2DX4: Microprocessor Systems Project

Final Project

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

Yazdan Rahman – L02 – rahmay1- 400199680

Interview Questions:

First Question:
https://drive.google.com/open?id=1nlV98vfEUFm6kacD5SM6UJhvtw1aYLrQ\

Second Question:
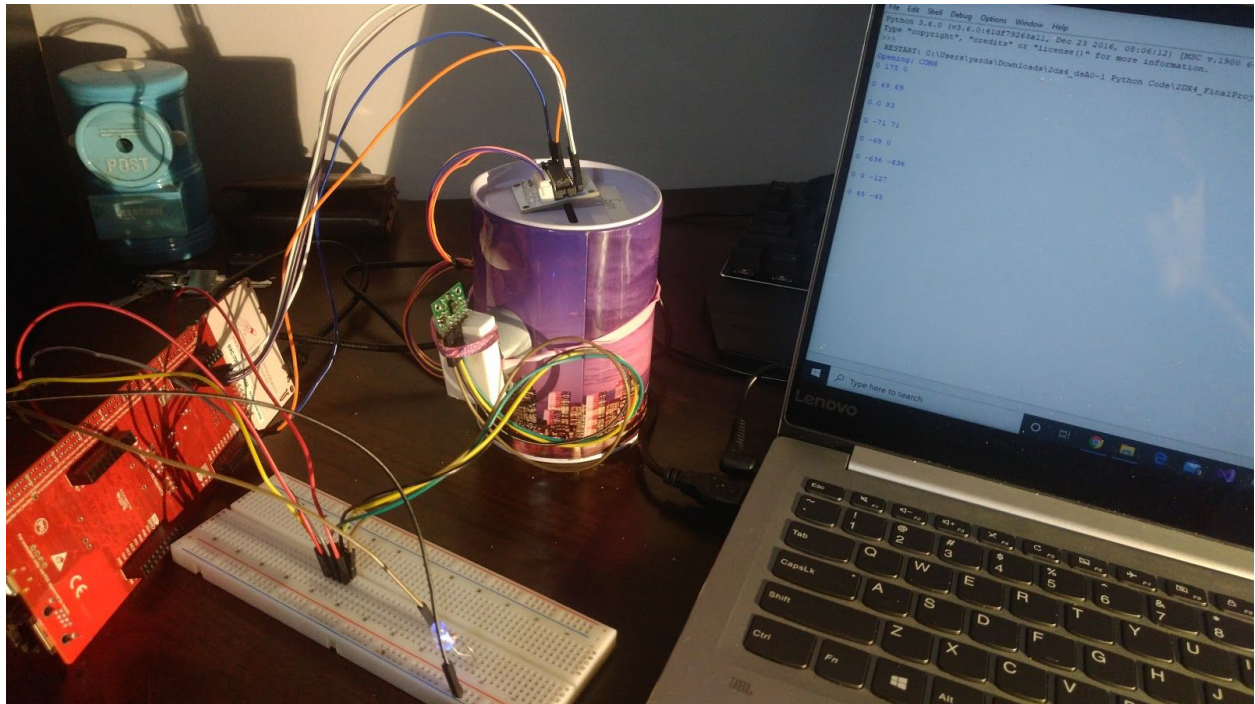https://drive.google.com/open?id=1dh8mTQWI4DhucySaJVPeuVRb05GrkkHG

Third Question:
https://drive.google.com/open?id=1RZsFvHQjOwdnS2T0lyIHVfjxYpMYT_e4

Design Overview:

Features:
- Module provides a measurement of the horizontal distance travelled by robot
- Module provides a 360 degree scan of the yz-plane
- Module provides LED indication of different modes
- Stores all the points onto a connected pc
- Generates a interactive 3D plot which shows an outline of the environment
- Module can used to track the robot's surrounding environment for an optimal route to take
- Module can used in order to track distance travelled by the robot
- 120MHz Arm Cortex M4F Processor (120MHz clock speed)
- 4.75V to 5.25V Operating Voltage
- 1024KB of Flash Memory, 256KB of SRAM, and 6KB EEPROM
- 20 shared analog input channels
- Two 12-bit SAR Based ADC Modules
- Each ADC has 2 Megasamples/second sampling rate
- Around $120 CAD for whole module set (MSP-EXP432E401Y Board, Sensors, Breadboard, Wires, and LEDs)
- Uses C++ Language (uVision IDE)
- Up to 15 Megabits/second UART Serial Baud Rate
- Uses 115200 bits/sec Baud Rate Serial Communication with a 1 terminator stop bit (by default)

General Description:

The Robot Module contains a MSP-EXP432E401Y Board, a VL53L1X Time of Flight Sensor, MPU 9250 IMU, and a 5V DC Stepper Motor.

The stepper motor is connected to the VL53L1X Time of Flight Sensor which rotates in the yz-plane. When the onboard button is pressed, the onboard LEDs on the MSP-EXP432E401Y indicate a rotational movement of the stepper motor. The stepper motor rotates the VL53L1X sensor and the robot gives 8 xyz coordinate points during the 360 degree sweep. It then sends the coordinates via Serial Communication and the python program reads and saves those coordinate points. The robot module now turns off, and waits for another button press to repeat the same steps again in addition to moving 20cm forward. The forward horizontal distance is tracked using the MPU 9250 IMU. Once the python program obtains two sets of 8 xyz coordinate points, it generates a 3D plot representing the movement of the robot from the first set of 8 xyz points to the second set to the third set, and so on, depending on how many sets are specified.

The VL53L1X Time of Flight Sensor works by sending out pulses of infrared light and measuring the time it takes for the pulse to bounce from the object back to the sensor. The sensor then uses the transducer to convert the nonelectrical signal into an electrical signal, signal conditioning then converts the electrical signal into a continuous digital signal. Finally ADC (Analog to Digital Converter) converts the continuous digital signal into a discrete digital signal then the signal is sent to the MSP-EXP432E401Y Board.

The serial communication uses a 115200 baud rate which transfers the xyz coordinate data from the MSP-EXP432E401Y Board to the computer through the USB. The running python program reads in this data input and processes it by saving it as a xyz file. The xyz formatted file is then read in from the python program as a cloud array using the open3d library package. After connecting some points together using lines, the points are plotted using open3d.

Block Diagram:
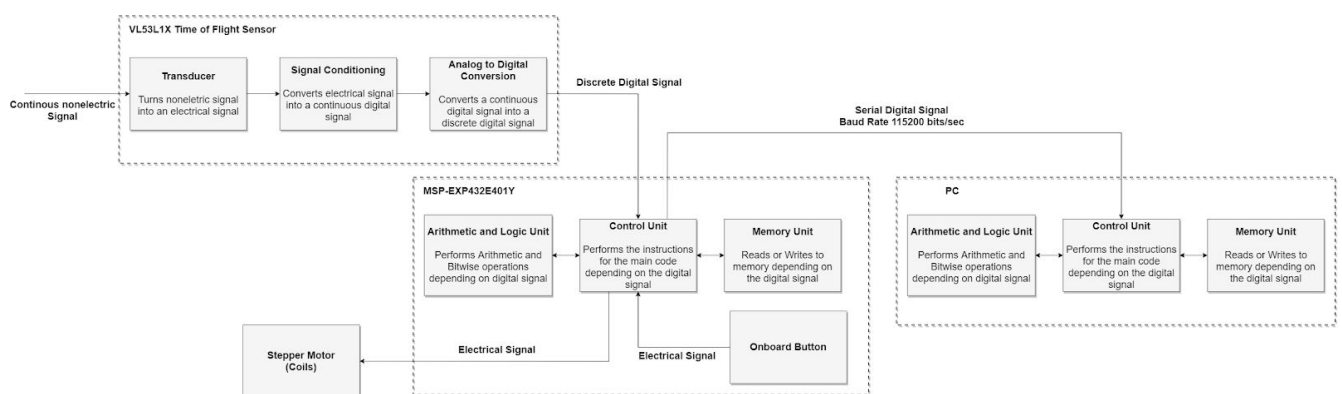
Device Characteristics Table:

To ensure the python code will work, the computer specific port needs to be set in the python program. This port can be found out by opening up and checking the realterm "Port" tab with a 115200 Baud rate while booting up the program code on the MSP-EXP432E401Y Board. By checking through all the ports on the realterm tab one by one, the correct port can be found by a brute force approach. The python program will then need to be edited to ensure that the correct port is opened by the program when running. The number of sample point sets needed to generate the plot needs to be also specified when running the python code. These steps are mentioned more in depth under Application Example. The MSP-EXP432E401Y Board has a max serial communication rate of 15 Megabits per second, far greater speeds than the python program operates in to detect the signal on time.

When running the code in the MSP-EXP432E401Y Board, the 4 Onboard LEDs will light up and start flashing in a certain order. This means that the program is initializing the VL53L1X Time of Flight Sensor to use for the next steps. Make sure to not open up the python code before this is done. When the external LED is lit up, the program is finished its Initialization Phase and is now in Idle Phase. Now, if you would like to run the python code to view the graphical visualization, make sure to run the python code before clicking the onboard PJ1 button. Now the program will enter the Sweeping Phase where the stepper motor will rotate 360 degrees, taking distance measurements every 45 degrees using the Time of Flight Sensor. After finishing sweep, the program will enter the Idle Phase again. Clicking the onboard PJ1 button again will again start the Sweeping Phase, and so on. Once the Idle Phase has been reached the specified number of times, the python program will generate the visual graph. A bus speed of 96MHz was used for the project program, in other words a PSYSDIV value of 4.

Refer to Schematic Figure 1 and Table 1 for more details on hardware connections

Detailed Description:

Distance Measurement:

The ToF sensor is initially set up with the SetupToF(void) function. This function first checks if the status of the ToF is ok by reading in data sent from the ToF after sending the default address of the ToF and certain indexes. This process uses I2C communication and accesses certain parts of memory in the ToF which store the ToF status. The boot state is then sent to the ToF followed by a while loop which checks if the ToF is booted up using the status checking mentioned previously. All Leds are flashed to signal successful bootup and the ToF interrupt is cleared and ranging mode is enabled for the ToF. Now in the main function a set of while loops are entered. The outer loop repeats forever, then there are two inner loops. Moving the stepper motor 45 degrees is the inner while loop and obtaining the ToF measurement at each 45 degrees and repeating the inner while loop 8 times for a full 360 rotation is the outer while loop. The ToF is called between these while loops using the function SendToFDistance(void). This function interrupts the ToF and waits until the ToF has ranged and acquired the distance yet, then it clears the ToF interrupt for the next interrupt call and returns the ToF distance. This data is stored in an array and after the stepper motor has rotated 360 degrees, the 2 inner while loops are exited and the program starts its calculation phase. This is how the y and z points are

calculated for each of the 8 distance measurements in the array (Note that only x coordinate points are manually increased by 20cm in the code after every 360 sweep):

Refer to Flowchart Figure 2 for more details about c++ program code

Since we are given an angle of 45 deg, and the hypotenuse (since the distance is at least how far the object is, AKA always greater than or equal to y or z), we can use the right angle triangle rule to get the y and z of any given distance. The only problem is that we need the same reference point, so the angle is not just 45 degrees, it is 0 deg, 45 deg, 90 deg, etc. The angle increases by 45 each time starting from zero until 360

$y = $ (Distance Measurement #) $ * (\sin(x))$
$z = $ (Distance Measurement #) $ * (\cos(x))$
Where x increases from 0 degrees, to 360 degrees.

For example: Say we are in the +y axis, we did a sweep of the coordinates from +y axis towards the +z axis, and we get the first 3 measurements, 200cm, 400cm, 300cm. Let's assume x = 20cm.

The first 3 set of coordinates are:
[x, (200) * (sin(0 deg)), (200) * (cos(0 deg))] = [200mm, 2000mm, 0mm]
[x, (400) * (sin(45 deg)), (400) * (cos(45 deg))] = [200mm, 2828.4mm, 2828.4mm]
[x, (300) * (sin(90 deg)), (300) * (cos(90 deg))] = [200mm, 0mm, 3000mm]

The y, and z coordinates are then transmitted using UART Serial Communication from the MSP-EXP432E401Y Board to the pc. This is sent using the sprintf() and UART_printf() function calls in the main function. The sprintf() function creates a c++ string (char array pointer) which contains the inputted value. The UART_printf() statement sends the string into the UART register one letter at a time. This is then sent through to the serial stream on the computer with 115200 baud rate. The receiving end is then the python code, which continuously reads from the input stream at a specific baud rate (115200). The python code continuously checks using a while loop which terminates if "End" string is received. The while loop also continuously inserts whatever points it receives into an xyz file. Thus whenever a set of points is required, it is sent using the UART_printf() statement followed by a "End" string on the onboard program. The python program repeats this while loop until however many sets of points are mentioned by the user and writes all the points into the xyz file. The python program now detects the points and stores them in the xyz file and continues the program when the "End" string is read from the serial stream shortly afterwards.

The x points are now opened from the xyz file using open3d, and stored into a cloud array. The cloud array contains all the sets of points in a variable. A lines array is then created which will store the lines connecting two points. Now all adjacent points in the same plane are connected to one another and stored in this line array. Then all points which are adjacent in both planes (for example, point 1 and point 9, point 2 and point 10, point 5 and point 13) are

connected by a line and stored in the line array. Now the lines are all rendered as a 3D plot using open3d visualization software.

Refer to Flowchart Figure 3 for more details about python program code

Displacement:

In order to get the IMU measurements there will be a few new sets of code added and changes, assuming IMU returns velocity in m/s. Two sets will be added, the function SetupIMU(void) which will set up the IMU so that it will be ready to read for the next IMU call. Then the function GetIMUReading(void) which will be a function which gets the current IMU reading in a while loop until while recording the time passed. The while loop will keep repeating while the distance calculated with the current IMU velocity and time passed does not exceed a certain output, in this case, 20cm. When the while loop repeats, the wheel motors on the robot are turned on until the while loop breaks. When the while loop breaks, state is set to 1,and the distance passed is set to the x variable. The SetupIMU() will be called during initialization like the other components. The GetIMUReading() will be called right under turning on the external LED. In addition to this the x +=20 manual incrementation of x will be deleted. Now the robot will sweep, then autonomously move, then sweep again, autonomously move again, and so on when the onboard PJ1 button is clicked.

Currently displacement is implemented using a manual 20cm displacement. When the Onboard PJ1 button is pressed, the program sweeps, then iterates 20cm manually (adds 20cm to x var), and stops. When then the button is clicked again, the sweep is repeated, which is the manual implementation currently used in the c++ program.

Refer to Flowchart Figure 2 for more details about c++ program code

Visualization:

Lenovo Ideapad 720s (PC Specifications):
- Windows 10 Home Edition
- Intel Core i7-8550U 2GHz CPU
- 8GB DDR4 Ram

Program Used:
- Python 3.6.0

Libraries Used (For Python 3.6.0):
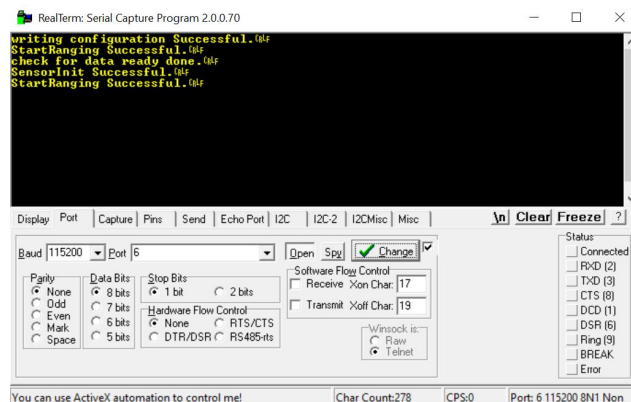- serial
- open3d
- numpy

The data is stored in an xyz file and processed using open3d for visual rendering. The points are then opened from the xyz file as a cloud array. Now a line array is defined which

contains the lines from one point to another and a line is made for every point with its adjacent point in the array all in the same plane. Now all points which are in the same modx position are also defined with lines connecting all of them (where x is the amount of data points specified by the user). For example; Point 1 and Point 9 are connected with lines, Point 7 and Point 15 are connected with lines, etc. Now a lineset is created using the lines array, and that lineset is visually plotted onto a 3D plane with the open3d library.

Application Example with Expected Output:

     1.) First you need to make sure the python program detects the right serial port. To do this, you need to open up realterm, click on the "port" tab section and change the baud rate to 115200.

     2.) Once done, you should either do nothing popup, or a few lines with the last line containing "StartRanging Successful" popup. If it does not change the port selection to a different number port until the success message does appear like the following:

     3.) When you have found the port, find where it says "s = serial.Serial("COM#", 115200)" in the python program and replace # with the correctly working port number which was found in step 2. For example, with a correct port number of 4, line 6 in the python program should be:
"s = serial.Serial("COM4", 115200)"

     4.) Once this is done, you will need to install the correct python libraries in order to run the python program. Make sure you install Python 3.6.0 using the installer from the python website. As well as the serial, numpy, and open3d libraries for python 3.6.0. Refer to https://www.python.org/about/help/ for more information.

     5.) Make sure to connect the correct ports on the ToF, M. Refer to Schematic Figure 1 and Table 1 for more details on hardware connections.
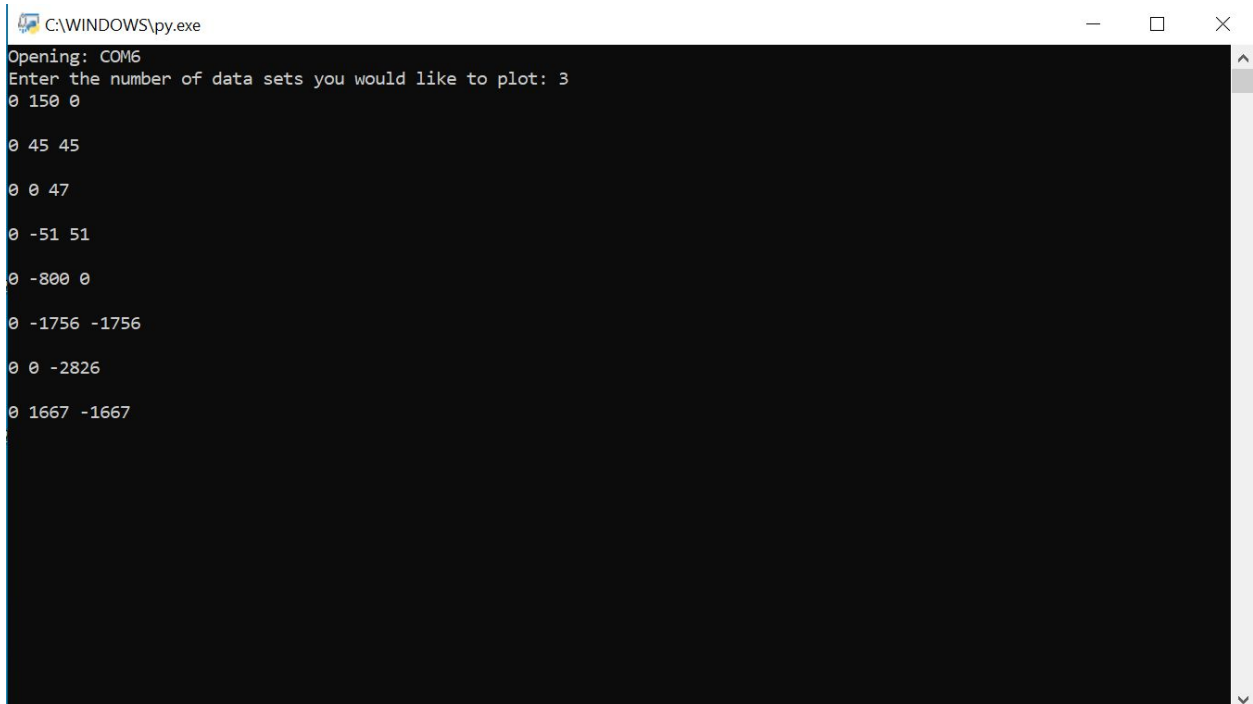
     6.) Now open up the uVision Project code and load it onto the MSP-EXP432E401Y Board but do not click the reset button yet.

7.) Click the reset button. You should now see the Onboard LED's blink in a certain sequence. Once that is done, the external LED should turn on and all Onboard LEDs should turn off.

8.) You can now double click the python program to run it. The cmd should popup with text that reads "Opening:COM#" with your port number # and "Enter the number of data sets you would like to plot:". This number represents the amount of data sets you want, essentially the number of 360 sweeps you would like to use for the plot. Make sure to specify this as an INTEGER value greater than 0. An input of 3 or greater is recommended.

9.) Now click the Onboard PJ1 button located on the side of the board (Labelled PJ1). The External LED should turn off and Stepper Motor should start rotating. The N0 Onboard LED (Labelled D2) should turn on for a brief moment every 45 degrees rotation and the Onboard LEDs should start to blink in sequence very fast. This Onboard LED blinking is telling the user that the program is currently communicating with the ToF sensor and is obtaining a distance value.

10.) The python program should now display something similar to:

```
C:\WINDOWS\py.exe                                           —   □   ×
Opening: COM6
Enter the number of data sets you would like to plot: 3
0 150 0

0 45 45

0 0 47

0 -51 51

0 -800 0

0 -1756 -1756

0 0 -2826

0 1667 -1667
```

Now repeat step 9 x amount of times where x is the data set input you inputted into the python program in step 8.

11.) Now open3d should popup with the display like the following:
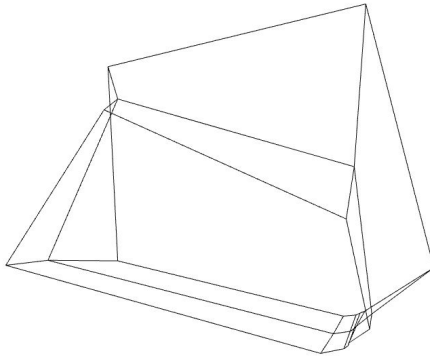
```
C:\WINDOWS\py.exe
400 0 -2576

400 2445 -2445

geometry::PointCloud with 24 points.
[[   0.   150.     0.]
 [   0.    45.    45.]
 [   0.     0.    47.]
 [   0.   -51.    51.]
 [   0.  -800.     0.]
 [   0. -1756. -1756.]
 [   0.     0. -2826.]
 [   0.  1667. -1667.]
 [ 200.   195.     0.]
 [ 200.    49.    49.]
 [ 200.     0.    44.]
 [ 200.   -34.    34.]
 [ 200.  -235.     0.]
 [ 200. -1032. -1032.]
 [ 200.     0. -2579.]
 [ 200.  2172. -2172.]
 [ 400.   269.     0.]
 [ 400.    54.    54.]
 [ 400.     0.    46.]
 [ 400.   -27.    27.]
 [ 400.  -272.     0.]
 [ 400.  -821.  -821.]
 [ 400.     0. -2576.]
 [ 400.  2445. -2445.]]
```

In these screenshots above, the open3d display is on the left, and the python cmd is on the right. You should get a plot as well (most likely different) to the above. This plot will show you the connected points from your first sweep to the second sweep. FYI the 3D plot is a rendering of the room with various furniture and exteriors, your points will vary.

Note: If you wish to repeat this task make sure to delete the points2dx4proj.xyz file that is located in the same folder as the python program executable. This is because the python program will load in any program with the same name as "points2dx4proj".

The sensor is assumed to start from the positive y axis, and sweep in the positive z axis direction. The x direction is assumed to be obtained from the IMU with a constant increment of 20cm.

Limitations:

1.) The accuracy of numbers in c++ and any other programming language spans from the floating numbers since assembly language can only use base 2 operations so the language compiler has to condense decimal numbers placed from base 10 into base 2 which is what a floating number is. The inaccuracy comes in when there is no exact multiple of a bit. For example; 1.2-1.0 in a program may result in 0.199999999999996 since 0.2 cannot be exactly represented by base 2. This causes inaccuracy in calculations and if there are repeated calculations taking place, this error increases accordingly. The inaccuracy affects the number much more greatly if the number is smaller which can lead to a program error.

2.) Equation for Max Quantization Error:
Max Quantization Error $(\Delta) = V_{IN} / 2^m$ (where V_{IN} is supply voltage of device and m is)

VL53L1X Time of Flight Sensor (ToF):
- V_{IN} = 5V since we hooked up the ToF V_{IN} channel to 5V (Refer to Figure 1)
- m = 12 from the ToF datasheet

$\Delta = V_{IN} / 2^m = 5V / (2^{12}) = 5V / (4096)$

$= 0.00122070312 \text{ or } \cong 0.0012207$

MPU 9250 Inertial Measurement Unit (IMU):
- $V_{IN} = 5V$ since we hooked up the IMU V_{IN} channel to 5V (theoretically)
- $m = 16$ from the ToF datasheet

$\Delta = V_{IN} / 2^m = 5V / (2^{\wedge}16) = 5V / (4096)$
$\quad = 7.62939453 \times 10^{-5} \text{ or } \cong 76.294\mu$

3.) The computer communicates with any ports using serial communication. This form of communication in any computer is often capped at around 115200 Bits/sec baud rate. This baud rate goes all the way back to the 1900s where certain Crystals oscillators were commonly used at the time due to their low price. These crystals had a frequency of around 315/88 MHz (or 3.579545 MHz) known as the color burst frequency. Then the UART chip divided the frequency into 16 for subsampling. So 3.579545GHz / 16 = 223.721563 MHz. This has to be further divided by 2 since the lowest divisor offered by the UART was 2. This gives 111.860782 MHz which is approximately 1118607 Bits/s. But since it was challenging to even implement a baud rate of 115200Bits/s, it became the standard at the time, and still is for certain PCs. This can also be proven by launching realterm and checking the maximum baud rate of the "Port" tab, Although this will vary per computer. A baud rate depends on the individual computer specs.

4.) The form of communication used between the MSP-EXP432E401Y Board and the VL53L1X Time of Flight Sensor is I²C (AKA inter integrated circuit) communication with a speed of 400kHz (Refer to ToF datasheet). This form of communication uses master-slave implementation where the master sends information bits usually 8 bits to inform the slave about information, the slave then sends 1 bit as an acknowledgement of that data. The SDA and SCL lines are connected in this communication form which represent the Serial Data and Serial Clock lines respectively. The SDA line is for sending data between the master and slave while the SCL is for syncing up their clocks for easier data transfer.

5.) When comparing all the components in the whole system, the slowest component would be the stepper motor. This is due to its crippling lowest frequency rating of around ~100 Hz. This is due to the motors slowest safe driving delay in between coil turns is 10ms. Since this is in seconds, it needs to be converted to Hz for comparison and it turns out to be 1/(10ms) which is 100Hz. Second slowest excluding motor would be the ToF sensor since it has a 400kHz speed as compared to a high speed of 120MHz for the MSP-EXP432E401Y Board . This is especially noticeable during initialization of the ToF which takes a couple of seconds (due to sensor booting up).

6.) The necessary sampling rate of the IMU needed will depend on the frequency and the IMU and the sampling rate of the onboard MSP-EXP432E401Y buses. Since the bus sampling rate of 96MHz was specified and the IMU sampling rate of 24MHz was found (from the datasheet). The bottleneck would be from the IMU and the maximum sampling rate would be that of the IMU. Since we are also using I²C connections for communications between the IMU and the MSP-EXP432E401Y Board. The maximum sampling rate will also depend on the I²C

buffer which 2^{10}. This means that my actual transfer rate will be Sampling Rate / Sampling Buffer which is 24MHz / 2^{10} = 23.4375kHz or 23.4375kBits/s.

Circuit Schematic:

Figure 1:
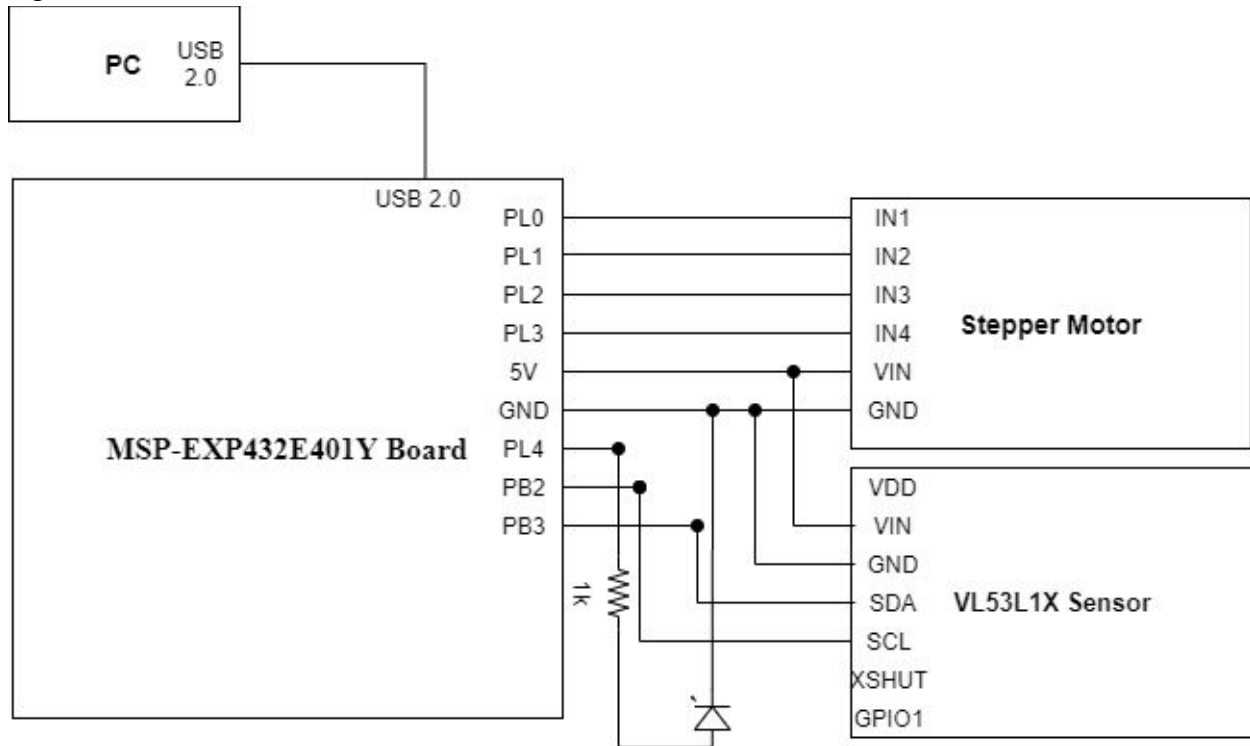
Table 1:

Pins Setup (MSP-EXP432E401Y Board | Other):	
PB2	SCL (ToF Sensor)
PB3	SDA (ToF sensor)
GND	GND (ToF sensor)
5V Supply	VIN (ToF Sensor)
PL4	VIN (External LED with Resistor)
GND	GND (External LED with Resistor)
PL0 - PL3	IN1 - IN4 (Stepper Motor)
5V Supply	VIN (Stepper Motor)

GND	GND (Stepper Motor)

Programming Logic Flowchart:

Figure 2 (C++ Program):

Figure 2 (Python Program):

Start Of Python Program

↓

Importing Library Packages: serial, numpy and open3d

↓

Initialize Variables

↓

Opening up the serial stream of 115200 Bits/sec on the port connected to the MSP-EXP432E401Y Board

↓

Create a new XYZ file called PointsToStore.xyz and open up python file writer to write to the file

↓

Start of For loop - For int i from 0 to numOfPlanes var - 1

↓

Start of For loop - Loop until "End" input from serial stream is found

↓

Is the input from the serial stream a set of points? —Yes→ **Write the input into the xyz document**

↓ No   ↓

Get the next input value from the serial stream ←─────────────┘

↓

End of Loop

↓

End of Loop

↓

Close python file writer

↓

Close serial input stream reader ─────────────┐
                                                   │
                                                   │
Use open3d to open xyz file and grab and store points into a 3D array ←── (from top)

↓

Create a an array var to store coordinate sets called "lines"

↓

Start of For loop - For int i from 0 to numOfPlanes var

↓

Start of For loop - For every point (in 3D array) in plane of index i

↓

Add the current point and its adjacent pair (current point + 1) to the lines array

↓

End of Loop

↓

End of Loop

↓

Start of For loop - For int i from 0 to numOfPlanes var - 1

↓

Start of For loop - For every point (in 3D array) in plane of index i

↓

Add current point with its pair in adjacent plane to the lines array

↓

End of Loop

↓

End of Loop

↓

Use open3D to create a geometry which contains all the lines in the lines array

↓

Use open3D to plot this geometry

↓

End Of Program
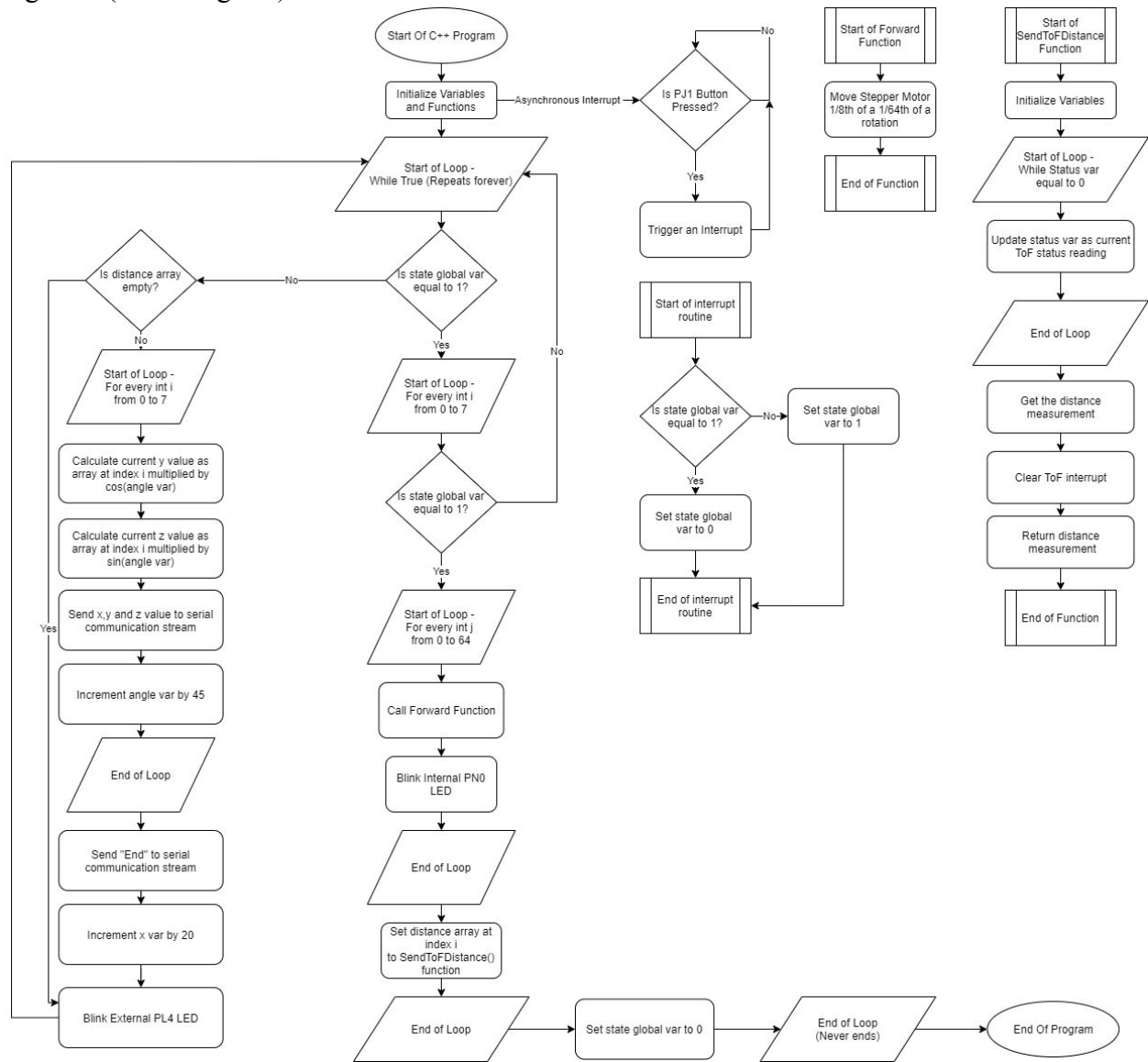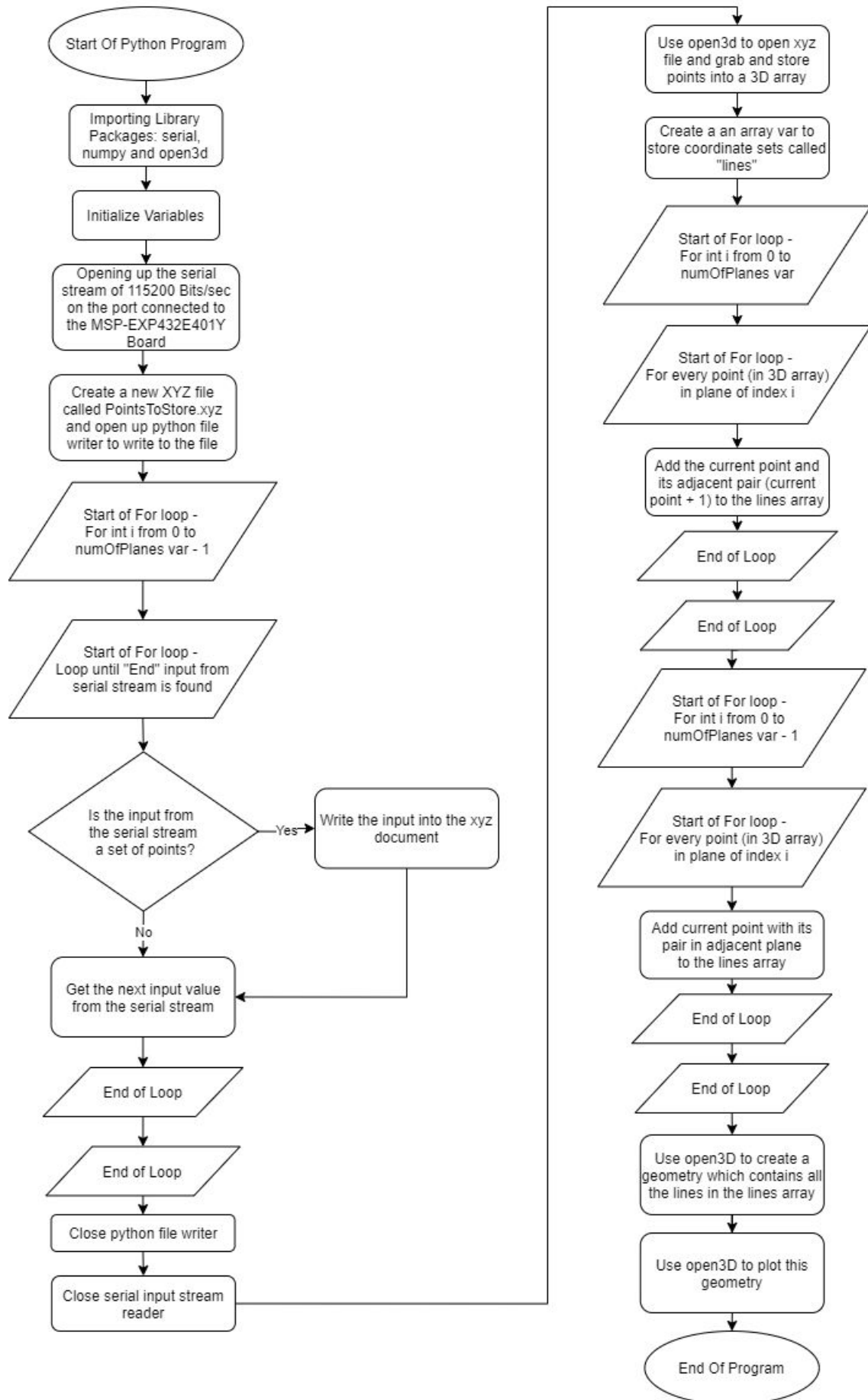
Components Specifications:

MSP-EXP432E401Y Board
- 120MHz Arm Cortex M4F Processor (120MHz clock speed)
- 4.75V to 5.25V Operating Voltage
- 1024KB of Flash Memory, 256KB of SRAM, and 6KB EEPROM
- 20 shared analog input channels
- Two 12-bit SAR Based ADC Modules
- Each ADC has 2 Mega Samples per second sampling rate
- $70.58 CAD per board according to mouser.ca
- Uses SimpleLink MSP-EXP432E401Y LaunchPad Development and Software Development Kit (SDK) software
- Up to 15 Megabits per second UART Serial Baud Rate

VL53L1X Time of Flight Sensor
- 2.6V to 5.5V Operating Voltage
- 50Hz max sampling rate
- $16.67 CAD per board according to pololu.com
- Up to 400kHz I^2C serial communication

MPU 9250 Inertial Measurement Unit (IMU)
- 2.4V to 3.6V Operating Voltage
- Three 16-bit ADC Modules
- Each ADC has 8 Kilo Samples per second sampling rate
- $14.95 USD per board according to sparkfun.com
- Up to 400kHz I^2C or 1MHz SPI serial communication

Sources:

"Electronic Components," *Mouser Electronics Canada - Electronic Components Distributor*. [Online]. Available: https://www.mouser.ca/ProductDetail/Texas-Instruments/MSP-EXP432E401Y?qs=j%2B1pi9Td xUbU/FmadQLPsg. [Accessed: 14-Apr-2020].

"MSP432E401Y: Maximum ADC Sample Frequency," *E2E*. [Online]. Available: http://e2e.ti.com/support/microcontrollers/msp430/f/166/t/677767?MSP432E401Y-Maximum-A DC-Sample-Frequency. [Accessed: 14-Apr-2020].

"Pololu - VL53L1X Time-of-Flight Distance Sensor Carrier with Voltage Regulator, 400cm Max," *Pololu Robotics & Electronics*. [Online]. Available: https://www.pololu.com/product/3415. [Accessed: 14-Apr-2020].

J. Steele, "SparkFun IMU Breakout - MPU-9250," *SEN-13762 - SparkFun Electronics*. [Online]. Available: https://www.sparkfun.com/products/13762. [Accessed: 14-Apr-2020].

Patil, T. Agarwal, T. Agarwal, A. Bhangale, T. Agarwal, T. Agarwal, Arjun, T. Agarwal, T. ElProCus, and T. ElProCus, "Electronic Communication Protocols Basics and Types with Functionality," *ElProCus*, 02-Oct-2014. [Online]. Available: https://www.elprocus.com/communication-protocols/. [Accessed: 14-Apr-2020].