Report on ELEC230 Assignment 2 2021-22

Academic Integrity Declaration

I confirm that I have read and understood the University's definition of plagiarism and collusion from the Code of Practice on Assessment. I confirm that I have neither committed plagiarism in the completion of this work nor have I colluded with any other party in the preparation and production of this work. The work presented here is my own and in my own words except where I have clearly indicated and acknowledged that I have quoted or used figures from published or unpublished sources (including the web). I understand the consequences of engaging in plagiarism and collusion as described in the Code of Practice on Assessment (Appendix L).

Contents

Part 1 (50%) Range Finder

Introduction, Method and Design

Introduction: Theory and background about the laser rangefinder.

The laser rangefinder used in the lab sessions was VL53L1X manufactured by STMicroelectronics (ST). It is integrated on a board, VL53L1X-SATEL which is also manufactured by ST, with a 2.8 to 5 V voltage regulator and a signal interface level shifter; hence it can be supplied by the Raspberry Pi Zero W's 3.3V or 5V supply pin [1]. The VL53L1X supports up to 4 meters distance measurement with accuracy in millimetre and up to 50Hz ranging frequency [1].

The working principle behind the VL53L1X is called Time of Flight (ToF). On the VL53L1X are two gold squares of the same size. They are emitter and receiver respectively. The emitter emits photons which are reflected back to the receiver after hitting an obstacle. Then, the distance between the VL53L1X and the obstacle can be obtained by multiplying the speed of light by half the time it takes for the photon to be emitted, hits the obstacle, and return.

Normally, measurement of rangefinders may be disturbed by environment conditions, such as objects colour and material roughness [1]. The same applies to VL53L1X. However, ST claims that its ST FlightSense technology used in VL53L1X overcomes several interferences well, so the results are reliable [1].

Method: Interfacing the range finder with the Pi via the I2C bus

Inter-Integrated Circuit (I2C) is a communication protocol. It is a low-cost method to integrate sensors systems into a robotic system [2]. I2C is serial, which means in this protocol, data is sent one bit per time compare to parallel communication which sends several bits as a whole. This implies the I2C only needs one or two buslines to transfer data. Actually, I2C uses two buslines. One transfers data is called SDA; the other conveys clock signal is called SCL [2].

By connecting the SDA and SCL pin of VL53L1X-SATEL to the I2C SDA and SCL pin of the Raspberry Pi respectively, and connect VCC to Pi's 3.3V supply pin, GND to Pi's GND pin, Raspberry Pi becomes an I2C controller, while the VL53L1X-SATEL is its peripheral.

Each peripheral needs to have an I2C address in order for the controller to access it via the I2C protocol. As per the user manual, the default I2C address for VL53L1X-SATEL is 0x29 [3]. If the controller, Raspberry Pi, can see this address, then it sees VL53L1X-SATEL. This logic can be used to detect whether the VL53L1X-SATEL is successfully connected with Raspberry Pi via I2C. i2cdetect was used to carry out this test:

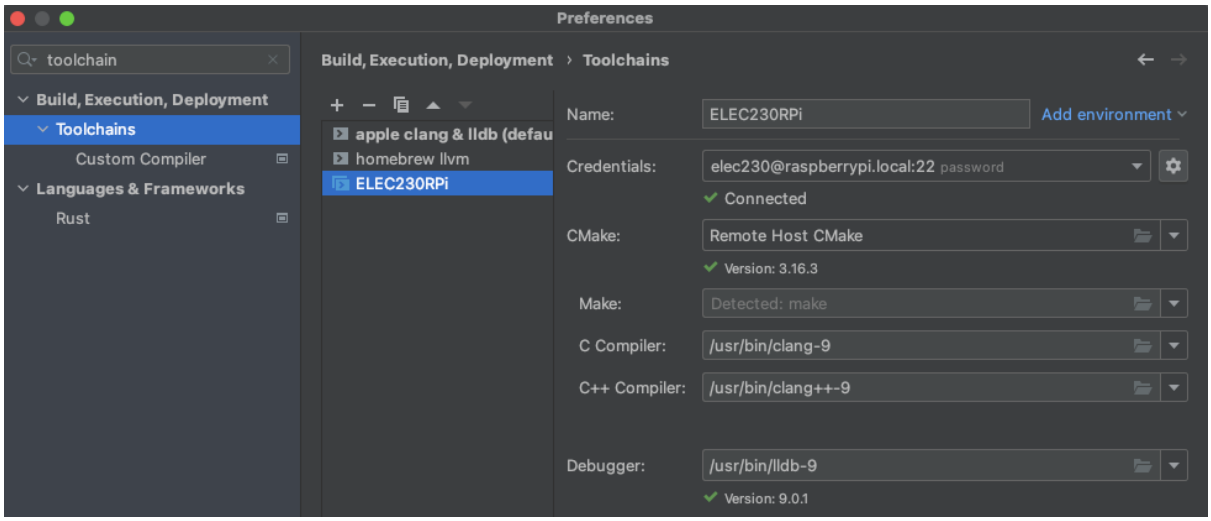
```
& elec230 @ raspberrypi in ~/projs/ass2/cmake-build-
~ sudo i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- 29 -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

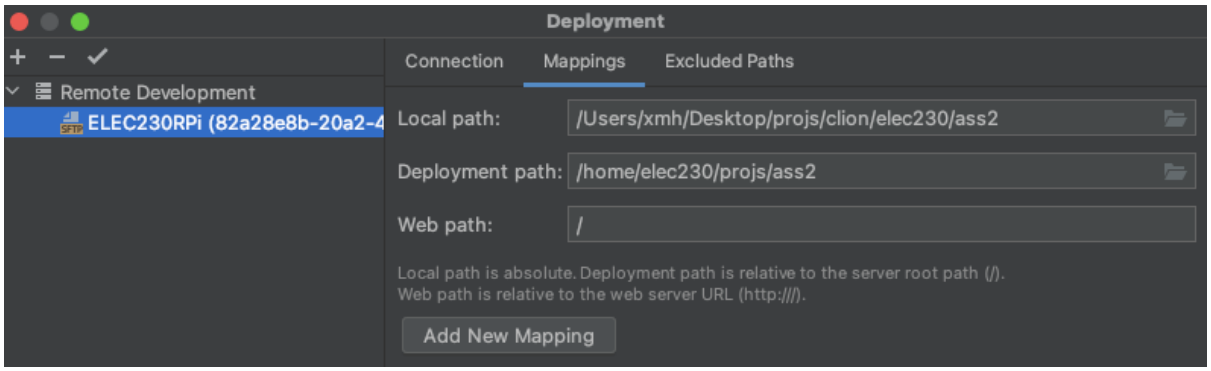
29 appeared in the table indicates a successful connection.

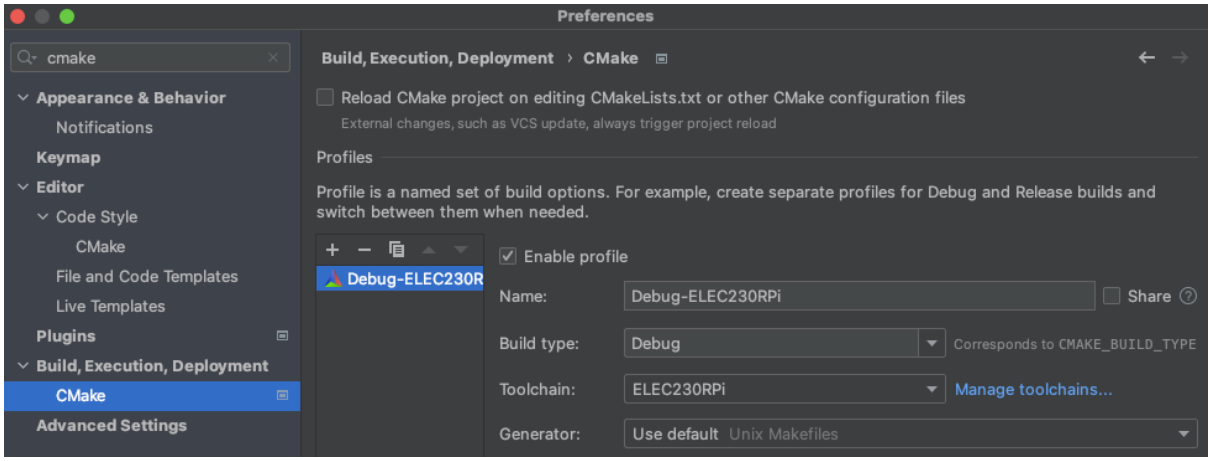
In I2C, each peripherals share the same wire must have a unique address [2]. From the table, it can be found that there are only 117 addresses. Therefore, in this case, one I2C controller which occupies GPIO 3 and 5 can have up to 117 peripherals.

Method: obtaining range measurements

Initially, a download manager, wget, an archive manager, 7z, and a build tool, GNU make, is used to download, extract, and build the demo code. The code is on the Raspberry Pi making it difficult to for developer to read and modify the code. CLion's features help to improve this development environment. I set up remote toolchain and remote deployment on CLion so as to implement a remote development workflow.

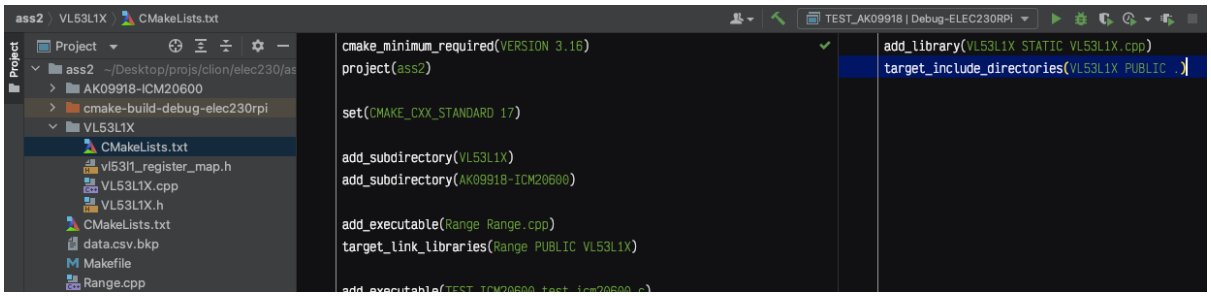






I also tried VSCode, but its remote feature does not support the Raspberry Pi Zero W's architecture.

I rearrange the file structure to split application which is Range.cpp from libraries, and I wrote a CMakeLists.txt to improve the problem of repetitive compilation reducing debug efficiency; If I only change the application, Range.cpp, it will only recompile the Range.cpp, not the unmodified lib, VL53L1X, and vice versa.



Interestingly, I found that the demo code actually does not depend on the lib wiringPi.

After building, CLion outputs a binary file, Range, into remote machine's build directory. Run with sudo permission:

```
& elec230 @ raspberrypi in ~/projs/ass2/cmake-build-debug-elec230rpi 0 [21:51:55]
~ sudo ./Range
[sudo] password for elec230:
Distance(mm): 13
Distance(mm): 13
Distance(mm): 13
Distance(mm): 14
Distance(mm): 15
Distance(mm): 11
Distance(mm): 14
Distance(mm): 12
Distance(mm): 12
Distance(mm): 11
Distance(mm): 14
Distance(mm): 12
Distance(mm): 12
Distance(mm): 13
^C

& elec230 @ raspberrypi in ~/projs/ass2/cmake-build-debug-elec230rpi 0 [21:52:01]
~
```

The unit is in millimetre.

Design: 5 drops of the ping pong ball

The aim of the experiment was to measure the displacement of a ping pong ball as it bounced up and down after being lowered from the air. The experiment requires the use of a ping pong ball, a tube slightly larger than the ping pong ball to restrict the ball to only do up and down movements, and the rangefinder. A total of five measurements were taken. The measurements were appended to a CSV file.

The ping pong ball was not held close to the rangefinder but at a distance from it that can barely be read by the rangefinder before it is released, because the rangefinder cannot measure at very close distances.

Further Work (includes method, results, and discussion)

In order to know the measurement rate, I specified the sample size and recorded the measurement time using lib chrono, then divided the sample size by the measurement time (See Range.cpp).

Surprisingly, the measurement rate was only about 11.1Hz, well below 50Hz.

```
Distance(mm): 12
Distance(mm): 12
Distance(mm): 12
Distance(mm): 13
Distance(mm): 13
Distance(mm): 12
Distance(mm): 12
Distance(mm): 13
Time(s): 8.93956
Sampling Rate: 11.1862 per second

& elec230 @ raspberrypi in ~/projs/ass2/cmake-build-debug-elec230rpi 0 [21:52:49]
~
```

I consulted the official ST API with its documentation and found that the rangefinder initialisation method was inside the for loop and suspected that it was slowing things down, so I brought it out of the loop (See Range.cpp). The readings become surprisingly fast, well above 50 Hz. However, I found that this was simply because the method that gets data reads old data in the register, and the initialisation method that proposes the loop happens to have the function of blocking the loop until the new data is written. I wrote a method, clearInterrupt(), based on the official documentation and the code on GitHub, by changing the state of register, VL53L1_SYSTEM__INTERRUPT_CLEAR which is defined in vl53l1_register_map.h, to 0x01, to make the inside while loop loops until new data is written (See Range.cpp). Unfortunately, the measurement rate returned to about 11.1Hz.

I2C utilises baud rate to control data transfer speed, so I guess it is possible to increase the measurement rate by increasing the baud rate. The default baud rate for the Raspberry Pi is 100k [4, 5]. No change in measurement rate after raising it to 900k.

```
& elec230 @ raspberrypi in ~/projs/a
~ cat /boot/config.txt | rg 900000
dtparam=i2c_arm_baudrate=900000
```

Reading through the official documentation and third-party code on GitHub, I found that **intermeasurement period** and **timing budget** could be set and might be related to the measurement rate, but I didn't have time to figure out.

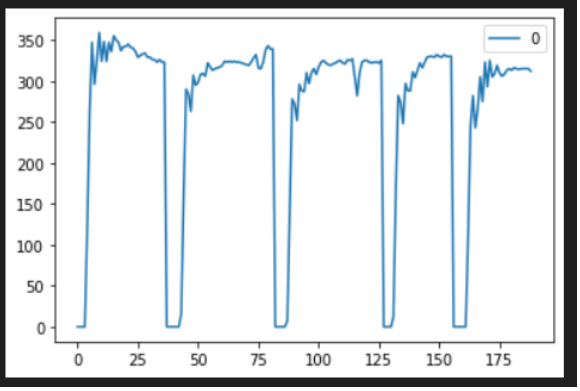
Printing the data on stdout is not a good way to analyse the data after the experiment, so I used lib fstream to save the data in csv format (See Range.cpp).

Results

Results: Ping pong experiment [and add subtitles for your other work]

```
import pandas as pd
data = pd.read_csv("data.csv")
data.plot()
```

```
<AxesSubplot:>
```



The unit for vertical axis is millimetre.

Discussion

Discussion: Ping pong experiment [and add subtitles for your other work]

The reason there is no reading when target is very close to the rangefinder is that the photon flight time is very short and a finite length variable is not sufficient to record such a very short time, so for the program the photon flight time is equivalent to zero, and the distance is zero too.

The fluctuations at the left end of the curves obtained from the five measurements represent the bouncing up and down of the ping pong ball. This process gradually decreases until it stops, i.e. the curve finally stabilises at the value of 300mm.

The first measurement is very different from the other ones. This is probably due to the fact that the hand-held rangefinder cannot be held in place very well.

The sampling frequency of the sensor is only 11.1, so the measured curve can only reflect the approximate bouncing of the table tennis ball up and down.

According to the documentation, the rangefinder actually measures a cylinder rather than a line [3]. This also explains the large deviations between measurements.

Discussion: analysis of the demo code

From the point of view of structure, Range.cpp is an application, while vl53l1_register_map.h, VL53L1X.cpp, and VL53L1X.h forms a library. To be more specific, vl53l1_register_map.h records the address of all registers; VL53L1X.h uses the object-oriented programming paradigm to abstract the rangefinder into a class by defining its behaviours and a private attribute deviceAddress; VL53L1X.cpp implements the behaviours. Then in the Range.cpp, the class VL53L1X is instantiated so as to operate the real VL53L1X.

In terms of functionality, the library encapsulates the modification and reading of the VL53L1X's register state, while the Range.cpp implements a basic logic of one measurement.

Discussion: Comparison of results with theory

No idea.

Part 2 (30%) IMU

Introduction, Method and Design

Introduction: Theory and background about the IMU

IMU used in the experiment consists of an accelerometer, gyroscope and magnetometer with a total of nine degrees of freedom. The accelerometer measures acceleration in three orthogonal directions, X, Y, and Z. The gyroscope measures the degree of rotation around the X, Y, and Z axis. The magnetometer measures the strength of the magnetic field along the X, Y, and Z axis.

The combination of gyroscope and accelerometer can already be used to estimate the relative position, velocity and acceleration of moving objects [6]. However, because of the integration operation involved in the measurement, the error is slowly amplified over time [6]. This is called drift [6]. Hence, it needs to be combined with other sensors to correct the drift. The camera and GPS can be used, but here we use a magnetometer which could correct the drift by pointing north [6].

Method: interfacing the IMU via the I2C bus

Theoretically, the IMU could share two baselines with the rangefinder and thus be controlled by a single I2C controller. However, the sensors do not work properly, probably because the pull-up resistance is too small when connected. Alternatively, the other two GPIO ports on the Raspberry Pi can be used to create a second I2C controller by editing /boot/config.txt and then rebooting.

```
& elec230 @ raspberrypi in ~/projs/ass2/cmake-build-debu
~ cat /boot/config.txt | rg i2c-gpio
dtoverlay=i2c-gpio,i2c_gpio_sda=5,i2c_gpio_scl=6,bus=2
```

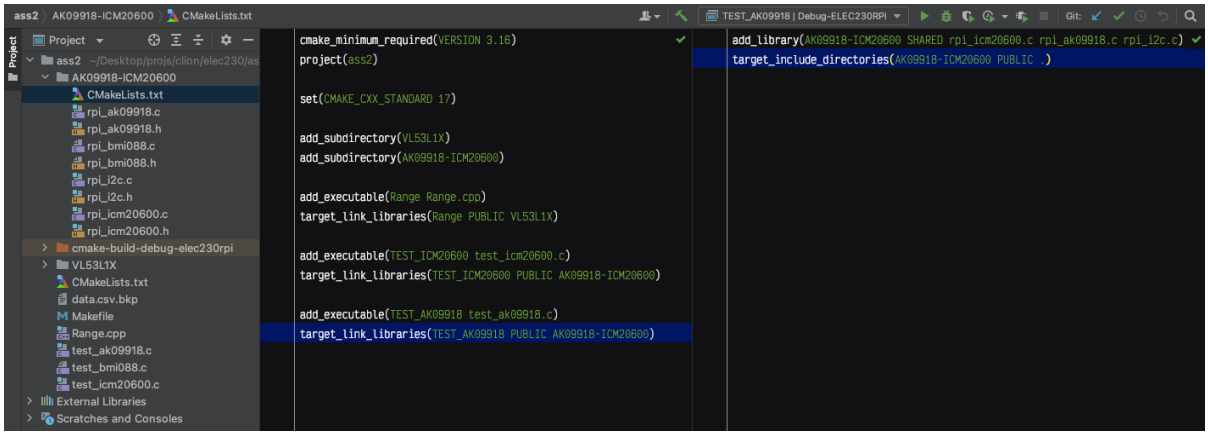
As in the part one, check whether the second controller has successfully established an I2C connection with the IMU.

```
& elec230 @ raspberrypi in ~/projs/ass2/cmake-build-
~ sudo i2cdetect -y 2
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- 0c -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- 69 -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

Two addresses appeared in the table. This is because the IMU consists of two sensors, ICM20600 and AK09918. ICM20600 has accelerometer and gyroscope functions, and AK09918 is magnetometer. The default I2C addresses of the two sensors are marked on the bottom of the IMU and are consistent with the test result, indicating a correct connection.

Method: obtaining measurements with the accelerometer and gyroscope

Same as part 1, separate the app, test_ak09918.c and test_icm20600.c, from the lib and write CMake to compile.



Compile and run TEST_ICM20600 with sudo privilege.

```
& elec230 @ raspberrypi in ~/projs/ass2/cmake-build
~ sudo ./TEST_ICM20600
Device ID: 0x11
Thermo  =   25.39 C
ACCEL X = -240.23 mg Y = -946.29 mg Z = -225.10 mg
GYRO  X =   -1.04dps Y =   -0.12dps Z =    0.06dps
Thermo  =   25.39 C
ACCEL X = -235.35 mg Y = -950.20 mg Z = -229.00 mg
GYRO  X =   -0.61dps Y =   -0.37dps Z =    0.24dps
Thermo  =   25.29 C
ACCEL X = -236.82 mg Y = -952.15 mg Z = -227.54 mg
GYRO  X =   -0.37dps Y =   -0.43dps Z =    0.18dps
Thermo  =   25.29 C
```

Method: obtaining measurements with the magnetometer

Compile and run TEST_AK09918 with sudo privilege.

```
& elec230 @ raspberrypi in ~/projs/ass2/cmake-build-debug-elec230r
~ sudo ./TEST_AK09918
Device ID: 0x480C
SELF TEST RESULT --- AK09918_ERR_OK: OK
AK09918_ERR_NOT_RDY: Not ready(check too quickly) usleep = 8001 us
COMPASS X =    4.65 uT Y =   38.40 uT Z =   20.55 uT
COMPASS X =    4.05 uT Y =   38.25 uT Z =   19.65 uT
COMPASS X =    3.90 uT Y =   37.05 uT Z =   19.80 uT
COMPASS X =    4.80 uT Y =   37.50 uT Z =   18.90 uT
COMPASS X =    4.50 uT Y =   39.00 uT Z =   18.15 uT
COMPASS X =    4.95 uT Y =   38.40 uT Z =   20.10 uT
COMPASS X =    3.90 uT Y =   37.05 uT Z =   20.85 uT
```

Design: experiments with the three IMU sensors

The IMU is marked with the positive direction of XYZ axes and rotation.

Accelerometer: Point the XYZ axes towards the ground and observe the accelerometer reading. Shake the IMU and observe the reading.

Gyroscope: Slowly rotate the board around XYZ axes in the positive direction marked on the board and observe the gyroscope reading.

Magnetometer: Point the XYZ axes to the North Pole and observe the magnetometer reading. Hold the electronic device close to the magnetometer and observe the reading.

Results and Discussion

Results and Discussion: accelerometer

The reading of the axis pointing to the ground is approximately -1000 mg. If the axis is reversed, the reading will be 1000 mg. Therefore, the reading is 1000 mg when the axis points out of and perpendicular to the ground. For example, when the IMU is flat, the Z-axis is in this state.

```
& elec230 @ raspberrypi in ~/projs/ass2/cmake-build
~ sudo ./TEST_ICM20600
Device ID: 0x11
Thermo  =   25.10 C
ACCEL X =   -5.86 mg Y =    0.00 mg Z =  998.05 mg
GYRO  X =   -1.16dps Y =   -0.67dps Z =    0.00dps
Thermo  =   25.44 C
ACCEL X =  -10.74 mg Y =   -8.30 mg Z = 1000.98 mg
GYRO  X =   -0.31dps Y =   -0.61dps Z =    0.00dps
Thermo  =   25.10 C
ACCEL X =   -8.79 mg Y =   -8.30 mg Z =  999.51 mg
GYRO  X =   -0.31dps Y =   -0.31dps Z =    0.24dps
Thermo  =   25.10 C
ACCEL X =   -4.39 mg Y =   -4.88 mg Z = 1000.00 mg
GYRO  X =   -0.61dps Y =   -0.18dps Z =    0.12dps
Thermo  =   25.10 C
ACCEL X =   -5.37 mg Y =   -8.79 mg Z =  999.02 mg
GYRO  X =   -0.12dps Y =   -0.61dps Z =    0.06dps
Thermo  =   24.76 C
ACCEL X =   -6.84 mg Y =   -6.84 mg Z =  998.05 mg
GYRO  X =   -0.43dps Y =   -0.37dps Z =    0.37dps
^C
```

The unit mg is the gravitational acceleration times ten to the power of minus three. Hence, 1000mg means one gravitational acceleration.

The only acceleration an object at rest has is the gravitational acceleration, which is consistent with the reading.

When none of the three axes of the accelerometer are perpendicular to the ground, the reading on each axis is the component of the gravitational acceleration.

When shaking the IMU along an axis, a larger reading can be observed for this axis.

Results: gyroscope

Zero reading in each axis when the gyroscope is at rest, regardless of the stance of the IMU.

```
& elec230 @ raspberrypi in ~/projs/ass2/cmake-build
~ sudo ./TEST_ICM20600
Device ID: 0x11
Thermo  =   25.10 C
ACCEL X =   -5.86 mg Y =    0.00 mg Z =  998.05 mg
GYRO  X =   -1.16dps Y =   -0.67dps Z =    0.00dps
Thermo  =   25.44 C
ACCEL X =  -10.74 mg Y =   -8.30 mg Z = 1000.98 mg
GYRO  X =   -0.31dps Y =   -0.61dps Z =    0.00dps
Thermo  =   25.10 C
ACCEL X =   -8.79 mg Y =   -8.30 mg Z =  999.51 mg
GYRO  X =   -0.31dps Y =   -0.31dps Z =    0.24dps
Thermo  =   25.10 C
ACCEL X =   -4.39 mg Y =   -4.88 mg Z = 1000.00 mg
GYRO  X =   -0.61dps Y =   -0.18dps Z =    0.12dps
Thermo  =   25.10 C
ACCEL X =   -5.37 mg Y =   -8.79 mg Z =  999.02 mg
GYRO  X =   -0.12dps Y =   -0.61dps Z =    0.06dps
Thermo  =   24.76 C
ACCEL X =   -6.84 mg Y =   -6.84 mg Z =  998.05 mg
GYRO  X =   -0.43dps Y =   -0.37dps Z =    0.37dps
^C
```

The unit dps means degree per second.

When rotating around an axis in the positive direction indicated on the IMU, the reading is positive. The opposite is negative.

Results: magnetometer

See above. Clearly your focus will be on the 7th and 8th bullet points plus any other points you think are interesting [‘Bonus’].

When one axis points to the ground, the sum of the components of the two axes parallel to the ground points to the North Pole. In practice, however, this is not the case. The magnetometer cannot indicate which direction is north probably because the geomagnetic field itself is very weak and is disturbed by the magnetic fields generated by the surrounding electronics and the steel in the building.

When an electronic device is placed close to the magnetometer, the reading increases significantly. The magnetic field of the electronic device itself should be small. This indicates that the magnetometer is susceptible to interference from the ambient magnetic field.

```
COMPASS X =   44.70 uT Y = -147.00 uT Z =   53.40 uT
COMPASS X =   45.45 uT Y = -145.65 uT Z =   52.65 uT
COMPASS X =   44.25 uT Y = -146.70 uT Z =   51.15 uT
COMPASS X =   44.55 uT Y = -146.10 uT Z =   51.00 uT
COMPASS X =   43.35 uT Y = -146.10 uT Z =   51.75 uT
COMPASS X =   44.70 uT Y = -145.65 uT Z =   51.90 uT
COMPASS X =   44.10 uT Y = -147.30 uT Z =   51.30 uT
COMPASS X =   44.25 uT Y = -147.00 uT Z =   50.85 uT
COMPASS X =   44.85 uT Y = -146.10 uT Z =   51.00 uT
COMPASS X =   44.85 uT Y = -146.10 uT Z =   51.75 uT
COMPASS X =   45.60 uT Y = -146.55 uT Z =   53.10 uT
COMPASS X =   44.55 uT Y = -145.80 uT Z =   52.05 uT
COMPASS X =   45.75 uT Y = -148.20 uT Z =   51.60 uT
COMPASS X =   44.85 uT Y = -147.90 uT Z =   51.00 uT
COMPASS X =   44.55 uT Y = -146.85 uT Z =   50.25 uT
COMPASS X =   44.85 uT Y = -145.80 uT Z =   50.70 uT
COMPASS X =   43.80 uT Y = -147.60 uT Z =   50.25 uT
COMPASS X =   44.85 uT Y = -146.85 uT Z =   51.00 uT
COMPASS X =   44.55 uT Y = -146.10 uT Z =   50.25 uT
COMPASS X =   44.70 uT Y = -146.40 uT Z =   51.60 uT
COMPASS X =   45.60 uT Y = -146.55 uT Z =   50.70 uT
COMPASS X =   45.15 uT Y = -146.10 uT Z =   51.30 uT
```

The reading is in uT, which means ten to the power of minus six Tesla.

The northern hemisphere is at the south pole of the geomagnetic field, so the geomagnetic field slopes downwards. This explains why the value becomes larger when X axis is pointed towards the ground.

Part 3 (20%) Supplementary Questions

1. Gyroscopes and accelerometers are proprioceptive because angular velocity and acceleration they measure are information about the object itself. Magnetomers are exteroceptive because the magnetic field is information external to the object.
2. Gyroscopes, accelerometers and magnetometers determine their values by measuring energy from the outside world and do not require energy consumption of their own, whilst infrared sensors emit infrared radiation which is a type of energy to receive the environmental reaction which return back by the infrared radiation. One other example of an active sensor is optical encoder. It emits and receives light shone through a coded disk to measure rotational motion.
3. In order to let GPS achieve high accuracy, base stations need to be set up nearby, which is inconvenient. Magnetometers are susceptible to interference by strong magnetic fields in the environment. Mechanical gyroscope would accumulate error over time.
4. Concurrency does not require multiple tasks to be computed simultaneously, but rather means that multiple tasks are executed without sequential relationships. It is possible to use threads to allow multiple programs to share a single processor and have the effect of appearing to be calculating simultaneously.
5. C++ and C have no garbage collector. Memory needs to be managed by the programmers themselves. If there is a need for a garbage collector, the programmer can implement it himself. There are third-party implementations, for example, Boehm–Demers–Weiser garbage collector.
6. Pipe is classified as unnamed and named, but (|) represents only the unnamed pipe. Unnamed and named pipe are both used to redirect standard output, input, or error of a command. However, unnamed pip is created temporarily, for example, cat log | less, whilst named pipe is a special file which exists until it is removed and is created by mkfifo.
7. This is not a good analogy. Firstly, four chargers can charge simultaneously, whereas threads just make tasks appear to be running at the same time, and behind the surface, one computing resource switches between them quickly. On the other hand, four chargers do not run into race condition, whilst threads do. Moreover, the cars and the semaphores are completely unrelated. Semaphore manages how many threads can access a resource at the same time. In this case, the number of vacant chargers is more appropriately analogous to the semaphore. Last but not least, a thread is created for a task, and after the task is done, the thread is released; whereas the chargers are constantly present and waiting for tasks. In combination with the first point, the charging station is more like a process pool which has four workers.

Reference

[1] ELEC230 Lecture 9b PPT

[2] ELEC230 Lecture 10a PPT

[3] https://www.waveshare.com/wiki/VL53L1X_Distance_Sensor

[4] https://www.raspberrypi.com/documentation/computers/processors.html

[5] https://datasheets.raspberrypi.com/bcm2835/bcm2835-peripherals.pdf

[6] Intro to autonomous mobile robots