Lab4 - FYS4240

In this document you can find answers to the following tasks for lab4:

- **Exercise 1:** 1.3, 1.4, 1.5 and 1.6
- **Exercise 4:** 4.3, 4.4 and 4.6

Exercise 1

For exercise 1 the IMU library file is `LSM9DS1.cpp` is substituted with `BMI270.cpp` and the IMU datasheet can be found here: `BMI270 datasheet`

Note to self:

In establishing the big picture it is vital to understand the IMU and its components:

- 3-axis accelerometer
- 3-axis gyroscope
- Configuring the BMI280 using writing operations with specific values to its registers using I^2C or SPI (communication protocols used between micro-controller and sensor)

That is why it is important to understand register addresses and values in the datasheet and how they are implemented in the IMU library file.

```
bmi2_i2c_read() //for reading data from a Bosch BMI270 sensor over I2C
                //Typical implemntation of an I2C read operation
following common
                //pattern of first sending the register address, then
reading back
                //the requested data in a seperate transaction

bmi2_i2c_write() //handles writing data to hte BMI270 sensor over I2C

int BoschSensorClass::begin(CfgBoshSensor_t cfg)
// Initialized the BMI270 IMU and BMM150 magnetometer sensors
// 1. I2C sets up communication interfaces for both sensors
// 2. Assigns I2C address and communicaiton functions with interface
pointers and device info

//functions for reading the IMU sensors:

int BoschSensorClass::readAcceleration(float& x, float& y, float& z)
int BoschSensorClass::readGyroscope(float& x, float& y, float& )
int BoschSensorClass::readMagneticField(float& x, float& y, float& z)
```

1.3

The information about acceleration sensor can be found in section *5.2.41* and *5.2.42*. The register map ca be found in section *5.2*. The data was found from the datasheet. Assuming `sample rate = output data rate` the necessary adjustments are listed in Table 1:

Address	Name	Value
0x40	ACC_CONF	`0x07 = odr_50`
0x41	ACC_RANGE	`0x00 = range_2g`

Table 1: Value adjustments with respect to addresses

1.4

The necessary changes can be made to `int BoschSensorClass::readAcceleration()`-function. The altered code function below has the following changes:

1. Substitutes the float-datatype with int for x,y,z as input parameters.
2. Removed division of acceleration sensor data, i.e. removed the `INT16_to_G` as it is used to convert the data to g. What we end up with are raw integer format which should be later scaled.

```
// Accelerometer
int BoschSensorClass::readAcceleration(int& x, int& y, int& z) {
  struct bmi2_sens_data sensor_data;
  auto ret = bmi2_get_sensor_data(&sensor_data, &bmi2);
  #ifdef TARGET_ARDUINO_NANO33BLE
  x = -sensor_data.acc.y;
  y = -sensor_data.acc.x;
  #else
  x = sensor_data.acc.x;
  y = sensor_data.acc.y;
  #endif
  z = sensor_data.acc.z;
  return (ret == 0);
}
```

1.5

The main advantage is **faster transmission** since we reduce the data size by using integers. Floating-points take up 4 bytes (32bits) in memory, while compared to integers, e.g. `int8_t` and `int16_t`, respectively take up 1 byte(8bits) and 2 bytes(16bits). Since serial communication can be relatively slow compared to internal processing on micro controllers. The effect is a reduced number of transmitted bytes and decrease in transmission time.

I think this answers the question, but there are other benefits to mention, such as processing efficiency as integer operations are considered faster than floating-point arithmetic, where the latter is more prone to precision loss and rounding errors, since floating-points cannot represent all decimal values precisely. Take decimal fractions as an example: 0.1 in decimal is hard to represent in binary (0.0001100110011001100110011001100110011001100110011001).

1.6

Serial data would be sent from the Arduino using `Serial.write()` rather than `Serial.println()` and split the number into a low byte and a high byte, i.e. using a Little-Endian Byte Order, with the most significant byte first, at the lowest byte address. Arduino Cookbook presents an example for this by sending first a header followed by two integer(two-byte) values as binary data, using `lowByte()` and `highByte()`.

Exercise 4

4.4

4.5

4.6

Given that the transformation matrix $R_S^B = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{pmatrix}$. Determine how the basis of M is represented in B:

1. **M's x-axis => $B's$ negative x-axis:**

$X_B = -X_M$: first column of R_M^B is $\begin{pmatrix} -1 \\ 0 \\ 0 \end{pmatrix}$

1. **M's y-axis => $B's$ y-axis:**

$Y_B = Y_M$: first column of R_M^B is $\begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$

1. **M's z-axis => $B's$ z-axis:**

$-Z_B = -Z_M$: first column of R_M^B is $\begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix}$

Thus the resulting transformation matrix, $R_M^B = \begin{pmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{pmatrix}$.

Assuming that the z-axis pointing upwards gives it a negative component.

Exercise 4

Exercise 4.3

For this exercise I used the data generated in **4.2** using the `SerialDataReadWrite_3CH_parallel.vi`. The data can be accessed through `magnetometer_readings`-file. The magnetometer was rotated 360 degrees in the horizontal plane. From Figure 1 we can read the offsets ΔB_x and ΔB_y to be $(-18.0570, -3.2854)\text{uT}$.

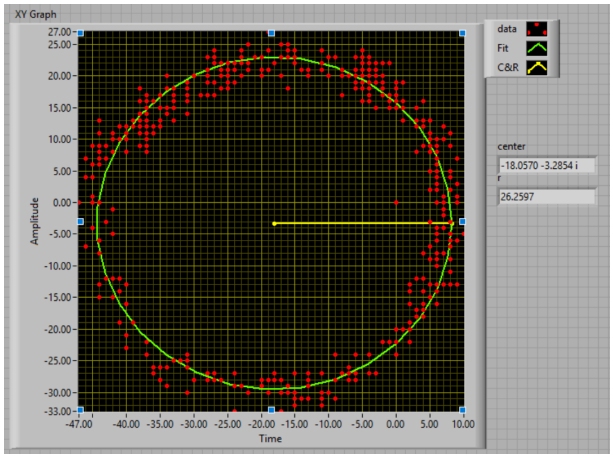

Figure 1: Readings

Exercise 4.6

The indoors are usually constrained in terms of space and most likely contain several electromagnetic sources that can distort the magnetometer readings. These sources can be many things, for example ferrous metal objects: cables and wirings, iron beams, refrigerators and other furnitures - objects made out of ferrous materials. Also electrical appliances are worth mentioning. I read a spike in the readings when touching the Arduino with cable that connects it to the laptop. Lastly, we can also find electronic devices in indoor that can also induce magnetic fields. All these factors can make the measured magnetic north unreliable.