2DX4: Microprocessor Systems Project
Final Project

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

Glen Tsui – L05 – tsuig – 400201284

Submitted by: Glen Tsui

2DX4 Microprocessor Systems Project

CompEng 2DX4, McMaster University

Submitted on: April 8, 2020

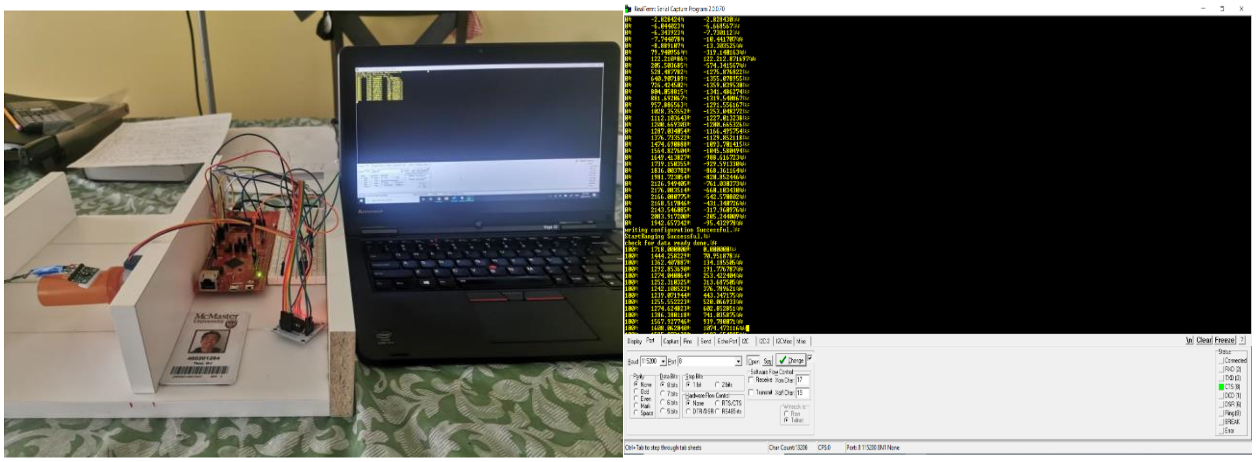
Contents

Table of Figures

Video Link

https://drive.google.com/open?id=1P-akEWJKQBAvJf0W4iErBhKFKgf7VU4z

Device Overview


RealTerm: Serial Capture Program 2.0.0.70

```
0     -2.828424     -2.828430
0     -6.044023     -6.668567
0     -6.343923     -7.730112
0     -7.744078     -10.441707
0     -8.889107     -13.303525
0     79.940956     -319.140163
0     122.210086    122.212.071697
0     205.503605    -574.341567
0     528.407702    -1275.876822
0     640.907109    -1355.078955
0     726.424502    -1359.039530
0     804.058815    -1341.406274
0     881.692067    -1319.540067
0     957.806563    -1291.556167
0     1028.353552   -1253.040272
0     1112.103643   -1227.013230
0     1200.669303   -1200.665326
0     1287.034054   -1166.495754
0     1376.733522   -1129.852118
0     1474.690088   -1093.781415
0     1564.827604   -1045.580494
0     1649.413827   -988.616723
0     1739.150355   -929.591330
0     1836.003782   -868.361164
0     1981.723054   -820.852446
0     2126.949405   -761.030173
0     2176.003514   -660.103430
0     2166.000775   -542.570002
0     2168.517046   -431.340726
0     2143.546085   -317.960976
0     2083.917200   -205.244009
0     1942.657342   -95.432970
writing configuration Successful.
StartRanging Successful.
check for data ready done.
100   1718.000000   0.000000
100   1444.258229   70.951878
100   1362.407087   134.185505
100   1292.853690   191.776787
100   1274.040064   253.422404
100   1252.310325   313.687505
100   1242.100522   376.789621
100   1239.071944   443.347175
100   1255.552223   520.066933
100   1274.624023   602.852851
100   1386.380118   741.035875
100   1567.927746   939.780071
100   1608.062040   1074.473116
```

Figure 1: Entire device with Microcontroller, Sensor, and PC; and PC displaying information

Features

This device is an embedded spatial measurement system that utilizes time-of-flight to measure distance 360 degrees along the Y-Z plane. The stored data can be used to construct a 3D model of the samples for presentation purposes using applications on a personal computer. This system is similar to Light Detection and Ranging (LIDAR) equipment but commercial LIDAR products are often large in size and expensive in price. This device is a smaller and less expensive alternative for indoor spatial measurements.

Hardware/Software Specifications

- Texas Instruments MSP-EXP432E401Y Microcontroller
  - 120MHz Default Bus Speed with individualized Bus Speed of 30 MHz
  - Input Power Supply 3.3 VDC
  - 1024KB Flash Memory
  - 256KB of SRAM with Single-Cycle Access
  - 6KB EEPROM
  - Two 12-Bit SAR-Based ADC Modules (2 Msps)
  - Uses programming language C in Keil software for implementation
- VL53L1X Time-Of-Flight laser-ranging sensor
  - Utilizes I2C interface (up to 400 kHz)
  - Accurate distance ranging up to 400cm
- 28BYJ-48 Stepper Motor with Velleman VMA401 ULN2003 Motor Driver
- 115200 baud rate for serial communication in RealTerm software
- Anaconda software for Python Open3D point cloud visualization
- Meshlab software for conversion to .stl file
- Total Cost: $145.00

General Description

This digital device undergoes 4 main stages: Data Collection, Data Handling, Data Writing, and Data Visualization. Data Collection begins by acquiring samples of a number of distance measurements by the Time-Of-Flight sensor per revolution of the stepper motor. The Time-Of-Flight sensor works by calculating the distance from the time it takes for a laser to hit a target and reflect back. The Time-Of-Flight sensor automatically performs transduction, signal conditioning, and analog/digital conversion automatically. Each measurement uses the I2C connection between the sensor and the microcontroller to serially transmit data under a baud rate of 115200. At the end of each revolution, the user can press the button again to start another series of measurements while incrementing the X-value. Before the data is displayed on the RealTerm software, the sample distance measurements are processed to calculate the corresponding Y and Z values within the Keil software. The phase of processing of data is called Data Handling. When the user has accumulated a sufficient number of measurements, the Data Writing phase begins. The X, Y, and Z coordinates can be retrieved from RealTerm and written into an .xyz file which will be used in the final phase which is Data Visualization. The .xyz file is used by the Python library Open3D to visualize a point cloud by connecting the points of each plane and connecting the planes together with lines. For a proper 3D visualization, the software MeshLab can be used to create a mesh which can be exported as an .stl file for presentation.

Block Diagram

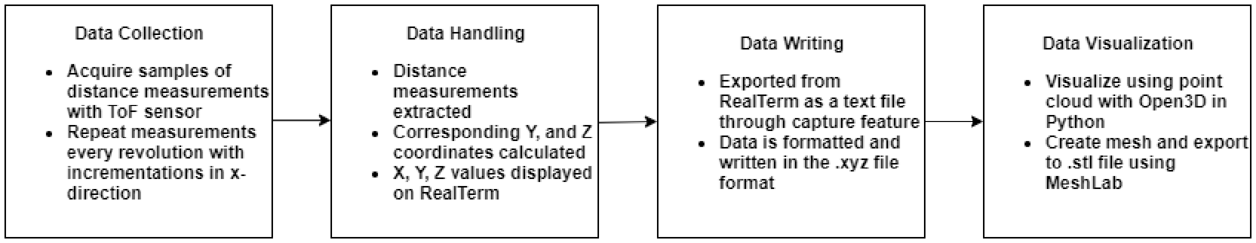

Data Collection
Acquire samples of distance measurements with ToF sensor
Repeat measurements every revolution with incrementations in x-direction
Data Handling
Distance measurements extracted
Corresponding Y, and Z coordinates calculated
X, Y, Z values displayed on RealTerm
Data Writing
Exported from RealTerm as a text file through capture feature
Data is formatted and written in the .xyz file format
Data Visualization
Visualize using point cloud with Open3D in Python
Create mesh and export to .stl file using MeshLab

Figure 2: Block Diagram of stages of the Project

Device Characteristics

First, the Time-Of-Flight sensor uses 4 pins for SCL, SDA, Vin, and GND; which is connected to PB2, PB3, 5V, and GND respectively on the microcontroller. For the Motor Driver, the pins IN1, IN2, IN3, IN4, V+, and V- are connected to PM2, PM3, PM4, PM5, 5V, and GND respectively on the microcontroller. The port used for the external LED for displacement status is PL4 and the on-board LED port used for distance status is PN0. The port used for the button is PM0 and the individualized bus speed used is 30MHz. Refer to Figure 11: Circuit Schematic for the full circuit schematic. PB2 and PB3 are two of the ten I2C-enabled pins for serial communication and allows for four different transmission speeds (100kbps, 400kbps, 1Mbps, and 3.33Mbps). The communication speed is standard 100kbps. The following programs were used with the Windows operating system. The availability of these software programs for other operating systems are unknown. The main software for configuring and programming the microcontroller is Keil (the correct debugger CMSIS-DAP Debugger and JTAG Adapter XDS110 with CMSIS-DAP must be selected). For serial communication and displaying of data, RealTerm must be installed. Ensure that the pySerial library is installed to locate the port used for communication. Python 3.6.0 must be installed with the Python libraries numpy and Open3D 0.9.0.0 for Point Cloud visualization from the .xyz file. MeshLab can also be installed for a proper mesh and a conversion option to an .stl file.

ToF to Microcontroller				Motor Driver to Microcontroller						External LED	On-Board LED
SCL	SDA	Vin	GND	IN1	IN2	IN3	IN4	V+	V-	PL4	PN0
PB2	PB3	5V	GND	PM2	PM3	PM4	PM5	5V	GND		

Figure 3: Device Characteristics Table

Detailed Description

Distance Measurement

The ToF sensor stands for Time of Flight and it obtains a distance measurement by calculating the time it takes for a laser to hit a target and reflect back. The Transduction, Signal conditioning, and Analog to Digital Conversion processes are done automatically by the Time-Of-Flight sensor. It is able to calculate the distance from the time since the VL53L1X ToF sensor uses a VCSEL (Vertical Cavity Surface Emitting Laser) to emit an infrared laser of wavelength 940 nm. This sensor can measure accurately up to a maximum distance of 4m. The VL53L1X ToF sensor returns the distance of the target in millimeters. The output of the sensor is first given as a series of bits which is then given as a measurement in millimeters through the API which controls the sensor. For a Universal Asynchronous Receiver-Transmitter (UART), the data in bits are transmitted serially one at a time through the established serial communication between devices.

When the distance measurements are acquired (Data Collection in Figure 2), the corresponding X, Y, Z values are calculated using trigonometry and the current angle of the motor. The current angle can be found by taking the angle in radians of one motor step division and multiplying by the current motor step. For example, if the current motor step is 256 out of 512, the current angle would be one PI radians. The X value is automatically determined by the shifts of the device and incrementation by a set distance (in this case 100mm in the X direction per revolution). To calculate the Y and Z coordinates, the distance measurement and current angle is needed. For example purposes, the distance measurement could be 500mm and the current angle could be one PI radians. The Y coordinate is calculated by the distance measured multiplied by sine of the current angle, for example it would be 500 * sin(PI) = 0mm. Z coordinate is calculated by the distance measured multiplied by cosine of the current angle, for example it would be 500 * cos(PI) = -500mm. Below includes the relevant equations used as well as a flowchart that covers the acquisition, transmission, and conversion process of data from the microcontroller.

$$Current\ Angle = \left(\frac{2\pi}{512}\right) * Current\ Motor\ Step$$

$$X\ Coordinate = Shift\ Distance * Number\ of\ Revolutions\ Completed$$

$$Y\ Coordinate = Distance * \sin(Current\ Angle)$$

$$Z\ Coordinate = Distance * \cos(Current\ Angle)$$

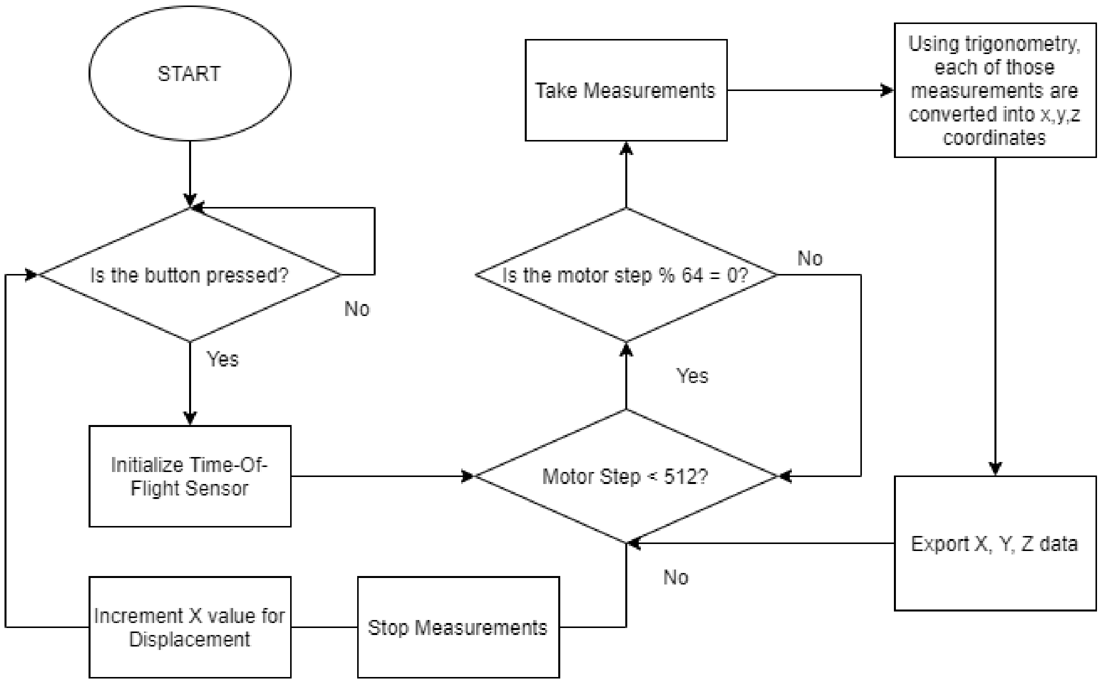

START
Is the button pressed?
No
Yes
Initialize Time-Of-Flight Sensor
Increment X value for Displacement
Stop Measurements
Take Measurements
Is the motor step % 64 = 0?
No
Yes
Motor Step < 512?
No
Using trigonometry, each of those measurements are converted into x,y,z coordinates
Export X, Y, Z data

Figure 4: Flowchart of Processes of Distance Data

Displacement

The MPU-9250 Inertial Measurement Unit (IMU) sensor is a component that can track position and orientation through the use of accelerometers and gyroscopes. Common uses for an IMU includes navigation, stabilization and control measures, mappings. The MPU-9250 uses 4 pins like the Time-Of-Flight sensor: SCL, SDA, VDD, and GND. There needs to be I2C-enabled ports so PG0 for SCL and PG1 for SDA can be used.

When the program starts, the initial reference position is at X = 0mm. The data from the IMU sensor is retrieved only when the displacement external LED is on, thus when one full revolution of measurements are taken. In doing so, when the device is displaced and the button is pressed again, the X coordinate value would be updated. Using an IMU would lead to more accurate results in terms of the distance between planes of measurements thus leading to a more accurate spatial measurement overall. Displacement is actually implemented using a simple incrementation of a specified distance in the X direction. For example, if at the end of each full revolution of measurements, the device is displaced 100mm; then the X coordinate would be incremented by exactly 100mm every full turn. This implementation is more accurate due to the fact that the displacement of the device may not be always precise. As the number of displacements increase, small discrepancies between each measurement of the X coordinate would impact the overall shape of the point cloud.

SCL	SDA	VDD	GND
PG0	PG1	5V	GND

Figure 5: Ports Arrangements for MPU-9250

Visualization

Specifications of Computer and Programs

- Lenovo ThinkPad Yoga12
    - Intel Core i7 (5th Gen) 5600U / 2.6 GHz Processor
    - Intel HD Graphics 5500
    - 12.5" HD Touchscreen Display
    - 8 GB RAM
- Python 3.6.0
    - Open3D 0.9.0.0 Library
    - Numpy 1.15.1 Library
    - Anaconda Jupyter Notebook 5.6.0
- C99 Programming Language
    - Keil uVision5
- RealTerm Serial Capture Program 2.0.0.70
- MeshLab 2020.03

Once all data measurements are compiled into an .xyz file, the Data Visualization stage in Figure 2 can begin. Visualization can be done with Python or with MeshLab. The Python method imports the Python libraries Open3D and Numpy to create a point cloud visualization. Numpy includes better functions for manipulation of arrays and Open3D can allow for reading of xyz files as point clouds and connections between the measurements in the point clouds, creating a line-set visualization. In the Python code there are two sets of nested for loops, one for the connections between the measurements of each plane and the other for the connections between each plane. For the first nested loop, it sets two variables as points, one point is one step greater than the other. In the .xyz file it would connect each point one by one until the end of the revolution where the final point is connected back to the first. This is repeated N times where N is the number of full planes of measurements done. The second nested loop connects the point to its equivalent point in the plane next to it and increments so all points would be connected adjacent to it. This is repeated N-1 times because there are only N-1 connections between N number of planes. Using the visualization function in Open3D, a line-set visualization can be created using the assigned points on the .xyz file.

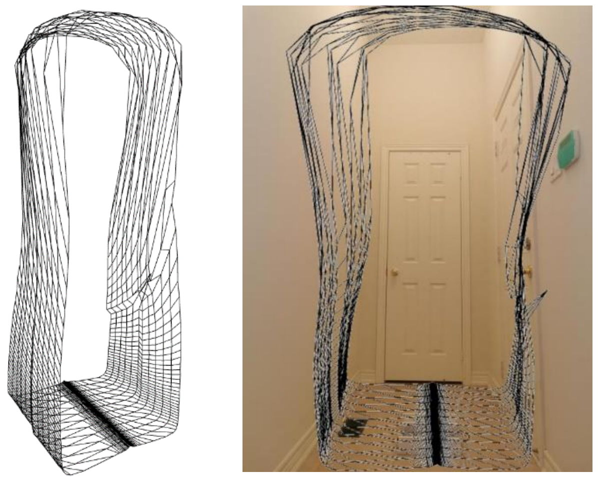


Figure 6: Line-Set Visualization and Spatial Comparison

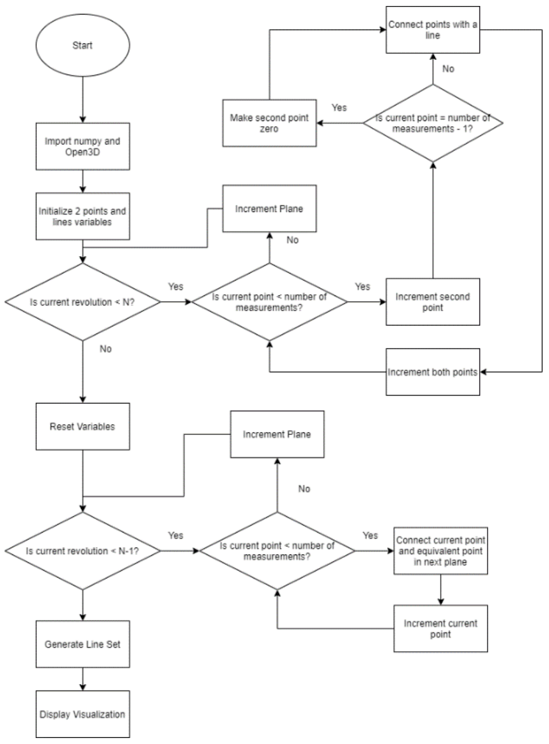

Start
Import numpy and Open3D
Initialize 2 points and lines variables
Is current revolution < N?
Yes
No
Is current point < number of measurements?
Yes
No
Increment Plane
Increment second point
Is current point = number of measurements - 1?
Yes
No
Make second point zero
Connect points with a line
Increment both points
Reset Variables
Increment Plane
Is current revolution < N-1?
Yes
Is current point < number of measurements?
Yes
No
Connect current point and equivalent point in next plane
Increment current point
Generate Line Set
Display Visualization

Figure 7: Flowchart for Python Point Cloud Visualization

Using MeshLab for visualization is a much simpler process. After installing MeshLab, the .xyz file can be opened. First the program must estimate the surface normal from the point cloud dataset. This is important in setting the underlying shape of the mesh. This setting can be found in Filters -> Normals, Curvatures and Orientations -> Compute Normals for set Points. Once the surface approximations are set in place, the point cloud is ready to be meshed. In Filters -> Remeshing, Simplification and Reconstruction -> Surface Reconstruction: Ball Pivoting, the Clustering Radius can be adjusted to the user's liking. Once a desired mesh is created, it can be exported as an .stl for 3D printing or presentation purposes.

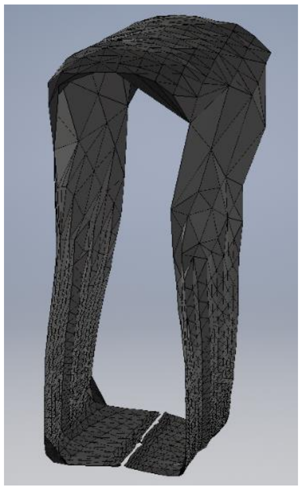
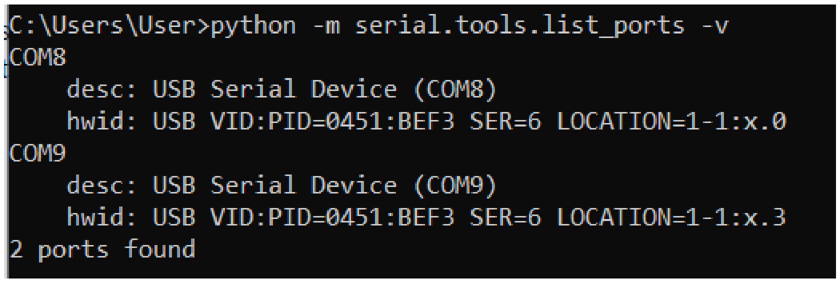


Figure 8: MeshLab 3D .stl Visualization

Application Example

An application for this device is using it to map the spatial area of a hallway. The following steps to acquiring signal and mapping using this device following a specific set of specifications as outlined by the Device Characteristics and the Specifications of Computer and Programs in Visualization.

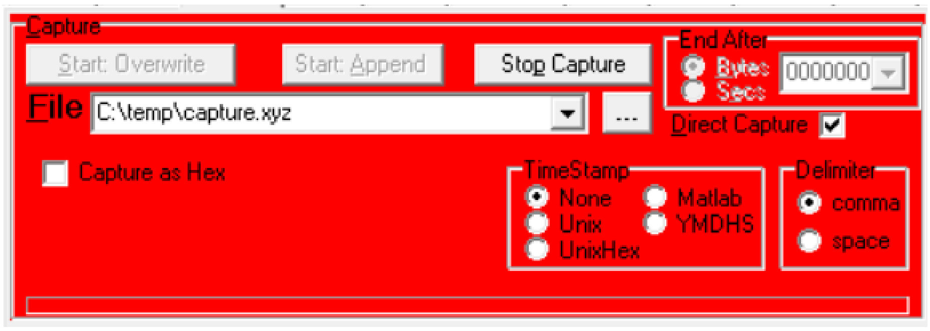
1. Ensure Python 3.6.0, PySerial, Open3D, and Numpy is installed
2. Connect the microcontroller to the computer and upload the C code in Keil by pressing Translate -> Build -> Download.
3. Open up command prompt and type in “python -m serial.tools.list_ports -v”
4. It will then output a list of ports, remember the port that is used for serial communication. In this case it would be COM8. Refer to Figure 9: Output of Ports List for Communication.



```
C:\Users\User>python -m serial.tools.list_ports -v
COM8
    desc: USB Serial Device (COM8)
    hwid: USB VID:PID=0451:BEF3 SER=6 LOCATION=1-1:x.0
COM9
    desc: USB Serial Device (COM9)
    hwid: USB VID:PID=0451:BEF3 SER=6 LOCATION=1-1:x.3
2 ports found
```

Figure 9: Output of Ports List for Communication

5. Run RealTerm and click the Port tab. Select 115200 and the chosen port number for the Baud and Port dropdown menus respectively.
6. Then click the Capture tab and click the 3 dots next to the file dropdown menu. Select a location to store the measurements on the computer.
7. Click Start: Overwrite and reset the microcontroller. The Capture section as in Figure 10: RealTerm Capture Screen will turn red to indicate that it is recording the measurements.



File: C:\temp\capture.xyz

Figure 10: RealTerm Capture Screen

8. Press the external button to start the motor and sensor for data acquisition. The motor rotates along the Y-Z plane. After one full revolution, shift the device a specified distance in the direction perpendicular to the direction of the motor rotation (X-direction). Press the button and repeat the process until the entire room is measured.
9. Click Stop Capture on RealTerm and locate the .xyz file created. The steps in Visualization can be proceeded.

Limitations

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

   Since trigonometric functions such as sine and cosine are used to calculate the Y and Z coordinates, the accuracy of the dimensions depends on the number of decimal digits. The MSP432E401Y has a floating-point unit to carry out operations regarding floating-point numbers. Within the floating-point unit of the microcontroller, it has only 32 single-precision registers, s0 to s31. Each single-precision register can store a 32-bit integer or a single-precision floating-point value. Also, datatypes in C programming like double are up to 64 bits which would limit decimal counts to 64 bit precision.

2. Calculate your maximum quantization error for each of the IMU and ToF modules.

   The maximum quantization error is equal to the difference in min-max operating voltages divided by two times the number of ADC bits. For the Time-Of-Flight sensor (VL53L1X), there are 16 ADC bits, V_{max} is 3.5, and V_{min} is 2.6V. Thus the maximum quantization error for the VL53L1X is 0.028125V. For the Inertial Measurement Unit sensor (MPU-9250), there are also 16 ADC bits, V_{max} is 3.6V, and V_{min} is 2.4V. Thus the maximum quantization error for the MPU-9250 is 0.0375V.

$$Maximum\ Quantization\ Error = \frac{V_{max} - V_{min}}{2 * NumOfBits_{ADC}}$$

$$Maximum\ Quantization\ Error\ (ToF) = \frac{3.5 - 2.6}{2 * 16} = 0.028125V$$

$$Maximum\ Quantization\ Error\ (ToF) = \frac{3.6 - 2.4}{2 * 16} = 0.0375V$$

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

   The maximum selectable baud rate in RealTerm is 115200. This means that the rate that information is transferred in a serial communication channel is at a maximum of 115200 pulses per second. A single pulse may consist of many bits. For personal computers, serial ports generally supports speeds up to 115200. For the microcontroller the support of IrDA SIR encoder and decoder functions for data rates go up to 115.2 kbps half-duplex[1].

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

   The communication method used between the microcontroller and the ToF and IMU modules is the I2C protocol. The baud rate is 115200 and the communication speed is a standard 100kbps.

[1] MSP432E401Y Microcontroller Data Sheet page 107

5. Reviewing the entire system, which element is the primary limitation on speed? How did you test this?

   The primary limitation on speed is the bus clock speed of the microcontroller. For this device, the microcontroller's bus speed is set to 30 MHz and serves as a primary limitation on the speed of every process. This limitation can be proved by changing the PSYSDIV value in PLL.h to a value corresponding to the System Clock. Lowering the System Clock is equivalent to having a greater limitation on the overall speed of the entire system.

6. Based upon the Nyquist Rate, what would be the necessary sampling rate required for the displacement module? What happens when your input signal exceeds this frequency?

   The Nyquist Rate is equal to two times the highest frequency of the sample signal. It is a minimum rate that the sampler must sample or else the signal may alias. For the MPU-9250 displacement module, the highest output frequency is 4kHz so the sampling rate must be at least 8kHz which is the Nyquist Rate. If the input signal is below 8kHz, the frequencies would overlap, cause aliasing, overall leading to mixing up and losing information. However, when the input signal exceeds 8kHz, the information can still be recovered and can be replaced without any loss.

Circuit Schematic

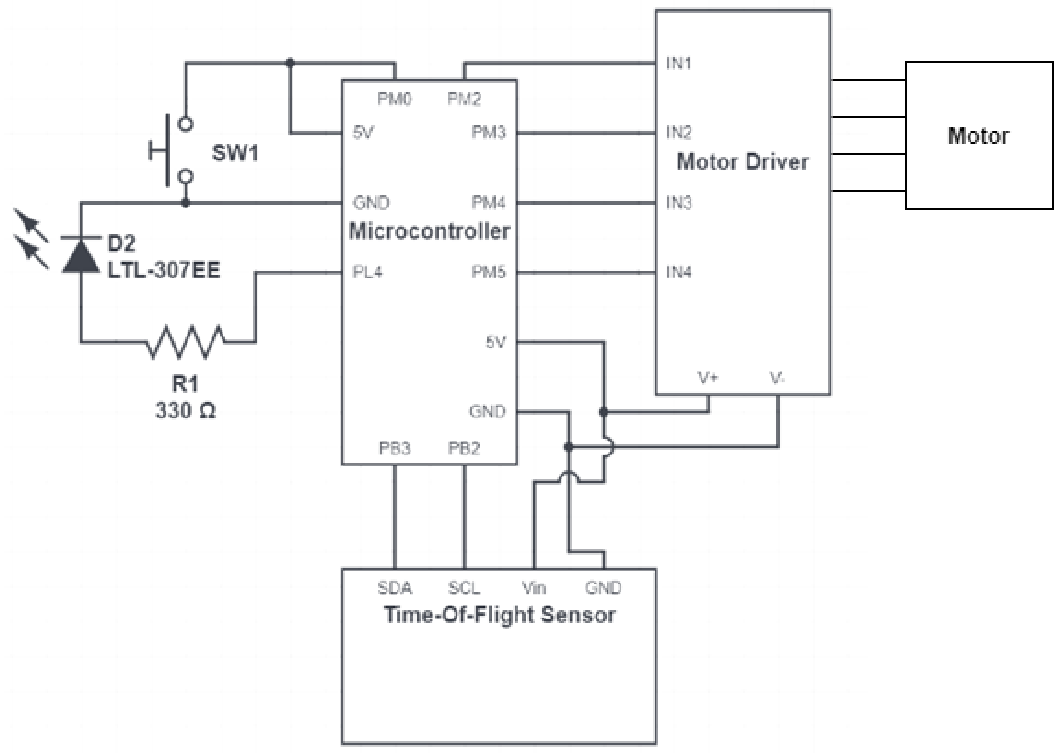


Figure 11: Circuit Schematic