Exercise Manual
For
SC2104/CE3002
Sensors, Interfacing and Digital Control

Practical Exercise #2:
Operating Characteristics of IMU's Accelerometer and Gyroscope

Venue: SCSE Labs

COMPUTER ENGINEERING COURSE

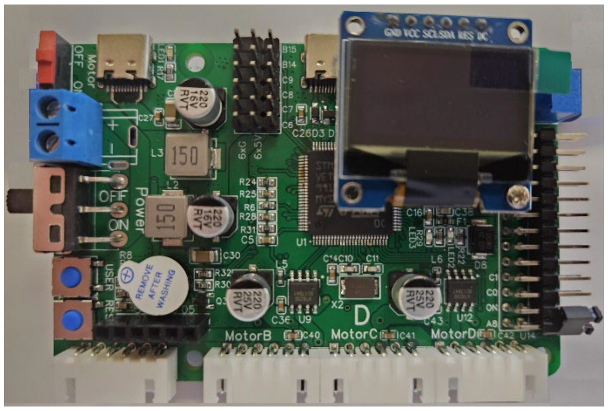
SCHOOL OF COMPUTER SCIENCE AND ENGINEERING
NANYANG TECHNOLOGICAL UNIVERSITY

Learning Objectives

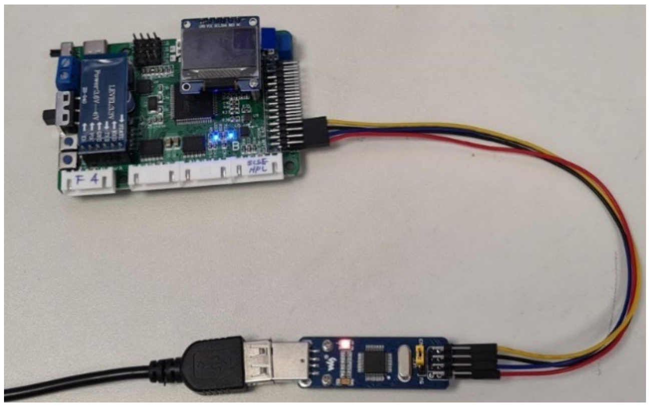
Continue from Exercise #1, you will now develop programs to access the on-board IMU's Accelerometer and Gyroscope devices on the STM32F4 platform, and to examine and understand the characteristics of these devices during operations.

Equipment and accessories required

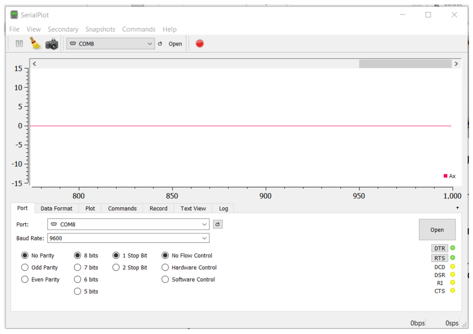
i) One desktop computer installed with STM32CubeIDE, PuTTY and SerialPlot software.
ii) One STM32F4 board (Version D)
iii) One ST-LINK SWD board



STM32F4 board



ST-Link for downloading and debugging code



Port: COM8
Baud Rate: 9600
[✓ No Parity] Odd Parity Even Parity
[✓ 8 bits] 7 bits 6 bits 5 bits
[✓ 1 Stop Bit] 2 Stop Bit
[✓ No Flow Control] Hardware Control Software Control

SerialPlot's UI

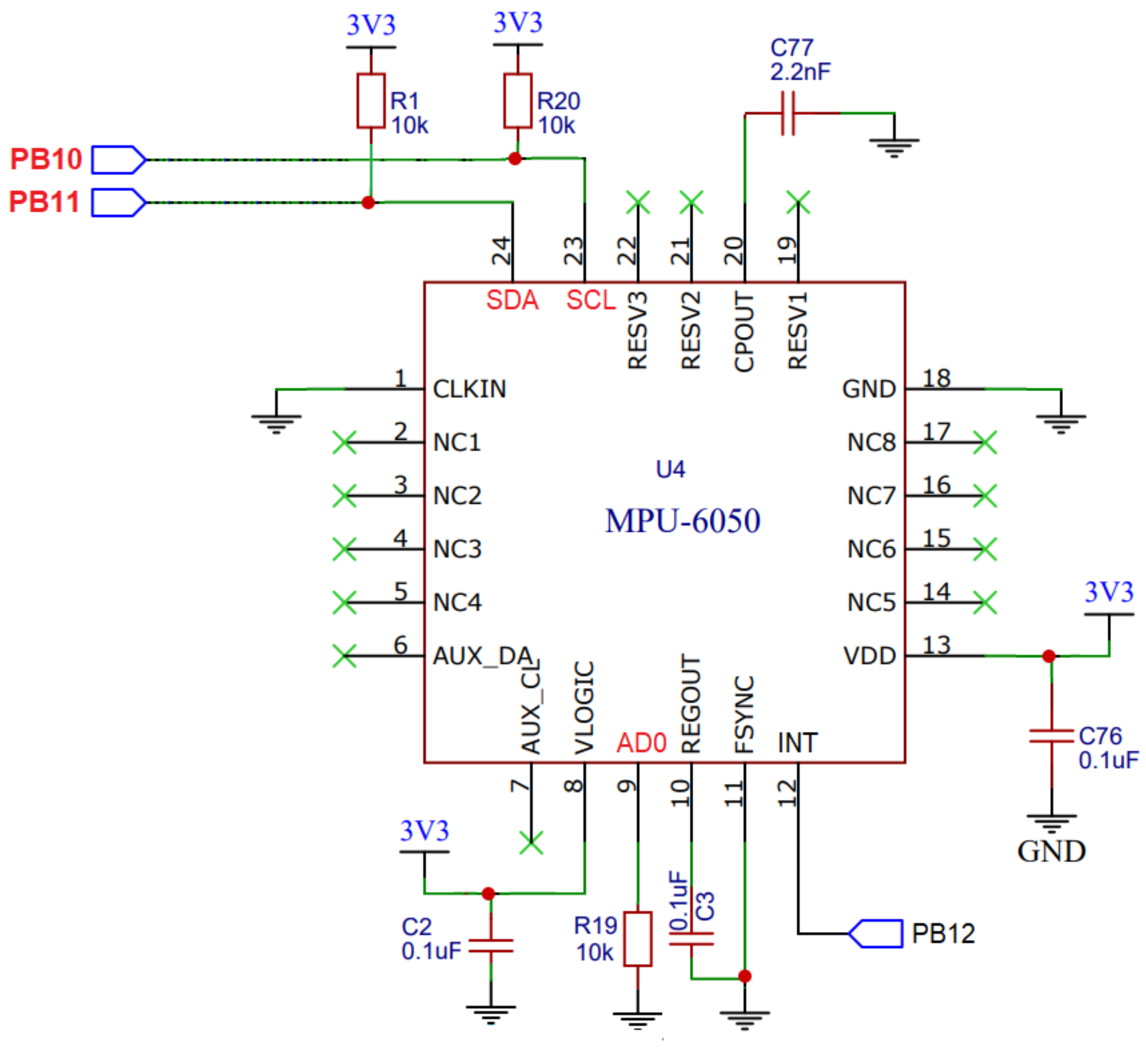
Introduction

Typical IMU contains devices such as Accelerometer, Gyroscope and Magnetometer that are used widely in many equipment and machines, especially those that are designed to operate autonomously such as drones and vehicles for navigation purpose. However, to use these devices effectively, it is importance to first understand the characteristics and limitation of these devices.

In the following exercises, you will first learn how to access an IMU that are connected to the microcontroller through an I^{2}C interface. You will then examine the devices' real time operating characteristics, by collecting and sending their measurements through the STM32F4's serial port and display them on the computer using the "SerialPort" software.

1. I²C interface of the IMU

The STM32F4 (Version D) board contains a 6-axis IMU device, MPU-6050, which is connected to the microcontroller based on the I²C interfaces as shown in the following schematic diagram.



As such, you will need to first configure the appropriate microcontroller's I/O pins to function as the I²C interface pins in order to access the IMU.

1.1. Configuration of the I²C interface STM32CubeIDE

Study the schematic diagram above and configure the appropriate I/O pins for the IMU interface using the STM32CubeIDE's GUI based configuration feature. (Hint: you will find that the relevant I/O pins to be used have built-in support for I²C functions.)

If this is done correctly, you would notice that a function `MX_I2C1_Init();` is generated and added into the main.c file. You will also notice the following declarations been generated and stored in the file (which you need to use when calling the IMU related functions.)

```
/* Private variables ---------------------------------------------*/
I2C_HandleTypeDef  hi2c2;    // this is for the I²C
UART_HandleTypeDef huart3;  // this is for ??
```

1.2. IMU Driver code

The code that are used to access and to obtain the IMU's measurements are available in NTULearn course-site, which consists of two files: `"MPU6050.c"` and its header file `"MPU6050.h"`. The following are brief descriptions of some relevant information in these two files that can be used in your program to operate the IMU.

(i) You will need to declare a variable with the structure type `IMU_Data`;; E.g., `IMU_Data imu`; Declaration of the `IMU_Data` structure is in the `"MPU6050.h"` file which is as shown below:

```
typedef struct {
   I2C_HandleTypeDef *i2cHandle;
   UART_HandleTypeDef *uart;
   float acc[3];   // store the 3 axes Accelerometer measured values
   float gyro[3];  // store the 3 axes Gyroscope measured values
  } IMU_Data;
```

(ii) The following function is to be called from your program to initialize the IMU device before its operation.

```
uint8_t* IMU_Initialise(IMU_Data *dev, I2C_HandleTypeDef *i2cHandle, UART_HandleTypeDef *uart)
```

E.g., `uint8_t status = IMU_Initialise(&imu, &hi2c2, &huart3);`

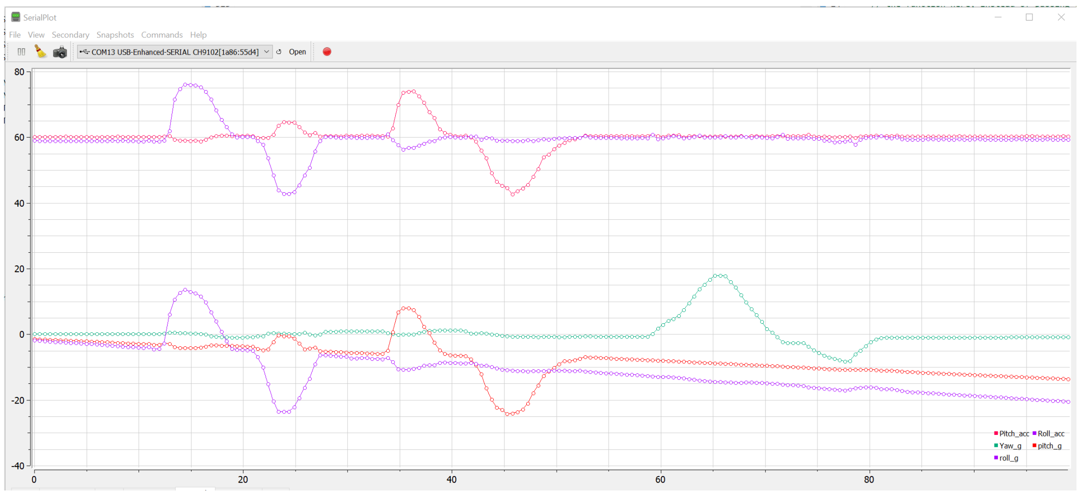
The return value `status` is a code that indicates the execution status of the function (0 = no error). (You can study the code of the function in the `"MPU6050.c"` file to find out all the possible return values).

(iii) The function `IMU_AccelRead(IMU_Data *dev)` can be used to read the accelerometer values.

(iv) The function `IMU_GyroRead(IMU_Data *dev)` can be used to read the gyroscope values

1.3. Display of readings on SerialPlot

Once you obtain the readings from the Accelerometer and Gyroscope, they can be sent through the uart serial port to the computer and displayed using the SerialPlot software.


SerialPlot
File View Secondary Snapshots Commands Help
COM13 USB-Enhanced-SERIAL CH9102[1a86:55d4]
Open
80
60
40
20
0
-20
-40
0
20
40
60
80
Pitch_acc
Roll_acc
Yaw_g
pitch_g
roll_g

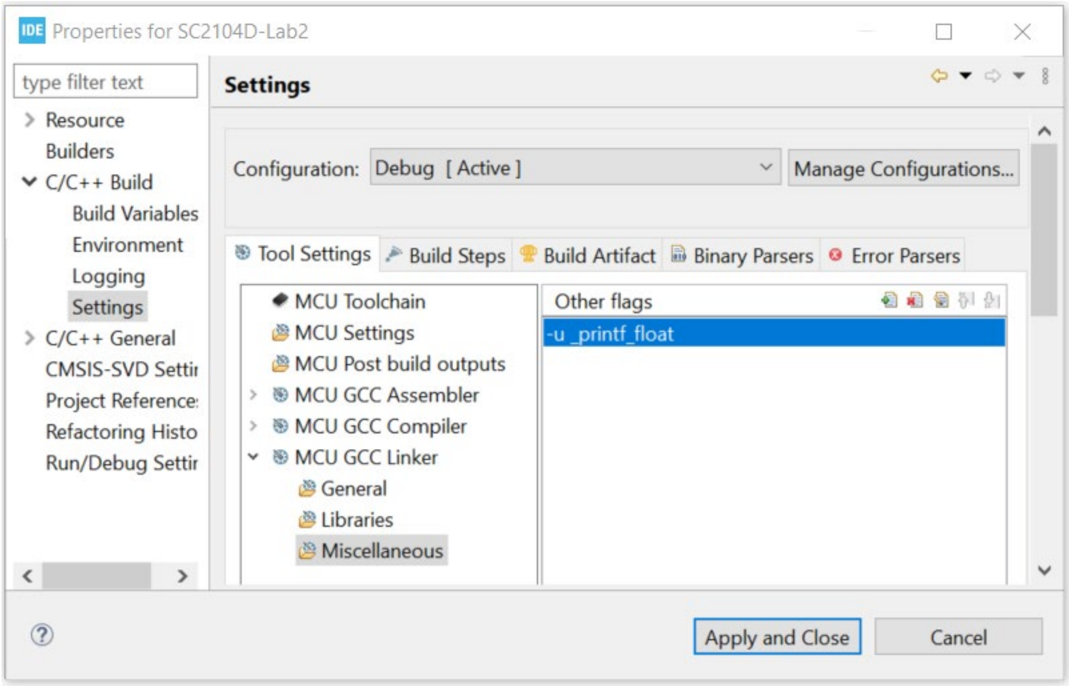
The UART function (that you had used in Exercise #1) `HAL_UART_Transmit()` can only transmit unsigned 8-bit integer data type. On the other hand, the data values from the IMU are of the **float** data type. As such, you will need to convert the float data type to char, before calling the UART function.

One way to do so is to use the `sprintf()` function such as the following:

```
sprintf(sbuf, "%5.2f, ", imu.acc[i]);
HAL_UART_Transmit(&huart3, sbuf, 8, HAL_MAX_DELAY);
```

with the declaration `char sbuf[10];`

However, the floating-point support for `printf()` is not enabled by default in STM32CubeIDE. To do so, we need to add the flag `-u _printf_float` to its linker option. (`-u` flag is to force linking and loading of library module that is undefined, in this case, `_printf_float`)


Properties for SC2104D-Lab2

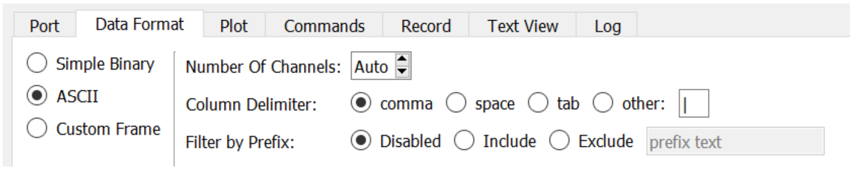
Settings

Configuration: Debug [Active]

C/C++ Build > Settings > Tool Settings > MCU GCC Linker > Miscellaneous

Other flags: -u _printf_float

The SerialPort program that is used to display the data (sent by the microcontroller) on the computer also need to be configured with appropriate settings such as the following.


Data Format: ( ) Simple Binary [✓ ASCII] ( ) Custom Frame

Number Of Channels: Auto

Column Delimiter: [✓ comma] ( ) space ( ) tab ( ) other:

Filter by Prefix: [✓ Disabled] ( ) Include ( ) Exclude

Note the Column Delimiter option which you can select to be used by the SerialPort software to display the individual data it receives.

Hints: (i) You will need to further separate the readings into individual groups, e.g. separate them into different rows for each round of readings.

(ii) You can use Putty first to observe whether the data you sent are in the expected format.

2. Practical Exercises

Based on the above information (and lecture notes), you will now develop the code to perform the various operations as described below.

2.1. Configure the I²C interface pins of the Microcontroller

As described in section 1.1, configure the appropriate microcontroller I/O pins to enable access of the IMU using the I²C based interface.

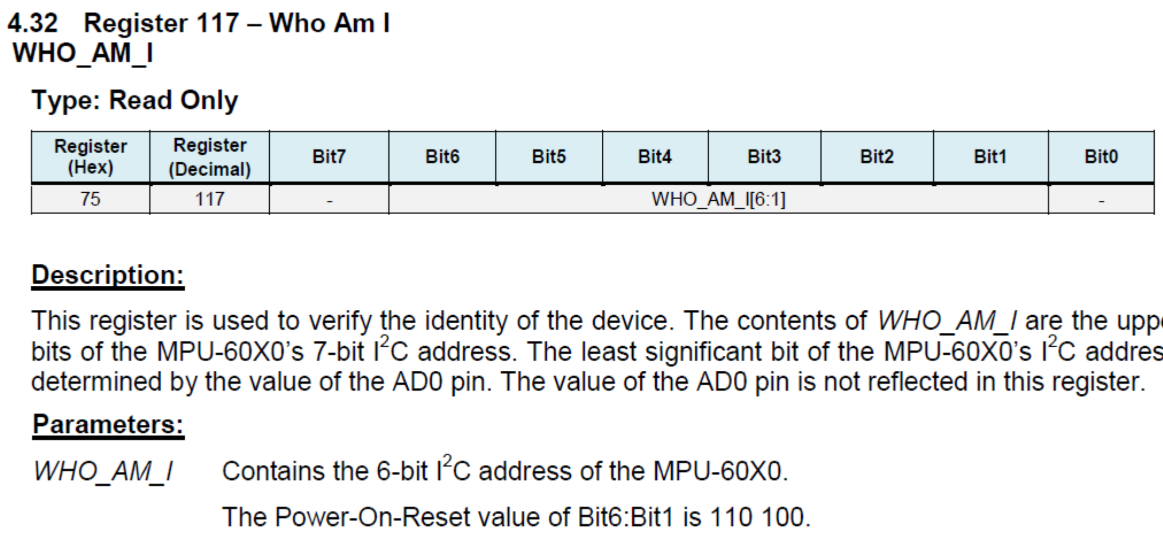
2.2. Defining the IMU address of the MPU6050

I²C bus interface makes use of unique I²C device address to communicate between the Master (i.e., the microcontroller) and the slave (i.e., the IMU device). (Refer to lecture note on I²C for detail.)

As such, you need to define the appropriate I²C address (7-bit) of MPU6050 IMU in your program by including the following statement in your `main.c` file:

```
uint8_t IMU_ADDR = ????<<1;  // address of MPU6050, and left shift by 1 bit(why?)
```

The following is the description of the MPU6050's register about its (default) address setting.



4.32 Register 117 – Who Am I
WHO_AM_I

Type: Read Only

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
75	117	-	WHO_AM_I[6:1]						-

Description:

This register is used to verify the identity of the device. The contents of *WHO_AM_I* are the upper 6 bits of the MPU-60X0's 7-bit I²C address. The least significant bit of the MPU-60X0's I²C address is determined by the value of the AD0 pin. The value of the AD0 pin is not reflected in this register.

Parameters:

WHO_AM_I	Contains the 6-bit I²C address of the MPU-60X0. The Power-On-Reset value of Bit6:Bit1 is 110 100.

Based on the above description, determine the value of the I²C address that you will use for the MPU-6050 device in your program (Hint: also refer to the schematic diagram on page 3 for relevant information).

2.3. Accessing the Accelerometer

Code the routines required to access the orientation values from the IMU's Accelerometer. Send the values to the SerialPort program to display them in real time on the computer.

Check that your code is working accordingly by rotating the STM32F4 board in different orientations and observe the changes in the output values (as displayed in the SerialPlot program).

2.4. Accessing the Gyroscope

Similarly, code the routines required to read the Gyroscope's values and display them on the SerialPort program to verify that the Gyroscope is operating accordingly.

2.5 Deriving the Pitch and Roll angles from Accelerometer

Add the instructions to derive the Pitch and Roll angles by using the appropriate data from the Accelerometer readings (as discussed in the lecture).

Hint: To execute the tan^{-1} operation, you can use the function `atan2(A/B)` which is available in the library `<math.h>`

- Send the values to the SerialPort program for display.
- Ascertain that the program is behaving accordingly when you pitch and roll the STM32F4 board at different angles.
- Observe the change in the displayed values when you move and shake the board. Add appropriate code to your program to reduce/overcome the effect observed.

2.6 Deriving the Yaw, Pitch and Roll angles from Gyroscope

Add the codes to derive the Yaw, Pitch and Roll angles using the readings obtained from the Gyroscope.

Note that the readings output by the Gyroscope is in unit of angular velocity (in degree/second). To find the rotated angle (in degree), you will need to integrate the readings with the elapse time (Δt) between reading of the values. The following instructions may be useful for you to implement the required operations.

```
uint32_t millisOld, millisNow;  // time value
float dt;// time elapse

millisNow = HAL_GetTick(); // get the current time
dt = (millisNow - millisOld)*0.001; // time elapsed in millisecond
millisOld = millisNow; // store the current time for next round
```

- Send the values to the SerialPort program for display. Ascertain that the program is behaving accordingly when you change the orientations of the STM32F4 board to different angles.
- Compare its response with the Accelerometer under similar operating conditions.

3. Summary of observations

Compare and comment on the accuracy and behavior of the Pitch and Roll angles derived from the Accelerometer and Gyroscope respectively.