MSC - Capteurs

Travaux Pratiques

F. Goutailler - T. Tang - N. Simond

Table des matières

Cet ensemble de 4 séances de TP a pour objectif global de compléter le cours de Capteurs, du module Electronique pour les Systèmes Embarqués.

Attention, le travail à fournir dépasse largement le cadre des 16h encadrées !

Les éléments abordés appartiennent donc à un large domaine, de la carte STM32 en tant que système embarqué (notion de BSP, liaison I2C, etc.) à la mise au point et l'utilisation d'une chaîne de mesure (sensibilité, bande-passante, calibration, ...). Le sujet est dans un premier temps très guidé, afin que tous les étudiants aient la même base de départ. La seconde partie du sujet est beaucoup moins détaillée.

Les outils matériel et logiciel à utiliser sont :

- une carte d'évaluation STM32 Nucleo-G431RB (`https://os.mbed.com/platforms/XXX`) ;
- une centrale inertielle IMU9250 comportant un accéléromètre, un gyromètre, un magnétomètre et un capteur de température
- STM32Cube IDE pour la gestion du code et l'interface avec la carte STM32 ;
- Doxygen pour l'édition automatique de l'aide de votre code.

Les objectifs en terme de compétences sont multiples :

- maîtrise de STM32Cube IDE (environnement de type Eclipse) ;
- apprentissage de Doxygen ;
- approfondissement des connaissances sur les cartes de type STM32 ;
- compréhension et mise en place d'une liaison par protocole I2C ;
- lecture et compréhension complète d'une *datasheet* ;
- modification des spécificités d'une chaîne de mesure en lien avec une application donnée ;
- calibration d'un capteur par différentes méthodes ;
- conception et implémentation d'un filtre complémentaire ...

3 livrables sont à fournir, après les séances paires (2 et 4), aux dates qui vous seront communiquées ;

- vos fichiers de code principaux : `main.c`, `functions.c` et `functions.h`. Attention de bien respecter les différentes indications de l'énoncé, afin que ces fichiers puissent être testés hors de votre projet, par votre encadrant de TP ;
- la documentation de vos fichiers, générée avec Doxygen, au format pdf ;
- un document de synthèse avec l'ensemble des réponses aux questions posées sous forme de diaporama afin de privilégier vos solutions à chaque question posée. Abuser de captures (avant/après) qui prouvent l'efficacité des solutions proposées. **Les questions apparaissent dans les parties du texte encadrées**.

L'évaluation sera étonnamment basée sur :

- le déroulement des séances : ponctualité, investissement en séance, ...,
- la qualité de chaque livrable,
- l'avancement du projet,
- le retour de la carte et du capteur en temps et en heure.

Les passages en bleu vous indique l'étape à atteindre pour chaque séance et donc le contenu attendu de chaque livrable. Les passages en orange sont destinés aux étudiants les plus avancés dans la séance ou à ceux qui veulent aller plus loin entre les séances.

1 Prise en main de la carte STM32 et des capteurs

1.1 Programmation en C d'un micro-contrôleur

Tous les préceptes présentés par L. Monchal en 1A et 2A demeurent valables. Nous attirons néanmoins votre attention sur le fait que sur μC l'allocation dynamique de mémoire n'est pas disponible (fonction `malloc` et ses dérivées), il est donc d'usage de travailler uniquement avec de l'allocation statique. Afin de limiter de trop nombreuses allocations en mémoire, il est d'usage de déclarer les variables principales en tant que globales dans les fichiers qui les utilisent principalement et de les référencer en tant qu'*extern* dans les fichiers secondaires où vous développerez vos fonctions. Ainsi, il devient inutile de passer en paramètre ces mêmes variables aux différentes fonctions qui les utiliseront.

Par exemple, une seule structure de type `I2C_HandleTypeDef` (configuration du périphérique I2C utilisé) est déclarée dans le `main.c` automatiquement suite à la configuration du fiochier *.ioc*. De même, il suffira de déclarer 3 variables de type tableau de `double` pour enregistrer les 3 composantes des 3 capteurs gyroscope, accéléromètre et magnétomètre.

1.2 Carte STM32 Nucleo G431RB et STM32Cube IDE

STM32Cube IDE est un logiciel complet permettant :

1. une configuration sous forme graphique de la carte d'évaluation utilisée : choix des horloges, activation des périphériques, mise en place des interruptions, calcul de la consommation, ...
2. une génération automatique du code C, à partir de la configuration choisie ;
3. une modification et un test du code C, dans un environnement Eclipse : listing des fonctions et variables, debugger, accès à la mémoire...

Pour gagner du temps et vous permettre d'aller plus loin dans la partie capteur de ce projet, toute la phase d'initialisation a déjà été faite (cf. cours de 2A) et mise sous forme de projet, nommé *G431RB-MSC-Sensors*.

Dans un premier temps, récupérez sous Moodle l'archive du projet (*G431RB-MSC-Sensors.zip*) et enregistrez-la dans votre répertoire de travail personnel.

Dans le logiciel STM32CubeIDE :

- cliquer sur *File / Import* ;
- choisir l'onglet *General / Existing Projects into Workspace*. Appuyer sur *next* ;
- dans la nouvelle fenêtre qui apparaît, il faut cliquer sur *Select Archive File* puis l'archive du projet. Il suffit de cliquer sur *Finish* pour procéder à l'importation ;
- le projet doit apparaitre dans la fenêtre *Project Explorer*.
- Tester la bonne compilation de ce dernier : *Project/Build All* ou icône en forme de marteau ou touche F9 ;
- par un double clic sur le fichier *G431RB-Sensor.ioc*, lancer l'éditeur de configuration et vérifier la configuration adoptée : bus I2C validé et broches associées... ;
- brancher la carte à votre PC via le port USB ;
- repérer dans le fichier *tools.c* les quelques lignes permettant le clignotement de la led verte ;
- Compiler votre programme et entrer en mode *DEBUG* (**F11** ou **Run/Debug**). Tester par une exécution pas à pas votre programme ;
- à l'instar de la plateforme MBED, STM32Cube IDE peut générer un fichier binaire exécutable qu'il suffit alors de copier/coller ou glisser/déposer sur la carte d'évaluation.

  Repérer dans votre répertoire projet, le fichier exécutable, avec l'extension **.bin**. Glisser/déposer ce fichier dans votre carte et vérifier la bonne exécution du programme. Vous disposez ainsi de 2 méthodes pour tester vos programmes, en fonction de vos besoins.

Pour sauvegarder votre projet, il suffit d'un clic droit sur le projet puis *Export*, onglet *(General / Archive File)*. Il suffit alors de laisser les options cochées par défaut et de choisir le répertoire de destination. Bien entendu, toute solution à partir de git/github fera aussi l'affaire.

Pour importer le projet sur un nouvel ordinateur, il suffit de répéter la procédure utilisée ci-dessus avec le fichier (*G431RB-MSC-Sensors.zip*).

Dernier conseils, avant de rentrer dans le monde merveilleux des capteurs : dans STM32Cube IDE, n'écrire du code qu'entre les balises de type *USER SECTION*. Tout ce que vous produirez devra apparaître dans le répertoire *Utils* : vous éviterez ainsi de modifier le *main* et tous les fichiers du répertoire *Core*, générés automatiquement. Vous pouvez générez un couple de fichiers source/en-tête part livrable par exemple. Une fois que vous êtes satisfait du résultat, pensez à encapsuler votre travail dans une fonction avec un nom explicite de façon à pouvoir l'appeler rapidement lors d'un test.

1.3 Prise en main de la carte

Il vous faut dans un premier temps allumer les 2 LEDs (rouge et verte) durant une seconde à l'aide des fonctions HAL. Cela s'avèrera relativement simple si vous prenez le temps de comprendre le contenu de *stm32f4xx_hal_gpio.c* et *stm32f4xx_hal.c*.

Commenter chacune des lignes déjà présentes dans le fichier *tools.c*. Compléter la fonction `test_LEDs` de façon à faire clignoter les leds avec des périodes différentes. Mettez en oeuvre une liaison série entre la carte et le PC de façon à afficher dans un terminal un message de bienvenue à votre convenance.

1.4 Premiers pas avec le bus I2C

Quand a été inventé le bus I2C et quelle est sa version actuelle ?
Combien de lignes nécessite le bus I2C ? Quelle est leur fonction ?
Le bus I2C est un bus série, synchrone, bidirectionnel et half-duplex. Donner la définition des ces 4 éléments.
Sur combien de bits est « classiquement » codée l'adresse d'un périphérique sur un bus I2C ? Combien de périphériques peuvent être alors connectés au même bus ?
Quelle est la vitesse de transmission possible des octets sur un bus I2C ?

Dans un premier temps, nous allons effectuer un scan du bus I2C, afin de vérifier les périphériques présents :

- brancher votre capteur à la carte STM. **Pensez à bien vérifier la plage d'alimentation de votre capteur** et les broches liées au port I2C choisi ;
- dans la documentation de la librairie HAL, au chapitre 34 (*HAL I2C Generic Driver*), chercher une fonction permettant de tester la présence de périphériques sur le bus I2C ;
- écrire les quelques lignes de code (à placer dans *User Code 2*) permettant d'utiliser cette fonction pour scanner le bus I2C et sauvegarder dans un tableau les adresses des périphériques trouvés. Compiler et tester votre programme.

Si tout est correct, vous devez voir apparaitre 6 adresses :

- 0 : permet de s'adresser à tous les périphériques du bus en même temps (*General call*) ;
- 1 : permet une synchronisation de l'ensemble des périphériques du bus ;
- 208, 209, 238 et 239.

Pour info, suivant la méthode employée (code exécuté), les 2 premières valeurs (0 et 1) n'apparaissent pas toujours suite au balayage des adresses accessibles. Si c'est votre cas, ne vous formalisez pas et poursuivez.

Expliquer l'origine des 4 dernières adresses, grâce aux données indiquées sur le capteur fourni.

Dans un second temps, nous allons vérifier l'identité du capteur MPU-9250 :

- dans la documentation du MPU-9250, chercher quel registre permet de vérifier l'identité du capteur utilisé. Quelle est la valeur de retour attendue lors de la lecture de ce registre ?
- dans la documentation de la librairie HAL, au chapitre 34 (*HAL I2C Generic Driver*), chercher une fonction permettant de lire directement un ou plusieurs octets dans un registre de périphérique ;

- écrire les quelques lignes de code permettant d'utiliser cette fonction pour lire le registre adéquat et vérifier la valeur renvoyée.

Une bonne habitude à prendre en programmation est de faire de la gestion d'erreurs. Lors de l'utilisation du bus I2C (lecture ou écriture d'octets), cela consiste à vérifier la valeur retournée par la fonction utilisée. Si cette valeur n'est pas `HAL_OK`, il faut alors appeler la fonction `Error_Handler();` (fonction déjà définie dans le bas de votre fichier *main.c*. Le traitement de l'erreur se fait ainsi au sein de cette fonction. Cela permet par exemple d'allumer une LED rouge ou d'afficher le fichier et le numéro de la ligne concernés par l'erreur. On peut aussi imaginer une procédure permettant de traiter ou « résoudre » l'erreur, en fonction de cette dernière : modification de paramètres, choix offert à l'utilisateur, *reset* du capteur...

Modifier vos fonctions précédentes en intégrant cette gestion d'erreur. Déclencher l'allumage de la LED rouge lors d'une erreur.

2 Acquisition des données

2.1 Lecture des données

2.1.1 Initialisation des capteurs

Avant de lire les premières données de température, d'accélération et de vitesse angulaire, il faut initialiser les différents capteurs. L'étape d'initialisation sera fera ici *a minima*, mais pourra être complétée par la suite, en fonction des besoins.

- créer un fichier *functions.c* et un fichier *functions.h* qui contiendront le prototype et le corps des différentes fonctions à venir ;
- écrire la fonction `InitSensors`.
  Cette fonction doit permettre les étapes suivantes :
  1. *hardware reset* : remise à zéro de l'ensemble des registres du capteur puis restauration des valeurs par défaut ;
  2. attente d'un délai de 100 ms (fonction *HAL_Delay* à utiliser par exemple) ;
  3. *clock selection* : choisir la meilleure horloge possible avec PLL.

Ne pas oublier de commenter cette fonction en respectant les balises Doxygen : @brief, @param, @return... Il en sera de même pour toutes les fonctions écrites dans la suite de ce projet.

Observer quelques trames I2C à l'oscilloscope numérique et vérifier leur cohérence : fréquence d'horloge, bit de *START*, bit de *STOP*...

2.2 Température

On s'intéresse dans un premier temps à la lecture des données de température.

Quel(s) registres faut-il utiliser pour lire la température mesurée par le MPU-9250 ? Comment passe-t-on des données « brutes » à la température en degrés Celsius ? Que représentent les paramètres *RoomTemp_Offset* et *Temp_Sensitivity* ? Quelles sont leurs valeurs ? Expliquer leurs unités.

Pour la lecture de le température, créer une fonction `TempMeasure` qui recueillera la température mesurée par le capteur.

Appeler votre fonction dans la boucle infinie de votre programme et testez la, en affichant le résultat de la mesure ou en lisant quelques mesures via le debugguer.

LIVRABLE 1

2.3 Accélération

On traite à présent les données d'accélération.

> Quel(s) registres faut-il utiliser pour lire les accélérations mesurées par le MPU-9250 ?
> Comment sont codées les données d'accélérations ? En déduire le passage des données « brutes » à l'accélération selon les 3 axes exprimées en g et non en $m.s^{-2}$.
> Quelle valeur d'accélération a_z vais-je mesurer si j'oriente l'axe (Oz) de l'accéléromètre vers le haut ? Même question vers le bas ?
> Si vous avez répondu -1 puis $+1$ aux 2 questions précédentes, prenez le temps de regarder le début de la vidéo suivante : `https://www.dailymotion.com/video/x28d9c7`.

Pour la lecture des données de l'accéléromètre, créer la fonction `AccMeasure` qui enregistrera les données de l'accéléromètre selon les 3 axes.

Le second argument de cette fonction, de type `double*`, pointera vers une zone mémoire contenant les informations d'accélération selon les axes $\vec{x}$, $\vec{y}$ et $\vec{z}$. **On fera dans un premier temps l'hypothèse d'une étendue de mesure fixe, égale à la valeur par défaut** $\pm 2g$.

Tester votre fonction par quelques mesures statiques ou dynamiques, en orientant le capteur de différentes manières. En incluant la librairie *math.h*, faire le calcul de la norme du vecteur gravité $||\vec{G}|| = \sqrt{a_x^2 + a_y^2 + a_z^2}$ (cas statique) et vérifier que la valeur obtenue est bien très proche de 1.

Afficher les données d'accélération sur le terminal.

Améliorer votre programme par une lecture automatique de la sensibilité de l'accéléromètre dans le registre idoine.

2.4 Vitesse angulaire

Pour la lecture des donnée du gyromètre, respecter une procédure similaire à celle de l'accéléromètre.

2.5 Champ magnétique

> Quelles sont valeurs des composantes du champ magnétique terrestre à Cergy-Pontoise ?
> Pourquoi le magnétomètre du MPU-9250 n'est pas directement accessible sur le bus I2C ? En lisant le document *MPU-9250 Product Specification*, proposer une solution pour le voir apparaitre sur le bus.
> Quelles adresses seront utilisées pour communiquer avec ce capteur en lecture et écriture ?

Modifier votre fonction `Init`, afin de permettre la communication avec le magnétomètre, via le bus I2C. Vérifier cette bonne communication en lisant le registre *WHO_AM_I* du capteur. Si cela fonctionne vous devez retrouver la valeur 0x48 ou 72.

Toujours dans la fonction `Init`, configurer le magnétomètre (registre *Control 1*), avec les paramètres suivants :

- *Continuous measurement mode 2* (mesure des données à une fréquence de 100Hz) ;
- *16-bit output*.

En lisant la *datasheet* du capteur **en détail** et notamment le paragraphe *6.4.3.2. Normal Read Sequence*, écrire une fonction de lecture des données du champ magnétique, selon les 3 axes. Voici le prototype à respecter :

```
void MagMeasure(I2C_HandleTypeDef*,double*)
```

Tester votre fonction, en faisant quelques mesures statiques. Faire le calcul de la norme du vecteur champ magnétique $||\vec{B_T}|| = \sqrt{B_x^2 + B_y^2 + B_z^2}$ (cas statique) et vérifier que la valeur obtenue est bien très proche de $50\mu T$.

Petit détour par la CEM... approcher votre capteur d'un ordinateur, d'une table en métal ou même de votre carte STM32 et regarder l'évolution des valeurs du champ magnétique mesuré.

3 Modification des paramètres métrologiques

3.1 Sensibilité

> Définir la sensibilité d'un capteur. Quelle différence faîtes-vous entre la sensibilité statique et la sensibilité dynamique ?
> Quels sont les réglages possibles ici pour la sensibilité de l'accéléromètre ?
> Quel registre faut-il modifier pour faire évoluer cette sensibilité ?

On place ici l'accéléromètre tel que son axe (Oz) soit approximativement dirigé à l'inverse du champ de pesanteur terrestre. Faire varier la sensibilité selon les valeurs possibles et relever la valeur « brute » des données, après une attente d'au moins 100ms. Comparer ces données entre elles et vérifier leur cohérence.

3.2 bande-passante

> Rappeler le lien entre bande-passante d'un capteur et valeur efficace du bruit de ce capteur.
> Quels sont les réglages possibles ici pour la bande-passante du gyromètre ?
> Quels registres faut-il modifier pour faire évoluer ce paramètre ?

Code une fonction de prototype `GyrNoise` qui mesure et sauvegarde (second paramètre de la fonction) la valeur efficace du bruit pour les 3 axes de mesure du gyromètre.

Dans le programme principal, faire évoluer la bande-passante du gyromètre de 3600 à 5Hz en 4 ou 5 valeurs et noter à chaque fois les valeurs efficaces du bruit pour les 3 axes. Faire un tracé (Matlab ou Octave par exemple) de l'évolution de la valeur efficace de bruit en fonction de la bande bande. Retrouvez-vous la forme de courbe attendue ?

3.3 Fréquence d'Echantillonnage

> Dans la *datasheet*, que désigne le terme *Output Data Rate (ODR)* pour l'accéléromètre ?
> Comment est-il possible de configurer ce paramètre ? Quel intervalle de valeurs peut-il prendre ?
> Dans une application industrielle, quels facteurs faudrait-il prendre en compte pour régler au mieux ce paramètre ?

Modifier votre fonction `Init` pour fixer le paramètre *ODR* à sa plus faible valeur (*i.e.* 3,91Hz). Pour faire cela, il faut écrire dans le registre *Sample Rate Divider* et le registre *Configuration*. En effet, le capteur n'acceptera de prendre en compte la valeur du *Sample Rate Divider* que si sa fréquence d'échantillonnage initiale est fixée à 1KHz. Le paramètre *DLPF_CFG* ne peut donc pas prendre n'importe quelle valeur.

Compiler et tester l'effet de cette faible fréquence d'échantillonnage.

LIVRABLE 2

4 Calibration

4.1 Procédure *single point* - gyromètre

Le peu de moyen de tests fiables pour la calibration du gyromètre oblige à utiliser une procédure dite *single point* ou *no turn*. Cette procédure consiste à acquérir un ensemble de N mesures ω_x, ω_y et ω_z, le gyromètre étant immobile. Les moyennes des séries de mesures $< \omega_x >$, $< \omega_y >$ et $< \omega_z >$ sont donc les *offsets* du capteur, pour les 3 axes de mesure.

La prise en compte de ces *offsets* peut se faire de 2 manières :

1. de manière logicielle : en soustrayant à chaque nouvelle mesure les *offsets* calculés ;
2. de manière matérielle : en écrivant dans les registres adéquats du MPU-9250.

Lire la note d'application *AN-1057*.
Quel est l'inconvénient majeur de la méthode de calibration *single point* ?
Quel type de matériel faudrait-il utiliser pour une calibration plus efficace ?
A l'aide de la datasheet du *MPU-9250*, détailler les registres à utiliser et la mise en forme à effectuer pour indiquer les *offsets* calculés.

Mise en place et test de la calibration :

- coder la procédure de calibration `GyrCalib` qui renverra les offsets calculés sur les 3 axes. Cette fonction permettra une prise en compte matériel des *offsets* calculés. Pour cela, il vous faut lancer une procédure d'aquisition des 3 axes sur N échantillons ;
- coder la fonction `Average` permettant de calculer la moyenne de N mesures réalisées selon les 3 axes ;
- coder la fonction `Variance` permettant de calculer la variance de N mesures réalisées selon les 3 axes ;
- vérifier que les offsets moyennés ont des valeurs proches de 0 après les N mesures.

4.2 Procédure complète - magnétomètre

La calibration complète d'un magnétomètre est une procédure complexe, mais très bien documentée. Il suffit de se référer aux notes d'application fournies par les fabricants de magnétomètre : Freescale, Analog Device... Par exemple : *Calibrating an eCompass in the Presence of Hard- and Soft-Iron Interference*, AN4246, NXP.

La procédure que vous allez utiliser ci-dessous est plus « légère » qu'une procédure complète, mais elle est efficace et relativement simple à mettre en œuvre.

4.2.1 Nécessité d'une calibration

Le magnétomètre fourni dans ce TP a déjà été calibré en usine, comme d'ailleurs l'ensemble des capteurs électroniques. Afin de mettre en évidence la nécessité de calibrer le magnétomètre, dans son environnement d'utilisation, il suffit d'enregistrer les données de champ magnétique selon les axes Ox, Oy et Oz quand l'utilisateur décrit des formes de 8, dans les 3 directions de l'espace. Si le magnétomètre est parfaitement calibré, l'affichage de ces données en 3D doit former une sphère parfaite, centrée en 0 et de rayon $R = 50$, si le champ magnétique terrestre est supposé être de $50\mu T$. L'affichage des données en 2D ($B_y = f(B_x)$, $B_z = f(B_y)$ et $B_x = f(B_z)$) doit de même donner des cercles, centrés en 0 et de rayon $R = 50$.

- dans votre fichier *main.c* (*USER CODE SECTION 2*), ajouter quelques lignes de code permettant de faire l'acquisition de N mesures de B_x, B_y et B_z, avec un écart de 50ms entre chaque mesure. Ces mesures seront stockées dans un ou plusieurs tableaux, de taille adéquate ;
- via le *debugger*, procéder à l'acquisition de ces données, en décrivant des formes de 8 dans l'espace. Exporter les sous le tableur de votre choix. Il suffit pour cela de faire apparaitre les variables idoines dans l'onglet *Expressions* du *debugger* et de procéder par copier/coller (clic droit + *Copy Expressions*). La procédure peut être un peu « longue », mais il n'en existe pas de plus rapide, sauf à utiliser un logiciel tiers, comme STM Studio, par exemple ;
- dans le tableau, mettre en forme le fichier de données, **en ne gardant que les données numériques** ;
- écrire un script *Matlab* permettant l'importation et l'affichage des données. Voici quelques commandes utiles : *load*, *reshape*, *plot3*, *plot*...
- il vous reste à conclure quant à la qualité de la calibration de votre magnétomètre. Avez-vous bien obtenu des formes de cercle ? Ces cercles sont-ils centrés en 0 ? Quel est leur rayon ?

Il serait étonnant que votre magnétomètre soit bien calibré pour 3 raisons :

1. le magnétomètre AK8963C est bien calibré en usine, mais il s'agit uniquement du capteur de type MEMS. Son placement sur la carte IMU10DOF peut entrainer l'apparition de perturbations électromagnétiques : alimentation de la carte, régulateur...

2. les industriels anticipent le fait que si l'utilisateur souhaite une précision de mesure la plus juste, il va procéder à une nouvelle calibration du capteur. La calibration en usine est donc « rapide » et corrélée au coût du composant ;
3. l'environnement d'utilisation et donc les sources éventuelles de perturbation ne sont évidemment pas prévisibles.

4.2.2 Procédure de calibration

Deux types de perturbation expliquent les données de champ magnétique obtenues :

1. *hard iron biases*. Ce type de perturbation est créé par des objets source de champ magnétique : un petit aimant, une enceinte, une boucle de courant, un appareil électronique... Ces perturbations sont responsables du décalage du 0, des cercles affichés ci-dessus. Pour compenser ces perturbations il suffit donc de mesurer la valeur moyenne du champ magnétique selon les 3 directions de l'espace, en décrivant des figures de 8 et de soustraire ensuite les *offsets* obtenus à chaque mesure ;
2. *soft iron biases*. Il s'agit d'objet pouvant déformer les lignes de champ naturelles de la Terre : table en fer ou en acier, objet en nickel... Ces perturbations entrainent une déformation des cercles, car le capteur n'a plus la même sensibilité apparente dans toutes les directions de l'espace. La « vraie » correction de ces distorsions fait intervenir un recalage sur la sphère idéale est des matrices 3x3 de correction. Nous nous contenterons ici d'appliquer un coefficient correcteur sur les différentes mesures, pour tendre vers une sensibilité apparente « moyenne ».

Voici la procédure à suivre :

- créer la fonction `MagCalib(I2C_HandleTypeDef*,double*,double*)`, où le premier paramètre de type `double*` sera un tableau de 3 entrées (de type `double`), servant à enregistrer les *offsets* calculés et le second, sera un tableau de 3 entrées (de type `double`), servant à stocker les coefficents correcteurs calculés. La suite des étapes décrites sera codée dans cette fonction ;
- en décrivant des formes de 8 dans l'espace, enregistrer la valeur minimum et la valeur maximum du champ magnétique selon Ox, Oy et Oz : $B_{x,min}$, $B_{x,max}$...
- calculer les 3 *offsets* : $mag\_bias\_x = \frac{B_{x,min}+B_{x,max}}{2}$ et stocker les dans un tableau ;
- calculer les 3 rayons maximum : $R\_x = \frac{B_{x,min}-B_{x,max}}{2}$ et stocker les dans un tableau ;
- calculer le rayon moyen : $R = \frac{R\_x+R\_y+R\_z}{3}$;
- calculer les coefficients correcteurs pour les 3 axes : $mag\_scale\_x = \frac{R}{R\_x}$ et stocker les dans un tableau ;
- modifier la fonction `MagMeasure`, afin de prendre en compte ces nouveaux coefficients et de passer d'une mesure « brute » du champ magnétique à une mesure corrigée ;
- refaire les tests de la première partie (acquisition de données et affichage sous Matlab) et conclure quant à la qualité de la calibration effectuée.

LIVRABLE 3

5 Boussole électronique robuste, multi-capteurs

5.1 Boussole compensée en inclinaison

Une boussole est un dispositif (électronique ou non) indiquant la direction du nord magnétique (pas géographique ! Pour ceux que ça intéresse : `https://www.nationalgeographic.fr/sciences/le-pole-nord-magnetique-se-deplace-plus-rapidement-que-prevu`).

L'objectif de cette partie est de créer une boussole électronique, compensée en inclinaison, à l'aide du magnétomètre et de l'accéléromètre. L'ensemble de la procédure est décrite dans la note d'application suivante (disponible sous Moodle) : *Implementing a Tilt-CompensatedeCompass using Accelerometer and Magnetometer Sensors*, AN4248, NXP.

Cette partie est **volontairement** moins guidée. Il vous appartient donc de :

- comprendre la note d'application ;
- créer la fonction permettant le calcul du nord magnétique, en utilisant la compensation d'inclinaison ;
- afficher cette information sur le terminal, d'une manière claire et lisible ;
- respecter l'ensemble des contraintes imposées depuis le début de ce projet : commentaire des fonctions, nom des variables...

5.2 Boussole multi-capteurs

L'objectif de cette partie est d'améliorer la mesure d'inclinaison (*tild sensing*) utilisée dans la boussole, en fusionnant les données de l'accéléromètre et du gyromètre, à l'aide d'un filtre complémentaire.

Le principe du filtre complémentaire est de filtrer passe-bas les données de l'accéléromètre et passe-haut les données du gyromètre. Cela donne donc :

$$\boxed{\theta[n] = K(\theta[n-1] + Te\omega[n]) + (1-K)\theta_{acc}[n]}$$

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

Te est la période d'échantillonnage, τ la constante de temps choisie pour les filtres, θ_{acc} l'angle θ calculé à partir des données de l'accéléromètre par relation trigonométrique.

Voici les étapes à suivre pour dans un premier temps réaliser la mesure d'inclinaison :

- proposer une méthode de calcul pour θ_{acc} ;
- choix les paramètres Te et τ en fonction des propriétés des capteurs et de l'application ;
- mettre en place et tester une interruption se déclenchant toutes les Te s ;
- dans la routine d'interruption, calculer $\theta[n]$ et afficher le résultat sur le terminal ;
- tester votre solution et ajuster les paramètres au besoin.

Une fois la mesure d'inclinaison fonctionnelle (valeur cohérente des angles de roulis et de tangage), il vous reste à fusionner le filtre complémentaire et l'algorithme de la première partie, indiquant l'angle de lacet (déviation par rapport au nord magnétique).

LIVRABLE 4