Prise en main du capteur Grove Accelometer gyroscope LSM6DS3

Partie 1 : Diagramme de Gantt du projet

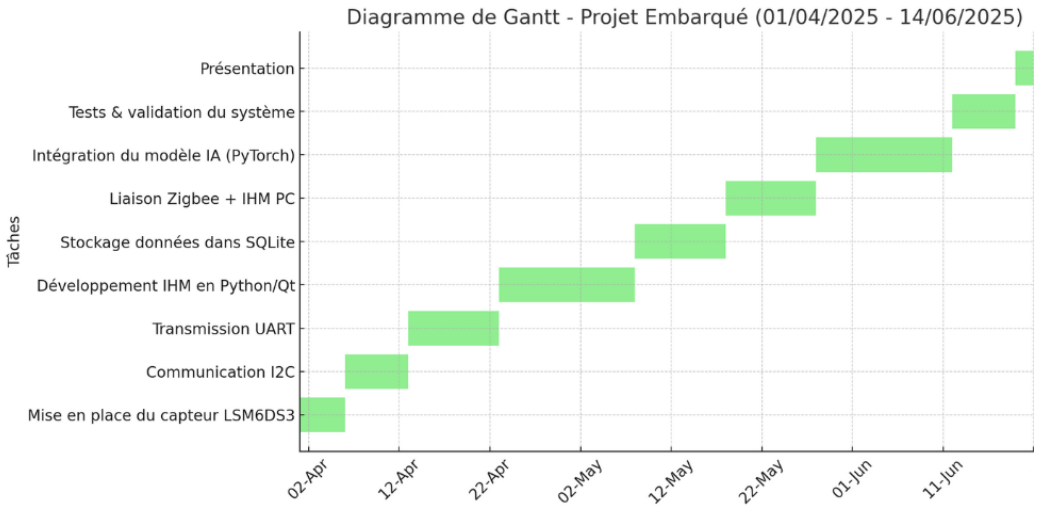

FIGURE 1 – Diagramme de Gant

Partie 2 : Prise en main du capteur LSM6DS3TR

1. Aspect physique et principe de fonctionnement

Le capteur **LSM6DS3TR** est un dispositif MEMS (Micro-Electro-Mechanical Systems) qui intègre un accéléromètre et un gyroscope sur une puce minuscule.

- **Accéléromètre** : mesure les accélérations linéaires selon 3 axes (X, Y, Z) en détectant les variations de capacité ou le déplacement d'une masse suspendue microscopique à l'intérieur du capteur.
- **Gyroscope** : mesure la vitesse angulaire (rotationnelle) sur 3 axes, généralement via l'effet Coriolis sur une structure vibrante microscopique.

Cette combinaison permet la détection précise des mouvements et des rotations dans l'espace, utile pour des applications variées telles que la navigation inertielle, la stabilisation d'image, ou encore la détection de gestes.

2. Présentation du capteur

Le **LSM6DS3TR** est un capteur MEMS 6 axes combinant :

- Un accéléromètre 3 axes ($\pm 2/\pm 4/\pm 8/\pm 16$ g),

— Un gyroscope 3 axes (±125/±245/±500/±1000/±2000 dps),

— Une sortie température intégrée.

Il communique via des interfaces **I²C** ou **SPI**, avec une consommation optimisée pour les applications *always-on*.

Applications typiques :

— Navigation inertielle

— Suivi d'activité / podomètre

— Détection de mouvement / chute

— Réalité virtuelle, IoT, smartphones

3. Connexion matérielle (hardware)

Brochage essentiel (boîtier LGA-14) :

— **VDD / VDD_IO** : 1.71 V à 3.6 V / 1.62 V (alimentation analogique / numérique)

— **GND** : masse

— **SCL / SCLK** : horloge I2C / SPI

— **SDA / SDI / SDO** : données I2C ou SPI

— **INT1 / INT2** : sorties d'interruption programmables

Des résistances de pull-up sont nécessaires pour la communication I²C.

4. Communication avec le capteur

I²C (adresse 7 bits par défaut : `0x6A`

— Lecture d'un registre : envoyer l'adresse + bit R/W = 1

— Écriture dans un registre : bit R/W = 0

5. Mise en route logicielle

Étapes de base :

1. **Vérifier l'identité du capteur**
   Lire le registre `WHO_AM_I` (`0x0F`) → doit retourner `0x69`.

2. **Configurer l'accéléromètre**
   Registre `CTRL1_XL` (`0x10`) : choix de la fréquence, bande passante et échelle.
   Exemple : `CTRL1_XL = 0x60` → 119 Hz, ±2g, filtrage par défaut.

3. **Configurer le gyroscope**
   Registre `CTRL2_G` (`0x11`) : fréquence + échelle
   Exemple : `CTRL2_G = 0x60` → 119 Hz, ±245 dps

4. **Lire les données**
   Accéléromètre : registres de `0x28` à `0x2D`
   Gyroscope : registres de `0x22` à `0x27`
   Chaque axe est codé sur 16 bits (Low + High)

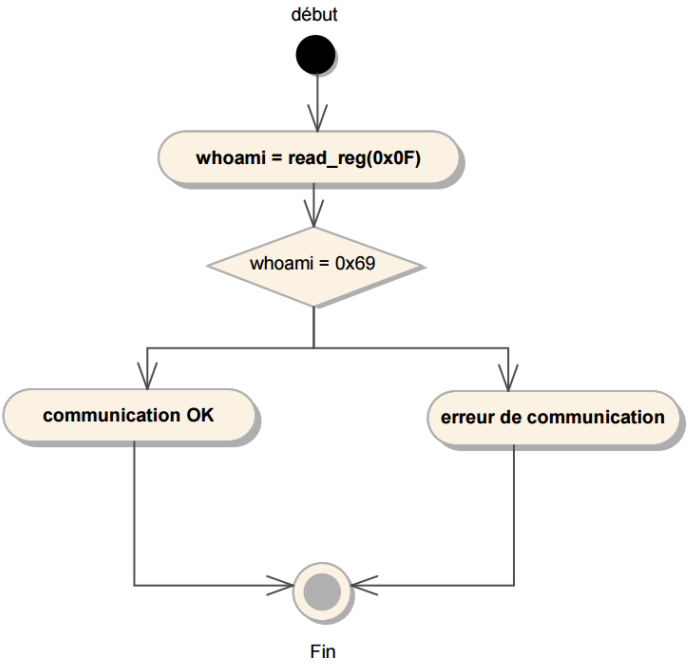

FIGURE 2 – Lecture du registre WHO_AM_I depuis l'interface I2C

6. Lecture de l'accéléromètre : explication de code

Voici une fonction C utilisant HAL (STM32Cube) pour lire les données de l'accéléromètre du LSM6DS3TR (version bloquante classique) :

Listing 1 – Lecture de l'accéléromètre

```
HAL_StatusTypeDef LSM6DS3_ReadAccelerometer(LSM6DS3_HandleTypeDef *dev,
    float *x, float *y, float *z) {
    uint8_t accel_data[6];
    HAL_StatusTypeDef ret;
    int16_t raw_x, raw_y, raw_z;
    float sensitivity = 0.061f; // mg/LSB for  2g

    // Envoyer adresse du registre de d but de lecture (OUT_X_L_XL = 0
       x28)
    ret = HAL_I2C_Mem_Read(dev->hi2c, LSM6DS3_I2C_ADDRESS, 0x28,
                           I2C_MEMADD_SIZE_8BIT, accel_data, 6,
                              HAL_MAX_DELAY);
    if (ret != HAL_OK) return ret;

    raw_x = (int16_t)((accel_data[1] << 8) | accel_data[0]);
    raw_y = (int16_t)((accel_data[3] << 8) | accel_data[2]);
    raw_z = (int16_t)((accel_data[5] << 8) | accel_data[4]);

    *x = (raw_x * sensitivity) / 1000.0f * 9.81f;
    *y = (raw_y * sensitivity) / 1000.0f * 9.81f;
    *z = (raw_z * sensitivity) / 1000.0f * 9.81f;

    *x = *x / 10;
    *y = *y / 10;
    *z = *z / 10;
```

```
    return HAL_OK;
}
```

Explication étape par étape :

- **accel_data[6]** : tableau recevant les 6 octets des données brutes (X, Y, Z sur 16 bits chacun).
- **HAL_I2C_Mem_Read(...)** : lit directement 6 octets à partir de l'adresse du registre `0x28`.
- **Conversion des données brutes** :
  - Les données sont lues en format little-endian : LSB d'abord.
  - On combine les 2 octets pour chaque axe en un entier signé 16 bits.
- **Sensibilité** : `0.061 mg/LSB` (valeur typique pour ±2g).
- **Conversion** :
  - Le résultat est multiplié par 9.81 pour passer de g à $\mathrm{m/s^2}$.
  - Ensuite, divisé par 10 (correction empirique ou unité spécifique).
- **Retour** : `HAL_OK` si tout s'est bien passé.

Remarque : cette fonction utilise un appel bloquant classique. Pour l'utilisation en interruption (IT), la logique est différente.

7. Prise en main du projet

Cette section explique les étapes à suivre pour connecter la carte, configurer les ports et lancer l'IHM de supervision.

1. Connexion du matériel

1. Connecter la carte embarquée (STM32, ESP32, etc.) au PC via USB.
2. Connecter également le module **Zigbee** via son port USB (dongle USB ou UART-USB).
3. Identifier les ports série utilisés (par exemple `COM3`, `COM4` sous Windows, ou `/dev/ttyUSB0` sous Linux).

2. Configuration du port série dans l'IHM

- Ouvrir le fichier de configuration ou le script Python de l'IHM.
- Modifier la variable ou paramètre correspondant au port série (par exemple : `port = 'COM4'` ou `'/dev/ttyUSB0'`).
- Vérifier que le port sélectionné correspond au module Zigbee.

3. Installation des dépendances Python

1. Ouvrir un terminal ou invite de commande.
2. Se placer dans le dossier contenant le code Python et le fichier `requirements.txt`.
3. Lancer la commande suivante :

```
pip install -r requirements.txt
```

4. Attendre la fin de l'installation des bibliothèques nécessaires (ex : `pyserial`, `tkinter`, `matplotlib`, etc.).

4. Lancement de l'IHM

— Une fois les dépendances installées, exécuter le script principal :

```
python main.py
```

— L'interface graphique doit s'ouvrir, permettant la visualisation des données transmises via Zigbee.

— S'assurer que les ports série ne sont pas déjà utilisés par un autre logiciel (comme un terminal série).

Conseil : utiliser un environnement virtuel Python pour isoler les dépendances :

```
python -m venv venv
source venv/bin/activate  # Linux/macOS
venv\Scripts\activate.bat # Windows
```