



Capteurs et chaîne de mesure

Compte-rendu de travaux pratiques

Elio FLANDIN
Kevin DUGARD

Sommaire

I - Prise en main

II - Acquisition des données

III - Modification des paramètres métrologiques

IV - Calibration

V - Boussole électronique



I - PRISE EN MAIN

Initialisation

```
/*
 * @brief Test du fonctionnement des LEDs verte et rouge.
 *
 * Allume la LED pendant 1 seconde puis l'éteint.
 * Permet de vérifier la configuration GPIO et la fonction HAL_Delay.
 */
void test_LED(void) {
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_SET); // LED verte ON
    HAL_Delay(1000);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET); // LED verte OFF
}
```

```
/**
 * @brief Initialisation UART + message d'accueil.
 * Affiche un message de bienvenue sur le terminal série
 * et prépare l'environnement de test.
 */
void setup(void) {
    strcpy((char*)message, "MSC 2025 - Capteurs\r\n");
    HAL_UART_Transmit(&hlpuart1, prompt, strlen((char*)prompt), HAL_MAX_DELAY);
    HAL_UART_Transmit(&hlpuart1, message, strlen((char*)message), HAL_MAX_DELAY);

    test_LEDs();
    I2C_scan();
}
```

```
/**
 * @brief Scan du bus I2C pour détecter les périphériques connectés.
 * Balaye toutes les adresses possibles (0x00 à 0xFF) et affiche
 * sur le terminal les adresses des périphériques répondant.
 * @note Utilise la fonction HAL_I2C_IsDeviceReady().
 */
void I2C_scan(void) {
    foundCount = 0;
    HAL_UART_Transmit(&hlpuart1, (uint8_t*)"r\n*** Début du scan I2C ***\r\n", 30, HAL_MAX_DELAY);

    for (uint8_t address = 0; address < 128; address++) {
        if (HAL_I2C_IsDeviceReady(&i2c1, (uint16_t)(address << 1), 1, 10) == HAL_OK) {
            foundAddresses[foundCount++] = address;
            sprintf((char*)message, sizeof(message), "Adresse détectée : %d\r\n", address);
            HAL_UART_Transmit(&hlpuart1, message, strlen((char*)message), HAL_MAX_DELAY);
        }
    }

    sprintf((char*)message, sizeof(message),
            "\r\nNombre total d'adresses détectées : %d\r\n", foundCount);
    HAL_UART_Transmit(&hlpuart1, message, strlen((char*)message), HAL_MAX_DELAY);
}
```

I²C

Date de création	Version actuelle	Nombre de lignes	Codage classique	Vitesse
1982	I ² C standard / Fast-mode	2 : SDA & SCL	Sur 7 bits = 128 périphériques	100 kbit/s à 3,4 Mbit/s

```
STM32G431 >> MSC 2025 - Capteurs  
  
*** Début du scan I2C ***  
Adresse détectée : 104 = MPU 9250  
Adresse détectée : 119 = BMP 280  
Nombre total d'adresses détectées : 2
```

Bonnes adresses mais sur 7 bits au lieu de 8

→ Fréquent dans les datasheets

Types de bus

Série	Synchrone	Bidirectionnel	Half-duplex
Données transmises bit par bit sur une seule ligne	Transmission cadencée par un signal d'horloge	Possibilité de transmission et de réception de données	Transmissions dans un seul sens à la fois

Vérification

```
/*
 * @brief Lecture du registre WHO_AM_I du MPU-9250 pour vérifier sa présence.
 *
 * Lit le registre 0x75 du capteur (adresse 0x68) via le bus I2C.
 * Affiche la valeur lue sur le terminal et allume la LED rouge en cas d'erreur.
 *
 * @param none
 * @retval none
 */
void checkMPU9250_ID(void) {
    uint8_t who_am_i = 0;

    // Lecture du registre WHO_AM_I (adresse 0x75)
    if (HAL_I2C_Mem_Read(&hi2c1, 0x68 << 1, 0x75, 1, &who_am_i, 1, HAL_MAX_DELAY) != HAL_OK) {
        Error_Handler();
    }

    sprintf((char*)message, sizeof(message), "WHO_AM_I = 0x%02X\r\n", who_am_i);
    HAL_UART_Transmit(&hlpuart1, message, strlen((char*)message), HAL_MAX_DELAY);

    // Vérification de l'identifiant
    if (who_am_i != 0x71) {
        HAL_UART_Transmit(&hlpuart1, (uint8_t*)"Erreur MPU-9250\r\n", 18, HAL_MAX_DELAY);
        Error_Handler();
    } else {
        HAL_UART_Transmit(&hlpuart1, (uint8_t*)"MPU-9250 détecté avec succès.\r\n\r\n", 32, HAL_MAX_DELAY);
    }
}
```

Addr (Hex)	Addr (Dec.)	Register Name	Serial I/F	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
75	117	WHO_AM_I	R					D[7:0]			
77	119	XA_OFFSET_H	R/W					WHOAM[7:0]			
								XA_OFFSETS [14:7]			

This register is used to verify the identity of the device. The contents of WHO_AM_I is an 8-bit device ID. The default value of the register is 0x71.

WHO_AM_I = 0x71
MPU-9250 détecté avec succès.
MPU-9250 initialise.

II - Acquisition des données

1 - Initialisation

```
/*
 * @brief Initialise le MPU-9250 (reset + sélection horloge PLL).
 *
 * 1. Reset complet du capteur
 * 2. Attente 100 ms
 * 3. Sélection de l'horloge PLL (meilleure stabilité)
 */
void InitSensors(void) {
    uint8_t data;

    // Étape 1 : Reset matériel
    data = 0x80; // bit7 = H_RESET
    if (HAL_I2C_Mem_Write(&hi2c1, 0x6B << 1, 0x6B, 1, &data, 1, HAL_MAX_DELAY) != HAL_OK)
        Error_Handler();

    HAL_Delay(100); // 100 ms d'attente

    // Étape 2 : Sélection horloge PLL (CLKSEL = 1)
    data = 0x01;
    if (HAL_I2C_Mem_Write(&hi2c1, 0x6B << 1, 0x6B, 1, &data, 1, HAL_MAX_DELAY) != HAL_OK)
        Error_Handler();

    HAL_UART_Transmit(&hlpuart1, (uint8_t*)"\\r\\nMPU-9250 initialise.\\r\\n", 23, HAL_MAX_DELAY);
}
```

Addr (Hex)	Addr (Dec.)	Register Name	Serial I/F	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
6B	107	PWR_MGMT_1	R/W	H_RESET	SLEEP	CYCLE	GYRO_STANDBY	PD_PTAT	CLKSEL[2:0]		

BIT	NAME	FUNCTION
[7]	H_RESET	1 – Reset the internal registers and restores the default settings. Write a 1 to set the reset, the bit will auto clear.

Code	Clock Source
0	Internal 20MHz oscillator
1	Auto selects the best available clock source – PLL if ready, else use the Internal oscillator
2	Auto selects the best available clock source – PLL if ready, else use the Internal oscillator
3	Auto selects the best available clock source – PLL if ready, else use the Internal oscillator
4	Auto selects the best available clock source – PLL if ready, else use the Internal oscillator
5	Auto selects the best available clock source – PLL if ready, else use the Internal oscillator
6	Internal 20MHz oscillator
7	Stops the clock and keeps timing generator in reset

(After OTP loads, the inverse of PU_SLEEP_MODE bit will be written to CLKSEL[0])

2 - Température

Registres	Conversion	RoomTempOffset	TempSensitivity
TEMP_OUT_H & TEMP_OUT_L	$T(^{\circ}C) = \frac{TEMPOUT - ROOMTEMP OFFSET}{Temp Sensitivity} + 21$	Décalage pour la température ambiante	Facteur de conversion (333.87 LSB/^{\circ}C)
	Unité	0 LSB : Décalage pour température ambiante (21^{\circ}C) (RoomTempOffset = 17 d'après la datasheet)	333.87 LSB/^{\circ}C : Nombre de LSB par degr{\`e}s Celsius

Addr (Hex)	Addr (Dec.)	Register Name	Serial I/F	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
41	65	TEMP_OUT_H	R								TEMP_OUT_H[15:8]
42	66	TEMP_OUT_L	R								TEMP_OUT_L[7:0]

Name: TEMP_OUT_H
Serial IF: SyncR
Reset value: 0x00 (if sensor disabled)

BIT	NAME	FUNCTION
[7:0]	D[7:0]	High byte of the temperature sensor output

Name: TEMP_OUT_L
Serial IF: SyncR
Reset value: 0x00 (if sensor disabled)

BIT	NAME	FUNCTION
[7:0]	D[7:0]	<p>Low byte of the temperature sensor output:</p> $TEMP_{degC} = ((TEMP_OUT - RoomTemp_Offset)/Temp_Sensitivity) + 21degC$ <p>Where Temp_degC is the temperature in degrees C measured by the temperature sensor. TEMP_OUT is the actual output of the temperature sensor.</p>

2 - Température

```
/**  
 * @brief Mesure la température et écrit le résultat dans un buffer  
 *  
 * @param buffer pointeur vers char[] pour recevoir le message  
 * @param size taille du buffer  
 * @return double température en °C  
 */  
double TempMeasureMsg(char *buffer, size_t size) {  
    uint8_t data[2];  
    int16_t raw;  
    double temperature;  
  
    if(HAL_I2C_Mem_Read(&hi2c1, 0x68 << 1, 0x41, 1, data, 2, HAL_MAX_DELAY) != HAL_OK)  
        Error_Handler();  
  
    raw = (int16_t)((data[0]<<8)|data[1]);      On récupère la valeur brute  
    temperature = ((double)raw)/333.87 + 21.0;  On la converti  
  
    int temp_int = (int)temperature;                Entier  
    int temp_frac = (int)((temperature - temp_int)*100); Décimales  
    if(temp_frac<0) temp_frac=-temp_frac;  
  
    sprintf(buffer, size, "\r\nTemperature = %d.%02d °C\r\n", temp_int, temp_frac);  
    return temperature;  
}
```

→ La partie décimale d'un nombre ne peut pas être négative

Temperature = 32.78 °C
Temperature = 33.02 °C
Temperature = 32.88 °C



Ces mesures ont été prise avec un **doigt** sur le capteur pour observer l'**augmentation de température**.

2 - Température

```
int temp_int = (int)temperature;
int temp_frac = (int)((temperature - temp_int)*100);
if(temp_frac<0) temp_frac=-temp_frac;
temp_int -= 5; // Mesure expérimentale
```

Temperature = 23.58 °C

Temperature = 23.57 °C

Temperature = 23.57 °C

Temperature = 23.57 °C

Temperature = 23.56 °C

On trouve la température mesurée plus tôt un peu **élevé** par rapport à la température réel de la salle, même avec un doigt sur le capteur (plus de 30°C alors qu'il fait environ 22-23°C dans la salle). Nous allons donc mettre un place un **offset expérimental** de -5°C.

3 - Accélération

Registres	Codage	Plage / Sensibilité	Conversion en g
ACCEL_XOUT_H (0x3C) ACCEL_YOUT_H (0x3E) ACCEL_ZOUT_H (0x40)	16 bits en LSB	± 2g / 16384 LSB/g	$a_x(g) = \frac{ACCEL\ XOUT}{ACCEL\ SENSITIVITY}$

Addr (Hex)	Addr (Dec.)	Register Name	Serial I/F	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
3B	59	ACCEL_XOUT_H	R								ACCEL_XOUT_H[15:8]
3C	60	ACCEL_XOUT_L	R								ACCEL_XOUT_L[7:0]
3D	61	ACCEL_YOUT_H	R								ACCEL_YOUT_H[15:8]
3E	62	ACCEL_YOUT_L	R								ACCEL_YOUT_L[7:0]
3F	63	ACCEL_ZOUT_H	R								ACCEL_ZOUT_H[15:8]
40	64	ACCEL_ZOUT_L	R								ACCEL_ZOUT_L[7:0]

Sensitivity Scale Factor	AFS_SEL=0		16,384	LSB/g
	AFS_SEL=1		8,192	LSB/g
	AFS_SEL=2		4,096	LSB/g
	AFS_SEL=3		2,048	LSB/g

3 - Accélération

```
/*
 * @brief Lit la sensibilité réelle de l'accéléromètre MPU-9250
 *
 * Lit le registre ACCEL_CONFIG et retourne le facteur de conversion
 * en LSB/g pour la plage configurée.
 *
 * @return double Sensibilité (LSB/g)
 */
double ReadAccelSensitivity(void) {
    uint8_t reg;
    double factor;

    // Lecture du registre ACCEL_CONFIG
    if (HAL_I2C_Mem_Read(&hi2c1, 0x68 << 1, 0x1C, 1, &reg, 1, HAL_MAX_DELAY) != HAL_OK)
        Error_Handler();

    // Bits [4:3] : FS_SEL
    reg = (reg >> 3) & 0x03;

    switch(reg) {
        case 0: factor = 16384.0; break; // ±2g
        case 1: factor = 8192.0; break; // ±4g
        case 2: factor = 4096.0; break; // ±8g
        case 3: factor = 2048.0; break; // ±16g
        default: factor = 16384.0; break;
    }

    return factor;
}
```

Accel range: ±2g



Sensibilité : 16384 LSB/g

Si l'on dirige l'accéléromètre vers le haut, on mesurera **+1g** (accélération due à la gravité) et **-1g** vers le bas

3 - Accélération

```
/*
 * @brief Mesure l'accélération selon les 3 axes du MPU-9250
 *
 * Lit les registres ACCEL_XOUT_H/L (0x3B/0x3C), ACCEL_YOUT_H/L (0x3D/0x3E),
 * ACCEL_ZOUT_H/L (0x3F/0x40) via I2C et convertit les valeurs brutes en g.
 *
 * @details
 * Les valeurs brutes sont codées en complément à 2 (int16_t).
 * Pour l'étendue par défaut ±2g, la conversion est :
 * ax = ACCEL_X_RAW / 16384.0
 * ay = ACCEL_Y_RAW / 16384.0
 * az = ACCEL_Z_RAW / 16384.0
 *
 * La fonction affiche également les valeurs mesurées sur le terminal UART.
 *
 * @param acc_data Pointeur vers un tableau de 3 doubles
 *                  pour stocker [ax, ay, az] en g.
 */
void AccMeasureMsg(double *acc_data, char *buffer, size_t size) {
    uint8_t raw[6];
    int16_t ax_raw, ay_raw, az_raw;
    double sensitivity = ReadAccelSensitivity();

    if(HAL_I2C_Mem_Read(&hi2c1, 0x68<<1, 0x3B, 1, raw, 6, HAL_MAX_DELAY) != HAL_OK)
        Error_Handler();

    ax_raw = (int16_t)((raw[0]<<8)|raw[1]);
    ay_raw = (int16_t)((raw[2]<<8)|raw[3]);
    az_raw = (int16_t)((raw[4]<<8)|raw[5]);

    acc_data[0] = (double)ax_raw / sensitivity;
    acc_data[1] = (double)ay_raw / sensitivity;
    acc_data[2] = (double)az_raw / sensitivity;

    double norm = sqrt(acc_data[0]*acc_data[0] + acc_data[1]*acc_data[1] + acc_data[2]*acc_data[2]);
    sprintf(buffer, size, "Accel - X: %.3f g, Y: %.3f g, Z: %.3f g\r\n||G|| = %.3f g\r\n",
            acc_data[0], acc_data[1], acc_data[2], norm);
}
```

```
Temperature = 31.01 °C
Accel - X: 0.030 g, Y: 0.121 g, Z: 1.038 g
||G|| = 1.046 g
```

```
Temperature = 30.72 °C
Accel - X: 0.027 g, Y: 0.117 g, Z: 1.035 g
||G|| = 1.042 g
```

4 - Gyroscope

Registres	Codage	Plage / Sensibilité	Conversion en g
GYRO_XOUT_H (0x3C) GYRO_YOUT_H (0x3E) GYRO_ZOUT_H (0x40)	16 bits en LSB	± 250dps / 131 LSB/dps	$v_x (\text{°. s}^{-1}) = \frac{\text{GYRO XOUT}}{\text{GYRO SENSITIVITY}}$

Addr (Hex)	Addr (Dec.)	Register Name	Serial I/F	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
43	67	GYRO_XOUT_H	R								GYRO_XOUT_H[15:8]
44	68	GYRO_XOUT_L	R								GYRO_XOUT_L[7:0]
45	69	GYRO_YOUT_H	R								GYRO_YOUT_H[15:8]
46	70	GYRO_YOUT_L	R								GYRO_YOUT_L[7:0]
47	71	GYRO_ZOUT_H	R								GYRO_ZOUT_H[15:8]
48	72	GYRO_ZOUT_L	R								GYRO_ZOUT_L[7:0]

[4:3]	GYRO_FS_SEL[1:0]	Gyro Full Scale Select: 00 = +250dps 01 = +500 dps 10 = +1000 dps 11 = +2000 dps
-------	------------------	--

4 - Gyroscope

```
/**  
 * @brief Lit la sensibilité réelle du gyroscope MPU-9250  
 *  
 * @return double Sensibilité (LSB/(°/s)) selon la plage configurée  
 */  
double ReadGyroSensitivity(void) {  
    uint8_t reg;  
    double factor;  
  
    if(HAL_I2C_Mem_Read(&hi2c1, 0x68 << 1, 0x1B, 1, &reg, 1, HAL_MAX_DELAY) != HAL_OK)  
        Error_Handler();  
  
    reg = (reg >> 3) & 0x03; // Bits FS_SEL  
    switch(reg) {  
        case 0: factor = 131.0; break;      // ±250 °/s  
        case 1: factor = 65.5; break;      // ±500 °/s  
        case 2: factor = 32.8; break;      // ±1000 °/s  
        case 3: factor = 16.4; break;      // ±2000 °/s  
        default: factor = 131.0; break;  
    }  
    return factor;  
}
```

Gyro range: ±250 °/s

Sensibilité : 131 LSB/dps

Peu importe comment l'on dirige le gyroscope, s'il ne **bouge pas**, on lira des valeurs proches de **zéro**.

4 - Gyroscope

```
/*
 * @brief Mesure la vitesse angulaire sur les 3 axes du MPU-9250
 *
 * Lit les registres GYRO_XOUT_H/L, GYRO_YOUT_H/L, GYRO_ZOUT_H/L
 * et convertit les valeurs brutes en °/s.
 *
 * @param gyro_data Pointeur vers un tableau de 3 doubles [gx, gy, gz] en °/s
 */
void GyroMeasureMsg(double *gyro_data, char *buffer, size_t size) {
    uint8_t raw[6];
    int16_t gx_raw, gy_raw, gz_raw;
    double sensitivity = ReadGyroSensitivity();

    if(HAL_I2C_Mem_Read(&hi2c1, 0x68<<1, 0x43, 1, raw, 6, HAL_MAX_DELAY) != HAL_OK)
        Error_Handler();

    gx_raw = (int16_t)((raw[0]<<8)|raw[1]);
    gy_raw = (int16_t)((raw[2]<<8)|raw[3]);
    gz_raw = (int16_t)((raw[4]<<8)|raw[5]);

    gyro_data[0] = (double)gx_raw / sensitivity;
    gyro_data[1] = (double)gy_raw / sensitivity;
    gyro_data[2] = (double)gz_raw / sensitivity;

    sprintf(buffer, size, "Gyro - X: %.2f °/s, Y: %.2f °/s, Z: %.2f °/s\r\n",
            gyro_data[0], gyro_data[1], gyro_data[2]);
}
```

```
Temperature = 30.77 °C
Gyro - X: -0.73 °/s, Y: 0.44 °/s, Z: -0.31 °/s
Accel - X: 0.027 g, Y: 0.121 g, Z: 1.040 g
||G|| = 1.047 g
```

```
Temperature = 30.92 °C
Gyro - X: -1.02 °/s, Y: 0.57 °/s, Z: -0.06 °/s
Accel - X: 0.025 g, Y: 0.121 g, Z: 1.044 g
||G|| = 1.051 g
```

4 - Magnétomètre

Composantes Champ B Paris	Norme Champ B Paris	Solution Bus	Adresses
X (Nord) \approx 22 μ T Y (Est) \approx 1.1 μ T Z (Vertical) \approx 42 μ T	\approx 47.6 μ T par le calcul \approx 50 μ T environ pour le champ B terrestre	BYPASS_EN	Lecture / Écriture : 0x0C (7b) Écriture : 0x18 (8b) Lecture : 0x19 (8b)

[1]	BYPASS_EN	When asserted, the i2c_master interface pins(ES_CL and ES_DA) will go into 'bypass mode' when the i2c master interface is disabled. The pins will float high due to the internal pull-up if not enabled and the i2c master interface is disabled.
-----	-----------	---

On ne peut pas accéder directement au magnétomètre sur le bus I2C car il est câblé sur le bus I²C auxiliaire interne du MPU-9250.

Pour faire apparaître le magnétomètre AK8963 sur le bus I2C, nous allons activer le Bypass.

Pass-Through mode is also used to access the AK8963 magnetometer directly from the host. In this configuration the slave address for the AK8963 is 0X0C or 12 decimal.

4 - Magnétomètre

```
/**  
 * @brief Lit la sensibilité nominale du magnétomètre AK8963 selon le mode de sortie.  
 *  
 * @param bit16_mode Si 1, mode 16 bits (0.15 µT/LSB). Si 0, mode 14 bits (0.6 µT/LSB)  
 * @return double Sensibilité du magnétomètre en µT/LSB  
 */  
  
double ReadMagSensitivity(uint8_t bit16_mode) {  
    double factor;  
  
    if(bit16_mode) {  
        factor = 0.15; // µT/LSB en 16 bits  
    } else {  
        factor = 0.6; // µT/LSB en 14 bits  
    }  
  
    return factor;  
}
```

- 14/16-bit selectable data out for each 3 axis magnetic components
 - Sensitivity: 0.6 µT/LSB typ. (14-bit)
0.15µT/LSB typ. (16-bit)

AK8963 Sensitivity Adj: X=1.188, Y=1.191, Z=1.145, sens=0.150 µT/LSB

```
/// == Activation du BYPASS (connexion du magnétomètre AK8963 sur le bus I2C principal) ==  
data = 0x02; // Bit1 = BYPASS_EN  
HAL_I2C_Mem_Write(&hi2c1, 0x68 << 1, 0x37, 1, &data, 1, HAL_MAX_DELAY);  
  
HAL_Delay(10);  
  
// == Vérification du WHO_AM_I du magnétomètre ==  
uint8_t who_am_i_mag = 0;  
HAL_I2C_Mem_Read(&hi2c1, 0x0C << 1, 0x00, 1, &who_am_i_mag, 1, HAL_MAX_DELAY);  
  
snprintf((char*)message, sizeof(message), "AK8963 WHO_AM_I = 0x%02X\r\n", who_am_i_mag);  
HAL_UART_Transmit(&hlpuart1, message, strlen((char*)message), HAL_MAX_DELAY);  
  
if (who_am_i_mag != 0x48) {  
    HAL_UART_Transmit(&hlpuart1, (uint8_t*)"Erreur magnétomètre AK8963\r\n", 29, HAL_MAX_DELAY);  
    Error_Handler();  
} else {  
    HAL_UART_Transmit(&hlpuart1, (uint8_t*)"Magnétomètre détecté avec succès.", 38, HAL_MAX_DELAY);  
}  
  
// == Configuration du magnétomètre : 16-bit output + mode continu 2 (100Hz) ==  
data = 0x16; // 0b00010110 : 16-bit (bit4=1) + mode2 (bits[3:0]=110)  
HAL_I2C_Mem_Write(&hi2c1, 0x0C << 1, 0x0A, 1, &data, 1, HAL_MAX_DELAY);
```

--- Initialisation du MPU9250 ---
MPU9250 WHO_AM_I = 0x71
MPU-9250 détecté avec succès.
AK8963 WHO_AM_I = 0x48
Magnétomètre détecté avec succès.

4 - Magnétomètre

```
/*
 * @brief Mesure le champ magnétique 3D via le magnétomètre AK8963.
 *
 * Lit les registres HXL à HZH (0x03 à 0x08), applique les coefficients
 * d'ajustement (ASAX, ASAY, ASAZ), puis convertit les valeurs brutes en µT.
 *
 * @param mag_data Pointeur vers un tableau de 3 doubles [Bx, By, Bz] (µT)
 * @param buffer Pointeur vers la chaîne de caractères à remplir
 * @param size Taille du buffer
 */
void MagMeasureMsg(double *mag_data, char *buffer, size_t size) {
    uint8_t raw[7];
    int16_t mx_raw, my_raw, mz_raw;
    static double mag_adj[3] = {0.0, 0.0, 0.0}; // facteurs d'ajustement
    static uint8_t initialized = 0;
    double sens; // sensibilité en µT/LSB

    // 1. Initialisation + Lecture ASAX, ASAY, ASAZ
    if (!initialized) {
        uint8_t asa[3];
        uint8_t mode = 0x0F;
        if (HAL_I2C_Mem_Write(&hi2c1, 0x0C << 1, 0x0A, 1, &mode, 1, HAL_MAX_DELAY) != HAL_OK)
            Error_Handler();
        HAL_Delay(10);

        if (HAL_I2C_Mem_Read(&hi2c1, 0x0C << 1, 0x10, 1, asa, 3, HAL_MAX_DELAY) != HAL_OK)
            Error_Handler();

        // 2. Passer en Continuous Measurement Mode 2 (16 bits, 100Hz)
        mode = 0x16;
        if (HAL_I2C_Mem_Write(&hi2c1, 0x0C << 1, 0x0A, 1, &mode, 1, HAL_MAX_DELAY) != HAL_OK)
            Error_Handler();
        HAL_Delay(10);
    }
}
```

```
// 3. Calcul des coefficients d'ajustement
for (int i = 0; i < 3; i++) {
    mag_adj[i] = ((double)(asa[i] - 128) / 256.0) + 1.0;
}

sens = ReadMagSensitivity(1); // 16 bits = 0.15 µT/LSB
initialized = 1;

char msg[100];
snprintf(msg, sizeof(msg),
    "AK8963 Sensitivity Adj: X=%3f, Y=%3f, Z=%3f, sens=%3f µT/LSB\r\n",
    mag_adj[0], mag_adj[1], mag_adj[2], sens);
HAL_UART_Transmit(&hluart1, (uint8_t*)msg, strlen(msg), HAL_MAX_DELAY);
} else {
    sens = ReadMagSensitivity(1);
}

// 4. Attendre de nouvelles données : ST1.DRDY
uint8_t st1;
do {
    if(HAL_I2C_Mem_Read(&hi2c1, 0x0C << 1, 0x02, 1, &st1, 1, HAL_MAX_DELAY) != HAL_OK)
        Error_Handler();
} while(!(st1 & 0x01)); // DRDY = 1

// 5. Lecture des registres de data
if(HAL_I2C_Mem_Read(&hi2c1, 0x0C << 1, 0x03, 1, raw, 7, HAL_MAX_DELAY) != HAL_OK)
    Error_Handler();

// 6. Vérifier l'overflow
uint8_t st2;
if(HAL_I2C_Mem_Read(&hi2c1, 0x0C << 1, 0x09, 1, &st2, 1, HAL_MAX_DELAY) != HAL_OK)
    Error_Handler();
if(st2 & 0x08) return; // données saturées : ignorer lecture
```

```
// 7. Assemblage des données brutes
mx_raw = (int16_t)((raw[1] << 8) | raw[0]);
my_raw = (int16_t)((raw[3] << 8) | raw[2]);
mz_raw = (int16_t)((raw[5] << 8) | raw[4]);

// 8. Conversion en µT : sensibilité + coefficients d'ajustement
mag_data[0] = (double)mx_raw * sens * mag_adj[0];
mag_data[1] = (double)my_raw * sens * mag_adj[1];
mag_data[2] = (double)mz_raw * sens * mag_adj[2];

// 9. Calcul de la norme du vecteur B
double norm = sqrt(mag_data[0]*mag_data[0] + mag_data[1]*mag_data[1] + mag_data[2]*mag_data[2]);

// 10. Écriture du message
snprintf(buffer, size,
    "Mag X: %.2f µT, Y: %.2f µT, Z: %.2f µT\r\n||B|| = %.2f µT\r\n",
    mag_data[0], mag_data[1], mag_data[2], norm);
```

Addr	Register name	D7	D6	D5	D4	D3	D2	D1	D0
Read-only register									
10H	ASAX	COEFX7	COEFX6	COEFXS	COEFX4	COEFX3	COEFX2	COEFX1	COEFX0
11H	ASAY	COEFY7	COEFY6	COEFSY5	COEFY4	COEFY3	COEFY2	COEFY1	COEFY0
12H	ASAZ	COEFZ7	COEFZ6	COEFZ5	COEFZ4	COEFZ3	COEFZ2	COEFZ1	COEFZ0
Reset									

Temperature = 26.79 °C
Gyro - X: -1.59 °/s, Y: -0.55 °/s, Z: 1.63 °/s
Accel - X: 0.910 g, Y: 0.228 g, Z: 0.461 g
||G|| = 1.045 g
Mag - X: 39.37 µT, Y: 9.83 µT, Z: -20.43 µT
||B|| = 45.43 µT

Addr	Register name	D7	D6	D5	D4	D3	D2	D1	D0
Read-only register									
03H	HXL	HX7	HX6	HX5	HX4	HX3	HX2	HX1	HX0
04H	HXH	HX15	HX14	HX13	HX12	HX11	HX10	HX9	HX8
05H	HYL	HY7	HY6	HY5	HY4	HY3	HY2	HY1	HY0
06H	HYH	HY15	HY14	HY13	HY12	HY11	HY10	HY9	HY8
07H	HZL	HZ7	HZ6	HZ5	HZ4	HZ3	HZ2	HZ1	HZ0
08H	HZH	HZ15	HZ14	HZ13	HZ12	HZ11	HZ10	HZ9	HZ8
Reset									

Temperature = 26.89 °C
Gyro - X: 17.15 °/s, Y: 5.74 °/s, Z: 4.98 °/s
Accel - X: 0.907 g, Y: 0.230 g, Z: 0.460 g
||G|| = 1.043 g
Mag - X: 40.43 µT, Y: 11.26 µT, Z: -20.77 µT
||B|| = 46.83 µT



III - Modification des paramètres métrologiques

1 - Sensibilité

La **sensibilité** est le rapport (ou la dérivée) entre la variation de la sortie et la variation de la grandeur mesurée, autour d'un point de fonctionnement.

$$S = \frac{\Delta y}{\Delta x} \quad \text{ou} \quad S(x) = \frac{dy}{dx}$$

Unité : unité de la sortie sur unité de l'entrée

Si le capteur suit une loi linéaire, la sensibilité est la pente.

Sensibilité statique : pente de la caractéristique en régime permanent

On la mesure en appliquant des entrées quasi constantes (paliers), on attend l'état stationnaire, puis on calcule $\Delta y/\Delta x$.

Pour notre accéléromètre, la sensibilité statique est de 16384 LSB/g, cad que passer de 0 à 1 g augmente la lecture d'environ 16384 counts.

Sensibilité dynamique : elle dépend de la fréquence des signaux variables dans le temps, c'est le gain du système entre l'entrée et la sortie.

on la mesure en excitant le capteur avec des sinusoïdes et en traçant le Bode.

1 - Sensibilité

Les réglages possibles pour la sensibilité de l'accélération vont de **16384 LSB/g** à **2048 LSB/g** comme montré ci-dessous :

Sensitivity Scale Factor	AFS_SEL=0	16,384	LSB/g
	AFS_SEL=1	8,192	LSB/g
	AFS_SEL=2	4,096	LSB/g
	AFS_SEL=3	2,048	LSB/g

Il faut donc modifier ce registre **AFS_SEL** avec la valeur correspondante à la sensibilité voulue.

1 - Sensibilité

```
/** @brief Teste l'accéléromètre sur toutes les plages de sensibilité et affiche Z
*
* Affiche : FS_SEL, plage (g), sensibilité (LSB/g), Z_raw et Z_g
*/
void TestAccelSensZ(void) {
    uint8_t reg;
    int16_t z_raw;
    double z_g;
    double sensitivity;
    double z_range;
    char msg[150];

    HAL_UART_Transmit(&hlpuart1, (uint8_t*)"\\r\\n", 2, HAL_MAX_DELAY);
    HAL_UART_Transmit(&hlpuart1, (uint8_t*)"\\r\\n*** Test de l'accéléromètre sur toutes les plages FS_SEL (Z) "
        "***\\r\\n", 72, HAL_MAX_DELAY);

    for (uint8_t fs_sel = 0; fs_sel <= 3; fs_sel++) {
        // Lire le registre ACCEL_CONFIG
        if (HAL_I2C_Mem_Read(&hi2c1, 0x68<<1, 0x1C, 1, &reg, 1, HAL_MAX_DELAY) != HAL_OK)
            Error_Handler();

        // Modification FS_SEL (bits 4:3)
        reg &= ~0x18;           // Clear bits 4:3
        reg |= (fs_sel << 3); // Set FS_SEL
        if (HAL_I2C_Mem_Write(&hi2c1, 0x68<<1, 0x1C, 1, &reg, 1, HAL_MAX_DELAY) != HAL_OK)
            Error_Handler();

        HAL_Delay(100); // Stabilisation

        // Lire la valeur brute de l'axe Z
        uint8_t data[2];
        if (HAL_I2C_Mem_Read(&hi2c1, 0x68<<1, 0x3F, 1, data, 2, HAL_MAX_DELAY) != HAL_OK)
            Error_Handler();
        z_raw = (int16_t)((data[0]<<8)|data[1]);
    }
}
```

```
switch(fs_sel) {
    case 0: sensitivity = 16384.0; z_range = 2.0; break; // ±2g
    case 1: sensitivity = 8192.0; z_range = 4.0; break; // ±4g
    case 2: sensitivity = 4096.0; z_range = 8.0; break; // ±8g
    case 3: sensitivity = 2048.0; z_range = 16.0; break; // ±16g
    default: sensitivity = 16384.0; z_range = 2.0; break;
}

z_g = (double)z_raw / sensitivity;

// Affichage UART
snprintf(msg, sizeof(msg),
    "FS_SEL=%d | Plage=±%.1f g | Sens=%g LSB/g | Z_raw=%d | Z_g=%f g\\r\\n",
    fs_sel, z_range, sensitivity, z_raw, z_g);
HAL_UART_Transmit(&hlpuart1, (uint8_t*)msg, strlen(msg), HAL_MAX_DELAY);

// Ligne vide après
HAL_UART_Transmit(&hlpuart1, (uint8_t*)"\\r\\n", 2, HAL_MAX_DELAY);
}
```

*** Test de l'accéléromètre sur toutes les plages FS_SEL (Z) ***

FS_SEL=0	Plage=±2.0 g	Sens=16384 LSB/g	Z_raw=-16444	Z_g=-1.004 g
FS_SEL=1	Plage=±4.0 g	Sens=8192 LSB/g	Z_raw=-8182	Z_g=-0.999 g
FS_SEL=2	Plage=±8.0 g	Sens=4096 LSB/g	Z_raw=-4074	Z_g=-0.995 g
FS_SEL=3	Plage=±16.0 g	Sens=2048 LSB/g	Z_raw=-2050	Z_g=-1.001 g

Varie selon la sensibilité

Z_g = Sens * Z_raw

≈-1g (0.5% de variation)
> Bruit, Imperfection, ...

2 - Bande-passante

Lien entre bande passante et la valeur efficace du bruit :

Une large bande passante implique la présence d'un bruit important dans les données mesurées.

Les réglages possibles pour la bande passante du gyroscope vont de **8800Hz à 5Hz**.

Les registres à modifier sont **FCHOICE** et **DLPF_CFG**. Avec FCHOICE, on peut choisir entre **8800Hz** et **3600Hz** et on choisit des bandes passantes entre **250** et **5Hz** avec DLPF_CFG (FCHOICE à 0b11)

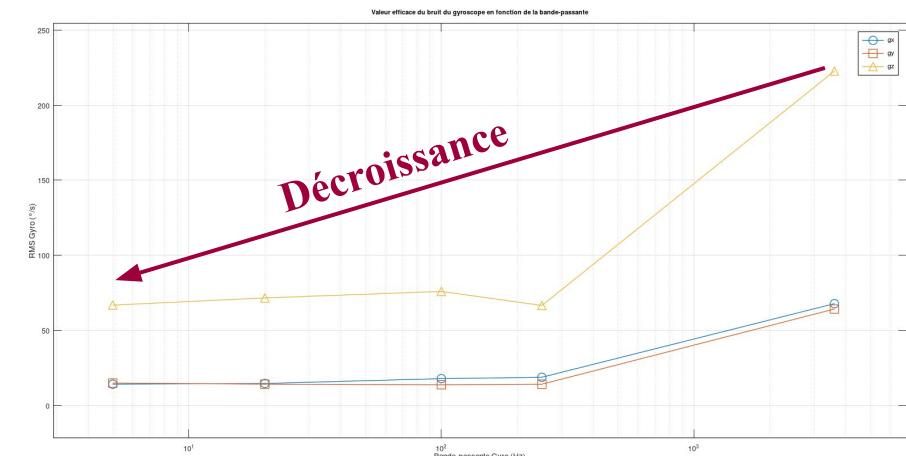
FCHOICE		DLPF_CFG	Gyroscope			Temperature Sensor	
<1>	<0>		Bandwidth (Hz)	Delay (ms)	Fs (kHz)	Bandwidth (Hz)	Delay (ms)
x	0	x	8800	0.064	32	4000	0.04
0	1	x	3600	0.11	32	4000	0.04
1	1	0	250	0.97	8	4000	0.04
1	1	1	184	2.9	1	188	1.9
1	1	2	92	3.9	1	98	2.8
1	1	3	41	5.9	1	42	4.8
1	1	4	20	9.9	1	20	8.3
1	1	5	10	17.85	1	10	13.4
1	1	6	5	33.48	1	5	18.6
1	1	7	3600	0.17	8	4000	0.04

2 - Bande-passante

```
BW = [3600, 250, 100, 20, 5]; % bande-passante
gx_rms = [67.742, 18.691, 17.782, 14.545, 14.087];
gy_rms = [64.168, 14.063, 13.692, 14.089, 14.843];
gz_rms = [222.745, 66.523, 75.906, 71.554, 66.774];

figure;
plot(BW, gx_rms, 'log', 'DisplayName','gx');
hold on;
plot(BW, gy_rms, 'log', 'DisplayName','gy');
plot(BW, gz_rms, 'log', 'DisplayName','gz');
set(gca, 'XScale','log'); % optionnel : échelle log pour mieux visualiser
xlabel('Bande passante Gyro (Hz)');
ylabel('RMS Gyro (°/s)');
title('Valeur efficace du bruit du gyroscope en fonction de la bande-passante');
legend show;
grid on;
```

```
*** Test du bruit du gyroscope sur différentes bandes passantes ***
Configuration bande passante Gyro: 3600 Hz
BW=3600 Hz | RMS Gyro: gx=67.742 °/s, gy=64.168 °/s, gz=222.745 °/s
Configuration bande passante Gyro: 250 Hz
BW=250 Hz | RMS Gyro: gx=18.691 °/s, gy=14.063 °/s, gz=66.523 °/s
Configuration bande passante Gyro: 100 Hz
BW=100 Hz | RMS Gyro: gx=17.682 °/s, gy=13.692 °/s, gz=75.906 °/s
Configuration bande passante Gyro: 20 Hz
BW=20 Hz | RMS Gyro: gx=14.545 °/s, gy=14.089 °/s, gz=71.554 °/s
Configuration bande passante Gyro: 5 Hz
BW=5 Hz | RMS Gyro: gx=14.087 °/s, gy=14.843 °/s, gz=66.774 °/s
```



Plus la bande passante **diminue**, moins il y a de bruit (*moins de puissance, qui est le carré de la RMSE*), donc plus sa valeur efficace est **faible**, c'est bien ce que nous voyons ici (*malgré une augmentation en passant de 250Hz à 100Hz selon l'axe z*)

Output Data Rate (ODR) est la fréquence à laquelle le capteur produit de nouveaux échantillons.

3 - Fréquence d'échantillonnage

Output Data Rate	Low power (duty-cycled)	0.24		500	Hz
	Duty-cycled, over temp		± 15		%
	Low noise (active)	4		4000	Hz

Table 2 Accelerometer Specifications

En mode **Low noise**



Le couple de registre **ACCEL_FCHOICE_B** et **A_DLDPF_CFG** règle le filtre DLDPF de l'accéléromètre et donc la bande passante. Nous avons 2 choix de configuration :

- **ACCEL_FCHOICE_B = 0** (DLDPF bypass) → **ODR interne = 4 kHz** et **BW = 1.13 kHz**.
- **ACCEL_FCHOICE_B = 1** et **A_DLDPF_CFG = [0,7]** (DLDPF actif) → **ODR interne = 1 kHz**, et la bande passante peut prendre des valeurs définies de 460 à 5Hz.

Accelerometer Data Rates and Bandwidths (Normal Mode)

ACCEL_FCHOICE	A_DLDPF_CFG	Output			
		Bandwidth (Hz)	Delay (ms)	Noise Density (ug/rthz)	Rate (kHz)
0	X	1.13 K	0.75	250	4
1	0	460	1.94	250	1
1	1	184	5.80	250	1
1	2	92	7.80	250	1
1	3	41	11.80	250	1
1	4	20	19.80	250	1
1	5	10	35.70	250	1
1	6	5	66.96	250	1
1	7	460	1.94	250	1

3 - Fréquence d'échantillonnage

En mode **Low power**



Un autre choix de configuration est possible, en mode **Low power**, l'accéléromètre peut seul échantillonner à une ODR programmable entre 0,24 à 500 Hz via le registre **LP_ACCEL_ODR** (0x1E).

Accelerometer Data Rates and Bandwidths (Low-Power Mode)

ACCEL_FCHOICE	ODR (Hz)	Output	
		Bandwidth (Hz)	Delay (ms)
0	0.24	1.1 k	1
0	0.49	1.1 k	1
0	0.98	1.1 k	1
0	1.95	1.1 k	1
0	3.91	1.1 k	1
0	7.81	1.1 k	1
0	15.63	1.1 k	1
0	31.25	1.1 k	1
0	62.50	1.1 k	1
0	125	1.1 k	1
0	250	1.1 k	1
0	500	1.1 kHz	1

3 - Fréquence d'échantillonnage

Choix des facteurs pour régler au mieux l'ODR dépend :

- De la fréquence de ce que l'on veut observer, par exemple geste humain (10-20Hz max), vibration d'une machine (100-1kHz), ou plus lent encore comme de la surveillance en continue ou un système fonctionnant en très low power.
- De la taille de notre bande passante (généralement une ou 1.5 fois la fréquence de mesure). SI le signal est trop bruité, on réduit la bande passante.
- L'ODR doit être plus grand ou égal à $2 f_{\max}$ afin de respecter le critère de Nyquist-Shannon.
- Du besoin d'avoir une réaction rapide ou non.
- De la saturation de notre microprocesseur car ODR haut → plus de données et plus d'énergie.

3 - Fréquence d'échantillonnage

```
// === Configuration DLPE pour ODR faible ===  
data = 0x06; // DLPE_CFG = 6 (accéléro et gyro à 1 kHz interne)  
HAL_I2C_Mem_Write(&hi2c1, 0x68 << 1, 0x1A, 1, &data, 1, HAL_MAX_DELAY);  
  
// === SRD pour 3,91 Hz ===  
data = 255; // SMPLRT_DIV = 255 → f_s = 1000 / (1 + 255) ≈ 3,91 Hz  
HAL_I2C_Mem_Write(&hi2c1, 0x68 << 1, 0x19, 1, &data, 1, HAL_MAX_DELAY);
```

InitSensors()

```
Temperature = 26.90 °C  
Gyro - X: -0.84 °/s, Y: 0.60 °/s, Z: -0.26 °/s  
Accel - X: 0.497 g, Y: -0.867 g, Z: -0.096 g  
||G|| = 1.004 g  
Mag - X: 94.94 µT, Y: 33.78 µT, Z: -69.87 µT  
||B|| = 122.62 µT
```

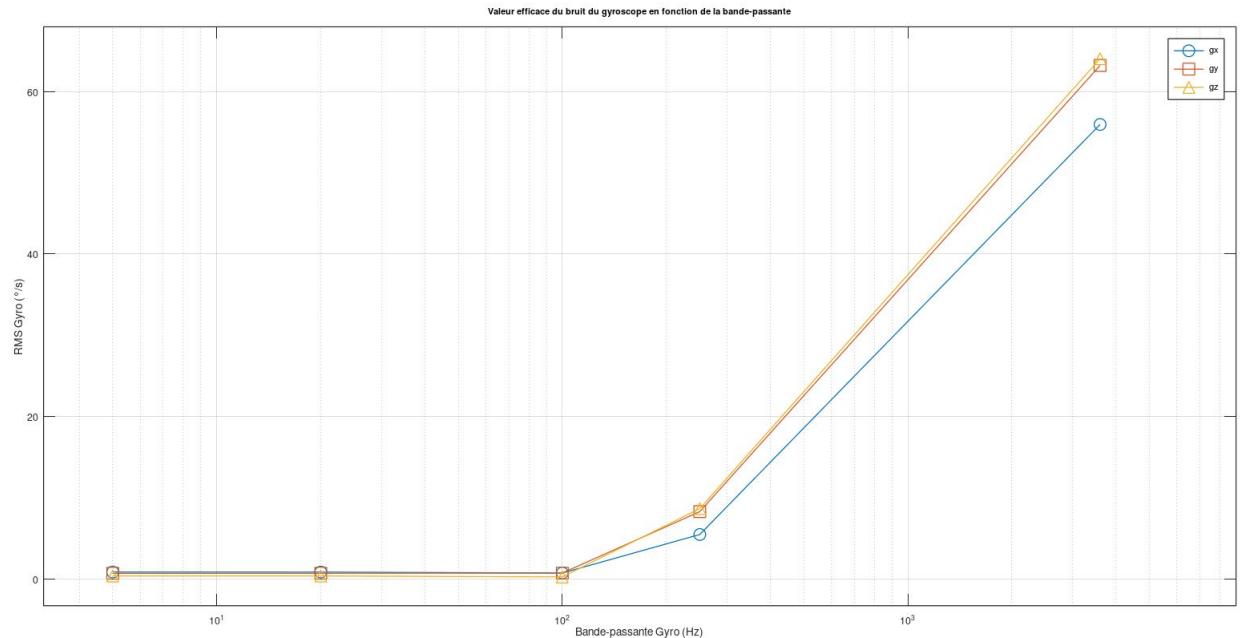
```
Temperature = 26.90 °C  
Gyro - X: -0.82 °/s, Y: 0.59 °/s, Z: -0.24 °/s  
Accel - X: 0.497 g, Y: -0.870 g, Z: -0.097 g  
||G|| = 1.006 g  
Mag - X: 94.41 µT, Y: 33.96 µT, Z: -70.73 µT  
||B|| = 122.75 µT
```

```
Temperature = 26.90 °C  
Gyro - X: -0.79 °/s, Y: 0.58 °/s, Z: -0.22 °/s  
Accel - X: 0.497 g, Y: -0.869 g, Z: -0.100 g  
||G|| = 1.006 g  
Mag - X: 94.05 µT, Y: 35.03 µT, Z: -69.70 µT  
||B|| = 122.19 µT
```

On observe **moins de variations** pour la température et des **variations plus douce** pour l'accélération, le gyroscope ou même encore le champ magnétique que précédemment.

3 - Fréquence d'échantillonnage

```
*** Test du bruit du gyroscope sur différentes bandes passantes ***
Configuration bande passante Gyro: 3600 Hz
BW=3600 Hz | RMS Gyro: gx=55.963 °/s, gy=63.229 °/s, gz=64.064 °/s
Configuration bande passante Gyro: 250 Hz
BW=250 Hz | RMS Gyro: gx=5.452 °/s, gy=8.271 °/s, gz=8.643 °/s
Configuration bande passante Gyro: 100 Hz
BW=100 Hz | RMS Gyro: gx=0.704 °/s, gy=0.701 °/s, gz=0.221 °/s
Configuration bande passante Gyro: 20 Hz
BW=20 Hz | RMS Gyro: gx=0.822 °/s, gy=0.680 °/s, gz=0.338 °/s
Configuration bande passante Gyro: 5 Hz
BW=5 Hz | RMS Gyro: gx=0.828 °/s, gy=0.670 °/s, gz=0.333 °/s
AK8963 Sensitivity Adj: X=1.188, Y=1.191, Z=1.145, sens=0.150 µT/LSB
```



Résultat du RMSE du bruit gyroscope beaucoup **plus pertinent** que celui obtenu précédemment :

- Bruit décroissant avec la bande passante décroissante
- Bruit presque nul à partir d'une certaine bande passante

Fin du livrable 2

IV - Calibration

1 - Procédure single point - gyromètre

- Inconvénient majeur de la procédure “single point / no-turn” :

Cette méthode ne permet pas de corriger la dérive d’angle si le gain n’est pas calibré. Une erreur de sensibilité du capteur peut quand même produire un erreur même après calibration. Par exemple, cette calibration n’empêche pas une mesure de $10,5^\circ/\text{s}$ au lieu de $10^\circ/\text{s}$ réel si le problème vient du gain et nous de l’offset au départ.

- Pour une calibration plus efficace :

On peut faire tourner le capteur sur un objet qui tourne à vitesse angulaire connue et constante pour avoir le gain sur chaque axe.

On peut aussi faire des mesures sous plusieurs températures différentes pour appliquer une correction en température par la suite et ainsi limiter encore un peu plus l’erreur.

- Il faut tout d’abord obtenir les valeurs du gyroscope immobile, présentent dans les registres GYRO_XOUT_H (0x43), GYRO_XOUT_L (0x44), GYRO_YOUT_H (0x45), ..., GYRO_ZOUT_L (0x48).

Ensuite, nous utilisons les fonctions Average et Variance pour calculer les offsets pour chaque coordonnées et enfin nous soustrayons les offsets aux valeurs obtenues pour que les gyroscope nous indique $(0, 0, 0)^\circ/\text{s}$ dans les 3 directions.

Fonctions Mathématiques

```
/*
 * @brief Calcule la moyenne de N mesures gyroscopiques.
 * @param data Tableau contenant les mesures (X, Y, Z)
 * @param N Nombre d'échantillons
 * @param avg Tableau de sortie [3] pour stocker les moyennes
 */
void Average(float data[][3], int N, float avg[3]) {
    avg[0] = avg[1] = avg[2] = 0.0f;
    for (int i = 0; i < N; i++) {
        avg[0] += data[i][0];
        avg[1] += data[i][1];
        avg[2] += data[i][2];
    }
    avg[0] /= N;
    avg[1] /= N;
    avg[2] /= N;
}
```

```
/*
 * @brief Calcule la variance des N mesures gyroscopiques.
 * @param data Tableau [N][3]
 * @param N Nombre d'échantillons
 * @param avg Moyennes calculées pour chaque axe
 * @param var Tableau de sortie [3] pour stocker les variances
 */
void Variance(float data[][3], int N, float avg[3], float var[3]) {
    var[0] = var[1] = var[2] = 0.0f;
    for (int i = 0; i < N; i++) {
        var[0] += powf(data[i][0] - avg[0], 2);
        var[1] += powf(data[i][1] - avg[1], 2);
        var[2] += powf(data[i][2] - avg[2], 2);
    }
    var[0] /= N;
    var[1] /= N;
    var[2] /= N;
}
```

Des fonctions permettant de calculer la **moyenne** et la **variance** nous permettent différents types de calibrations. Ici on ne se limite qu'à trois valeurs puisque nous sommes dans le **plan 3D**.

1 - Procédure single point - gyromètre

```
/**  
 * @brief Lit les offsets matériels du gyroscope dans le MPU-9250 et les affiche.  
 * @note Permet de vérifier si les offsets écrits sont bien proches de 0 après calibration.  
 * @return None  
 */  
  
void ReadOffsetGyro(void){  
    uint8_t data[6];  
    int16_t gyro_offset_raw[3];  
    float gyro_offset_dps[3];  
  
    // Lit les 6 registres d'offset (X, Y, Z)  
    if (HAL_I2C_Mem_Read(&hi2c1, MPU9250_ADDRESS, 0x13, 1, data, 6, HAL_MAX_DELAY) != HAL_OK){  
        HAL_UART_Transmit(&hlpuart1, (uint8_t*)"Erreur lecture offsets gyroscope\r\n", 35, HAL_MAX_DELAY);  
        return;  
    }  
  
    // Conversion en valeurs signées  
    gyro_offset_raw[0] = (int16_t)((data[0] << 8) | data[1]);  
    gyro_offset_raw[1] = (int16_t)((data[2] << 8) | data[3]);  
    gyro_offset_raw[2] = (int16_t)((data[4] << 8) | data[5]);  
  
    // Conversion en °/s (selon sensibilité ±250°/s)  
    gyro_offset_dps[0] = (float)gyro_offset_raw[0] / 131.0f;  
    gyro_offset_dps[1] = (float)gyro_offset_raw[1] / 131.0f;  
    gyro_offset_dps[2] = (float)gyro_offset_raw[2] / 131.0f;  
  
    // Affichage des résultats  
    char msg[128];  
    sprintf(msg, sizeof(msg), "Offsets Gyro (°/s): X=% .3f | Y=% .3f | Z=% .3f\r\n",  
            gyro_offset_dps[0], gyro_offset_dps[1], gyro_offset_dps[2]);  
    HAL_UART_Transmit(&hlpuart1, (uint8_t*)msg, strlen(msg), HAL_MAX_DELAY);  
  
    // Vérification : proche de 0 ?  
    if (fabs(gyro_offset_dps[0]) < 0.5 && fabs(gyro_offset_dps[1]) < 0.5 && fabs(gyro_offset_dps[2]) < 0.5){  
        HAL_UART_Transmit(&hlpuart1, (uint8_t*)"Offsets gyroscope proches de 0\r\n", 36, HAL_MAX_DELAY);  
    } else {  
        HAL_UART_Transmit(&hlpuart1, (uint8_t*)"Offsets gyroscope NON nuls\r\n", 33, HAL_MAX_DELAY);  
    }  
}
```

```
/**  
 * @brief Lit les vitesses angulaires brutes (°/s) sur les 3 axes.  
 *  
 * @param gx Pointeur vers la valeur X  
 * @param gy Pointeur vers la valeur Y  
 * @param gz Pointeur vers la valeur Z  
 */  
  
void ReadGyro(float *gx, float *gy, float *gz) {  
    uint8_t raw[6];  
    int16_t gx_raw, gy_raw, gz_raw;  
    double sensitivity = ReadGyroSensitivity();  
  
    if (HAL_I2C_Mem_Read(&hi2c1, 0x68 << 1, 0x43, 1, raw, 6, HAL_MAX_DELAY) != HAL_OK)  
        Error_Handler();  
  
    gx_raw = (int16_t)((raw[0] << 8) | raw[1]);  
    gy_raw = (int16_t)((raw[2] << 8) | raw[3]);  
    gz_raw = (int16_t)((raw[4] << 8) | raw[5]);  
  
    *gx = (float)(gx_raw / sensitivity);  
    *gy = (float)(gy_raw / sensitivity);  
    *gz = (float)(gz_raw / sensitivity);  
}
```

Ici ça permet de récupérer 1000 **valeurs brutes** de notre gyroscope puis de les **convertir** (*code de droite*) puis d'en déduire des **offsets** que l'on affichera (*code de gauche*).

1 - Procédure single point - gyromètre

```
/*
 * @brief Calibre le gyroscope en mode "single point" (no turn).
 *
 * Cette fonction mesure N échantillons lorsque le capteur est immobile,
 * calcule les offsets pour chaque axe et les écrit dans les registres
 * matériels de compensation du MPU-9250.
 *
 * @param N Nombre d'échantillons utilisés pour la moyenne
 */
void GyrCalib(uint16_t N) {
    float gyro_data[N][3];
    float avg[3];

    // --- Acquisition des N mesures ---
    for (int i = 0; i < N; i++) {
        ReadGyro(&gyro_data[i][0], &gyro_data[i][1], &gyro_data[i][2]);
        HAL_Delay(10); // selon ODR
    }

    // --- Calcul des offsets moyens ---
    Average(gyro_data, N, avg);

    char msg[100];
    snprintf(msg, sizeof(msg), "Offsets mesurés : X=% .3f, Y=% .3f, Z=% .3f\r\n", avg[0], avg[1], avg[2]);
    HAL_UART_Transmit(&hlpuart1, (uint8_t*)msg, strlen(msg), HAL_MAX_DELAY);

    // --- Conversion en valeurs brutes ---
    float sensitivity = 131.0f; // ±250°/s par défaut
    int16_t off_x = (int16_t)(-avg[0]);
    int16_t off_y = (int16_t)(-avg[1]);
    int16_t off_z = (int16_t)(-avg[2]);

    uint8_t data[2];
```

```
// --- Ecriture dans les registres matériels ---
data[0] = (off_x >> 8) & 0xFF;
data[1] = off_x & 0xFF;
HAL_I2C_Mem_Write(&hi2c1, 0x68 << 1, 0x13, 1, data, 2, HAL_MAX_DELAY);

data[0] = (off_y >> 8) & 0xFF;
data[1] = off_y & 0xFF;
HAL_I2C_Mem_Write(&hi2c1, 0x68 << 1, 0x15, 1, data, 2, HAL_MAX_DELAY);

data[0] = (off_z >> 8) & 0xFF;
data[1] = off_z & 0xFF;
HAL_I2C_Mem_Write(&hi2c1, 0x68 << 1, 0x17, 1, data, 2, HAL_MAX_DELAY);

HAL_UART_Transmit(&hlpuart1, (uint8_t*)"Calibration gyroscope terminée.\r\n", 34, HAL_MAX_DELAY);
```

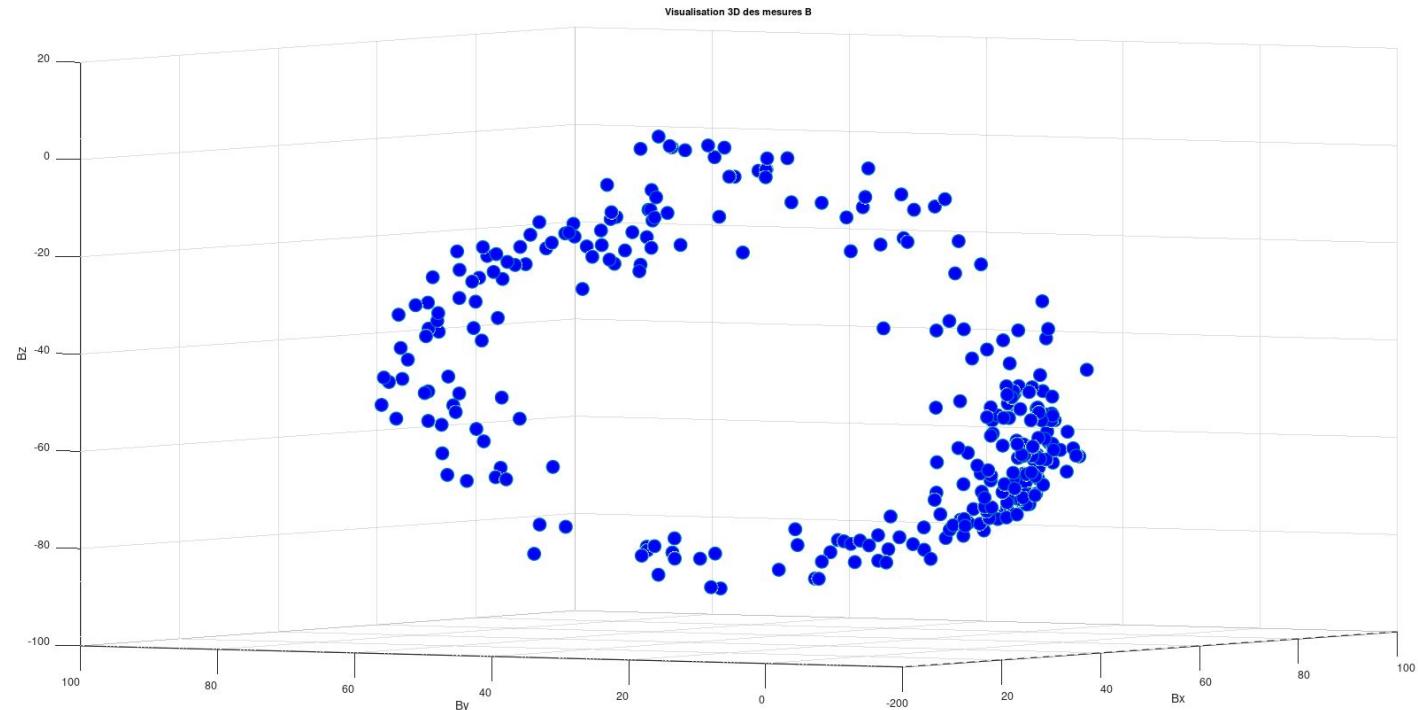
Début de la calibration du gyroscope...
Offsets mesurés : X=-0.812, Y=0.511, Z=-0.138
Calibration gyroscope terminée.
Calibration terminée !
Offsets Gyro (°/s): X=0.000 | Y=0.000 | Z=0.000
 Offsets gyroscope proches de 0

Sans mouvement, on fait la mesure sur un ensemble de **1000 échantillons**. On voit que les valeurs de nos offsets sont de **zéro** (exactement égaux dans ce cas là mais très proche dans la majorité des cas), on a donc réussi notre calibration.

2 - Procédure complète - magnétomètre

2.1 - Nécessité d'une calibration

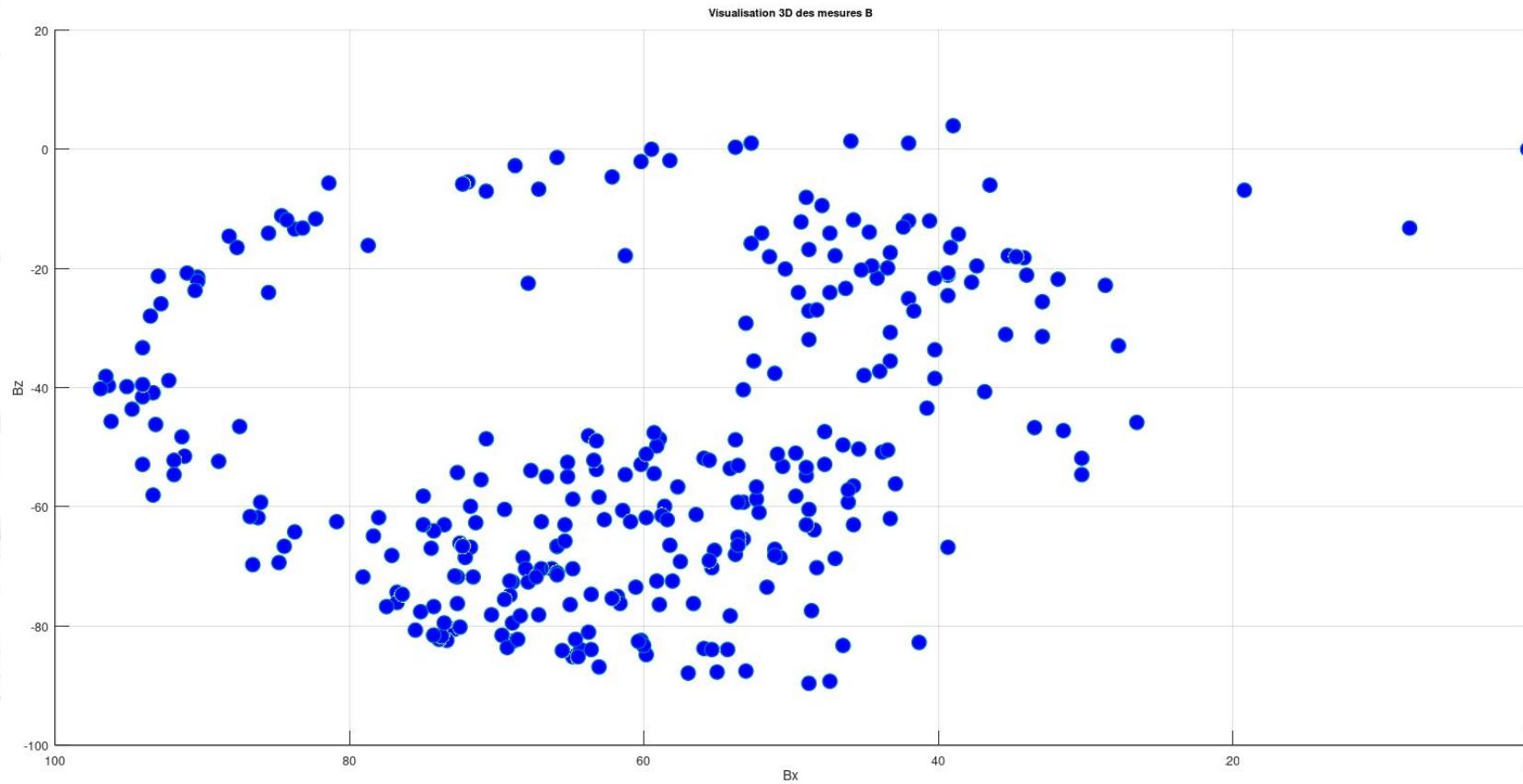
```
% Chemin vers le fichier CSV  
filename = 'C:/Users/kevin/Desktop/MeasureB.csv';  
  
% Lire le fichier CSV sans prendre en compte la première ligne  
data = csvread(filename, 1, 0);  
  
% Séparer les colonnes  
Bx = data(:,1);  
By = data(:,2);  
Bz = data(:,3);  
  
% === Graphe 3D des points ===  
  
figure;  
plot3(Bx, By, Bz, 'o', 'MarkerSize', 6, 'MarkerFaceColor', 'b');  
grid on;  
xlabel('Bx'); ylabel('By'); zlabel('Bz');  
title('Visualisation 3D des mesures B');  
  
% === Courbes 2D des composantes ===  
  
figure;  
plot(Bx, 'r', 'LineWidth', 1.5); hold on;  
plot(By, 'g', 'LineWidth', 1.5);  
plot(Bz, 'b', 'LineWidth', 1.5);  
xlabel('Index de mesure');  
ylabel('Valeurs');  
legend('Bx', 'By', 'Bz');  
title('Evolution des composantes B');  
grid on;
```



On retrouve bien l'idée d'une **sphère**, bien que loin d'être parfaite. Mais le fait que l'on est une erreur assez importante ici est logique puisque nous avons effectué ces tâches “à la main”

2 - Procédure complète - magnétomètre

2.1 - Nécessité d'une calibration



Prenons une vision selon l'axe y, si l'on suppose que le rayon se détermine à partir des deux points extrêmes de notre forme particulière, on retrouve bien ici un rayon de quasiment **50µT** comme attendu (valeur du champ magnétique de la Terre). Cependant, nous ne sommes pas **centrés en zéro**, logique puisqu'il s'agit des données non-calibrées.

2 - Procédure complète - magnétomètre

2.2 - Procédure de calibration

```
/*
 * @brief Calibre le magnétomètre pour compenser les hard iron et soft iron biases.
 * @param hi2c Pointeur vers la structure I2C (ex: &hi2c1)
 * @param mag_bias Tableau de 3 doubles pour stocker les offsets (hard iron) calculés
 * @param mag_scale Tableau de 3 doubles pour stocker les coefficients correcteurs (soft iron) calculés
 * @retval None
 *
 * @note Utilise ReadMag() pour ne pas afficher de messages UART inutiles.
 */
void MagCalib(I2C_HandleTypeDef *hi2c, double mag_bias[3], double mag_scale[3]){
    #define CALIB_SAMPLES 300
    #define CALIB_DELAY_MS 50

    double Bx_min = DBL_MAX, Bx_max = -DBL_MAX;
    double By_min = DBL_MAX, By_max = -DBL_MAX;
    double Bz_min = DBL_MAX, Bz_max = -DBL_MAX;

    for(uint16_t i = 0; i < CALIB_SAMPLES; i++)
    {
        float mx, my, mz;
        ReadMag(&mx, &my, &mz); // lecture sans UART

        if(mx < Bx_min) Bx_min = mx;
        if(mx > Bx_max) Bx_max = mx;

        if(my < By_min) By_min = my;
        if(my > By_max) By_max = my;

        if(mz < Bz_min) Bz_min = mz;
        if(mz > Bz_max) Bz_max = mz;

        HAL_Delay(CALIB_DELAY_MS);
    }
}
```

```
// offsets (hard iron)
mag_bias[0] = (Bx_max + Bx_min) / 2.0;
mag_bias[1] = (By_max + By_min) / 2.0;
mag_bias[2] = (Bz_max + Bz_min) / 2.0;

// rayons et coefficients correcteurs (soft iron simplifié)
double R_x = (Bx_max - Bx_min) / 2.0;
double R_y = (By_max - By_min) / 2.0;
double R_z = (Bz_max - Bz_min) / 2.0;
double R_mean = (R_x + R_y + R_z) / 3.0;

mag_scale[0] = R_mean / R_x;
mag_scale[1] = R_mean / R_y;
mag_scale[2] = R_mean / R_z;
```

Début calibration magnétomètre...

Bias: X=140.897 Y=57.545 Z=-23.005

Scale: X=0.964 Y=1.121 Z=0.934

Calibration terminée

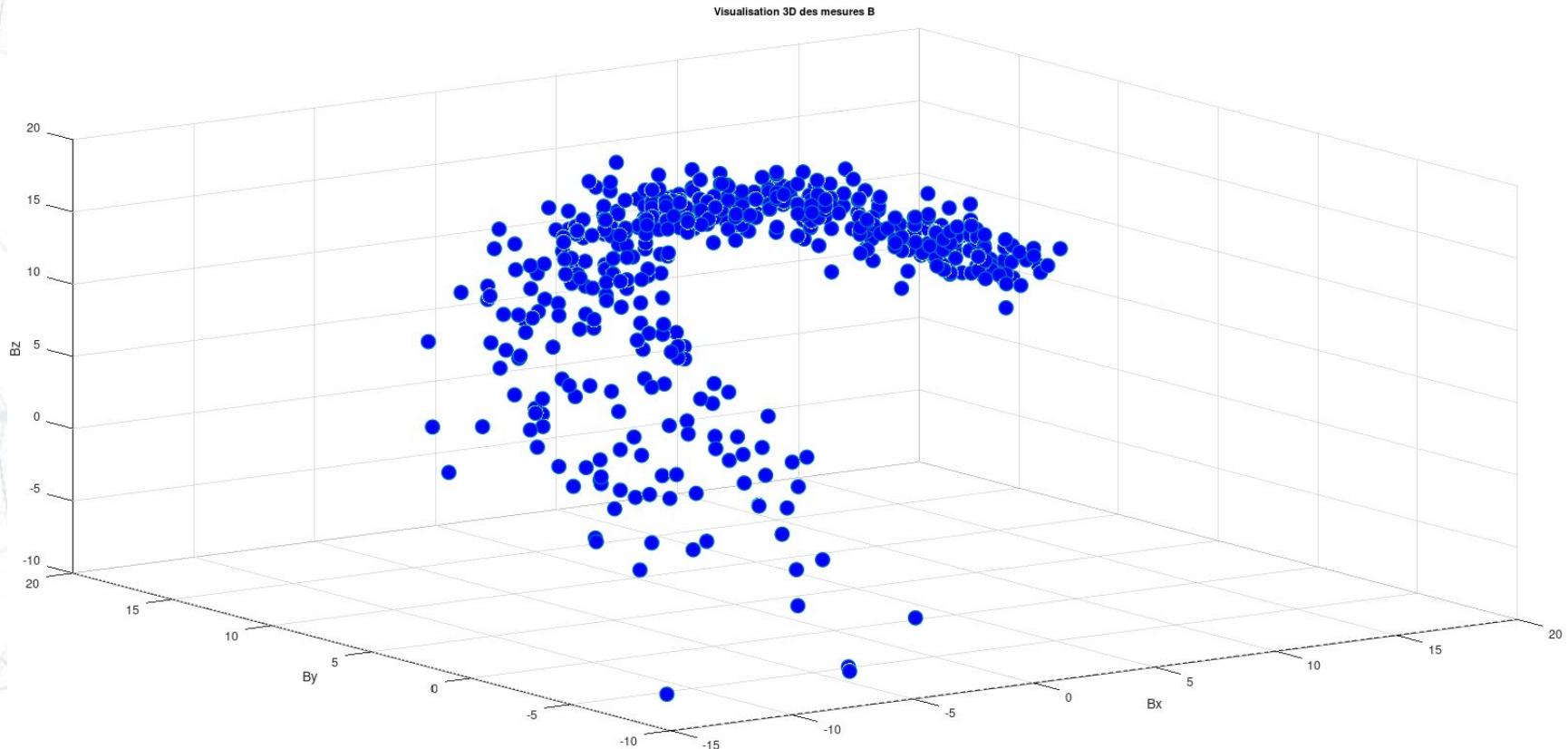
Maintenant que nous avons les **biais** et les **facteurs d'échelle**, nous n'avons plus qu'à les appliquer à nos valeurs brutes. Par la formule suivante :

$$B_{calib} = (B_{raw} - bias) * scale$$

Fin du TP3

2 - Procédure complète - magnétomètre

2.2 - Procédure de calibration

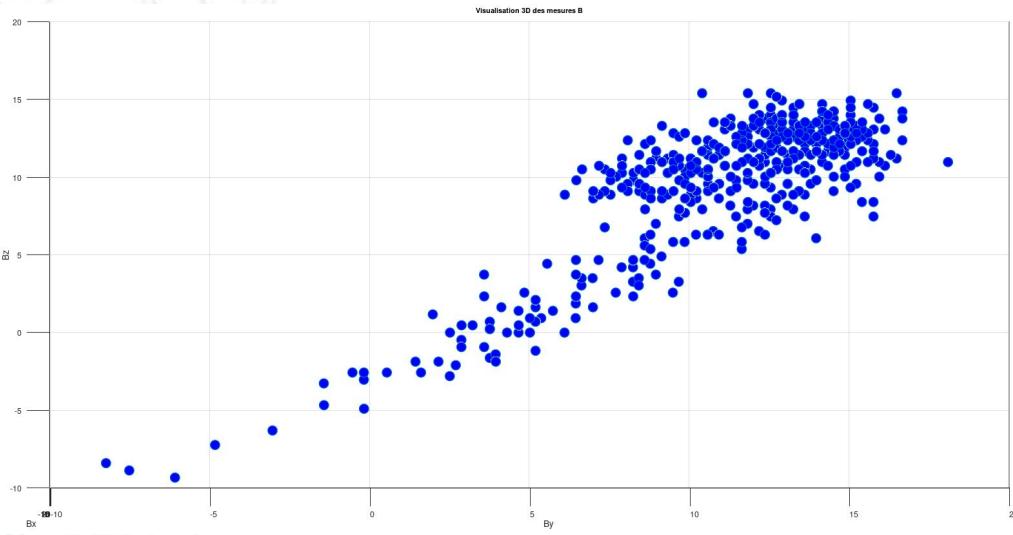
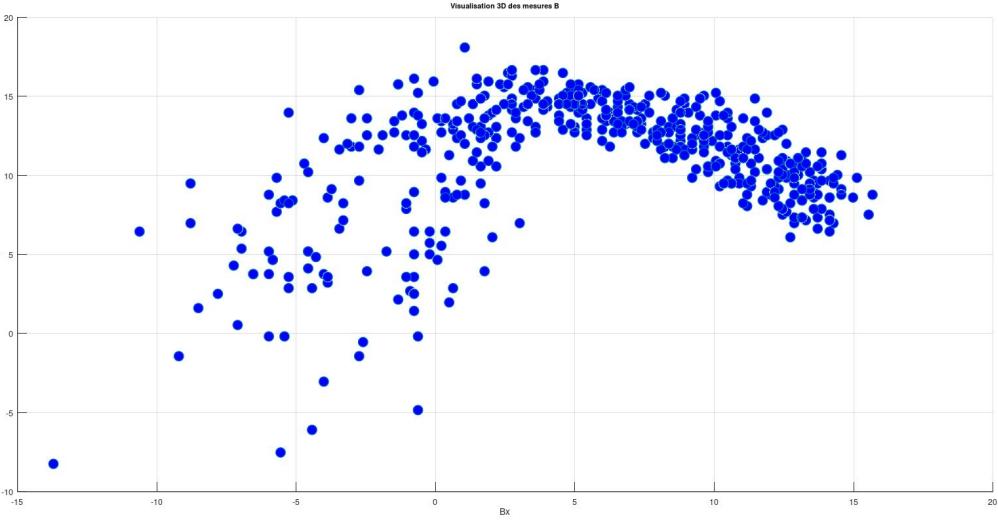
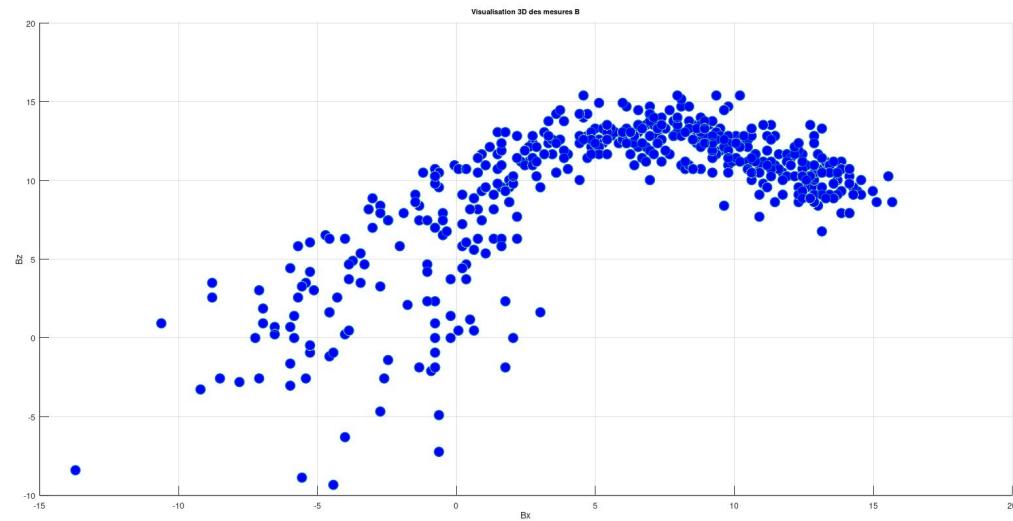


Cette fois-ci, on ne retrouve pas du tout un résultat **rond / ellipsoïdal**. On peut probablement expliquer cela par différentes raisons : **environnements** (perturbations) **different** par rapport à la dernière fois ainsi qu'une potentielle **calibration imprécise**.

On retrouve tout de même une idée d'ellipse partielle (assez vague).

2 - Procédure complète - magnétomètre

2.2 - Procédure de calibration



Malgré l'obtention d'un **résultat particulier**, nous terminons cette partie. Désormais, notre **gyromètre** ainsi que notre **magnétomètre** sont calibrés, nous pouvons passer à la réalisation d'une **boussole électronique**.

Fin du livrable 3

V - Boussole électronique

1 - Boussole compensée

```
/**  
 * @brief Calibre un accéléromètre avec une méthode simple single-point.  
 *  
 * Cette fonction utilise une calibration basique :  
 * - X et Y supposés à 0 g  
 * - Z supposé à 1 g  
 * Elle calcule uniquement l'offset et applique un gain unitaire.  
 *  
 * @param acc Tableau de 3 doubles contenant les mesures brutes de l'accéléromètre (X, Y, Z)  
 * @param calib Pointeur vers une structure CompassCalib_t où seront stockés les offsets et gains  
 */  
  
void CompassCalibSinglePoint(const double acc[3], CompassCalib_t *calib)  
{  
    calib->offset[0] = acc[0]; // X  
    calib->offset[1] = acc[1]; // Y  
    calib->offset[2] = acc[2] - 1.0; // Z en g  
  
    calib->gain[0] = 1.0;  
    calib->gain[1] = 1.0;  
    calib->gain[2] = 1.0;  
}
```

```
// Structure pour stocker les offsets et gains  
typedef struct {  
    double offset[3];  
    double gain[3];  
} CompassCalib_t;  
  
#define PI 3.14
```

Pour rendre notre code plus lisible nous avons créé une **structure** : **CompassCalib_t** qui stocke les **offset** pour chacun des axes (X, Y, Z) ainsi que leurs **gains**.

On calibre notre boussole comme si elle était en **(0 , 0)** et ne subissais que la **force du poids** (soit 1g).
Puis nous fixons un **gain unitaire**.

1 - Boussole compensée

```
/*
 * @brief Calcule et formate les angles d'inclinaison à partir des données de l'accéléromètre.
 *
 * Cette fonction applique une calibration sur les axes, puis calcule :
 * - theta : angle dans le plan XY (atan2(Y, X))
 * - psi   : angle dans le plan XZ (atan2(X, Z))
 * - phi   : angle de rotation par rapport à l'axe Z (atan2(sqrt(X²+Y²), Z))
 *
 * Les angles sont ensuite convertis en degrés et stockés dans une chaîne de caractères.
 *
 * @param acc Tableau de 3 doubles contenant les mesures brutes de l'accéléromètre (X, Y, Z)
 * @param msg Chaîne de caractères où sera écrit le message
 * @param msg_len Taille maximale de la chaîne msg
 * @param calib Pointeur vers la structure CompassCalib_t contenant les offsets et gains à appliquer
 */
void CompassMeasureMsg(const double acc[3], char *msg, size_t msg_len, CompassCalib_t *calib)
{
    double ax = (acc[0] - calib->offset[0]) * calib->gain[0];
    double ay = (acc[1] - calib->offset[1]) * calib->gain[1];
    double az = (acc[2] - calib->offset[2]) * calib->gain[2];

    // Calcul des angles (radians)
    double theta = atan2(ay, ax);                      // Plan (XY)
    double psi   = atan2(ax, az);                      // Plan (XZ)
    double phi   = atan2(sqrt(ax*ax + ay*ay), az);    // Rotation par rapport à Z

    // Conversion en degrés
    theta *= (180.0 / PI);
    psi   *= (180.0 / PI);
    phi   *= (180.0 / PI);

    // Affichage
    sprintf(msg, msg_len,
            "Theta(XY)=%2f deg | Psi(XZ)=%2f deg | Phi(Z)=%2f deg\r\n",
            theta, psi, phi);
}
```



```
Temperature = 28.17 °C
Gyro - X: -0.85 °/s, Y: 0.54 °/s, Z: -0.35 °/s
Accel - X: -0.005 g, Y: -0.995 g, Z: -0.003 g
||A|| = 0.995 g
Mag - X: 139.12 µT, Y: -4.83 µT, Z: -0.17 µT
||B|| = 139.20 µT
Theta(XY)=-180.00 deg | Psi(XZ)=-180.00 deg | Phi(Z)=180.00 deg
```



```
Temperature = 28.15 °C
Gyro - X: -0.95 °/s, Y: 0.49 °/s, Z: -0.28 °/s
Accel - X: -0.008 g, Y: -0.987 g, Z: 0.002 g
||A|| = 0.987 g
Mag - X: 140.36 µT, Y: -4.65 µT, Z: 0.34 µT
||B|| = 140.44 µT
Theta(XY)=-180.00 deg | Psi(XZ)=-0.00 deg | Phi(Z)=0.00 deg
```

Ici, on voit que selon l'**inclinaison** du capteur nous arrivons à faire varier correctement les **angles** (*le 0° étant initialisé lors de la calibration*). Mais pour l'instant nous n'affichons que les valeurs **0°** et **180°**, précisons donc cela afin d'obtenir le **roll**, le **pitch** et le **yaw** (*roulis, tangage, lacet*).

1 - Boussole compensée

```
/*
 * @brief Calcule l'orientation complète compensée en inclinaison
 * @param acc (calibré) : tableau de 3 valeurs de l'accéléromètre [X,Y,Z] en g
 * @param mag (calibré) : tableau de 3 valeurs du magnétomètre [X,Y,Z] en µT
 * @param roll Pointeur vers variable qui recevra le roulis (deg)
 * @param pitch Pointeur vers variable qui recevra le tangage (deg)
 * @param heading Pointeur vers variable qui recevra l'azimut nord magnétique (deg)
 * @retval None
 *
 * @note
 * - Roulis et tangage sont calculés à partir de l'accéléromètre
 * - Azimut (heading) est compensé pour l'inclinaison
 * - Toutes les valeurs sont renvoyées en degrés
 * - Formules trouvées dans un document dans ../docs
 */
void TiltCompensatedCompass(const double acc[3], const double mag[3],
                            double *roll, double *pitch, double *heading){
    // Calcul roulis et tangage à partir de l'accéléromètre
    // Roulis = Rotation autour de X
    *roll = atan2(acc[1], acc[2]) * (180.0 / PI);

    // Tangage = Rotation autour de Y
    *pitch = atan2(-acc[0], sqrt(acc[1]*acc[1] + acc[2]*acc[2])) * (180.0 / PI);

    // Conversion en radians pour les calculs de compensation
    double roll_rad = *roll * (PI / 180.0);
    double pitch_rad = *pitch * (PI / 180.0);

    // Compensation magnétomètre
    double Xh = mag[0]*cos(pitch_rad) + mag[2]*sin(pitch_rad);
    double Yh = mag[0]*sin(roll_rad)*sin(pitch_rad) + mag[1]*cos(roll_rad) - mag[2]*sin(roll_rad)*cos(pitch_rad);

    // Azimut compensé (nord magnétique)
    *heading = atan2(Yh, Xh) * (180.0 / PI);

    if(*heading < 0) *heading += 360.0;
}
```

```
Theta(XY)=-180.00 deg | Psi(XZ)=0.00 deg | Phi(Z)=0.00 deg
Roulis=-5.04 deg | Tangage=1.36 deg | NordMag=74.71 deg

Theta(XY)=-0.00 deg | Psi(XZ)=0.00 deg | Phi(Z)=0.00 deg
Roulis=-2.23 deg | Tangage=-0.25 deg | NordMag=128.26 deg

Theta(XY)=0.00 deg | Psi(XZ)=0.00 deg | Phi(Z)=0.00 deg
Roulis=0.43 deg | Tangage=-1.32 deg | NordMag=171.97 deg

Theta(XY)=0.00 deg | Psi(XZ)=0.00 deg | Phi(Z)=0.00 deg
Roulis=0.68 deg | Tangage=-1.02 deg | NordMag=169.93 deg

Theta(XY)=0.00 deg | Psi(XZ)=0.00 deg | Phi(Z)=0.00 deg
Roulis=0.36 deg | Tangage=-1.58 deg | NordMag=168.97 deg

Theta(XY)=-180.00 deg | Psi(XZ)=-0.00 deg | Phi(Z)=0.00 deg
Roulis=-2.24 deg | Tangage=4.70 deg | NordMag=302.69 deg
```



Affichage en faisant **tourner** (selon l'axe vertical) doucement le capteur. **Mesure cohérentes** car les valeurs de roulis et de tangage restent **proches de 0** (capteur à plat) et l'éloignement en degré au Nord magnétique augmente bien pendant notre demi tour.

On a désormais une **boussole électronique** permettant de lire l'**angle de roulis** et de **tangage**, de **compenser l'azimut** et de nous donner l'angle par rapport au **nord magnétique**.
Essayons désormais d'améliorer cette dernière.

2 - Boussole multi-capteurs

TIM2 Mode and Configuration

Mode

Clock Source Internal Clock

Configuration

Reset Configuration

NVIC Settings DMA Settings

Parameter Settings User Constants

Configure the below parameters :

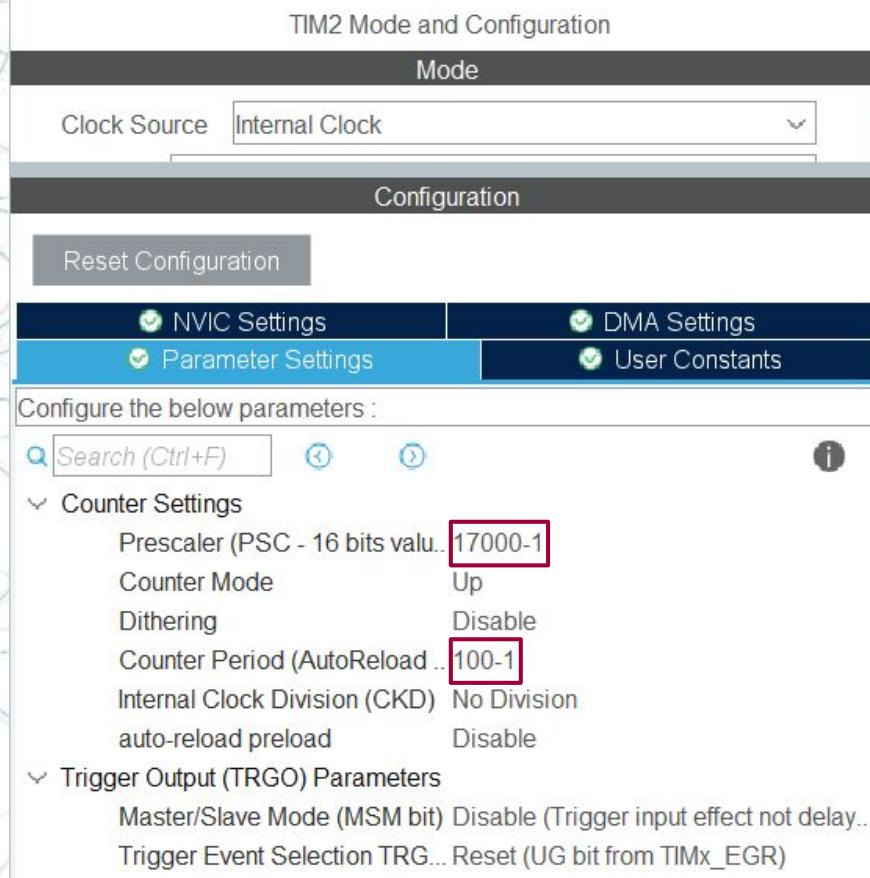
Search (Ctrl+F)

Counter Settings

- Prescaler (PSC - 16 bits val..) **17000-1**
- Counter Mode Up
- Dithering Disable
- Counter Period (AutoReload ..) **100-1**
- Internal Clock Division (CKD) No Division
- auto-reload preload Disable

Trigger Output (TRGO) Parameters

- Master/Slave Mode (MSM bit) Disable (Trigger input effect not delay...)
- Trigger Event Selection TRG... Reset (UG bit from TIMx_EGR)



170 APB1 timer clocks (MHz) → Notre timer fonctionne à la fréquence d'APB1 : **170MHz**

double Te = 0.01;
double tau = 0.1;

Période d'échantillonnage : **10ms**
Constante de temps : **100ms**

On a donc ici une fréquence d'échantillonage de **100Hz**. En choisissant un PSC de **16999**, notre timer compte désormais à **10kHz**. Et ce dernier déclenche une interruption toutes les ARR+1 périodes. Ainsi, en divisant ARR+1 par la fréquence que nous venons de calculer, nous obtenons une période de **10ms**, soit une fréquence de **100Hz**.

Et c'est exactement à cette fréquence que nos capteurs effectuent des mesures.

When continuous measurement mode 1 (MODE[3:0] = "0010") or 2 (MODE[3:0] = "0110") is set, sensor is measured periodically at 8Hz or **100Hz** respectively.

```
// Demarrage du Timer
HAL_TIM_Base_Start_IT(&htim2);
HAL_UART_Transmit(&hlpuart1, (uint8_t*)"Timer 100 Hz démarré\r\n", 26, HAL_MAX_DELAY);
```

2 - Boussole multi-capteurs

```
void TiltCompensatedCompassAngles(const double acc[3], const double mag[3],
                                  double *roll, double *pitch, double *heading) {
    static double roll_prev = 0.0;
    static double pitch_prev = 0.0;

    /* angles depuis acc (deg) */
    double roll_acc = atan2(acc[1], acc[2]) * (180.0 / PI);
    double pitch_acc = atan2(-acc[0], sqrt(acc[1]*acc[1] + acc[2]*acc[2])) * (180.0 / PI);

    /* utilisation d'un simple alpha pour lisser (exponential smoothing) */
    CompassInitParameters(); /* récupère K (on peut réutiliser K comme alpha si tu veux) */
    double alpha = 1.0 - K; /* si tau grand -> K proche de 1 -> alpha petit -> plus lissage */

    *roll = alpha * roll_acc + (1.0 - alpha) * roll_prev;
    *pitch = alpha * pitch_acc + (1.0 - alpha) * pitch_prev;

    roll_prev = *roll;
    pitch_prev = *pitch;

    /* heading via mag si disponible */
    if (mag != NULL && heading != NULL) {
        double roll_rad = (*roll) * (PI / 180.0);
        double pitch_rad = (*pitch) * (PI / 180.0);

        double Xh = mag[0]*cos(pitch_rad) + mag[2]*sin(pitch_rad);
        double Yh = mag[0]*sin(roll_rad)*sin(pitch_rad) + mag[1]*cos(roll_rad) - mag[2]*sin(roll_rad)*cos(pitch_rad);

        *heading = atan2(Yh, Xh) * (180.0 / PI);
        if (*heading < 0.0) *heading += 360.0;
    } else if (heading != NULL) {
        *heading = NAN;
    }
}
```

```
/*
 * @brief Initialise K si nécessaire.
 */
static void CompassInitParameters(void)
{
    if (K == 0.0) {
        if ((tau + Te) != 0.0) {
            K = tau / (tau + Te);
        } else {
            // fallback pour éviter la division par 0
            K = 0.98;
        }
    }
}
```

Initialisation et calcul de K

$$K = \frac{\tau}{\tau+Te}$$

Cette fonction était notre **premier jet** pour obtenir notre boussole électronique multi-capteurs mais malheureusement elle ne **fonctionnait pas réellement**, c'est même l'inverse, elle proposait des angles de roulis et de tangage bien moins précis.

2 - Boussole multi-capteurs

```
Theta(XY)=-0.00 deg | Psi(XZ)=0.00 deg | Phi(Z)=0.00 deg  
Roulis=-6.46 deg | Tangage=-1.18 deg | NordMag=189.38 deg  
Roulis amélioré =-6.32 deg | Tangage amélioré =-1.26 deg | NordMag amélioré =189.40 deg
```

```
Theta(XY)=-0.00 deg | Psi(XZ)=0.00 deg | Phi(Z)=0.00 deg  
Roulis=-6.37 deg | Tangage=-1.15 deg | NordMag=158.66 deg  
Roulis amélioré =-6.32 deg | Tangage amélioré =-1.25 deg | NordMag amélioré =158.69 deg
```

```
Theta(XY)=-0.00 deg | Psi(XZ)=0.00 deg | Phi(Z)=0.00 deg  
Roulis=-6.74 deg | Tangage=-1.26 deg | NordMag=146.44 deg  
Roulis amélioré =-6.36 deg | Tangage amélioré =-1.25 deg | NordMag amélioré =147.15 deg
```

```
Theta(XY)=-0.00 deg | Psi(XZ)=0.00 deg | Phi(Z)=0.00 deg  
Roulis=-6.50 deg | Tangage=-1.30 deg | NordMag=289.68 deg  
Roulis amélioré =-6.37 deg | Tangage amélioré =-1.26 deg | NordMag amélioré =289.68 deg
```

Voici ce que nous obtenions. Ici tout semble bien puisque les **angles** et les **angles améliorés** semblent **proches** donc on se dit qu'ils doivent être juste **plus précis**. Mais en réalité dès qu'on tournait le capteur, les angles améliorés ne **variait que peu** et donc étaient **très loin de la réalité**.

Cependant, l'angle du nord magnétique amélioré lui **variait autant que ce dernier non-amélioré**, peut-être y avait-il une **réelle amélioration**.

2 - Boussole multi-capteurs

```
void TiltCompensatedComplementary(const double acc[3], const double gyro_rates[3], const double mag[3],
                                    double *roll, double *pitch, double *heading)
{
    static double roll_prev = 0.0;
    static double pitch_prev = 0.0;

    // Initialisation de K
    CompassInitParameters();

    // Angles à partir de l'accéléromètre
    double roll_acc = atan2(acc[1], acc[2]) * (180.0 / PI);
    double pitch_acc = atan2(-acc[0], sqrt(acc[1]*acc[1] + acc[2]*acc[2])) * (180.0 / PI);

    // Intégration du gyroscope
    double roll_from_gyro = roll_prev + Te * gyro_rates[0];
    double pitch_from_gyro = pitch_prev + Te * gyro_rates[1];

    // Formule proposée par l'énoncé
    *roll = K * (roll_from_gyro) + (1.0 - K) * roll_acc;
    *pitch = K * (pitch_from_gyro) + (1.0 - K) * pitch_acc;

    // Sauvegarde des valeurs
    roll_prev = *roll;
    pitch_prev = *pitch;

    // Calcul du heading par mag
    if (mag != NULL && heading != NULL) {
        double roll_rad = (*roll) * (PI / 180.0);
        double pitch_rad = (*pitch) * (PI / 180.0);

        double Xh = mag[0]*cos(pitch_rad) + mag[2]*sin(pitch_rad);
        double Yh = mag[0]*sin(roll_rad)*sin(pitch_rad) + mag[1]*cos(roll_rad) - mag[2]*sin(roll_rad)*cos(pitch_rad);

        *heading = atan2(Yh, Xh) * (180.0 / PI);
        if (*heading < 0.0) *heading += 360.0;
    } else if (heading != NULL) {
        *heading = NAN;
    }
}
```

Nous avons donc proposé cette **deuxième version** que nous avons malheureusement pas eu le temps de tester afin de voir si elle marchait ou non.

C'est donc ici que nous nous arrêtons pour ce travail pratique de 16 heures.