


PROJET : BTS CIEL

Dossier Individuel pour le Projet Ballon Stratosphérique (TW5)

sommaire :

1. Analyse détaillée	2
1.1 Responsabilité technique dans le système embarqué	2
1.2 Description fonctionnelle du programme de traitement	4
1.3 Maquette ou prototype de l'interface utilisateur (IHM)	5
2. Ingénierie du système embarqué	5
2.1 Langage de programmation : C++	5
2.2 Bibliothèques logicielles utilisées 	6
2.3 Outils de test et de validation	7
• Affichage console (via terminal) :	7
• LED témoin sur le MPU6050 :	7
• Vérification mécanique :	8
2.4 Pertinence des choix technologiques	8
2.5 Modélisation UML (avec technologie) :	9
• Rôles principaux du MPU6050 :	12
• Attributs notables du MPU6050 :	12
• Communication du MPU6050 :	13
• Rôles principaux GestionFile :	13
• Particularités GestionFile :	13
• Rôles principaux I2C :	14
• Interactions I2C :	14
3. Évaluation fonctionnelle du dispositif	14
3.1 Test unitaire du MPU6050	14
3.2 Résultats des Tests et Observations	17
3.3 Problèmes rencontrés et ajustements nécessaires.....	17
4. Bilan et perspectives	17
4.1 Bilan sur la planification personnelle	17
4.2 Retards ou réajustements	18
4.3 Apport personnel du projet	18
• Compétences acquises :	18
• Difficultés rencontrées :	19
• Solutions mises en place :	19
• Améliorations possibles du projet ballon stratosphérique :	19
Annexes :	20

1. Analyse détaillée .

1.1 Responsabilité technique dans le système embarqué .

Ce projet s'inscrit dans le cadre d'un vol en ballon stratosphérique destiné à **récolter des données environnementales** et à détecter des événements critiques pendant la descente. La nacelle embarque plusieurs composants électroniques, dont **le capteur MPU6050**, qui constitue **le cœur de la partie que j'ai développée**.

L'objectif principal de cette partie est de permettre la **détection automatique de deux événements majeurs** :

- **La chute libre** du ballon (éclatement), détectée via une **accélération quasi nulle** → *FreeFall*.
- **L'atterrissage** de la nacelle, détecté par une **absence prolongée de mouvement** → *ZeroMotion*.

Une fois ces événements détectés, un **signal radio est envoyé via un module LoRa RA-02 (433 MHz)** pour permettre aux opérateurs de localiser rapidement la nacelle. Ce choix de fréquence est compatible avec l'usage radio amateur en Europe.



Sachant que les composants et leur fonction sont présentés ci-dessous en format tableau pour mieux visualiser ce projet et de mettre un accent sur l'essentiel de ma partie personnelle qui sera plus approfondie dans les recherches .

Composant	caractéristique	choix
BME280	Mesure température , humidité et pression ,i2c .	3 capteurs en 1, prix abordable .
Pi cam	Connexion via port CSI Capture photos	Parfait pour faire de la vision photo
Raspberry Pi 0	Microcontrôleur ARM monocœur 1 GHz . Très compact, faible consommation.	Ultra léger, bon marché, parfait pour des projets IoT , portables ou avec espace limité
LoRa RA-02		
MPU6050	3 axes accéléromètre (3 axes gyroscope) I ² C (adresse 0x69) Plages de mesure : Accéléro: ±2g, ±4g, ±8g, ±16g Tension : 3.3V	

1.2 Description fonctionnelle du programme de traitement .

Le système repose sur un **programme C++ embarqué**, dont la fonction principale est de surveiller en temps réel les données d'accélération fournies par un capteur **MPU6050**. Ce capteur, connecté via une interface **I2C**, permet de mesurer les accélérations sur les trois axes (X, Y, Z) à ultra haute fréquence.

Dès son lancement, le programme initialise le capteur MPU6050, configure les paramètres de communication I2C et **calibre automatiquement les axes d'accélération** pour corriger les biais initiaux. Cette calibration permet d'obtenir des mesures plus fiables, nécessaires pour une détection précise des événements.

Le cœur du système repose ensuite sur l'**activation de deux interruptions matérielles** intégrées au capteur :

- **Free Fall (chute libre)** : cette interruption est déclenchée lorsqu'une **variation brutale** ou une **chute significative de l'accélération** est détectée sur au moins un des axes. Elle peut correspondre à une situation de chute réelle, comme celle rencontrée lors de l'**éclatement d'un ballon stratosphérique** ou d'un largage en altitude.
- **Zero Motion (immobilité prolongée)** : cette interruption se déclenche lorsque les mesures d'accélération deviennent **quasi nulles sur une durée continue prédéfinie**. Cela indique que le dispositif est resté stable sans mouvement, ce qui peut être interprété comme un **atterrissage ou une stabilisation après un impact**.

Lorsque l'une de ces interruptions est déclenchée, le programme exécute automatiquement une **fonction de rappel (callback)** spécifique. Ces fonctions convertissent l'événement physique détecté en un **message texte explicite**, tel que *"Free fall detected"* ou *"Zero Motion detected"*. Ces messages sont ensuite transmis au système principal via un **mécanisme d'IPC (communication inter-processus)**, en l'occurrence une **file de messages système**. Cette file permet une **communication fiable et asynchrone** avec d'autres modules logiciels, notamment pour l'envoi radio via LoRa.

L'algorithme mis en œuvre vise donc à **interpréter les données brutes issues du capteur en événements concrets et exploitables**, dans le contexte d'un système embarqué autonome :

- Une **chute libre** est interprétée comme un **éclatement** ou une **descente rapide**.
- Une **immobilité prolongée** est interprétée comme un **atterrissage** ou une **stabilisation** au sol.

Ce traitement permet une prise de décision immédiate, sans intervention humaine, et une **réactivité optimale dans des environnements contraints** en ressources (énergie, calcul, communication).

En résumé, ce système constitue une **surveillance intelligente des mouvements** embarquée dans un microcontrôleur. Il est capable de détecter de manière autonome des événements critiques dans des scénarios comme le suivi de ballon-sonde, les missions environnementales, ou les tests de dispositifs déployés en altitude. Il représente ainsi une **brique essentielle d'un système embarqué réactif et communicant**, s'inscrivant pleinement dans le cadre des technologies IoT et des capteurs intelligents.

1.3 Maquette ou prototype de l'interface utilisateur (IHM) .

Le projet étant spécifiquement conçu pour un système embarqué autonome à ressources limitées, aucune interface graphique utilisateur (GUI) n'est intégrée dans la version finale du dispositif. En effet, l'objectif principal du système est de fonctionner de manière totalement indépendante, sans intervention humaine directe ni affichage visuel sophistiqué embarqué.

Cependant, dans le cadre de la phase de développement, de test et de validation du système, une **interface console textuelle** a été mise en place. Cette interface, accessible via le terminal du système hôte (par exemple une Raspberry Pi ou un PC de développement), permet un **suivi en temps réel** de l'activité du capteur et de l'algorithme de détection des événements.

Plus précisément, cette interface affiche les messages suivants:

- L'état de détection actuel , sous forme de messages lisibles par un opérateur :
 - “ FreeFall detected ” → interprété comme une **chute libre**, pouvant correspondre à l'éclatement d'un ballon stratosphérique ou à une perte de sustentation.
 - “ZeroMotion detected” → interprété comme une **immobilité prolongée**, potentiellement causée par un **atterrissage** ou un impact au sol.

2. Ingénierie du système embarqué .

2.1 Langage de programmation : C++ .

Le langage **C++** a été retenu pour le développement du programme embarqué en raison de ses performances en temps réel, de son faible niveau d'abstraction (proche du matériel), et de sa richesse en bibliothèques orientées objets. C++ permet une maîtrise fine des ressources système tout en conservant une architecture modulaire et évolutive. Il est particulièrement adapté aux systèmes embarqués nécessitant une gestion précise du matériel, comme la communication I2C, la gestion des interruptions, ou encore l'accès bas niveau aux périphériques.

Avantages	Avantages	Inconvénient
MPU6050	✓ Intègre accéléromètre et gyroscope sur une seule puce (6	✗ Pas de capteur de pression intégré pour des mesures d'altitude .

Avantages	Avantages	Inconvénient
	<p>axes) .</p> <ul style="list-style-type: none"> ✓ Communication simple via bus I2C . ✓ Large documentation et support communautaire ✓ Coût très faible, facile à se procurer 	<ul style="list-style-type: none"> ✗ Sensible au bruit électrique (nécessite des précautions de câblage) . ✗ Nécessite calibration soignée pour éviter les faux positifs . ✗ Précision limitée comparée à des capteurs professionnels haut de gamme . ✗ Pas de filtrage matériel intégré, il faut gérer les dérives en logiciel .
Rasoberry PI 0	<ul style="list-style-type: none"> ✓ Format compact, idéal pour systèmes embarqués (ballon, drone, etc.) . ✓ Interface I2C intégrée, facilitant la communication avec les capteurs . ✓ Large compatibilité logicielle (Linux, bibliothèques C++) . ✓ Faible consommation électrique, adaptée aux systèmes alimentés sur batterie . ✓ Suffisante pour exécuter un programme C++ de détection en temps réel . 	<ul style="list-style-type: none"> ✗ Performances limitées pour des traitements lourds ou multitâches complexes . ✗ Absence de connecteur GPIO soudé d'origine, nécessite des soudures pour certains projets . ✗ Moins de ports disponibles (USB, GPIO) comparé à d'autres modèles Raspberry Pi . ✗ Démarrage plus lent qu'un microcontrôleur nu (ex :STM32, Arduino). ✗ Pas de système temps réel natif (nécessite Raspbian ou OS optimisé)

2.2 Bibliothèques logicielles utilisées

Le programme est développé en **C++** en environnement **Linux**, ce qui permet un contrôle fin des performances et une compatibilité directe avec les interfaces bas niveau de la Raspberry Pi. Les bibliothèques utilisées sont :

Bibliothèque utiliser	Rôle	Avantage
<chrono>	Gestion du temps (temporisation, délais, durées)	Précision élevée pour mesurer ou attendre des durées sans surcharge système.
<thread>	Mise en attente de l'exécution (temporisation via sleep_for)	Permet de gérer des temporisations non bloquantes, utile après détection d'événements (ex. : délai après une interruption).

Bibliothèque utiliser	Rôle	Avantage
<iostream>, <iomanip>	Affichage formaté sur la console	Affichage clair et formaté sur la console pour faciliter les tests et le débogage.
<stdexcept>	Gestion des erreurs et des exceptions	Gestion propre des erreurs avec des messages explicites en cas de défaillance (I2C, capteur, etc.).
MPU6050	Classe personnalisée pour la gestion du capteur MPU6050	Abstraction du matériel, simplifie la configuration, la calibration, et la gestion des interruptions du capteur.
GestionFile	Classe personnalisée pour la communication via file IPC	Interface simple avec les files IPC Linux pour communiquer les événements détectés à d'autres processus (ex : transmission LoRa).

2.3 Outils de test et de validation .

Le projet, étant embarqué, nécessite des méthodes simples mais efficaces pour valider son bon fonctionnement à chaque étape. Les outils utilisés sont principalement matériels et logiciels, adaptés à un environnement à ressources limitées.

- **Affichage console (via terminal) :**
 - Permet d'afficher en temps réel les informations essentielles pendant l'exécution :
 - Offsets mesurés à la calibration.
 - États détectés par les interruptions (Free Fall, Zero Motion).
 - Trames préparées pour la transmission.
 - Utilisé comme interface de débogage lors des phases de développement et de test.
- **LED témoin sur le MPU6050 :**
 - Le capteur MPU6050 dispose généralement d'une LED qui s'allume lorsque le module est alimenté.
 - Vérifier visuellement si :
 - Le capteur est bien alimenté.
 - La connexion physique est fiable (module bien enfiché, pas de faux contact).

- **Vérification mécanique :**

- Un simple appui sur le module MPU6050 permet de tester :
 - La stabilité de la connexion physique.
 - Les éventuelles déconnexions ou faux contacts sur les broches.

2.4 Pertinence des choix technologiques .

Élément	Choix	Avantages	Inconvénients/Limites
Carte principale	Raspberry Pi Zero	- Très compacte, idéale pour les systèmes embarqués - Accès aux interfaces GPIO/I2C - Système Linux complet - Compatible C++	- Moins puissante qu'une Pi 4 - Moins de ports disponibles
Rôle de la Pi Zero	Transmission entre MPU6050 et LoRa RA-02	- Sert uniquement de passerelle (lecture capteur & envoi via IPC) - Évite une complexité réseau directe	- Dépendance à un module externe pour la communication LoRa
Capteur	MPU6050 (accéléromètre 3 axes)	- Très répandu et bien documenté - Gestion native des interruptions (Free Fall, Zero Motion) - Précision correcte pour le projet	- Données parfois bruitées sans filtrage - Gyroscope inutilisé (surcapacité non exploitée dans ce projet)
Langage	C++	Performant pour l'embarqué - Maîtrise fine de la mémoire et des interruptions - Compatible Linux/Pi	- Nécessite rigueur de gestion mémoire
Communication	I2C		- Distance limitée - Sensible aux interférences

Élément	Choix	Avantages	Inconvénients/Limites
		<ul style="list-style-type: none"> - Interface simple à deux fils - Bien supportée sur Raspberry Pi - Faible consommation 	électriques
Bibliothèques C++	chrono, thread, iostream, stdexcept	<ul style="list-style-type: none"> - chrono pour la gestion du temps sans blocage - thread pour la pause en callback - iostream pour l'affichage - Fiabilité 	- Plus verbeux que d'autres langages comme Python
Affichage & Tests	Console texte + LED MPU6050	<ul style="list-style-type: none"> - Affichage direct des événements en console - LED visible sur capteur utile pour vérifier le fonctionnement (ex: chute ou stabilisation) 	<ul style="list-style-type: none"> - Pas d'interface graphique utilisateur - Difficulté de visualisation en conditions de vol ou embarqué réel

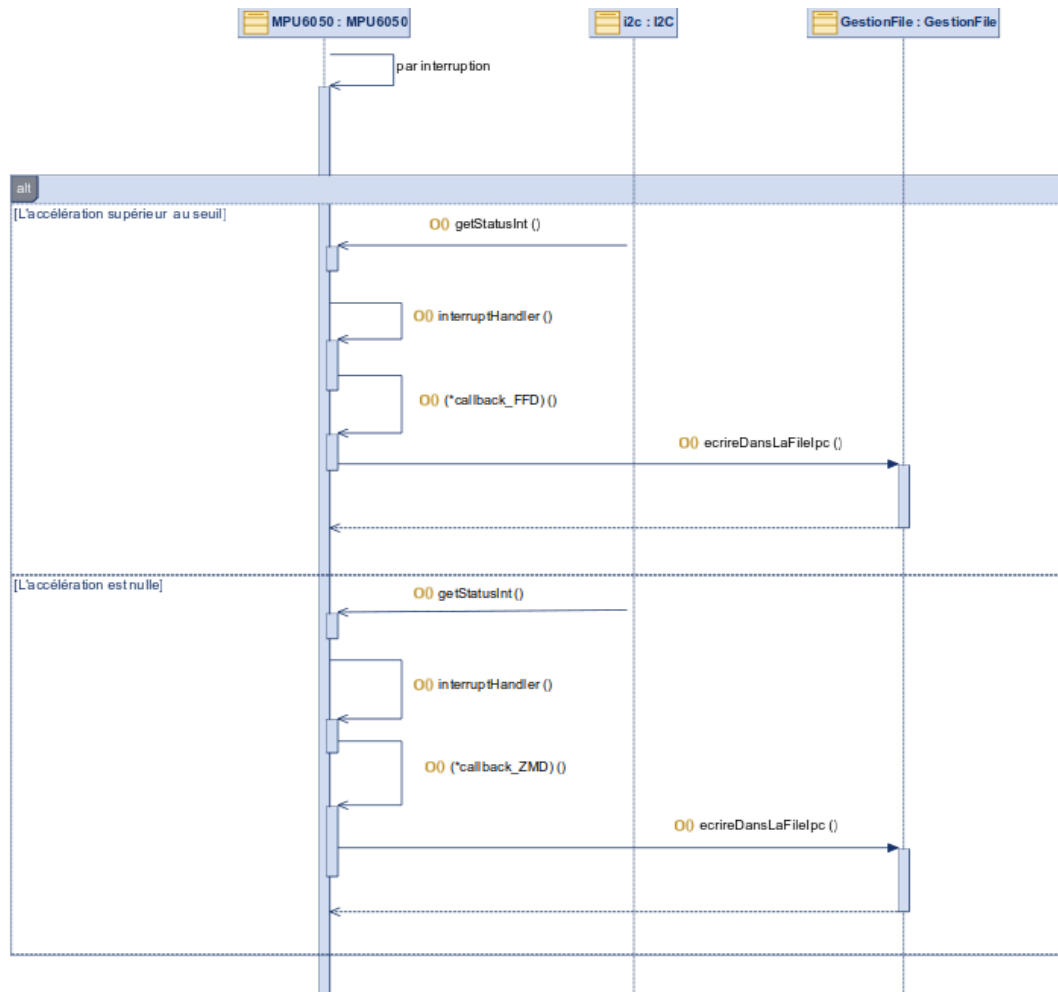
- Le **MPU6050** est précis et très répandu, avec une documentation et des exemples nombreux.
- **Raspberry Pi** offre une plateforme flexible avec accès GPIO, I2C, et réseau.

• **Composants partagés utilisés :**

- Pas de base de données , mais les données peuvent être **journalisées dans des fichiers .csv**.
- Pas d'API externe, mais une version réseau pourrait utiliser MQTT pour signaler les événements.

2.5 Modélisation UML (avec technologie) :

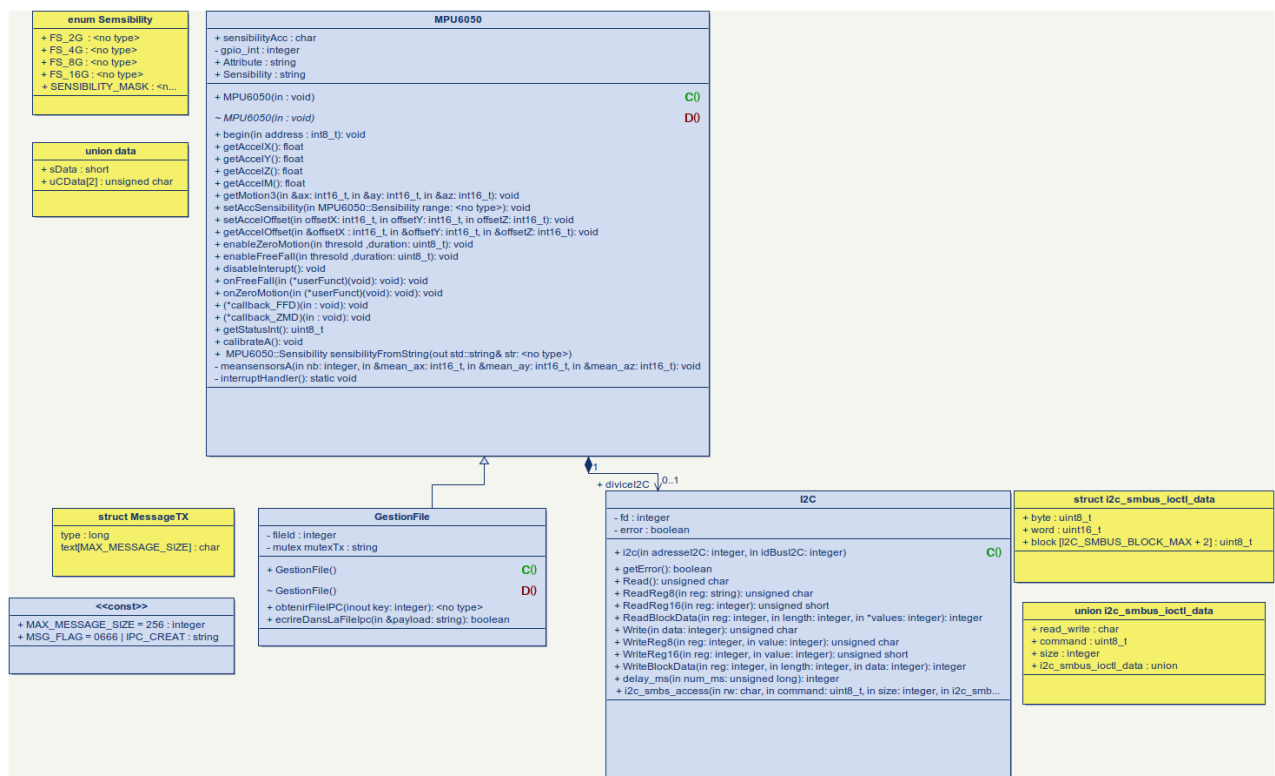
Ci-dessous représente d'un diagramme de séquence :



Ce diagramme de séquence illustre le comportement du système lorsqu'une interruption est générée par le capteur **MPU6050** via le bus **I2C**. Deux types d'événements sont gérés : le Free Fall Detection – FFD et le Zero Motion Detection – ZMD qui réagissent en temps réel à ces événements.

- **Dans le cas d'une chute libre**, une interruption est d'abord captée par le MPU6050. Le programme interroge alors le capteur via la méthode **getStatusInt()** pour vérifier la source de l'interruption. Ensuite, le gestionnaire d'interruptions **interruptHandler()** est appelé et exécute la fonction de rappel **callback_FFD()**, propre à la détection de chute. Cette fonction écrit ensuite un message dans une file IPC à l'aide de **ecrireDansLaFileIpc()**, afin que l'information soit transmise à un autre processus, par exemple pour être envoyée via LoRa ou enregistrée.
- **Dans le cas d'un zéro mouvement**, le même mécanisme d'interruption se déclenche. Le programme lit le statut de l'interruption avec **getStatusInt()**, puis passe par **interruptHandler()**. Cette fois, c'est la fonction **callback_ZMD()** (Zero Motion Detection) qui est appelée. Elle envoie également un message via la file IPC (**ecrireDansLaFileIpc()**), indiquant que le capteur détecte une absence totale de mouvement, utile par exemple pour signaler une stabilisation ou un atterrissage.

Ci-dessous ça représente le diagramme des class les 3 autres sert à voir se qui il y a de dans les class :



Le diagramme de class est présenté illustre l'architecture globale d'un système embarqué basé sur un capteur inertiel **MPU6050**.

Ce dernier est au cœur du dispositif et permet de détecter des événements physiques tels que la chute libre ou l'immobilité prolongée.

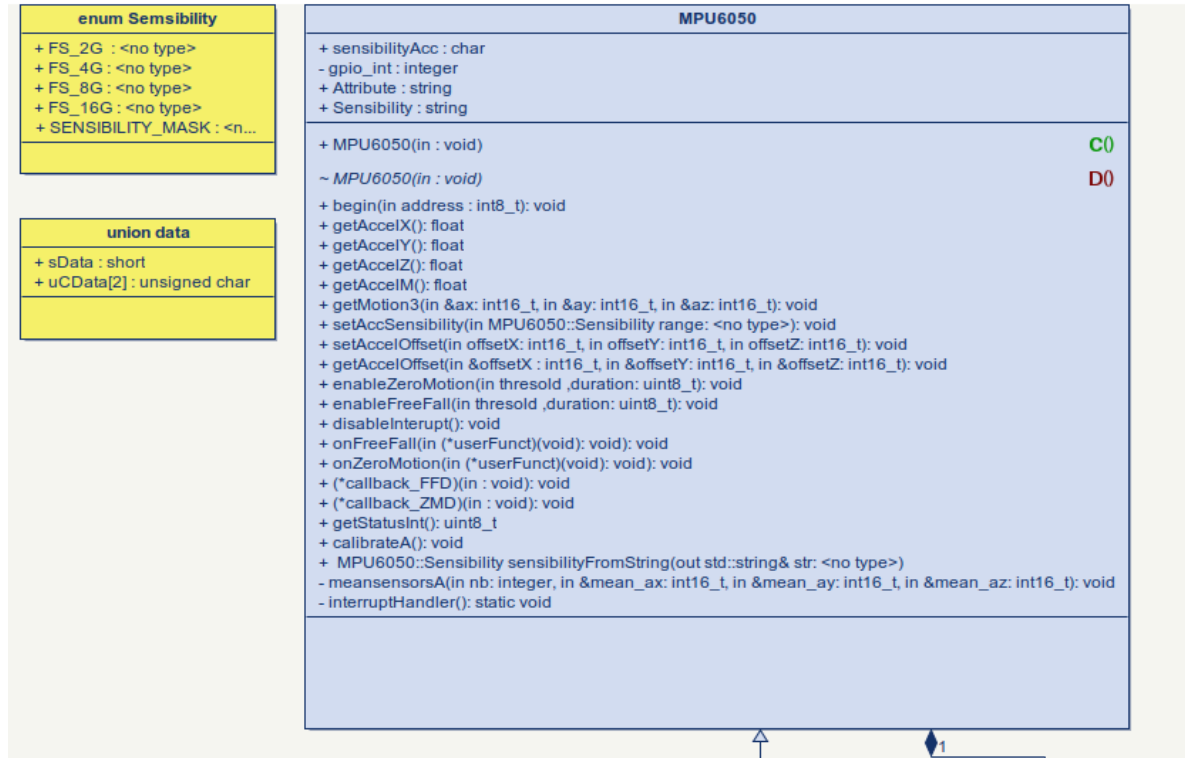
Pour fonctionner, le MPU6050 communique via le protocole **I2C**, encapsulé dans une classe dédiée nommée **I2C**.

Cette classe assure la lecture et l'écriture des registres du capteur à l'aide d'un descripteur de fichier et de fonctions bas niveau.

Une autre composante essentielle est la classe **GestionFile**, qui permet de transmettre les événements détectés par le capteur à un autre processus à l'aide d'un système de **file de messages IPC**.

Le diagramme met également en évidence des structures auxiliaires comme **MessageTX** pour le formatage des messages, et **i2c_smbus_ioctl_data** pour la configuration fine des échanges I2C à travers les appels système Linux. Les relations entre ces éléments sont clairement définies : la classe **MPU6050** dépend de la classe **I2C** pour ses communications matérielles et utilise **GestionFile** pour transmettre les données traitées.

Ce découpage modulaire permet une meilleure organisation du code, une maintenance facilitée, et une séparation nette entre les couches matérielles, logicielles et de communication.

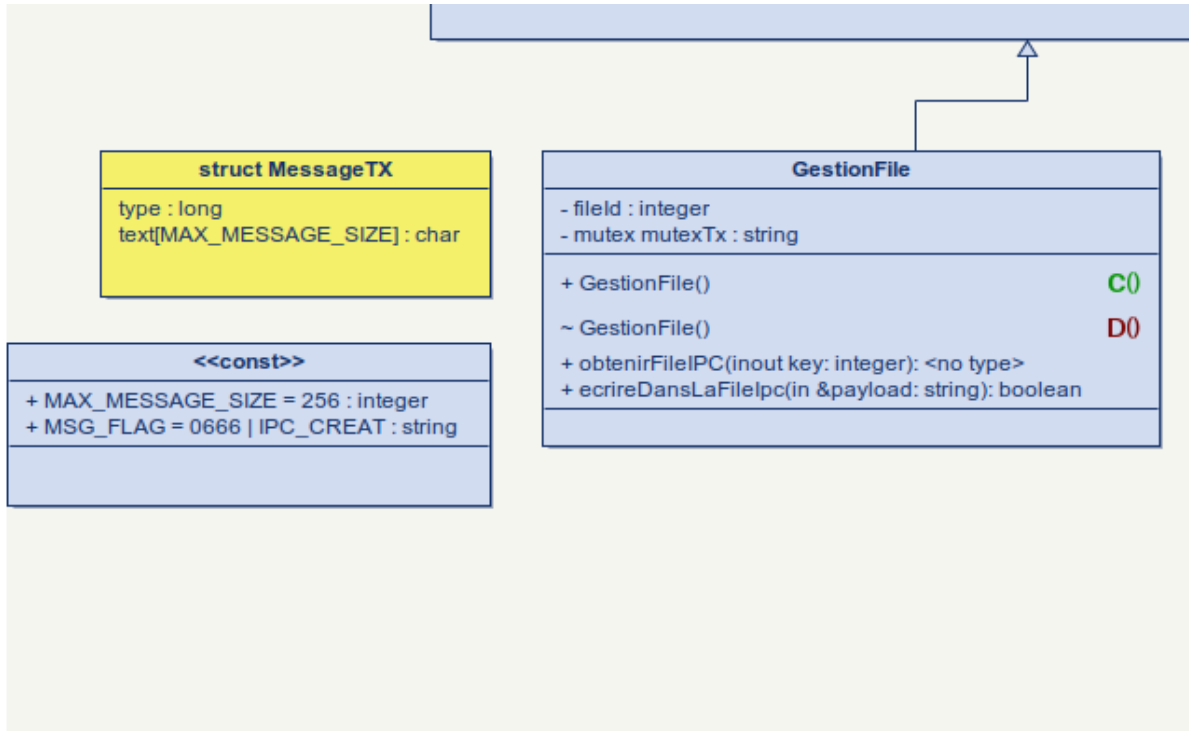


La classe MPU6050 représente la **gestion logicielle du capteur inertiel**. Elle encapsule toutes les fonctionnalités liées à l'accéléromètre intégré dans le MPU6050.

- **Rôles principaux du MPU6050 :**
 - **Initialisation et calibration** du capteur (MPU6050(), calibrateA() , begin()),
 - Accès aux données d'accélération (getAccelX() , getAccelY() , getAccelZ()),
 - **Configuration de sensibilité** et d'offsets (setAccelSensitivity() , setAccelOffset),
 - Activation des **interruptions intelligentes** :
 - enableZeroMotion() : détection d'immobilité prolongée,
 - enableFreeFall() : détection de chute libre,
 - onFreeFall() / onZeroMotion() : pour attacher des callbacks utilisateurs.
- **Attributs notables du MPU6050 :**
 - **sensibilityAcc** , **gpio_int** : paramètres de configuration ,
 - **Sensibility** , **Attribute** : chaînes de configuration ,
 - **interruptHandler()** : méthode interne gérant les interruptions.

- **Communication du MPU6050 :**

- Utilise un objet **I2C** (association vers la classe **I2C**) pour accéder physiquement au capteur sur le bus I2C,
- Transmet les données interprétées à la classe **GestionFile** sous forme de messages.



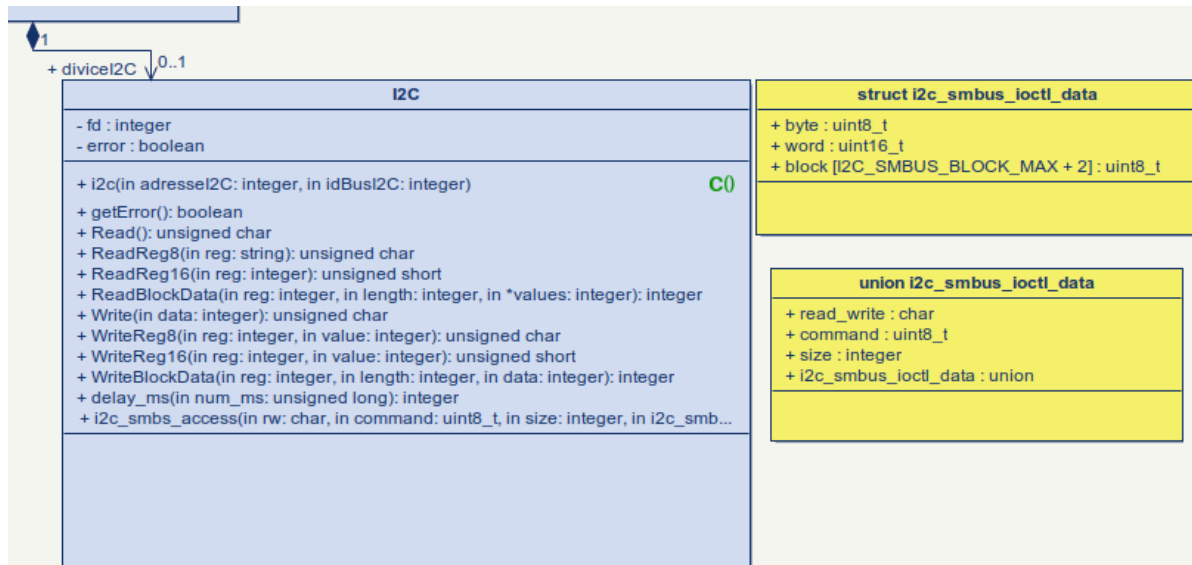
La classe GestionFile est responsable de la **communication inter-processus (IPC)** à l'aide de **files de messages POSIX/SYS V**. Elle fait le lien entre le traitement local des données et leur transmission éventuelle (par exemple via LoRa).

- **Rôles principaux GestionFile :**

- Écriture et lecture dans une file de messages : **ecrireDansLaFileIpc()** et **obtenirFileIpc()**.
- Encapsulation de la structure **MessageTX** qui contient :
 - Un champ `type` (entier long).
 - Un champ `text` pour stocker le message à transmettre.

- **Particularités GestionFile :**

- Attributs internes de gestion : **fileId**, **mutexMtxTX**,
- Utilisée par **MPU6050** pour **envoyer les événements détectés** ("Free Fall", "Zero Motion") vers d'autres composants logiciels ou physiques.



La classe I2C est une classe bas niveau assurant la communication matérielle avec le capteur **MPU6050** par le **protocole I2C**.

- **Rôles principaux I2C :**

- Ouverture et configuration du bus I2C (**i2cInit()**),
- Opérations de lecture/écriture de registres :
 - Lecture : **ReadReg8()**, **ReadBlockData()**, etc.
 - Écriture : **WriteReg8()**, **WriteBlockData()**, etc.
- Gestion des échanges de données binaires entre le processeur (Raspberry Pi) et le capteur.

- **Interactions I2C :**

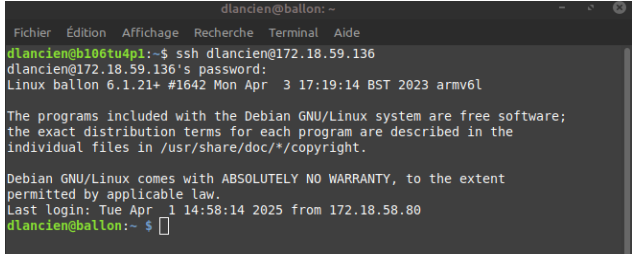
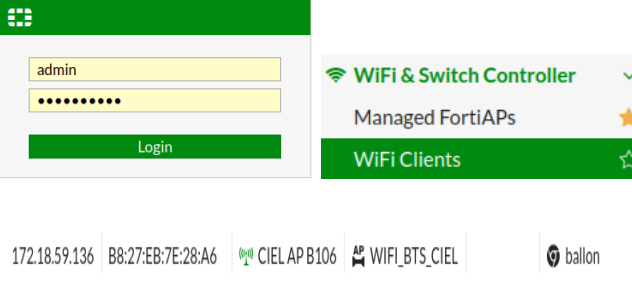
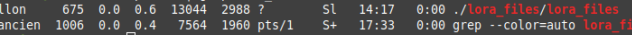
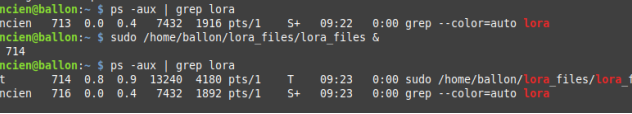
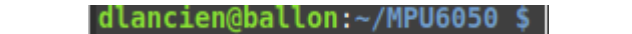
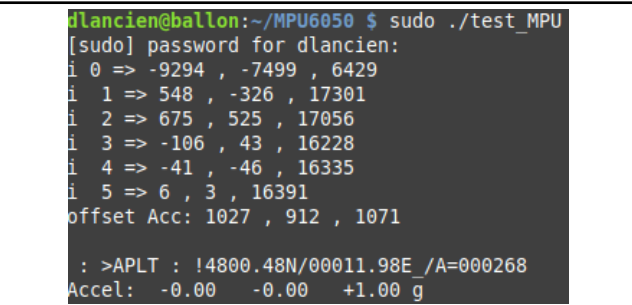

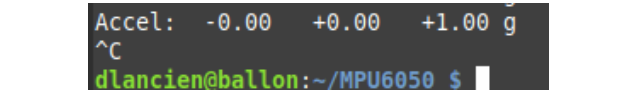
- Fournit une **interface I2C** fiable à la classe **MPU6050**, qui l'utilise pour accéder aux registres du capteur,
- Gère les erreurs de communication via l'attribut **error**.

3. Évaluation fonctionnelle du dispositif .

3.1 Test unitaire du MPU6050 .

<h2 style="text-align: center;">Fiche des tests unitaires du MPU6050.</h2>
--

Nature :	<p>Test de <code>mpu.getAccelX()</code>, <code>mpu.getAccelY()</code>, <code>mpu.getAccelZ()</code></p> <p>Vérifier que les valeurs retournées sont dans une plage attendue (ex. : entre -4g et +4g si <code>FS_4G</code> est utilisé).</p>	Référence :	TU - Et4 - 001
	<p>Test de <code>rappel_FF()</code> et <code>rappel_ZM()</code></p> <p>Vérifier que les interruptions de détection fonctionnent et envoient bien un message dans la file IPC.</p>		
	<p>Test de <code>mpu.calibrateA()</code></p> <p>Vérifier que les offsets sont bien mis à jour après calibration.</p>		
	<p>Test de <code>gestionFile.ecrireDansLaFileIPC()</code></p> <p>Vérifier que le message est bien écrit dans la file IPC.</p>		
Module :	MPU6050	Auteur :	Lancien Dorian
Date de création / mise à jour :		jeudi 6 mars / mardi 25 mars	
Objectif :	<p>Vérifier que les seuils de détection de chute libre et d'atterrissage sont bien calculés. S'assurer que le programme envoie la trame LoRa uniquement dans les conditions attendues.</p> <p>Tester les conditions aux limites (valeurs de l'accélération).</p>		
Condition du test			
État initial du module :		Environnement du test :	
Ordinateur	Raspberry PI 0	sur Linux Debian	
Programme	Netbeans / Qt creator	IDE 22 / 12.0.1 : C++	
Procédure du test			
Lancement de L'application :		vérifier que le branchement du chargeur soit mis correctement sur la nacelle car il s'effectue sur PWR . si c'est pas le cas et qu'elle est mise sur le clavier ça fonctionnera .	
Repère	Opérations	Résultats attendus	

1	aller mettre des commandes sur le terminal. <code>ssh dlancien@172.18.59.136</code> mot de passe est Dodo	
2	Si il se trouve pas 172.18.59.136 alors faut vérifier dans fortigate du lycée bts ciel pour le ballon, il faut mettre cet adresse sur url du navigateur est mettre 172.18.58.30. Le login c'est admin et le mot de passe touchard72 après aller sur WiFi & Switch Controller sur WiFi Clients et regarder l'adresse IP du ballon. donc remplacer le IP.	
3	faire des commandes ps a ou ps aux sachant que grep cherche ce qu'on souhaite ex: lora_files ps aux grep lora_files	
4	lancer manuellement lora en mettant la commande <code>sudo /home/ballon/lora_files/lora_files &</code>	
5	être dans le dossier MPU6050 avec la commande cd et après cd MPU6050.	
6	pour lancer le test unitaire faire la commande <code>sudo ./test_MPU</code> . avec le mot de passe Dodo .	
7	pour voir si la trame est bien envoyée, aller sur aprs.fi sur navigateur et se connecter . sur le site aprs.fi il faut aller voir dans la légende Autres vues: cliquez sur trames packet et sur Indicatif générateur: à côté faut mettre F4KMN-8 pour avoir que se de la nacelle en cliquant sur rechercher.	
8	pour arrêter le programme aller sur le terminal ou vous avez lancer le MPU est faire un <code>ctrl + c</code> pour arrêter le programme MPU .	

3.2 Résultats des Tests et Observations .

Les tests effectués sur banc ont permis de confirmer que la détection fonctionne correctement **à condition que le capteur MPU6050 soit correctement calibré.**

- Il est essentiel d'ajuster précisément le **seuil d'accélération** pour qu'il corresponde au **comportement réel du montage.**
- Une stabilité de **1 seconde** continue est requise pour valider une absence de mouvement (fonction de Zero Motion), ce qui permet de filtrer certains parasites de mesure.

3.3 Problèmes rencontrés et ajustements nécessaires

- **Faux contact** constaté au niveau du câblage, pouvant provoquer un fonctionnement erratique. Après vérification et sécurisation des connexions, le problème a été résolu.
- **Erreur** dans le **contenu du message** envoyé : une mauvaise rédaction ou un format incorrect a conduit à un message invalide. Une fois corrigée, la communication s'est faite correctement.
- Lors de **l'assemblage final des modules du programme**, un **warning** est apparu à la compilation. Il était dû à l'utilisation du même nom de macro **CONFIG** dans deux parties du code, ce qui provoque un conflit. Le problème a été résolu en renommant la macro spécifique au capteur en **CONFIG_MPU**, évitant ainsi toute ambiguïté lors de la compilation.

4. Bilan et perspectives .

4.1 Bilan sur la planification personnelle .

L'objectif principal du projet — à savoir la **détecter une chute et d'atterrissage** — a été atteint avec succès dans les délais impartis.

Les tests en conditions simulées ont confirmé le bon fonctionnement du système, sous réserve d'un **calibrage rigoureux** du capteur.

Le planning initial a globalement été respecté, malgré quelques ajustements nécessaires en cours de développement, notamment lors de **l'intégration des différentes parties du code** et la **résolution de conflits liés aux définitions de macros.**

Des **fonctionnalités complémentaires** sont envisagées pour la suite, telles que :

- L'enregistrement des événements dans un fichier ou une base de données,
- L'ajout d'une alerte sonore ou lumineuse,

- L'intégration dans un système de communication embarqué (ex. : LoRa, GSM).

Ces évolutions permettront de rendre le dispositif plus complet et mieux adapté à une utilisation en conditions réelles.

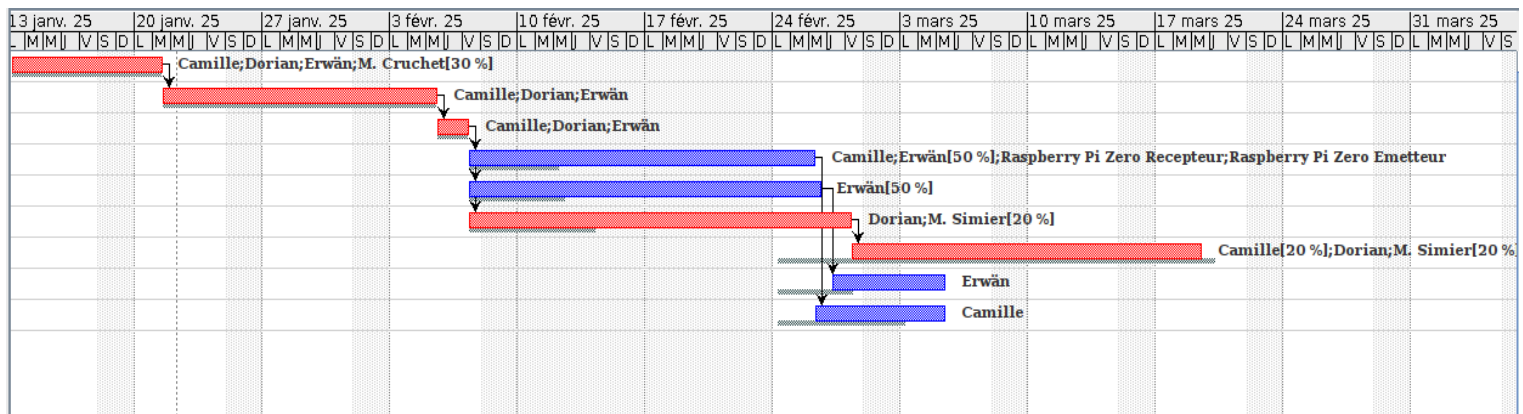
4.2 Retards ou réajustements .

Une période d'absence a nécessité un réajustement de l'organisation. Malgré cela, j'ai poursuivi le travail en autonomie en réalisant des **recherches personnelles à distance** et en **communiquant régulièrement avec mon collègue Camille**, ce qui a permis de **maintenir les délais et d'assurer le suivi du projet**.

Deux points techniques ont également nécessité des ajustements :

- Une **trame mal construite** a provoqué une erreur lors de l'envoi des données, rapidement corrigée après relecture du format.
- Un **faux contact au niveau du capteur MPU6050** a causé des détections erronées, résolu après vérification du câblage.

Ces incidents n'ont pas compromis l'avancement global du projet, mais ils ont mis en lumière l'importance d'une **vérification rigoureuse du matériel et de la structuration des messages**.



4.3 Apport personnel du projet .

- **Compétences acquises :**

Au cours du projet, plusieurs compétences techniques ont été développées :

- **Lecture et interprétation de données brutes** issues d'un capteur inertiel (MPU6050),
- **Communication via le protocole I²C** entre microcontrôleur et capteur,
- **Traitement de signal simple**, notamment pour extraire des événements pertinents,
- **Détection d'événements en temps réel**, avec gestion de seuils et conditions de déclenchement.

- **Difficultés rencontrées :**

Certaines étapes ont présenté des obstacles techniques :

- **Compréhension des unités et conversion des données brutes en valeurs physiques** exploitables,
- **Présence de bruit dans les mesures**, rendant instable la détection dans certaines conditions,
- **Faux contact matériel** sur le capteur MPU6050, provoquant des données incohérentes.

- **Solutions mises en place :**

Des solutions concrètes ont été apportées :

- **Calibration initiale du capteur à l'arrêt** pour obtenir une référence stable,
 - **Utilisation de seuils dynamiques et ajustables** dans le code pour affiner la détection,
 - **Vérification et sécurisation du câblage**, pour corriger les problèmes de faux contacts.
- **Améliorations possibles du projet ballon stratosphérique :**

Ajout de capteurs environnementaux complémentaires :

- Capteurs de qualité de l'air (particules fines, CO₂, ozone), utiles pour la surveillance atmosphérique.
- Capteur de rayonnement UV pour étudier l'exposition en altitude.
- **Automatisation de la récupération :**
Implémenter un système de balise radio ou de parachute guidé pour faciliter

la récupération du ballon après son atterrissage.

- **Analyse des données en temps réel avec alertes :**
Sur la page web, intégrer des alertes automatiques (ex. seuils critiques de pression, température, etc.) pour réagir rapidement en cas d'anomalies.
- **Optimisation énergétique :**
Utiliser des modes basse consommation pour prolonger la durée du vol ou utiliser une batterie avec un panneau solaire pour éviter une panne de courant et continuer à transmettre les données.

Annexes :

Vous trouverez ci-dessous les ressources complètes du projet, incluant les programmes sources, fichiers de configuration, et documents annexes.

- **Code source et documents – GitHub (branche **lancien**)**

Ce dépôt contient le code source principal du projet (MPU6050 + dossier lora_files), ainsi que les fichiers utiles à la compilation et au test sur Raspberry Pi.

 <https://github.com/estanislawski4/ProjetBallon/tree/lancien>

- **Fichiers complémentaires – Google Drive**

Ce dossier contient une copie du code source (MPU6050 et lora_files) ainsi que tous les documents de suivi, diagrammes, et représentations liés au projet.



https://drive.google.com/drive/folders/16ZonfM98yCZ4cKTGNqQUf4eB17klmcpb?usp=drive_link