

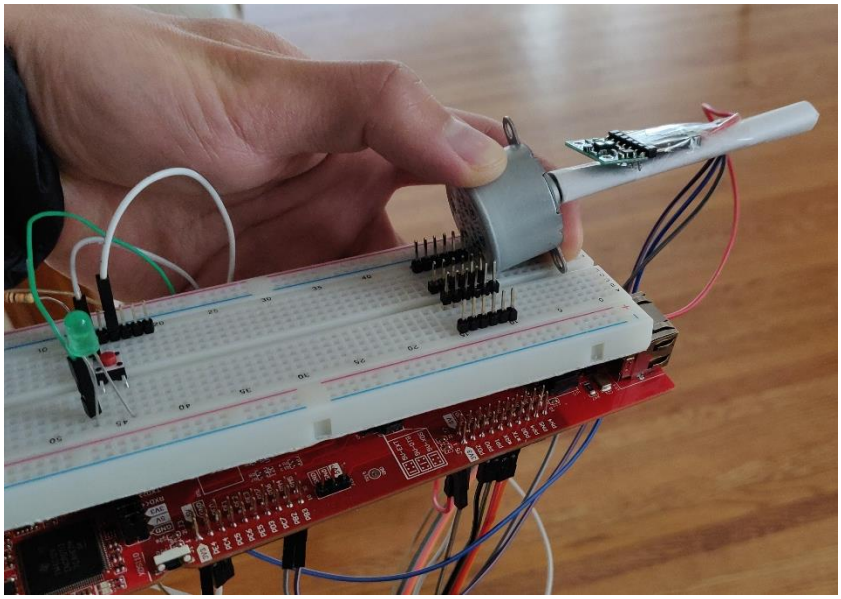

Figure 5: Holding the device with the Stepper motor's axle parallel to floor

5) To start data acquisition, stand in one spot and press the push button.
6) Once the stepper motor stops moving, one whole revolution has been measured, and the external LED should light back up again. The device is ready to begin another scan of data acquisition.
7) Steadily hold the device and move 1 cm in one direction. Keep moving in this direction for the entirety of the measurement.
8) Go back to step 5 to repeat data acquisition if required.
9) If no more data is required, the device can now be put down.
10) Run the second python program “FinalPointCloud.py” to generate a 3D visualization of the points measured.

5 Limitations

The device can only measure up to 4m away. The displacement of the device must be done manually, but precisely. The device must be connected to the PC to store the measurements.

1) The Time of Flight measurements are only taken to 4 significant digits. The math library providing the trigonometric functions only calculates up to 18 significant digits.
2) Max Quantization Error (MQE) = $V_{FS} / 2^m$
   MPU-9250 has an operating voltage range of 2.4-3.6V. It has three 16-bit ADCs.
   MPU MQE = $3.6/2^{16}$
   $= 5.4931 * 10^{-5}$
   VL53L1X has an operating voltage range of 2.6-3.5V. One distance measurement is 16 bits.
   VL53L1X MQE = $3.5/2^{16}$
   $= 5.3405 * 10^{-5}$
3) The maximum standard serial communication rate able to be implemented is 921600 bps with the PC. This value was found by checking the highest available baud rates on the software Application RealTerm.

4) The communication method used was by using Serial Simplex communication and the speed was 115200 bps.
5) The VL53L1X was the primary limitation on speed since the stepper motor always must pause for a bit for the VL53L1X to take a measurement and reconfigure and prepare itself for the next measurement.
6) Nyquist Rate: $f_{sample} >= 2*f_{signal}$
   Typically, the motion processing algorithms should be run at a high rate, often around 200Hz. 400Hz >= 2* 200Hz. Therefore, the sample frequency should be greater than 400Hz.
   If the input signal exceeds this frequency, the sample frequency becomes insufficient to produce an accurate signal and aliasing will occur.

6 Circuit Schematic

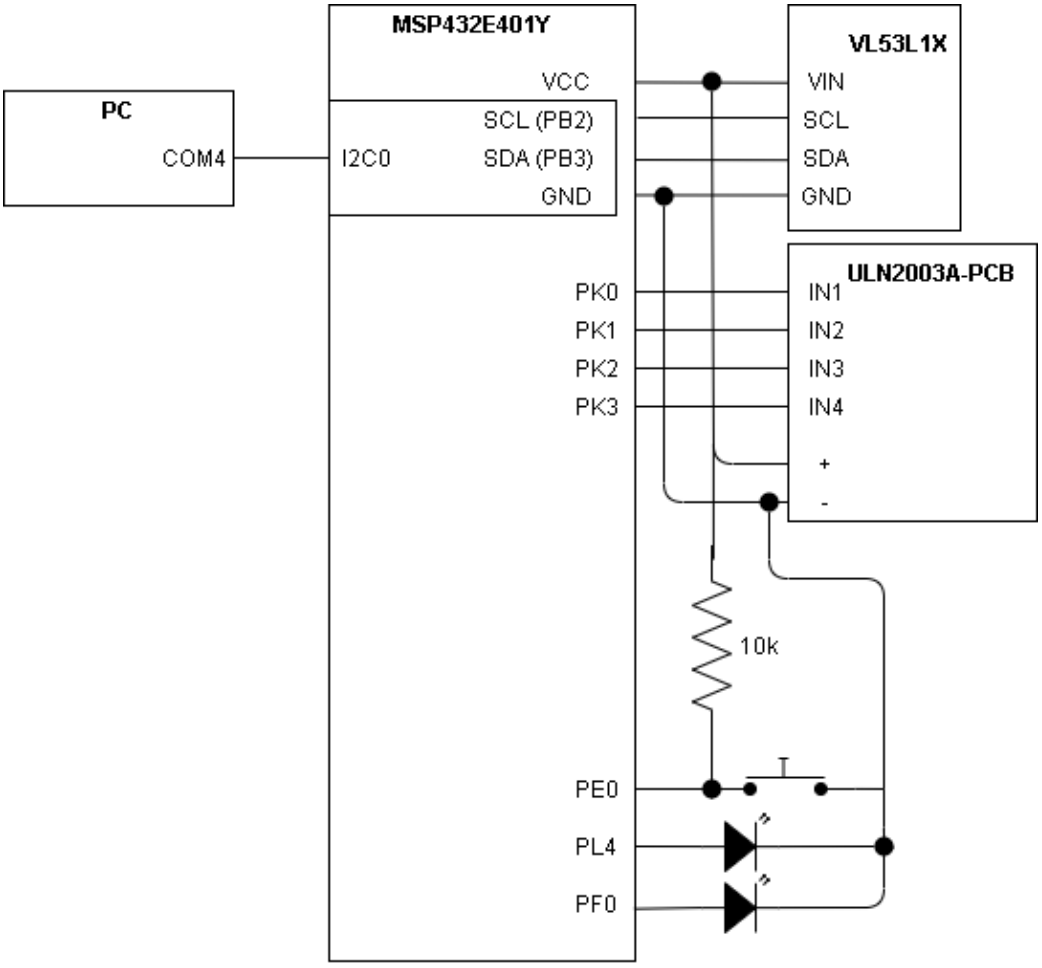

Figure 6: Circuit Schematic

7 Programming Logic Flowchart

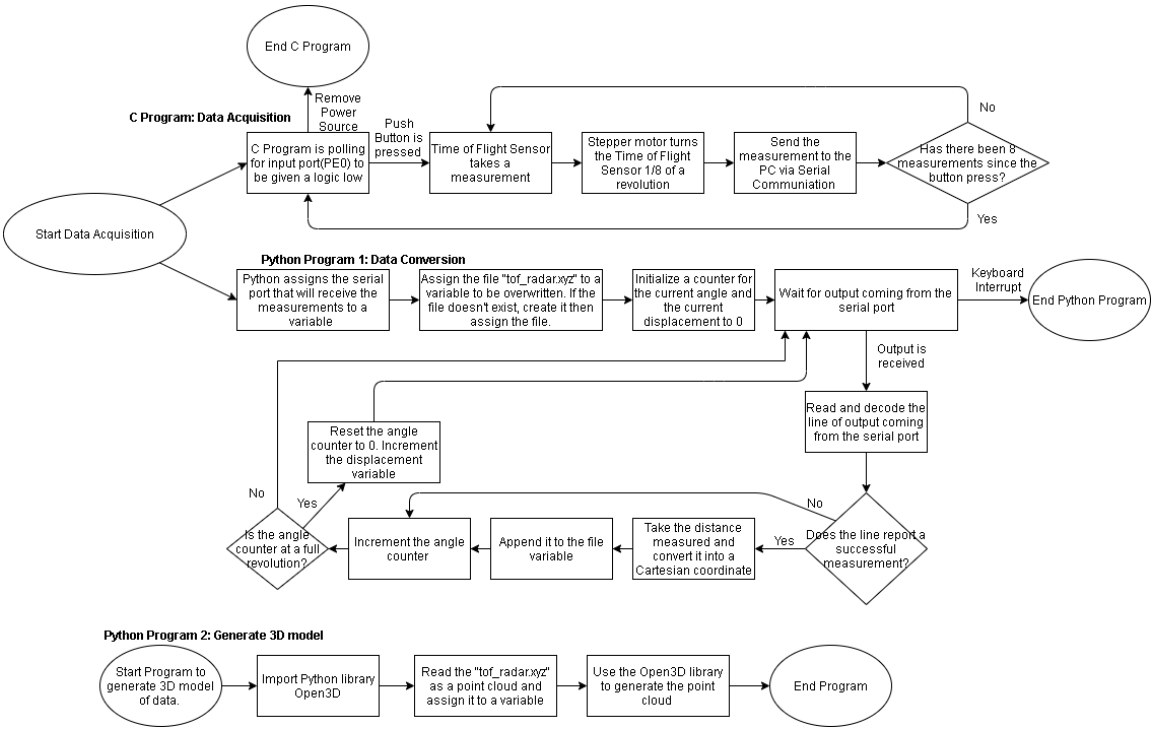

Figure 7: Programming Logic Flowchart