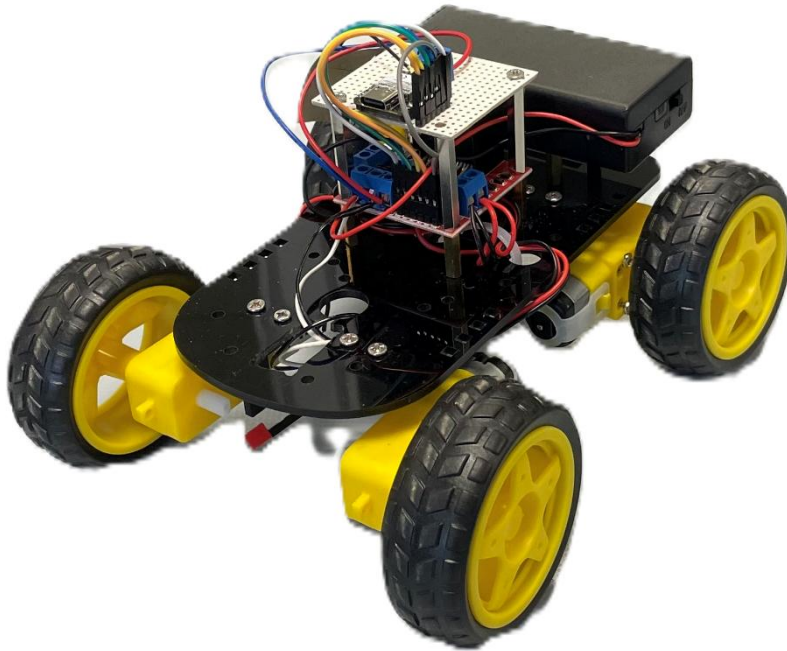


SAÉ : Centrale inertielle



Encadré par : Philippe Lucidarme

Quentin Adet
Dimitri Léguillon

IUT Angers :
4, Boulevard Lavoisier, 49100 Angers

Groupe 240

19 novembre 2025

Décharge de responsabilités

Ce rapport présente le travail réalisé par un groupe d'étudiants dans le cadre d'un projet pédagogique. Les auteurs et l'Université d'Angers ne garantissent pas que l'information, les documents, la méthodologie et le matériel présentés dans ce document soient complets, conformes à l'état de l'art et exacts ni n'assurent en toutes circonstances la sécurité des biens, des personnes et des utilisateurs. Les auteurs et l'Université d'Angers ne seront pas tenus responsables des dommages éventuels qui pourraient résulter de l'utilisation du contenu du présent rapport.

Licence

Léguillon Dimitri et Quentin Adet, auteurs du présent rapport, publions et divulguons celui-ci sous la Licence Creative Commons suivante « CC BY » pour le monde entier et pendant la durée légale de protection des droits d'auteur. Cette licence autorise la représentation, la reproduction, la modification, la création d'œuvres dérivées et l'utilisation y compris à des fins commerciales sous réserve de mentionner les noms et prénoms des auteurs.



A handwritten signature in black ink, appearing to read 'adet'.

Quentin Adet

A handwritten signature in black ink, appearing to read 'Dimitri Léguillon'.

Dimitri Léguillon

Sommaire

REMERCIEMENTS.....	4
INTRODUCTION.....	5
CAHIER DES CHARGES :.....	6
SCHEMA FONCTIONNEL DU PROJET :	10
SCHEMA STRUCTUREL DU PROJET :	11
DIAGRAMME FAST	12
LIVRABLE 1	13
<i>Le fonctionnement d'un IMU.....</i>	<i>13</i>
<i>Choix des composants</i>	<i>21</i>
LIVRABLE 2 (MISE EN ŒUVRE SUR BREADBOARD).....	30
<i>Schéma de câblage.....</i>	<i>30</i>
<i>Réalisation</i>	<i>30</i>
LIVRABLE 3	31
<i>Étude des communications réseaux.....</i>	<i>31</i>
LIVRABLE 4	35
<i>Récupération et mise en application des données de l'IMU</i>	<i>35</i>
LIVRABLE 5	39
<i>Conception carte électronique réceptrice (PCB).....</i>	<i>39</i>
LIVRABLE 6	44
<i>Contrôle du véhicule selon les données renvoyées par l'IMU</i>	<i>44</i>
LIVRABLE 7.....	45
<i>Création d'une Interface Homme Machine (IHM).....</i>	<i>45</i>
LIVRABLE 8	47
<i>Réalisation du code final de la télécommande</i>	<i>47</i>
<i>Réalisation de la commande du robot</i>	<i>49</i>
CONCLUSION ET PERSPECTIVES.....	51
ANNEXES	52
<i>Annexes A : Code communication HTTP</i>	<i>52</i>
<i>Annexes B : Code communication Point à Point</i>	<i>54</i>
<i>Annexes C : Code HTML cube 3D</i>	<i>57</i>
<i>Annexes D : Code émetteur cube 3D</i>	<i>59</i>
<i>Annexes E : Code IHM</i>	<i>64</i>
<i>Annexes F : Code émetteur Final.....</i>	<i>66</i>
<i>Annexes G : Code récepteur Final</i>	<i>67</i>
BIBLIOGRAPHIE.....	68
<i>Bibliographie sur les IMU</i>	<i>68</i>
<i>Bibliographie sur les microcontrôleurs</i>	<i>69</i>
<i>Bibliographie sur le filtre Mahony</i>	<i>69</i>
QUATRIEME DE COUVERTURE	70

Remerciements

Tout d'abord, nous tenons à remercier tout particulièrement et à témoigner toute notre reconnaissance aux personnes suivantes, pour leur soutien dans la concrétisation de ce projet :

- Nous tenons tout particulièrement à remercier M. Philippe LUCIDARME, le tuteur du projet, pour ses conseils judicieux, sa patience, sa disponibilité et la confiance qu'il nous a témoignée durant ce projet. Il a également mis à notre disposition tous les moyens nécessaires pour assurer la réussite de ce projet.
- Nous remercions le département GEII de l'IUT d'Angers-Cholet pour sa collaboration tout au long de cette SAE et pour avoir mis ses ressources à notre disposition lors de la concrétisation de ce projet.
- M. BREVARD et M. BOUCHERAU pour leur conseil avisé et leur coopération.

Introduction

Aujourd'hui, tout est fait pour que nous ayons de moins en moins d'efforts à fournir, et cela se voit partout autour de nous. Entre le développement des intelligences artificielles qui automatisent nos tâches, les drones qui se déplacent à notre place et, plus concrètement au quotidien, les robots aspirateurs qui réalisent les tâches ménagères, beaucoup d'actions qui étaient autrefois notre fardeau sont déléguées aux machines.

Pourtant, une question se pose : comment ces technologies parviennent-elles à faire ce que nous faisons nous-mêmes, alors que nous, êtres humains, pensons et avons conscience de notre corps et de notre environnement ? Comment reproduisent-elles, d'une certaine manière, ces sensations de mouvement et de position ?

Pour commencer à répondre à cette question, il faut revenir à la base : avant même de pouvoir « penser » pour s'adapter, une machine doit d'abord être capable de se mouvoir et de prendre conscience de ses propres mouvements. C'est sur cette première étape que se concentre ce projet. Au cœur de cette « conscience du mouvement » se trouve un composant aussi discret par sa taille que crucial par son rôle : la centrale inertielle, ou IMU.

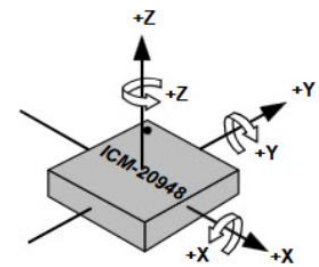


Figure 1 : Représentation IMU

Dans le cadre du parcours GEII, spécialité Électronique et Systèmes Embarqués (ESE), cette centrale inertielle est étudiée et intégrée au sein d'un système complet. L'IMU est utilisée comme principal moyen de pilotage d'un robot mobile à quatre roues, afin de mettre en évidence, dans une application concrète, ses performances et ses limites.

Le déplacement du robot sera commandé manuellement à partir des données d'orientation délivrées par une centrale inertielle, liée à un microcontrôleur. Les mesures des capteurs sont traitées pour en déduire son mouvement et sa position, puis transmises par Wi-Fi au robot. Ce dernier interprète les données reçues afin de générer les signaux de commande requis pour piloter les moteurs.

Ce projet englobe ainsi l'ensemble de la chaîne de traitement : acquisition des données inertielles, filtrage, interprétation du mouvement, transmission sans fil, et commande des moteurs.

Ce prototype est une approche concrète et complète des défis des systèmes embarqués modernes, allant de l'intégration matérielle à l'étalonnage des capteurs, en passant par la communication sans fil et les techniques de commande des moteurs.



Figure 2 : Robot mobile à 4 roues

Cahier des charges :

Livrable 1

Besoin principal :

Recherche et analyse

1. Le fonctionnement des IMU
2. Choix de l'IMU
3. Choix du microcontrôleur
4. Choix des batteries

Objectif :

Le premier livrable a pour objectif d'établir une base théorique et technique solide pour la suite du projet en étudiant les éléments essentiels du système de commande inertielle

Livrable 2

Besoin principal :

Mise en œuvre sur une carte de prototypage (type breadboard)

1. Schéma de câblage
2. Réalisation

Représentation du besoin :

Objectif :

Le second livrable a pour objectif de réaliser l'intégration matérielle et logicielle de la centrale inertielle sur une carte de prototypage, ainsi que de développer les outils nécessaires à l'affichage et à la visualisation des données en temps réel.

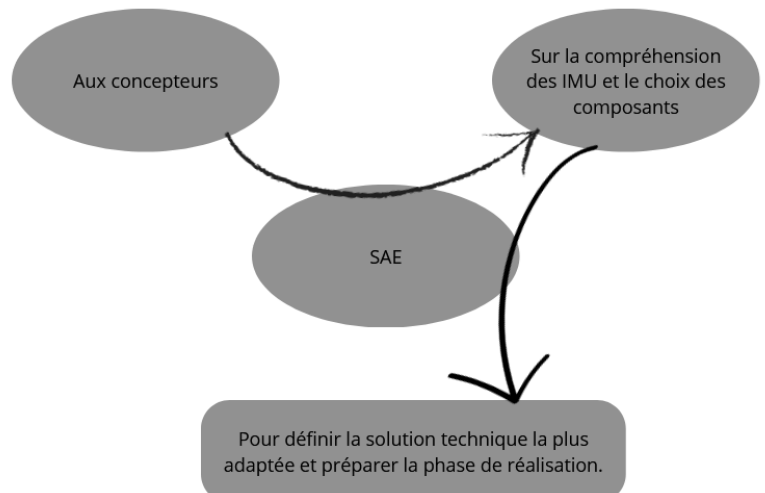


Figure 3 : bête à corne livrable 1

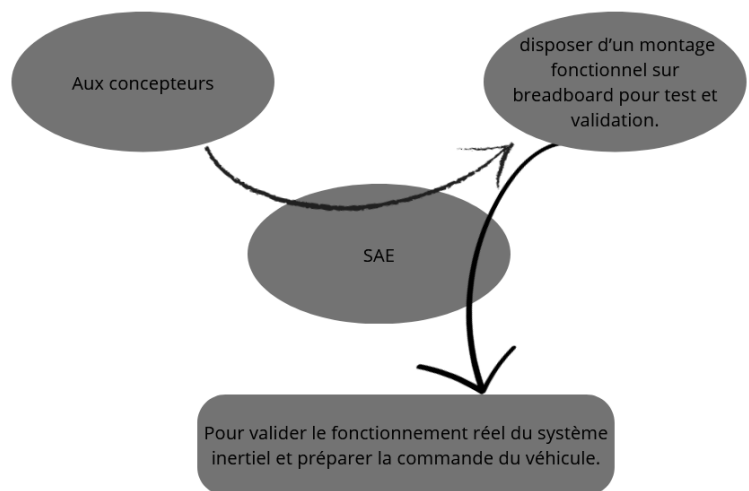


Figure 4 : bête à corne livrable 2

Livrable 3

Besoin principal :

Communication sans fil entre deux microcontrôleurs

Etude des types de communications

- a. Wifi
- b. Bluetooth

Représentation du besoin :

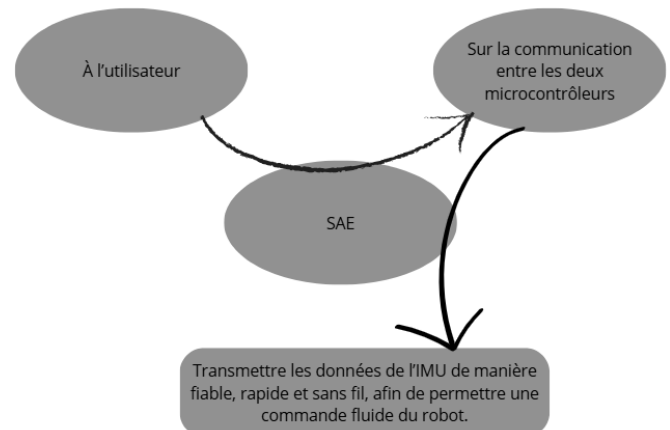


Figure 5 : bête à corne livrable 3

Objectif :

L'objectif consiste à établir une liaison sans fil entre deux modules ESP32. Le choix entre les protocoles WiFi et Bluetooth s'effectuera en fonction des contraintes du projet relatives à la portée, la consommation énergétique et le débit de transmission.

Livrable 4

Besoin principal :

Récupération des valeurs des capteurs de l'IMU et traitement des données via un AHRS

Représentation du besoin :

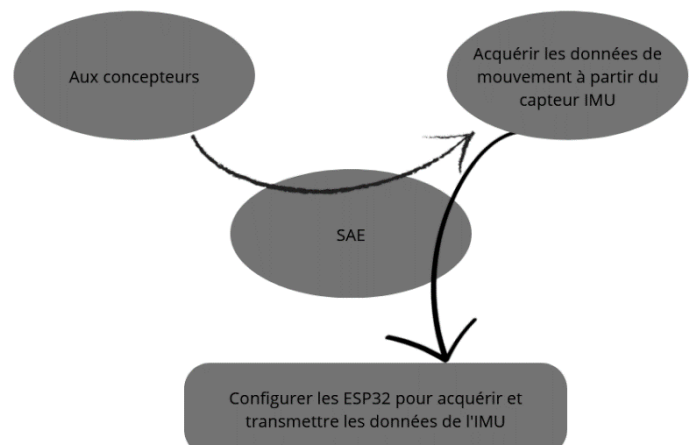


Figure 6 : bête à corne livrable 4

Objectif :

L'objectif consiste à récupérer les données de l'IMU et à les traiter afin de pouvoir les exploiter

Livrable 5

Besoin principal :

Conception et réalisation des cartes électroniques (PCB)

Représentation du besoin :

Objectif :

Le livrable 5 a pour objectif de concevoir deux PCB sur mesure destinés à remplacer les breadboards du livrable 2. Le choix et l'implantation de chaque composant seront réalisés avec minutie afin d'obtenir des cartes les plus compactes possible. Une étude de la consommation du système devra également être menée afin de dimensionner correctement les batteries.

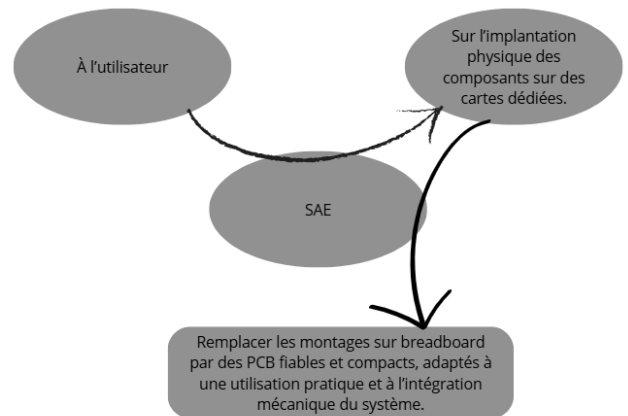


Figure 7 : bête à corne livrable 5

Livrable 6

Besoin principal :

Contrôler le véhicule selon les données renvoyées par l'IMU

Représentation du besoin :

Objectif :

Le livrable 6 a pour objectif de concevoir une commande afin de contrôler le robot mobile via les données renvoyées par l'IMU de manière progressive et fiable

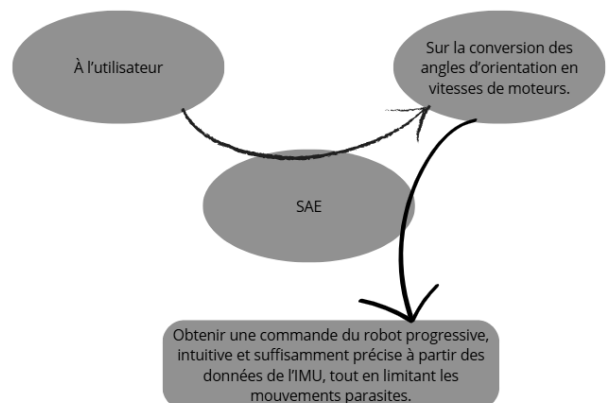


Figure 8 : bête à corne livrable 6

Livrable 7

Besoin principal :

Création d'une Interface Homme Machine (IHM)

Représentation du besoin :

Objectif :

Le livrable 7 a pour objectif de concevoir une interface homme-machine afin de pouvoir visualiser facilement les informations du capteur et la direction prise par la machine afin de faciliter l'utilisation du système et son débogage

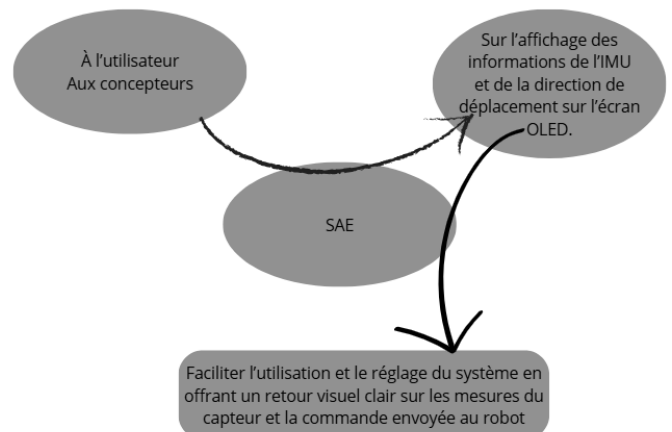


Figure 9 : bête à corne livrable 7

Livrable 8

Besoin principal :

Réalisation du code final afin de diriger le robot mobile

- Réalisation de la récupération et de l'envoi de la position de la main
- Réalisation de la commande du robot

Représentation du besoin :

Objectif :

L'objectif du livrable 8 est de générer les codes finaux, allant de la récupération de la position de la main grâce à l'IMU jusqu'au contrôle du robot par le biais d'un signal PWM pour les moteurs. Ce qui permet un premier prototypage du robot et l'amélioration des derniers soucis relatifs au code ou au robot.

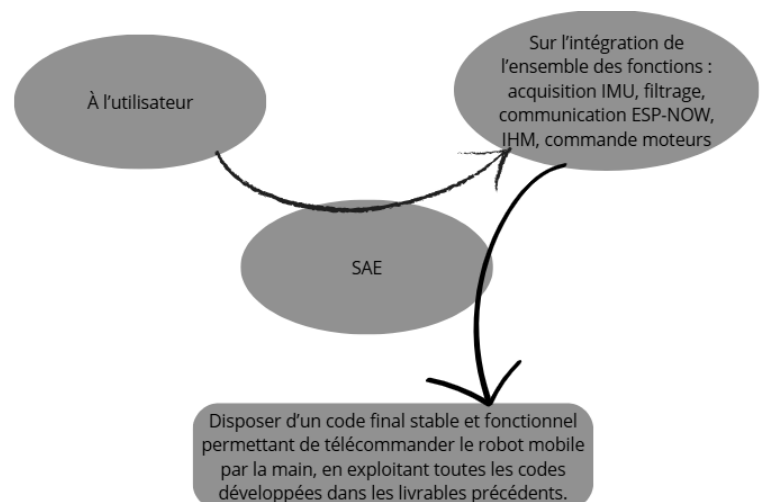


Figure 10 : bête à corne livrable 8

Schéma fonctionnel du projet :

Le schéma fonctionnel présente l'architecture globale du système réparti autour de deux microcontrôleurs ESP32-C3 mini communiquant entre eux via une liaison Wi-Fi. La première carte assure l'acquisition et l'affichage des données : elle reçoit les mesures de l'IMU ICM-20948 et pilote l'écran OLED via le bus I²C. La seconde carte est dédiée au contrôle de la puissance. Elle génère les signaux PWM nécessaires à la commande du pont en H, lequel pilote les moteurs. Chaque module est clairement identifié ainsi que le type de signal échangé (I²C, PWM, binaire), ce qui permet de visualiser simplement les rôles, flux d'information et interactions au sein du système embarqué.

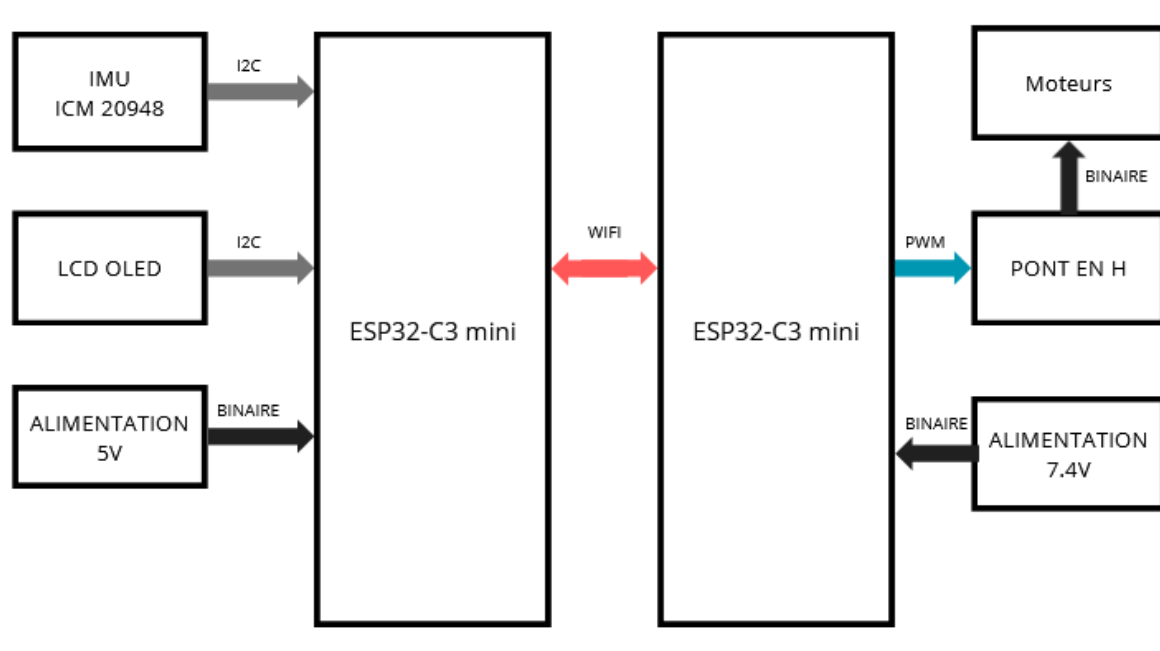


Figure 11 : schéma fonctionnel du projet

Schéma structurel du projet :

L'architecture matérielle définit l'organisation physique du système en spécifiant les éléments constitutifs et leurs interconnexions. La première unité ESP32-C3 mini gère l'acquisition des informations et leurs affichages via le capteur ICM-20948 et l'écran OLED, tous deux connectés via le bus I²C et alimentés sous 3.3 V. La seconde carte gère la gestion de puissance : elle produit les signaux PWM destinés au pont en H, lequel commande les moteurs alimentés sous 7,4 V. Ce schéma met en évidence les modules fonctionnels, les interfaces de communication et la distribution énergétique, fournissant une représentation structurée de l'implémentation matérielle du système.

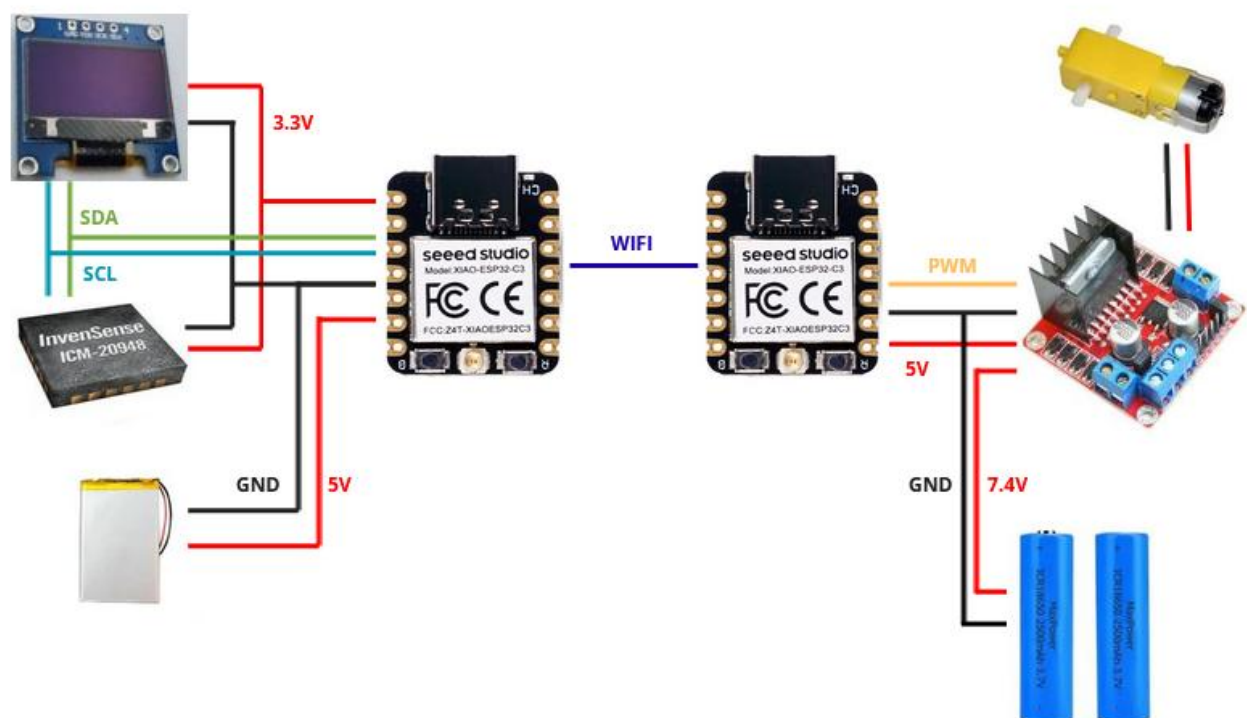


Figure 12 : schéma structurel du projet

Diagramme Fast

Le diagramme FAST (Function Analysis System Technique) est utilisé ici pour l'analyse fonctionnelle du projet. Cet outil permet de structurer les différentes fonctions du système en les décomposant en sous-fonctions, tout en mettant en évidence les relations de cause à effet entre elles. Cette méthode offre une vision claire de l'architecture fonctionnelle du projet, facilite la compréhension de son fonctionnement global, et guide les choix de conception en vue d'une optimisation plus cohérente et efficace.

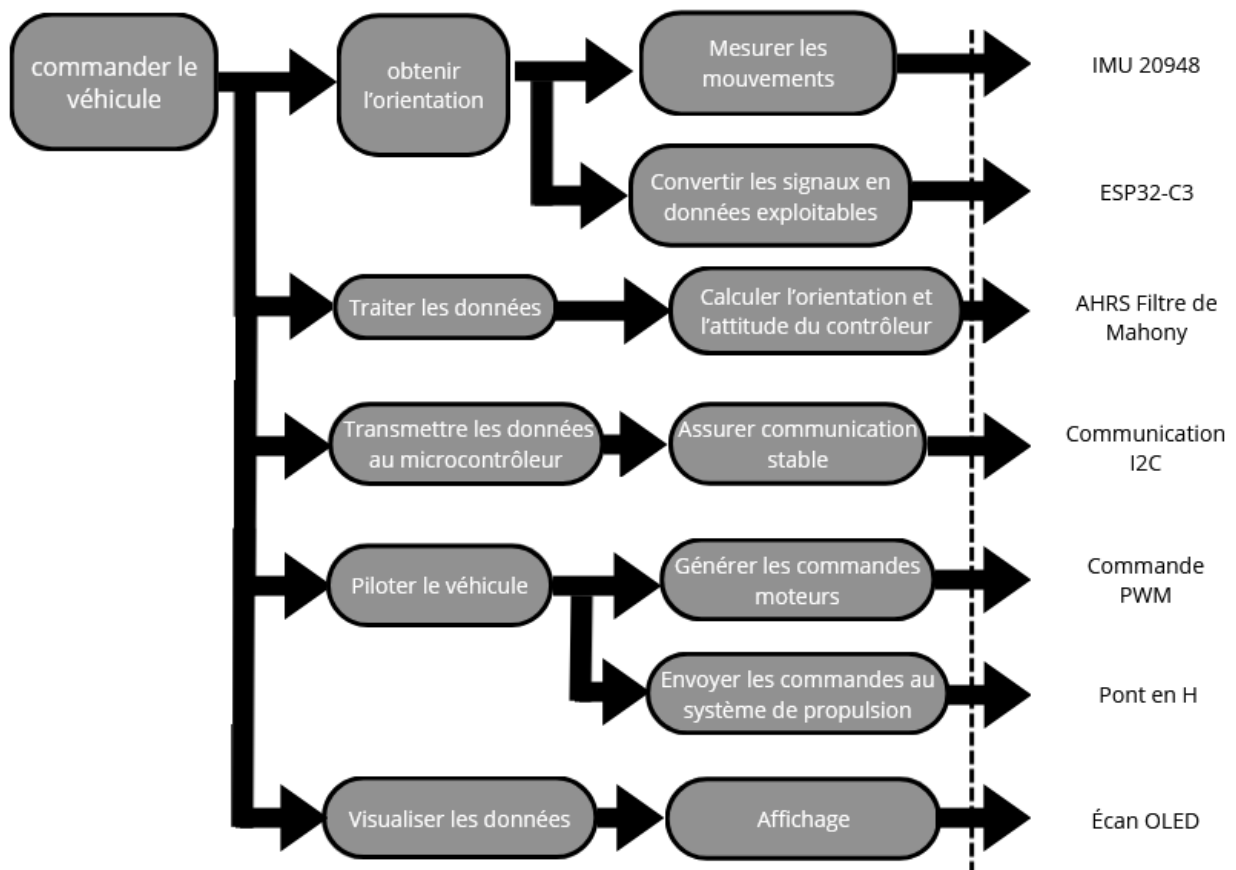


Figure 13 : *diagramme Fast du projet*

Livrable 1

Le fonctionnement d'un IMU

Pour utiliser correctement une IMU (Inertial Measurement Unit), il est primordial de comprendre son principe de fonctionnement interne. Une IMU est un dispositif électronique capable de mesurer les mouvements et l'orientation d'un objet dans l'espace en combinant plusieurs capteurs inertiels. Ces mesures permettent de déterminer la position, la vitesse angulaire et l'orientation d'un objet, même en l'absence de repères externes.

La suite de ce livrable est concentrée principalement sur les IMU MEMS (Micro-Electro-Mechanical Systems) et détaille le fonctionnement de leurs composants clés : accéléromètre, gyroscope et magnétomètre, ainsi que la manière dont ils collaborent pour fournir des informations précises sur le mouvement et l'orientation.

IMU MEMS :

Lorsque l'on étudie la composition d'une IMU MEMS, on met principalement en évidence trois familles de capteurs : l'accéléromètre, le gyroscope et le magnétomètre. Chacun d'eux repose sur une architecture interne similaire, organisée en plusieurs niveaux de conversion permettant de convertir un mouvement physique en donnée numérique.

Architecture de la chaîne de mesure

Indépendamment du type de capteur, une IMU MEMS suit généralement la même chaîne fonctionnelle :

1. Structure mécanique :
 - Masse mobile pour un accéléromètre
 - Structures ou poutres vibrantes pour un gyroscope
 - Résonateur magnétostrictif pour un magnétomètre
2. Conversion : transformation du mouvement mécanique en signal électrique (souvent capacitif ou piézoélectrique).
3. Conditionnement analogique : amplification, filtrage et mise en forme du signal issu de la conversion.
4. Conversion numérique : numérisation du signal via un convertisseur analogique-numérique (ADC).
5. Interface de communication : envoi des données au microcontrôleur via un bus série (I²C ou SPI).

Cette organisation modulaire garantit une mesure inertielle précise, stable et suffisamment rapide pour les applications de commande et de navigation en temps réel.

Accéléromètre :

Ce capteur mesure les changements de vitesse. Il peut détecter une accélération linéaire, ce qui permet de déterminer la vitesse à laquelle un objet se déplace dans une direction spécifique.

L'interface multiphysique *Electromécanique* du module MEMS. Le modèle se compose de trois sous-ensembles :

La masse libre, les ressorts fixés qui soutiennent cette masse et le réseau d'électrodes. Lorsque l'appareil subit une accélération, la masse libre se déplace, ce qui fait varier la capacité entre les électrodes fixes et mobiles. Cette variation de la capacité est proportionnelle à l'accélération (voir schéma ci-dessous Figure 15 et Figure 16).

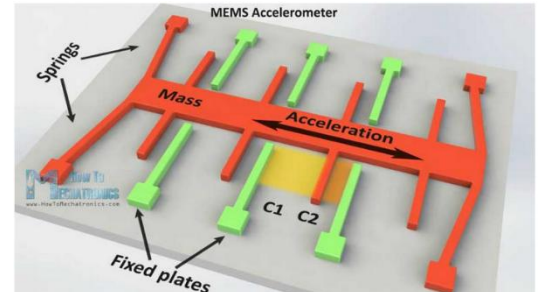


Figure 14 : [Accéléromètre IMU](#)

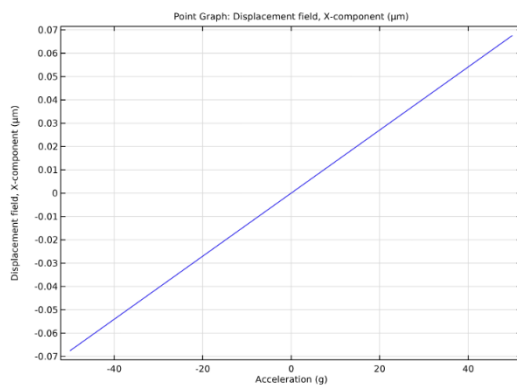


Figure 15 Displacement vs acceleration.

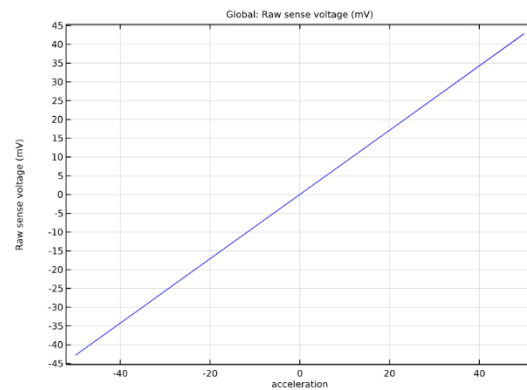


Figure 16 Sensed voltage vs. applied acceleration.

Cette relation reste approximativement linéaire, permettant une mesure précise. La tension de sortie du circuit de conditionnement est ensuite convertie en valeur d'accélération (en g ou m/s^2).

Erreurs et limitations des accéléromètres MEMS

Biais (offset) : Décalage constant de la mesure, typiquement de quelques mg (milligrammes de gravité). Ce biais doit être mesuré au repos et soustrait lors du traitement.

Bruit de mesure : Pour l'ICM-20948, la densité de bruit est de $230 \mu g/\sqrt{Hz}$ ce qui génère un léger "jitter" dans les mesures instantanées.

Impact sur la commande du robot : Un biais non compensé s'intègre au fil du temps et provoque une dérive de la position estimée, ainsi l'utilisation seule de l'accéléromètre n'est pas possible afin de garantir la commande du robot. C'est pourquoi la calibration et la fusion avec d'autres capteurs (filtre Mahony) sont essentielles.

*Le jitter = le tremblement ou la variabilité aléatoire du signal provoquée par le bruit.

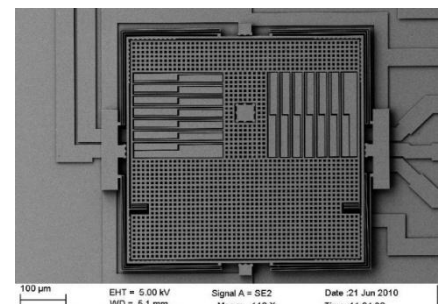


Figure 17 : [Image Accéléromètre](#)

Gyroscope

Le gyroscope mesure la vitesse de rotation, ce qui signifie qu'il vous indique la vitesse à laquelle un objet tourne autour de son axe. Ceci est vital pour les applications où la compréhension de l'orientation est essentielle, comme la stabilisation de drones ou le maintien d'une caméra stable.

Le gyroscope est composé d'une partie centrale fixée (la suspension) et de deux paires de tiges : les tiges d'entraînement et les tiges de détection. Ces tiges sont équipées d'électrodes qui permettent d'envoyer et de recevoir des signaux électriques (voir Figure ci-dessous).

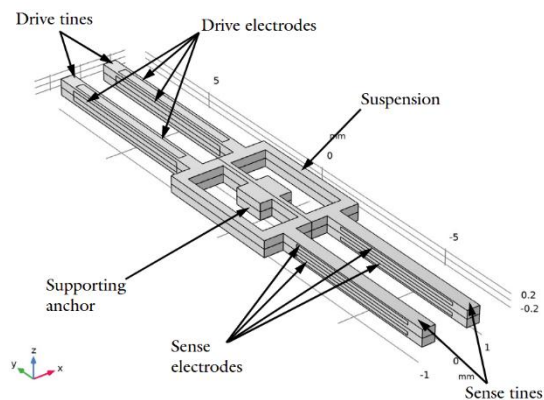


Figure 18 : [Gyroscope IMU](#)

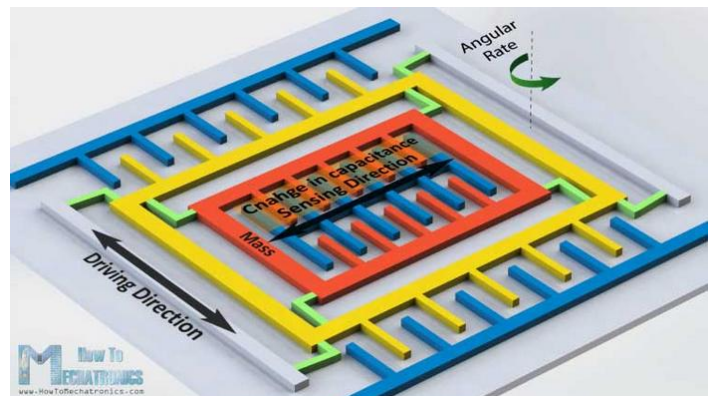


Figure 19 : [Gyroscope IMU](#)

Quand le gyroscope est immobile ou en mouvement régulier : Les tiges d'entraînement vibrent dans le plan horizontal (xy) grâce à un courant électrique appliqué. Les tiges de détection restent immobiles dans ce plan.

Quand le gyroscope tourne autour de l'axe y : La rotation provoque une force appelée force de Coriolis, qui fait vibrer les tiges de détection hors du plan horizontal (dans la direction z) (voir figure ci-contre).

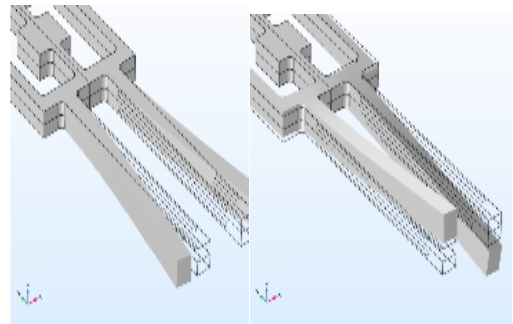


Figure 20 : [Gyroscope IMU](#)

Principe physique : Force de Coriolis

La force de Coriolis s'exprime par la relation $F_{\text{Coriolis}} = 2m(\Omega \times \vec{v})$, où :

- m est la masse des tiges de détection
- Ω est la vitesse angulaire à mesurer (rotation du capteur)
- v est la vitesse de vibration imposée aux tiges d'entraînement

La sortie électrique du gyroscope (courant piézoélectrique) est donc proportionnelle à Ω , permettant de mesurer la vitesse de rotation en degrés par seconde ($^{\circ}/s$) (voir figure ci-contre).

En résumé : les tiges d'entraînement créent des vibrations à certaines fréquences, la rotation fait bouger les tiges de détection, et ce mouvement est transformé en signal électrique pour mesurer la rotation.

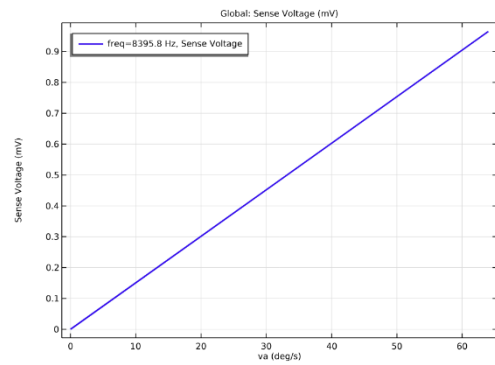


Figure 21 Sense voltage vs. angular acceleration at a drive voltage amplitude of 2 V and a frequency of 8396 Hz.

Erreurs et limitations des gyroscopes MEMS

Densité de bruit : Pour l'ICM-20948, le bruit est de $0.015 \text{ }^{\circ}/s/\sqrt{\text{Hz}}$. À 1 kHz de bande passante, cela donne environ $1.5 \text{ }^{\circ}/s$ de bruit RMS.

Dérive temporelle (bias instability) : Le biais varie lentement au cours du temps et avec la température, nécessitant une recalibration périodique.

Impact sur la commande du robot : Le gyroscope seul ne peut pas maintenir une orientation fiable sur une longue durée. C'est pourquoi le filtre de Mahony fusionne les données du gyroscope avec celles de l'accéléromètre (référence de gravité) et du magnétomètre (référence du nord magnétique) pour corriger cette dérive. Afin d'obtenir une commande précise

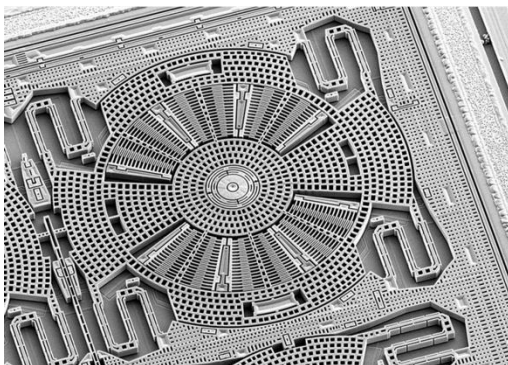


Figure 22 : [Image Gyroscope](#)

Magnétomètre (en option)

Ce capteur suit les champs magnétiques, qui sont souvent utilisés pour trouver le nord géographique. C'est particulièrement utile dans les systèmes de navigation, où il est essentiel de connaître votre orientation par rapport au champ magnétique terrestre.

Un magnétomètre MEMS est fabriqué en microscopie de précision sur un petit morceau de quartz ou de silicium.

Il contient :

- Deux poutres ou diapasons très fins (micromètres de largeur), capables de vibrer.
- Une couche magnétique déposée à la surface.
- Des électrodes piézoélectriques pour faire vibrer et mesurer la vibration.

On peut voir ça comme deux petits ressorts fixés à une base, chacun pouvant osciller.

Étapes de fonctionnement

1. Excitation mécanique
 - a. Les électrodes piézoélectriques font vibrer les deux poutres à leur fréquence de résonance naturelle (comme quand on pince une règle en plastique).
 - b. Sans champ magnétique, les deux vibrent exactement pareil.
2. Action du champ magnétique
 - a. La couche magnétique déposée sur les poutres a un moment magnétique.
 - b. Quand un champ magnétique terrestre agit dessus, il exerce un couple mécanique (comme une petite force de rotation. Voir figure ci-contre).
 - c. Ce couple pousse une poutre vers le haut et l'autre vers le bas.
3. Déformation et mesure
 - a. Les deux poutres se déforment légèrement dans des sens opposés.
 - b. Cela modifie leur fréquence de résonance (l'une augmente, l'autre diminue).
 - c. En mesurant la différence de fréquence entre les deux, on obtient une valeur proportionnelle au champ magnétique.

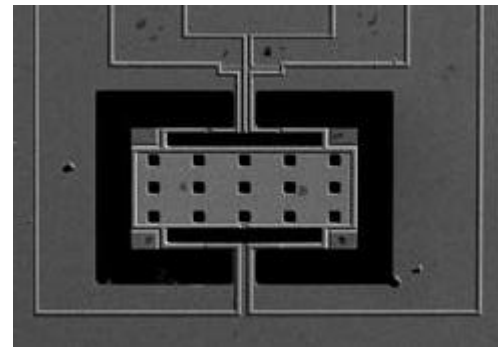


Figure 23 : [Image Magnétomètre](#)

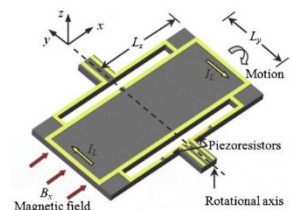


Figure 24 – Représentation 3D du design d'un magnétomètre MEMS AM à force de Laplace avec détection piézorésistive en torsion [42].

4. Conversion électronique

- L'électronique intégrée transforme ce décalage de fréquence en un signal électrique (voir figure ci-contre).
- Ce signal correspond à l'intensité (et direction, selon l'axe du capteur) du champ magnétique.

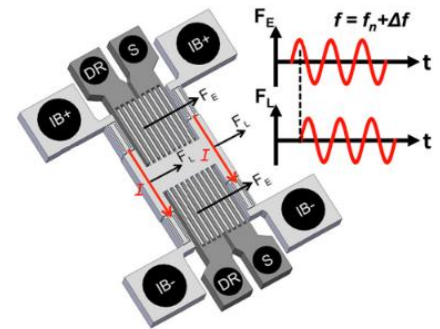


Figure 25 : Magnétomètre IMU

On obtient une valeur proportionnelle à l'intensité et à la direction du champ magnétique selon l'axe du capteur. Le magnétomètre de l'ICM-20948 possède une résolution de 16 bits et une plage de mesure de $\pm 4900 \mu\text{T}$ (microteslas).

Erreurs et calibration du magnétomètre

Hard-iron : Offset constant causé par les composants ferromagnétiques proches du capteur (batteries, moteurs). Correction : rotation complète sur 360° et calcul de l'offset moyen.

Soft-iron : Distorsions du champ magnétique causées par des matériaux magnétiquement « mous » *. Correction : matrice de compensation 3×3 .

Impact sur la commande du robot

Dans ce projet, le magnétomètre n'est pas employé en tant que capteur de commande principal, mais plutôt comme source d'informations additionnelles pour stabiliser l'orientation absolue. En théorie, il offre la possibilité de donner une direction vers le nord magnétique et d'optimiser la stabilité du lacet lors de mouvements prolongés en rectifiant la dérive du gyroscope. Dans la pratique, la présence de sources perturbatrices telles que les moteurs, les batteries ou les structures métalliques nécessite une calibration précise à la fois hard-iron et soft-iron ; sans cette procédure, les données pourraient être suffisamment altérées pour compromettre la précision de l'orientation estimée et donc celle du contrôle.

*Les matériaux magnétiquement « mous » déforment le champ magnétique sans créer leur propre champ

IMU basés sur un FOG

Les FOG utilisent des boucles de fibre optique et mesurent les interférences entre les faisceaux lumineux des boucles opposées afin de détecter la rotation de chaque axe. Le matériel utilisé est plus coûteux, plus volumineux et consomme généralement plus d'énergie, mais l'absence de pièces mobiles le rend moins sensible aux variations de température et aux vibrations mécaniques. Les FOG sont particulièrement adaptés aux environnements exigeants, à haute température et à fortes vibrations, comme l'exploitation minière, le pointage de machines industrielles et la stabilisation d'équipements lourds.

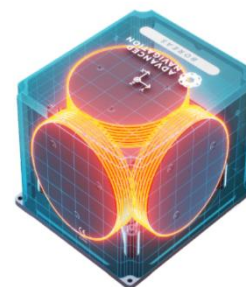


Figure 26 : [IMU FOG](#)

Fonctionnalité	IMU MEMS	IMU FOG
Technologie	Les systèmes micro électromécaniques (MEMS) utilisent des composants mécaniques miniaturisés.	Le gyroscope à fibre optique (FOG) utilise l'interférence lumineuse dans la fibre optique pour la mesure.
Taille et poids	Plus petit et plus léger, ce qui le rend idéal pour les applications limitées en espace comme les drones et les appareils portables.	Plus grand et plus lourd à cause de l'utilisation de fibres optiques, plus adapté aux applications aérospatiales ou industrielles.
Coût	Faible coût, largement disponible sur les marchés grand public et industriels.	Coût plus élevé en raison d'une technologie avancée et d'un processus de fabrication plus complexe.
Précision	Bonne précision pour les applications générales, mais la dérive dans le temps peut être importante dans les environnements de haute précision.	Précision extrêmement élevée avec une très faible dérive, ce qui le rend idéal pour les applications de navigation et aérospatiales.
Consommation d'énergie	Faible consommation d'énergie, adapté aux appareils alimentés par batterie.	Consommation d'énergie plus élevée grâce aux composants optiques, mieux adaptés aux systèmes disposant de ressources électriques disponibles.
Durabilité	Généralement plus résistant aux chocs et aux vibrations, souvent utilisé dans des environnements difficiles.	Fragile par rapport aux MEMS ; nécessite une manipulation soignée et est souvent installé dans des environnements à vibrations contrôlées.
Applications	Électronique grand public, drones, automobile, appareils portables, défense et utilisations industrielles générales.	Aérospatiale, défense, sous-marins et systèmes de navigation de haute précision.

Justification du choix MEMS pour notre application

Le choix d'un IMU MEMS (ICM-20948) plutôt qu'un FOG se justifie par les contraintes spécifiques de notre projet :

Environnement mécanique : La télécommande peut subir des chocs et vibrations modérés lors de son utilisation. Les IMU MEMS sont conçus pour résister à ces contraintes (spécification de choc : 20 000 g pour l'ICM-20948), alors que les FOG sont plus fragiles et nécessitent des montages antivibratoires.

Budget : Le coût d'un module ICM-20948 est d'environ 15-25 €, contre plusieurs milliers d'euros pour un FOG de qualité équivalente. Pour un projet pédagogique étudiant, ce facteur est déterminant.

Encombrement : L'ensemble du système émetteur (ESP32-C3 + ICM-20948 + écran OLED + batterie) doit tenir dans la main de l'utilisateur. Les dimensions du module ICM-20948 (3×3×1 mm pour la puce) permettent une intégration compacte impossible avec un FOG.

Consommation énergétique : L'ICM-20948 consomme 3,11 mA, un FOG consommerait plusieurs centaines de milliampères, nécessitant une batterie beaucoup plus volumineuse.

En conclusion, l'IMU MEMS représente le bon compromis entre performances, coût, taille, réactivité et consommation pour notre cahier des charges.

Choix des composants

Choix de l'IMU :

Afin de choisir un IMU adapté à nos besoins nous avons décidé d'en comparer plusieurs sur différents critères, ci-dessous vous pouvez trouver le tableau comparatif.

Critère	ICM-20948 	MPU-9250 	BNO055 
Prix	Relativement modéré / gamme moyenne, modules autour de ~15-25 €. (GO TRONIC 20,9 €.)	Souvent moins cher, surtout pour les modules clones ou anciens. ~8-20 €.	Souvent plus cher. ~30-45 €. (GO TRONIC 38,5 €.)
Qualité de mesure / bruit	Accéléromètre : densité de bruit $\approx 230 \mu\text{g}/\sqrt{\text{Hz}}$ Gyroscope : densité de bruit $\approx 0.015 \text{ }^\circ/\text{s}/\sqrt{\text{Hz}}$ Magnétomètre : résolution 16 bits, plage $\pm 4900 \mu\text{T}$	Accéléromètre : densité de bruit $\approx 300 \mu\text{g}/\sqrt{\text{Hz}}$, Gyroscope : densité de bruit $\approx 0.01 \text{ }^\circ/\text{s}/\sqrt{\text{Hz}}$	Accéléromètre : densité de bruit $\approx 150\text{--}190 \mu\text{g}/\sqrt{\text{Hz}}$ Gyroscope : bruit $\approx 0.014 \text{ }^\circ/\text{s}/\sqrt{\text{Hz}}$ Magnétomètre : offset calibré $\pm 2 \mu\text{T}$, précision capteur magnétique $\pm 2.5^\circ$
Consommation de courant	Très bon en faible consommation, adapté pour les applications portables. 3.11mA	Correct, souvent consommation 3.7mA	Peut-être plus consommateur selon le mode (orientation continue, fusion, etc.). 12.3mA
Fiches techniques	Voir bibliographie : Fiches techniques différentes IMU		

Synthèse comparative et décision finale

À l'issue de cette analyse comparative approfondie, l'ICM-20948 se distingue comme la meilleure solution alliant performances techniques et contraintes budgétaires du projet.

Avantages décisifs de l'ICM-20948 :

- Performances de mesure supérieures au MPU-9250 (bruit accéléromètre : 230 vs 300 $\mu\text{g}/\sqrt{\text{Hz}}$)
- Consommation énergétique optimisée (3,11 mA vs 12,3 mA pour le BNO055)
- DMP avancé permettant la fusion de capteurs 9 axes en temps réel
- Prix compétitif (15-25 €) largement inférieur au BNO055 (30-45 €)

Le BNO055 bénéficie certes d'un algorithme de fusion sensorielle intégré et d'une meilleure qualité de mesure, mais son coût élevé et sa consommation 4 fois supérieure ne justifient pas son emploi dans cette application du capteur.

Le MPU-9250 conserve un intérêt économique certain (8-20 €), mais ses performances globales (densité de bruit plus élevée, DMP moins avancé) et capacités de traitement restent inférieures à celles de l'ICM-20948 pour un écart de prix faible.

Par conséquent, la sélection de l'ICM-20948 s'avère être la décision technique la plus équilibrée et la plus conforme aux spécifications du projet, offrant le meilleur compromis entre performances, autonomie énergétique, et contraintes budgétaires.

Comparaison Microcontrôleur



Figure 27 : comparaison microcontrôleur

Il a été choisi d'utiliser le microcontrôleur ESP32, car il est le seul des trois microcontrôleurs à intégrer le Wi-Fi, fonctionnalité essentielle au projet. Bien qu'il soit possible d'ajouter des modules supplémentaires aux cartes Arduino et STM32, cette solution entraîne un coût additionnel.

L'ESP32 présente également de bonnes performances de calcul tout en conservant une faible consommation énergétique, ce qui en fait une solution adaptée aux systèmes embarqués exigeants. Il est programmable via l'IDE Arduino, ce qui facilite grandement son développement et sa prise en main.

En comparaison, des microcontrôleurs comme l'Arduino Uno ou le STM32 ne disposent pas de modules Wi-Fi et Bluetooth intégrés, nécessitant l'ajout de composants externes. C'est principalement ce critère d'intégration qui a motivé notre choix en faveur de l'ESP32.

Choix du modèle de microcontrôleur :

Après avoir retenu la famille de microcontrôleurs ESP32, il convient désormais de sélectionner le modèle le plus adapté à l'application. Il a été constaté que le modèle présentant la plus faible consommation moyenne tout en intégrant le Wi-Fi et le Bluetooth est l'ESP32-C3. Ce modèle a donc été retenu pour le projet.

Série	Architecture / Cœur	Wi-Fi / Bluetooth	Particularités	Consommation (typique)
ESP32-WROOM-32	Xtensa dual/single-core	Wi-Fi 2.4 GHz + Bluetooth classique & LE	Bonne puissance générale, large support	160 mA (mode actif) / 10 μ A (veille)
ESP32-S2	Xtensa single-core	Wi-Fi 2.4 GHz (pas de Bluetooth)	Intègre USB OTG, consommation réduite	160 mA (mode actif) / 20 μ A (veille)
ESP32-S3	Xtensa dual-core	Wi-Fi 2.4 GHz + Bluetooth LE	Support IA/ML, plus de GPIO disponibles	240 mA (mode actif) / 10 μ A (veille)
ESP32-C3	RISC-V single-core	Wi-Fi 2.4 GHz + Bluetooth LE	Faible conso, sécurisé	160 mA (mode actif) / 5 μ A (veille)
ESP32-C6	RISC-V single-core	Wi-Fi 6 + BLE + 802.15.4	Réseaux modernes (Thread, Zigbee), basse conso	200 mA (mode actif) / 10 μ A (veille)
ESP32-H2	RISC-V single-core	BLE + 802.15.4 (pas de Wi-Fi)	Idéal réseaux maillés, sans-fil basse énergie	100 mA (mode actif) / 1 μ A (veille)

Choix de l'écran :

L'objectif final est l'affichage de la direction de déplacement du robot à l'aide de flèches directionnelles. Il est également envisagé d'afficher les coordonnées X, Y et Z du capteur afin de faciliter le débogage du programme. Pour cela, deux technologies d'écran sont envisagées pour la réalisation de cet affichage :

- Écran à cristaux liquide

Cet écran présente l'avantage d'être simple à mettre en œuvre, mais il occupe beaucoup de place. Comme la taille disponible est limitée et que l'ensemble du système doit tenir dans la main, cela constitue un premier inconvénient. De plus, le nombre de caractères affichables ainsi que leur position sur l'écran sont très restreints. Pour ces raisons, ce choix n'a pas été retenu.

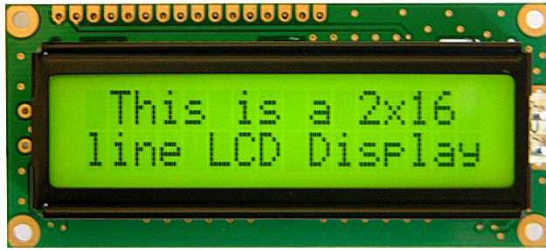
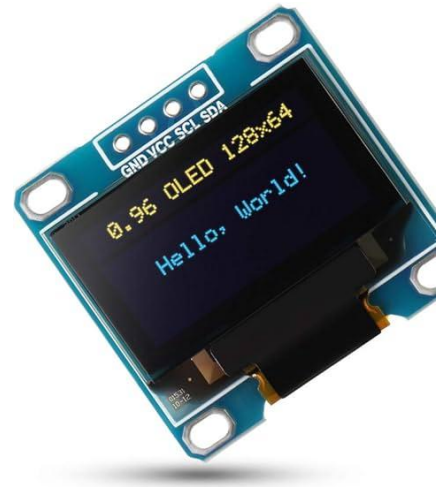


Figure 28 : Image d'un écran à cristaux liquide

- Écran OLED

L'écran OLED, quant à lui, est beaucoup moins contraignant : il permet d'afficher librement le contenu souhaité, avec la taille désirée et à l'emplacement choisi sur l'écran. Parmi les différentes tailles disponibles sur le marché, celle de 128×64 a été choisie car sa dimension est facilement intégrable dans un petit PCB.

Figure 29 : Image de l'écran OLED retenue



Choix des batteries :

Pour dimensionner les batteries, les consommations de chaque composant sont récupérées auprès de chaque constructeur. Si ces consommations ne sont pas fournies, elles seront calculées ci-dessous. Le total des consommations sera renseigné dans des tableaux qui serviront à dimensionner les batteries de chaque PCB.

Partie émettrice :

Esp32-C3 :

Wi-Fi Enabled Power Consumption

- **Active Model: <75 mA**
- **Modem-sleep Model: <25 mA**
- **Light-sleep Model: <4 mA**

La consommation de l'ESP32-C3 sur une durée d'une heure est supérieure à 75 mAh. Afin de prendre une marge de sécurité, la consommation est estimée à 100 mAh.

Écran Oled :

Données techniques

Jeu de puces	SSD1306
Couleur	Blanc
Angle visible	> 160°
Résolution	128 x 64 pixels
Tension de fonctionnement	3.3V / 5V
Température de fonctionnement	-30°C à +80°C
Faible consommation d'énergie	0,04W en fonctionnement normal
Commande	I2C / TWI
Affectation des broches	VCC, GND, SCL, SDA
Dimensions	env. 27mm x 27mm x 4,1mm

$$\begin{aligned} I &= U \times P \\ &= 3.3 \times 0.04 \\ &= 0.012 \text{ A} \end{aligned}$$

= 12 mA

La consommation de l'écran OLED pendant une heure est de 12 mAh.

ICM20948 :

Typical Operating Circuit of section 4.2, VDD = 1.8V, VDDIO = 1.8V, T_A=25°C, unless otherwise noted.

PARAMETER	CONDITIONS	MIN	TYP	MAX	UNITS	NOTES
SUPPLY VOLTAGES						
VDD		1.71	1.8	3.6	V	1
VDDIO		1.71	1.8	1.95	V	1
SUPPLY CURRENTS						
9-Axis (DMP disabled)	Low-Noise Mode; Compass in Continuous Mode		3.11		mA	2
Gyroscope Only (DMP, Barometer & Accelerometer disabled)	Low-Power Mode, 102.3 Hz update rate, 1x averaging filter		1.23		mA	2
Accelerometer Only (DMP, Barometer & Gyroscope disabled)	Low-Power Mode, 102.3 Hz update rate, 1x averaging filter		68.9		μA	2
Magnetometer Only (DMP, Accelerometer & Gyroscope disabled)	8 Hz update rate		90		μA	2
Full-Chip Sleep Mode			8		μA	2
TEMPERATURE RANGE						
Specified Temperature Range	Performance parameters are not applicable beyond Specified Temperature Range	-40		+85	°C	1, 3

La consommation de l'IMC20948 pendant une heure est de 3.11 mAh.

Consommation globale de la partie émettrice :

Composant	Consommation estimée
Esp32-C3	75 mAh
Écran Oled	12mAh
ICM20948	3.11mAh
Total	175.11mAh

Conclusion :

L'objectif est de garantir une autonomie minimale de deux heures lorsque le système est en fonctionnement. En intégrant une marge de sécurité, la consommation totale estimée de l'ensemble est de 300 mAh. Il est donc nécessaire de sélectionner une batterie présentant une capacité minimale de 600 mAh. Par ailleurs, la taille de la batterie est contrainte afin d'assurer un système aussi compact que possible. Une batterie de 3,7 V et de 1000 mAh a ainsi été retenue, comme indiqué ci-contre, ce qui permet de garantir une durée d'utilisation minimale estimée à six heures.



Figure 30 : Batterie L753626

Partie réceptrice :

Esp32-C3 :

Table 12.3. Current consumption depending on RF modes

Work mode	Description		Peak (mA)
Active (RF working)	TX	IEEE 802.11b, 1 Mbit/s, @21dBm	335
		IEEE 802.11g, 54 Mbit/s, @19 dBm	285
		IEEE 802.11n, HT20, MCS7, @18.5 dBm	276
		IEEE 802.11n, HT40, MCS7, @18.5 dBm	278
	RX	IEEE 802.11b/g/n, HT20	84
		IEEE 802.11n, HT20	87

Moteur DC :

Specifications

Operating Voltage	3V-12VDC
Maximum Torque	800g/cm max. @ 3V
Gear Ratio	1:48
Load Current	70mA (250mA max. @ 3V)
Weight	29g

4 moteurs sont utilisés, un pour chaque roue.

Donc :

$$250 \times 4 = 1000 \text{ mAh}$$

L298N

Symbol	Parameter	Test conditions	Min.	Typ.	Max.	Unit
V _S	Supply voltage (pin 4)	Operative condition	V _{IH} +2.5		46	V
V _{SS}	Logic supply voltage (pin 9)		4.5	5	7	V
I _S	Quiescent supply current (pin 4)	V _{en} = H; V _i = L; I _L = 0		13	22	mA
		V _{en} = H; V _i = H; I _L = 0		50	70	mA
		V _{en} = L; V _i = X			4	mA

Consommation globale de la partie réceptrice :

Composant	Consommation estimée
Esp32-C3	84 mAh
Moteur DC	1000 mAh
Pont en H L298	50mAh
Total	1134 mAh

Conclusion :

Un temps de fonctionnement minimum de 2 heures doit être assuré. La consommation globale est estimée à 1100 mAh, ce qui correspond, pour 2 heures d'utilisation, à une capacité de 2200 mAh. Une marge importante a été intégrée lors du calcul de la consommation des moteurs en considérant le courant maximal. De plus, il faut prendre en compte que les moteurs ne fonctionneront pas en continu, la consommation moyenne réelle étant donc largement inférieure à cette valeur.

Pour répondre à ces besoins en autonomie et en capacité, des batteries adaptées ont été sélectionnées. Il s'agit de cellules 18650 présentant une tension nominale de 3,7 V et une capacité de 2000 mAh. Deux cellules sont montées en série, ce qui permet d'obtenir une tension totale de 7,4 V, conforme aux exigences d'alimentation du système.

Livrable 2 (Mise en œuvre sur breadboard)

Schéma de câblage

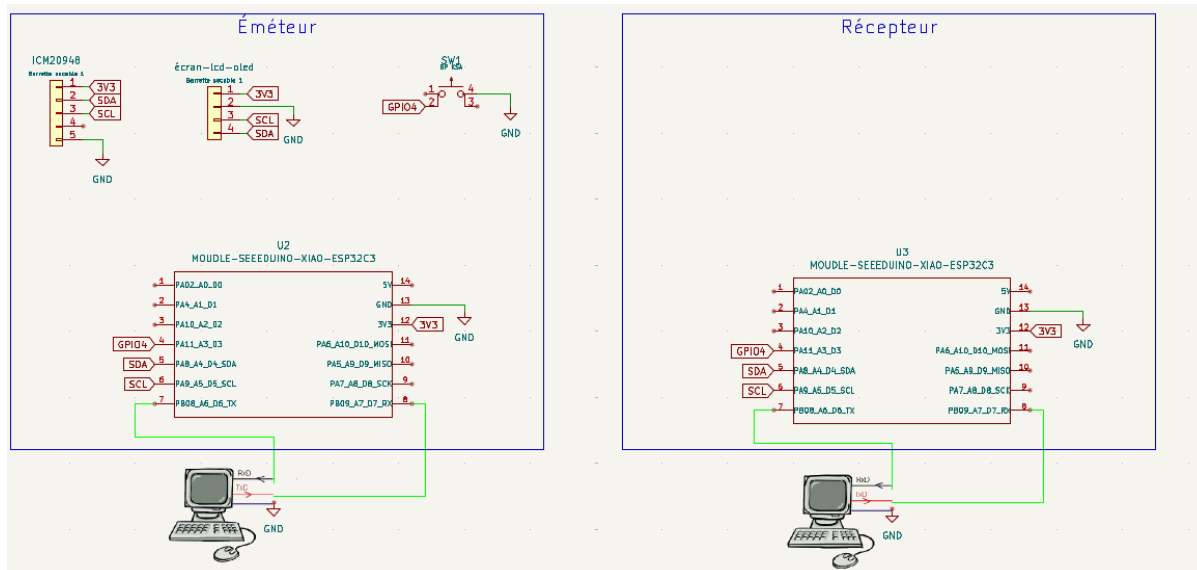
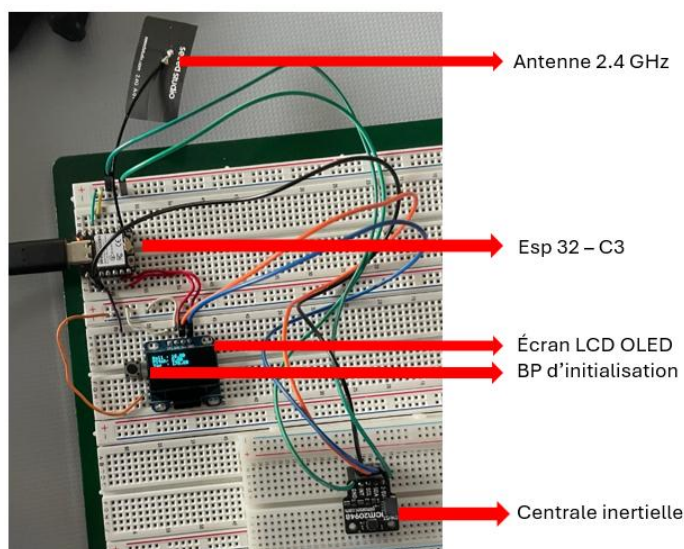


Figure 31 : schéma de câblage de la partie réceptrice et émettrice

Réalisation

Câblage de la partie émetteur



Câblage de la partie récepteur

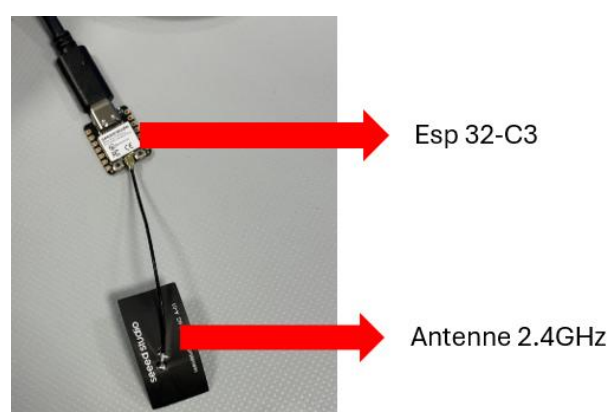


Figure 32 : image du câblage de la partie réceptrice et émettrice

Livrable 3

Étude des communications réseaux

Différentes méthodes : http, WebSocket, Point à point sont possibles lors de la communication entre deux ESP32, il est donc essentiel de les connaître afin de pouvoir choisir le plus adapté.

Fonctionnement HTTP avec ESP32

Architecture :

- Client / Serveur
 - L'un des ESP32 joue le rôle de client (initie la communication).
 - L'autre joue le rôle de serveur (attend et traite les requêtes).
1. Connexion réseau :
 - Les deux ESP32 doivent être connectés au même réseau Wi-Fi.
 - Chaque ESP32 se voit attribuer une adresse IP.
 2. Échange de données :
 - Le client envoie une requête HTTP à l'adresse IP du serveur.
 - Le serveur lit la requête, effectue une action ou collecte des données.

Caractéristiques :

- Communication unidirectionnelle par requête : le client initie toujours.
- Basée sur une connexion temporaire : chaque échange implique une ouverture, un traitement, puis une fermeture de connexion TCP.
- Les messages sont structurés selon le protocole HTTP (il respecte toute la concaténation).

En annexe A vous pouvez trouver un code pour communiquer avec ce protocole via Arduino IDE

Fonctionnement Web socket avec ESP32

Architecture :

- Relation Client / Serveur, mais avec une connexion persistante et bidirectionnelle.
- Les deux ESP32 se connectent initialement via HTTP, mais passent ensuite à un mode WebSocket.

1. Appairage :

- Un handshake* est effectué par le client (via HTTP).
- Une fois accepté, la connexion est changée vers WebSocket.
- Une connexion TCP persistante est alors maintenue entre les deux ESP32.

2. Échange de données :

- Chaque ESP32 peut envoyer ou recevoir des messages à tout moment.
- WebSocket s'appuie, comme HTTP, sur une connexion TCP dans la pile TCP/IP (Application → TCP → IP → Wi-Fi)
- Aucune reconnexion n'est nécessaire pour chaque échange.

Caractéristiques :

- Communication asynchrone et bidirectionnelle : les deux parties peuvent initier l'échange.
- Temps de latence très bas, car la connexion reste ouverte.
- Parfait pour des flux continus d'informations ou des applications interactives.

*En informatique, en télécommunications et dans les domaines associés, le handshaking (ou handshake ; en français, établissement d'une liaison) est un processus automatisé de négociation qui établit les paramètres d'une communication entre deux entités avant que la communication commence.

Fonctionnement Point à point avec ESP32

Architecture :

- Aucun routeur ou point d'accès nécessaire.
- Communication directe entre appareils.
- Chaque ESP32 connaît l'adresse matérielle (MAC) de l'autre.
- Un des ESP32 peut agir comme émetteur, l'autre comme récepteur, ou les deux peuvent alterner.

Fonctionnement (avec protocole ESP-NOW) :

1. Appairage :

- Chaque ESP32 configure son interface radio pour échanger directement avec un autre appareil connu via son adresse MAC.
- Pas d'adresses IP ni de négociation réseau traditionnelle.

2. Transmission des données :

- L'émetteur construit un petit paquet de données binaires ou encodées.
- Le paquet est envoyé directement, de façon très rapide, sur la couche radio.
- Le récepteur traite l'information à réception.
- L'ESP32 émetteur fait :
 1. Choisir les données utiles.
 2. Les regrouper proprement.
 3. Les convertir en format binaire ou encodé.
 4. Les envoyer

Caractéristiques :

- Communication bidirectionnelle.
- Communication très rapide et efficace : temps de transmission quasi instantané.
- Pas de réseau Wi-Fi requis : les ESP32 peuvent fonctionner isolés.
- Messages de petite taille (environ 250 octets maximum).

En annexe B se trouve un exemple de code permettant une communication point à point

Conclusion

La sélection du protocole ESP-NOW s'est justifiée par sa capacité à établir une connexion directe entre les deux ESP32 sans nécessité d'infrastructure réseau, offrant ainsi une latence quasi inexistante et un impact logiciel minimal. ESP-NOW, contrairement à une communication HTTP ou WebSocket qui requiert un réseau Wi-Fi et une couche protocolaire plus complexe, permet l'envoi de petits paquets de données en quasi-temps réel. Ceci est particulièrement adapté pour le transfert constant des angles d'orientation de la main vers le robot, cet envoie constant assure que même si un paquet est perdu il n'y a pas de répercussion directe sur le système. En termes de commande, cette option se manifeste par une réponse plus rapide du robot aux actions de l'utilisateur, une architecture allégée (absence de serveur ou de routeur à gérer) et une consommation énergétique plus faible côté télécommande, ce qui renforce la fiabilité et la praticité du système dans un contexte de projet embarqué.

Livrable 4

Récupération et mise en application des données de l'IMU

1.1 La récupération des données issues du capteur IMU

La première étape consiste à réussir à communiquer avec le capteur (ICM20948). Pour cela, nous développons un code simple qui lit les informations reçues sur le bus I²C et les écrit sur le port série de l'Arduino. Nous pourrions ensuite visualiser ces données sur un ordinateur grâce à l'IDE Arduino. Le code utilisé pour réaliser cette étape est présenté ci-dessous.

```
#include <Wire.h>
#include <ICM20948_WE.h>
#define ICM20948_ADDR 0x68
ICM20948_WE myIMU(ICM20948_ADDR);

void setup() {
  Serial.begin(115200);
  delay(1000);

  Wire.begin(6, 7); // Utilise les broches SDA/SCL
  Wire.setClock(10000);

  if (!myIMU.init()) {
    while (1); // stop si init échoue
  }
  // --- activation modules gyroscope accéléromètre---
  myIMU.enableAcc(true);
  myIMU.enableGyr(true);
  myIMU.enableCycle(ICM20948_NO_CYCLE);
  myIMU.enableLowPower(false);
  myIMU.setAccRange(ICM20948_ACC_RANGE_2G);
  myIMU.setGyrRange(ICM20948_GYRO_RANGE_250);
  // --- magnétomètre ---
  if (myIMU.initMagnetometer()) {
    myIMU.setMagOpMode(AK09916_CONT_MODE_100HZ);
  }
  // --- calibration ---
  Serial.println("Calibration du gyroscope et accelerometre...");
  myIMU.autoOffsets(); // mesure les offsets gyro/acc
  Serial.println("Calibration terminée !");
}

void loop() {
  xyzFloat acc, gyr, mag;
  myIMU.readSensor(); // met à jour les registres
  myIMU.getGValues(&acc); // en g
  myIMU.getGyrValues(&gyr); // en degrés/s
  myIMU.getMagValues(&mag); // en µTesla
  Serial.printf("A[g]: %.3f %.3f %.3f | G[°/s]: %.3f %.3f %.3f | M[µT]: %.3f %.3f %.3f\n",
    acc.x, acc.y, acc.z,
    gyr.x, gyr.y, gyr.z,
    mag.x, mag.y, mag.z);

  delay(300);
}
```

La fréquence de communication I²C est fixée à 10 kHz en se basant sur la fréquence d'échantillonnage interne maximale du gyroscope (9 kHz), celle-ci permet de mettre à jour régulièrement les mesures. La forme des signaux SCL et SDA observée à l'oscilloscope confirme le bon déroulement des transactions I²C entre l'ESP32-C3 et l'ICM20948, comme l'illustre la capture présentée où 68 est l'adresse de l'ICM

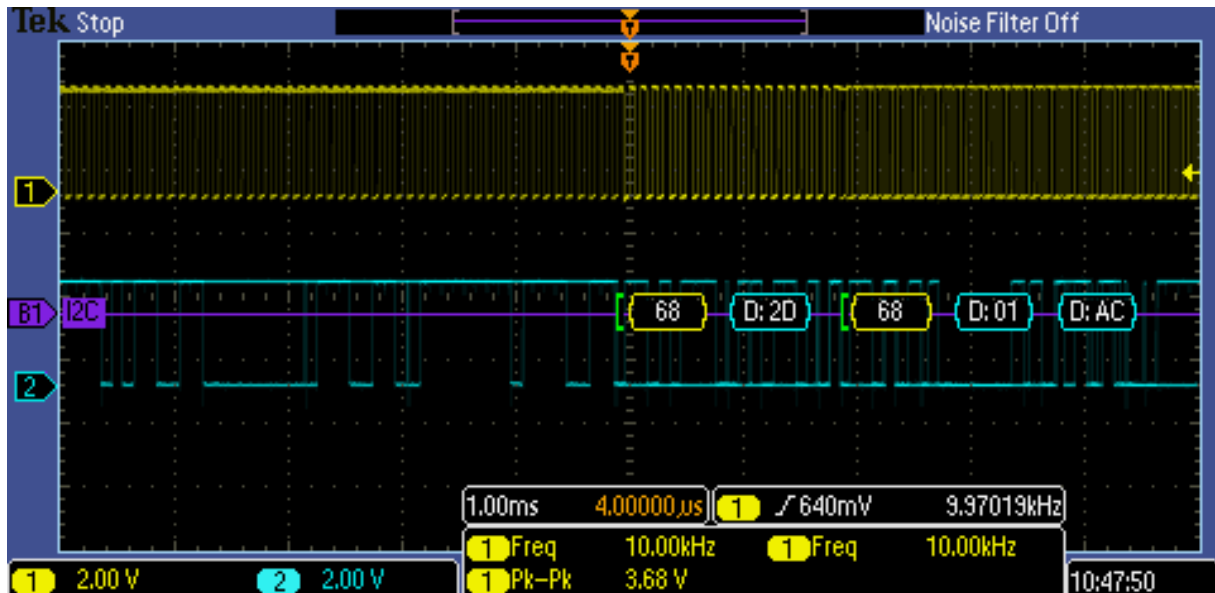


Figure 33 : capture d'écran d'oscilloscope affichant la trame de communication

1.2 Estimation d'orientation avec le filtre de Mahony

Un système AHRS (Attitude and Heading Reference System) est mis en œuvre afin d'estimer en continu l'orientation du capteur dans l'espace, sans recours à un repère externe autre que le champ magnétique terrestre.

Le filtre de Mahony constitue un algorithme de fusion inertielle destiné à estimer l'attitude (roulis, tangage, lacet) à partir d'un IMU 9 axes. Son principe repose sur une boucle de correction de type proportionnel-intégral (PI) qui compense la dérive du gyroscope en s'appuyant sur des vecteurs de référence mesurés (gravité et champ magnétique).

L'orientation interne du système est représentée sous forme de quaternion, puis éventuellement convertie en angles d'Euler pour l'affichage ou la génération des consignes de commande.

L'estimation d'orientation se décompose en deux étapes :

1. Prédiction

Les vitesses angulaires issues du gyroscope sont intégrées afin de mettre à jour le quaternion d'orientation. Cette étape fournit une estimation très réactive, capable de suivre rapidement les mouvements imposés au capteur, mais reste sujette à une dérive progressive en raison des biais et du bruit du gyroscope.

2. Correction

Les vecteurs de référence mesurés par l'accéléromètre (direction de la gravité) et par le magnétomètre (direction du champ magnétique terrestre) sont comparés aux directions attendues en fonction de l'orientation estimée. L'erreur entre vecteurs mesurés et vecteurs estimés est injectée dans un correcteur proportionnel afin de corriger instantanément le quaternion, et dans un correcteur intégral afin de compenser progressivement les biais lents du gyroscope.

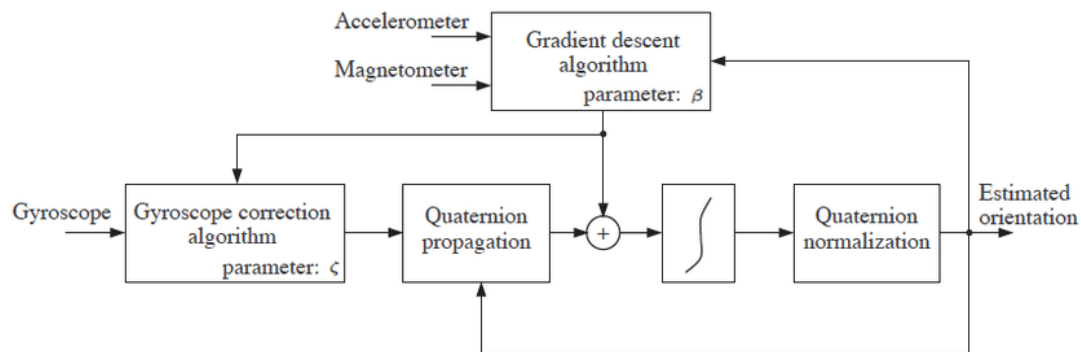


Figure 34 : *schéma simplifié du fonctionnement du filtre de Mahony*

Cette structure prédiction + correction permet d'obtenir une orientation stable, y compris avec un IMU MEMS telle que l'ICM-20948. Dans le projet, les 9 mesures de l'IMU sont fournies au filtre de Mahony, qui calcule en temps réel les angles de tangage, roulis et lacet utilisés pour la visualisation 3D

1.4 Rendu 3D temps réel d'un cube orienté selon les données de l'IMU

En lien se trouve la vidéo exemple : <https://youtu.be/pgnMbv1D5Xs>, en annexe C le code HTML du cube et en annexe D le code émetteur du cube

Le code HTML :

Ce programme implémente le contrôle d'un cube en 3D via une interface web, en exploitant des flux de données série.

Les composants principaux sont les suivants :

Structure HTML et p5.js : L'application utilise la bibliothèque p5.js pour générer un cube.

Interface série : Un sélecteur permet d'établir une connexion série entre l'ordinateur et un dispositif externe, tel qu'un microcontrôleur, afin de réceptionner des données.

Traitement des données : Après établissement de la liaison série, les informations transmises au format "tangage, lacet, roulis" respectivement pour les axes x y et z sont acquises en continu.

Mécanisme de rotation : Les valeurs angulaires de roulis, tangage et lacet provenant du port série déterminent l'orientation spatiale du cube. L'application procède à des transformations géométriques successives autour des trois axes principaux à chaque cycle de rafraîchissement : - rotateX(radians(tangage)) : Rotation suivant l'axe horizontal X - rotateY(radians(lacet)) : Rotation suivant l'axe vertical Y - rotateZ(radians(roulis)) : Rotation suivant l'axe de profondeur Z.

Ce code constitue donc une interface de graphique en temps réel qui traduit des mesures physiques de l'IMU en rotation d'un cube.

Le code émetteur :

Le programme situé en annexe D assure la lecture, la calibration et la fusion des données issues du capteur inertiel ICM-20948. Son rôle est de fournir les informations d'orientation (roulis, tangage et lacet) de l'IMU au système de visualisation (ordinateur).

L'architecture du code s'articule autour d'éléments principaux :

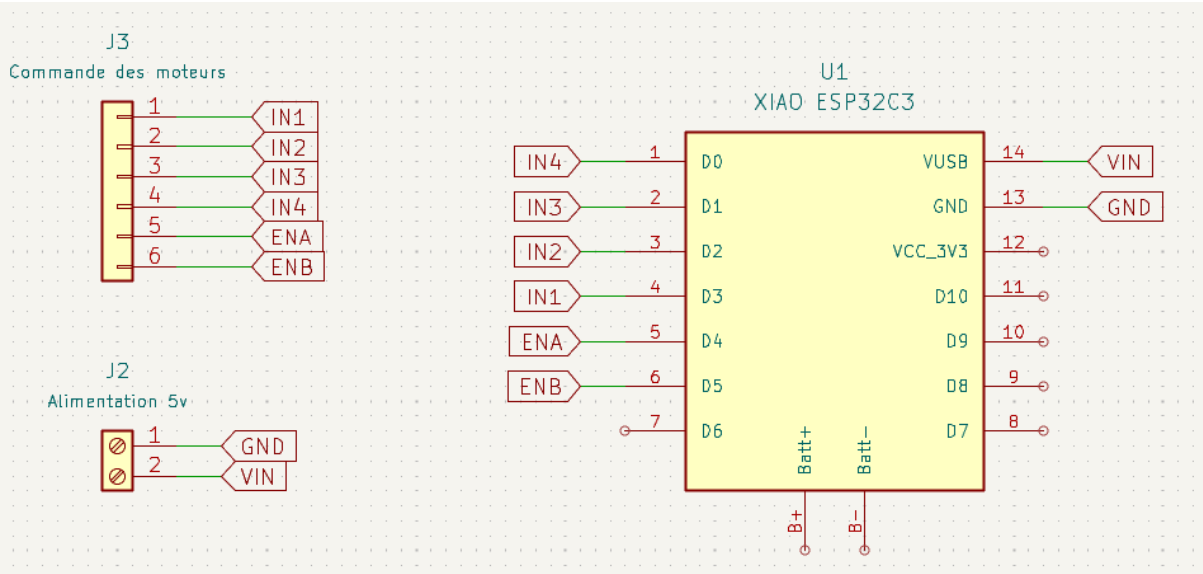
- Acquisition des mesures brutes : les données d'accélération, de vitesse angulaire et de champ magnétique sont récupérées via le bus I²C à partir du capteur ICM-20948.
- Gestion de l'interruption : la broche d'interruption de l'ICM est reliée à l'ESP32-C3 afin de déclencher automatiquement l'acquisition à chaque nouvelle mesure disponible. Ce fonctionnement par interruption garantit une mise à jour synchrone à la fréquence d'échantillonnage du capteur.
- Calibration : un bouton de commande permet de lancer à tout moment un calibrage de l'IMU. Le code intègre une fonction d'antirebond pour assurer une détection fiable du bouton, afin d'éviter les perturbations lors de l'appui.
- Filtrage et fusion de données : le filtre de Mahony est implémenté directement dans le code. Il fusionne les neuf mesures de l'IMU (3 accéléromètres + 3 gyroscopes + 3 magnétomètres) afin d'estimer un quaternion d'orientation. Ce quaternion est ensuite converti en angles exprimés en degrés (roulis, tangage, lacet).
- Communication série : les valeurs calculées sont transmises en continu sur le port série à 115200 bauds, sous forme de chaînes de caractères. Ces données sont ensuite interprétées par l'ordinateur, qui met à jour en temps réel l'orientation du cube 3D.

En résumé le microcontrôleur gère la calibration, la lecture capteur, la fusion de données et la communication, tandis que le poste récepteur (ordinateur) se limite à la visualisation graphique.

Livrable 5

Conception carte électronique réceptrice (PCB)

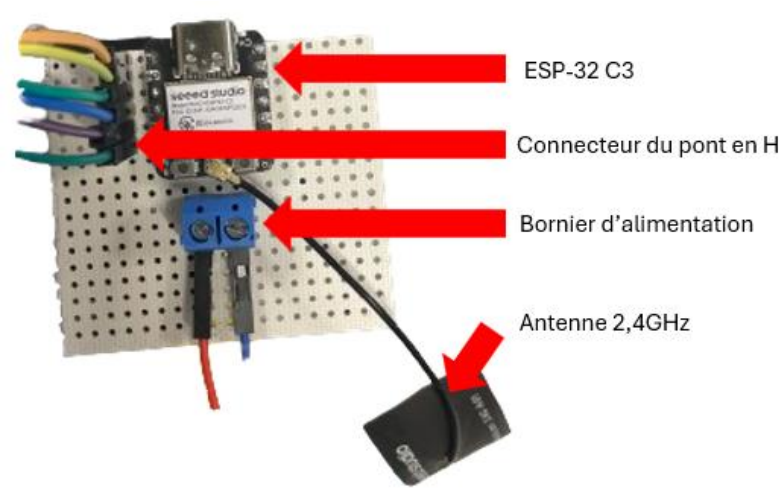
Schéma structurel :



Nomenclature :

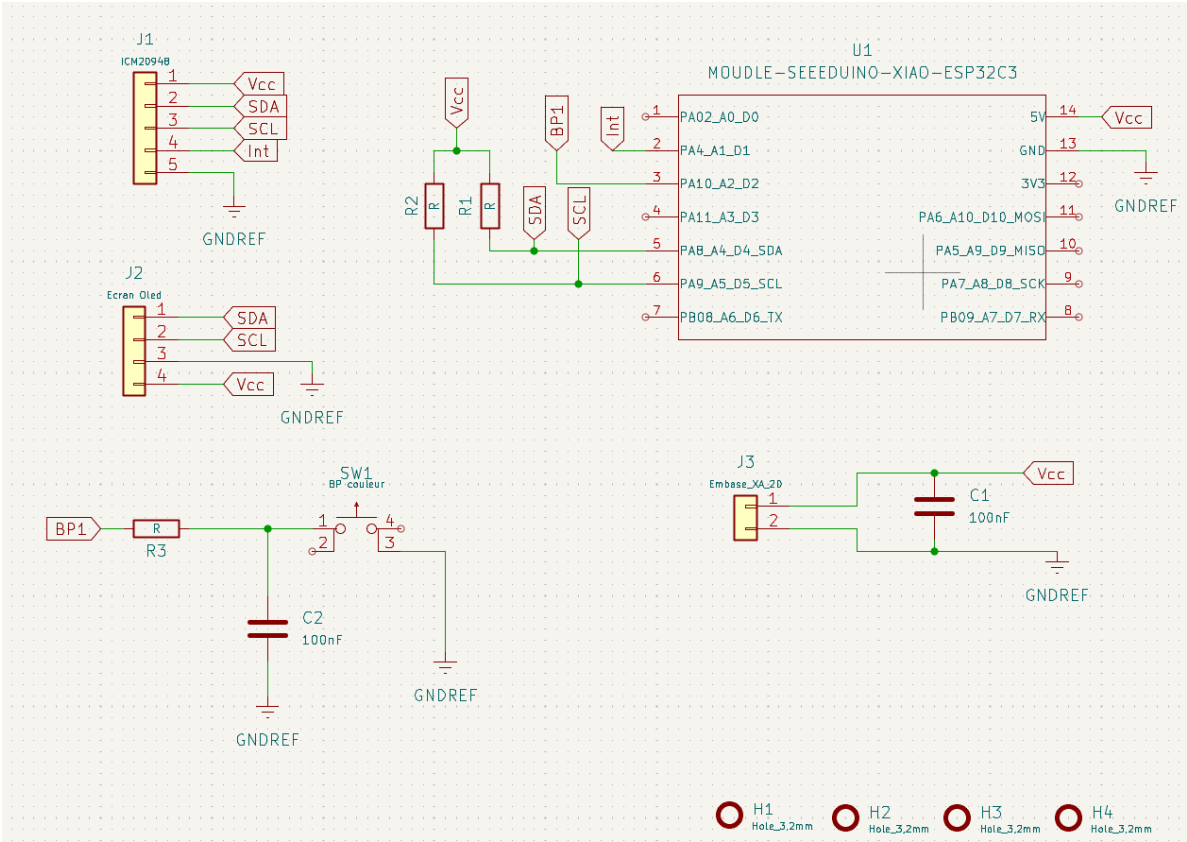
Parts	Value	Qty	Description	Package
U1	X	1	Esp32-C3 XIAO dev-kit	X
J1	X	1	Barrette sécable	BARRETTE_2
J2	X	1	Bornier A2p	AK500_2

Réalisation de la carte perforée :



Conception carte électronique émettrice (PCB)

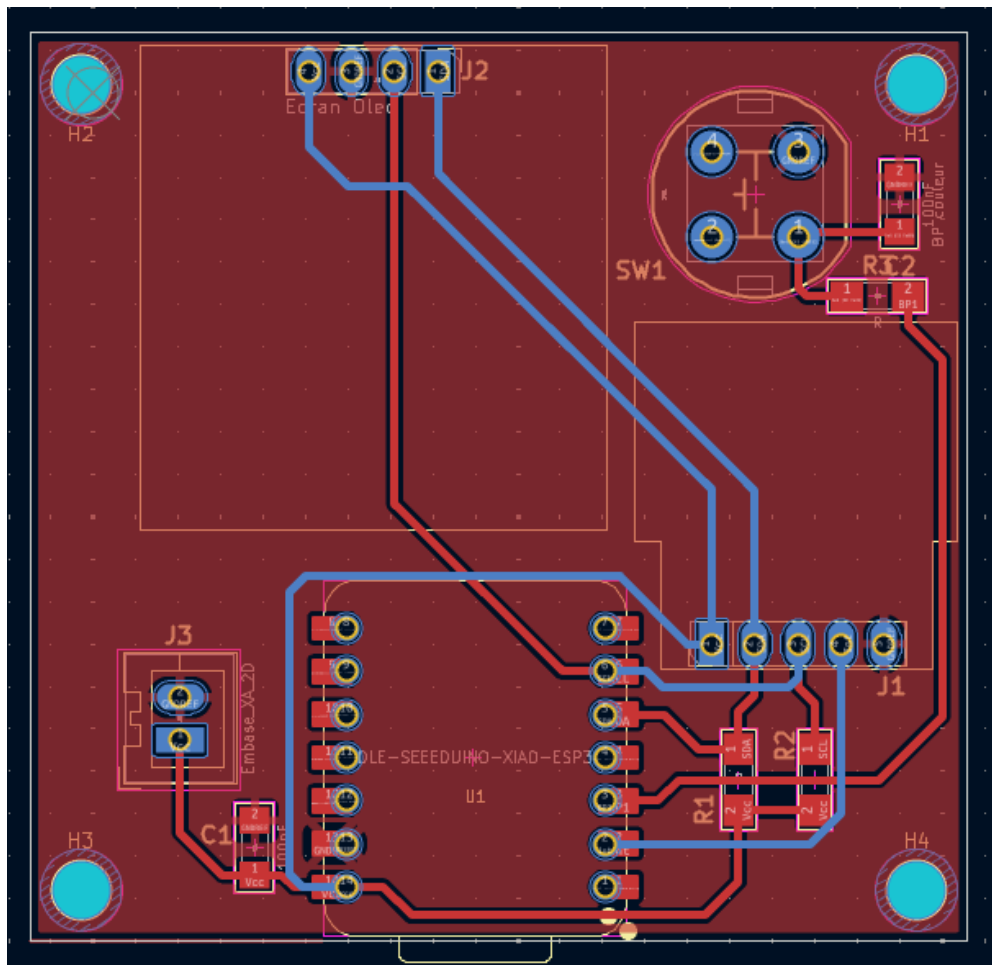
Schéma structurelle :



Nomenclature :

Parts	Value	Qty	Description	Package
U1	X	1	Esp32-C3 XIAO dev-kit	X
J1	X	1	Barrette sécable	BARRETTE_2
J2	X	1	Barrette sécable	BARRETTE_2
J3	X	1	Contact femelle	XA_2D
R1, R2	4.7kΩ	2	Résistance couche métallique	RCMS1206
R3	10kΩ	1	Résistance couche métallique	RCMS1206
C1, C2	100nF	2	Condensateur non polarisé	CCMS1206
SW1	X	1	Bouton poussoir	DT6
H1, H2, H3, H4	X	4	Trous de fixation	Hole_3,2mm

Routage :



Réalisation du PCB :



Procédure de test :

Test structurel :

Première étape : Avant la pose des composants.

Objectif : Recherche de court-circuit et/ou de coupure des pistes cuivre :

- Test visuel
- Test à l'ohmmètre (testeur de continuité)

Test BON 	Évolution à apporter sur la carte : Aucune	Évolution à apporter sur le schéma : Aucune
Test MAUVAIS		

Deuxième étape : Au fur et à mesure du câblage des composants, carte hors tension.

Objectif : Contrôler la qualité des soudures réalisées sur la carte.

- Test à l'ohmmètre (en testeur de continuité) de la continuité / isolation des alimentations / des pistes.
- Test visuel : qualité des brasures.

Test après la pose des composants traversants	Évolutions à apporter sur la carte : Aucune	Évolutions à apporter sur la carte : Aucune
Test BON 		
Test MAUVAIS		

Troisième étape : Mise sous tension de la carte.

Objectif : Vérifier la bonne distribution de l'alimentation sur l'ensemble de la carte.

- Alimenter la carte via la batterie 3.7 V
- Vérifier la tension sur les broches d'alimentation
- Vérifier la tension aux bornes des composants (IMU, ESP32, ÉCRAN OLED)

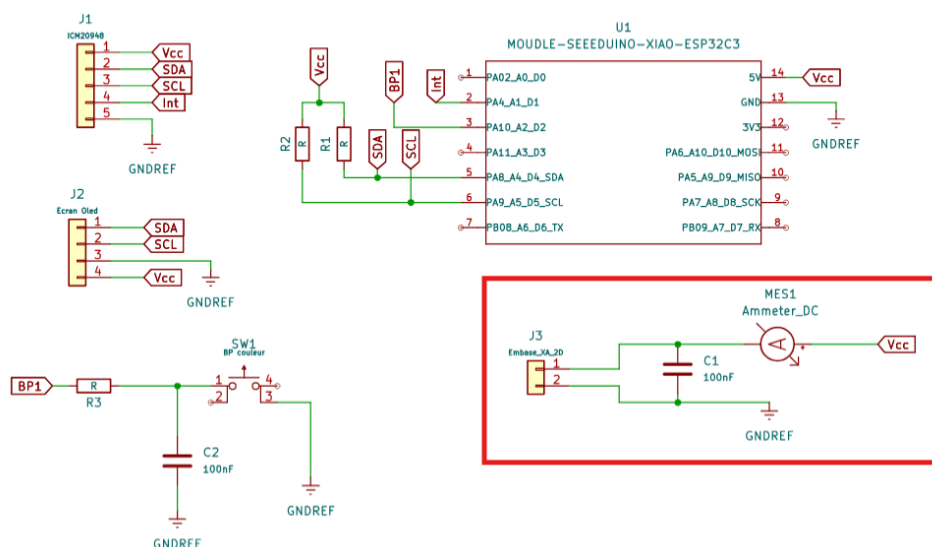
Test BON 	Évolution à apporter sur la carte : Aucune	Évolution à apporter sur le schéma : Aucune
Test MAUVAIS		

Quel est le courant consommé par la carte ?

Après ouverture du circuit et installation d'un ampèremètre conformément au schéma ci-dessous, une consommation de 160 mA a été mesurée pour la carte.

Les calculs théoriques réalisés à partir des fiches techniques conduisaient à une consommation d'environ 300 mA pour la partie émettrice. Les mesures effectuées lors de la procédure de test ont montré un courant réel d'environ 160 mA en fonctionnement nominal, nettement inférieur à cette première estimation.

Cette différence s'explique par le fait que tous les composants ne fonctionnent pas en permanence dans leurs pires conditions de consommation. Sur la base de la valeur mesurée, l'autonomie attendue de la télécommande est donc réévaluée à environ 4 heures, ce qui reste compatible avec les besoins du projet.



Livrable 6

Contrôle du véhicule selon les données renvoyées par l'IMU

Afin de contrôler le véhicule de manière progressive et proportionnelle à l'inclinaison de la main, une commande PWM appliquée au pont en H L298N présent sur le robot est mise en œuvre.

Ce module permet de piloter les moteurs à partir de trois signaux par voie : une entrée de PWM (ENA/ENB) pour la vitesse et deux entrées logiques (IN1/IN2, IN3/IN4) pour le sens de rotation (voir figure ci-contre).

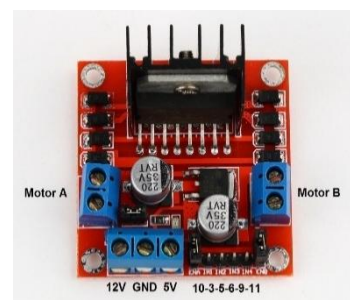


Figure 35 : [Image LN298N](#)

Les données d'orientation (lacet et roulis) calculées par l'IMU et transmises via ESP-NOW sont d'abord filtrées : une zone morte est appliquée sur le lacet afin d'ignorer les petites oscillations de la main et d'éviter des rotations parasites du robot autour de la position neutre.

À partir du lacet (avance/recul) et du roulis (gauche/droite), deux vitesses logiques, notées `speedLeft` et `speedRight`, sont calculées de manière à combiner translation et rotation pour les roues gauche et droite avec les formules $\text{speedLeft} = \text{lacet} - \text{roulis}$ et $\text{speedRight} = \text{lacet} + \text{roulis}$

Pour la génération des signaux PWM, les valeurs de vitesse sont transformées en intensités positives : une zone morte (DEADZONE) met la consigne à zéro sous un certain seuil, puis une fonction de mise à l'échelle map convertit la vitesse en rapport cyclique compris entre une valeur minimale et 255, avant limitation de sécurité à l'aide de la fonction constrain.

Les rapports cycliques obtenus (`dutyLeft` et `dutyRight`) sont appliqués aux entrées ENA et ENB du L298N, tandis que le signe de la vitesse détermine le sens de rotation via les fonctions `forward` ou `backward` de la bibliothèque de commande des moteurs (LN298N.h).

Cette stratégie permet d'obtenir un comportement à la fois intuitif et progressif : le robot reste immobile autour de la position neutre de la main, puis sa vitesse et sa direction évoluent de manière continue avec l'inclinaison, ce qui améliore la précision de pilotage et la sécurité d'utilisation.

Le code complet de la commande des moteurs est présenté en annexe G du rapport.

Livrable 7

Création d'une Interface Homme Machine (IHM)

1.1 Affichage des données sur un écran OLED

Le fonctionnement de l'écran est assuré par une bibliothèque logicielle qui simplifie la communication entre le microcontrôleur et l'écran. Cette bibliothèque fournit des fonctions pour initialiser l'écran, effacer l'affichage, dessiner des formes géométriques (lignes, cercles, rectangles), afficher du texte avec différentes tailles de caractères, et positionner précisément les éléments sur l'écran. Elle gère également le protocole de communication I²C ou SPI selon le modèle d'écran, ce qui permet d'envoyer les données d'affichage de manière fiable. Grâce à cette bibliothèque, l'affichage devient modulable et flexible, facilitant l'adaptation du contenu à l'application souhaitée.

Code qui permet d'initialiser l'écran et afficher 'Initialisation...'

```
if (!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) {  
    Serial.println("Échec init OLED");  
    while (true); // Stoppe si écran non détecté  
}  
  
display.clearDisplay();    // Efface l'écran  
display.setTextSize(1);    // Taille du texte  
display.setTextColor(SSD1306_WHITE); // Couleur du texte  
display.setCursor(0, 0);    // Position du curseur  
display.println("Initialisation..."); // Texte à afficher  
  
display.display();    // Met à jour l'écran  
delay(1000);    // Pause d'une seconde
```

1.2 Création d'une interface graphique

Pour avoir une meilleure expérience utilisateur, il est important que l'interface graphique soit claire et présente les informations importantes de manière lisible. Cette interface s'inspire des affichages tête haute utilisés dans les cockpits des avions de chasse (voir image ci-contre). Ces affichages permettent aux pilotes de connaître l'orientation de l'avion par rapport à l'horizon.



Figure 36 : [affichage tête haute](#)

Dans le cadre de cette application, l'interface affichera la direction de déplacement du robot, les coordonnées de l'IMU sur les axes X et Y, ainsi que l'état de la connexion Wi-Fi avec le robot. L'interface graphique a été générée en C++ avec l'aide de Claude, une intelligence artificielle, qui a permis de concevoir cette interface de manière propre et efficace. Voir l'image ci-contre représentant l'interface graphique réalisée.



Figure 37 : IHM partie émettrice

Se référer à l'annexe G pour le code permettant la création de l'interface graphique.

Au-delà de l'affichage pour l'utilisateur, l'IHM a joué un rôle important dans le débogage du système. L'affichage des angles de roulis, tangage et lacet a notamment permis de détecter des inversions d'axes lors des premiers essais, ainsi que de vérifier visuellement la cohérence entre le mouvement de la main et les valeurs fournies par le filtre de Mahony.

De même, l'indication de direction du robot à l'écran a facilité le réglage des seuils de zones mortes et des signes de commande des moteurs, en mettant en évidence les cas où le robot démarrait ou tournait alors que la main était proche de la position neutre.

Livrable 8

Réalisation du code final de la télécommande

Pour le code final, les différents programmes développés au fil des livrables ont été regroupés et adaptés afin de fonctionner au sein d'un unique code.

La carte émettrice assure l'acquisition de l'orientation de la main, l'affichage d'informations sur l'écran OLED et l'envoi des données vers la carte de commande du robot.

La bibliothèque ICM20948WE est utilisée pour configurer l'accéléromètre, le gyroscope et le magnétomètre, calibrer le capteur et calculer directement les angles de tangage et de roulis, ce qui simplifie fortement la partie traitement de l'IMU par rapport au code initial du cube 3D.

Sur cette base, le code de communication point à point ESP-NOW est intégré afin de transmettre en temps réel les angles calculés vers l'ESP32 récepteur, tandis que l'écran OLED affiche à la fois les informations utiles au débogage et une indication de direction pour l'utilisateur.

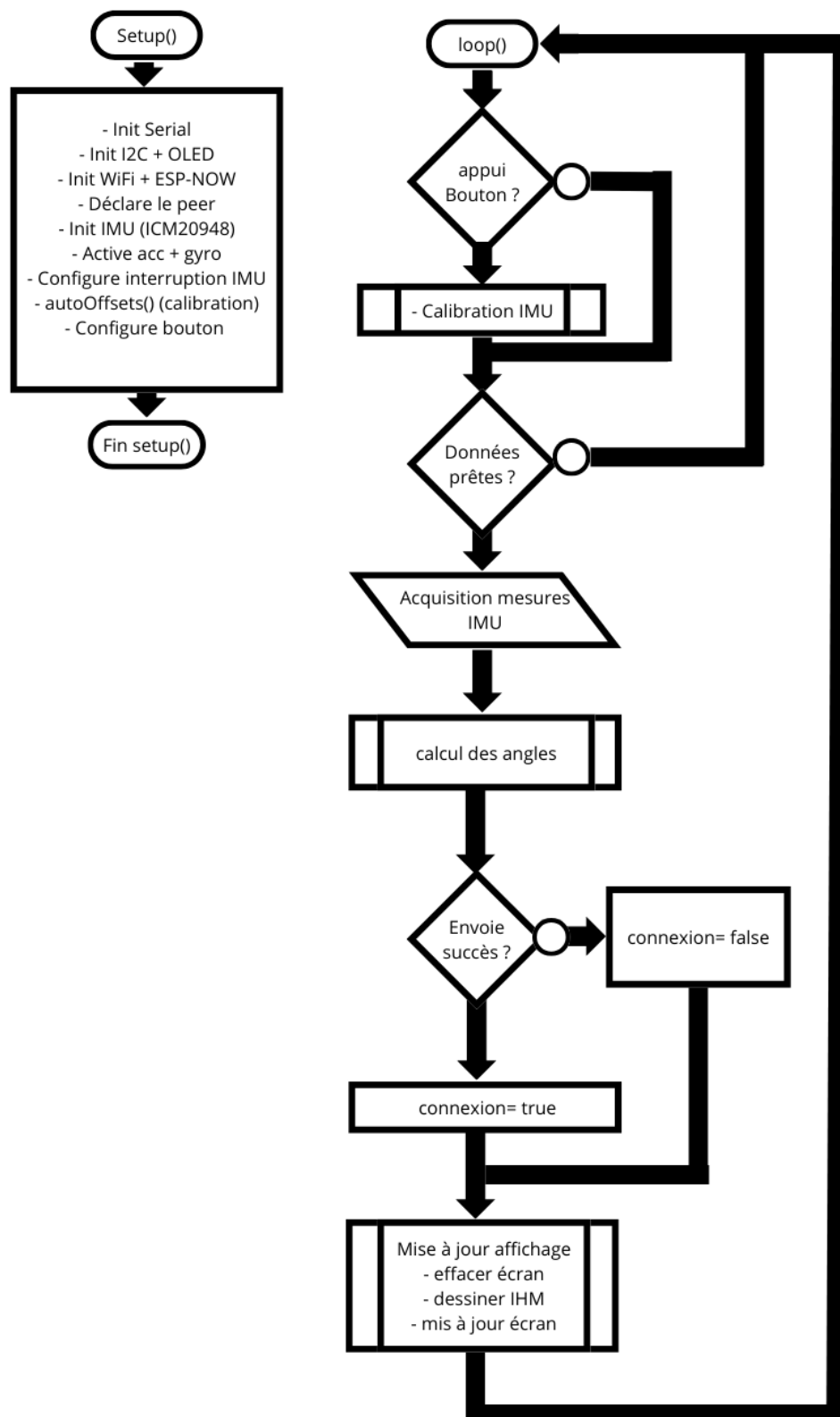
L'organigramme présenté à la page suivante illustre le déroulement complet du code émetteur, depuis l'initialisation jusqu'à l'envoi des données au robot, et le code complet est fourni en annexe F du rapport.

Les tests réalisés ont montré que la combinaison IMU + filtre de Mahony + ESP-NOW permet de conserver une orientation stable de la main sur toute la plage de mouvements utilisés pour la commande, sans dérive gênante pendant plusieurs minutes d'utilisation continue.

En pratique, les principales limites observées concernent les déplacements diagonaux, moins précis et parfois inexistant que les mouvements purement avant/arrière ou gauche/droite, ce qui a conduit à ajuster les coefficients de combinaison lacet/roulis et les zones mortes afin d'améliorer la commande.

Organigramme du code :

Cet organigramme présente la structure du code émetteur : initialisation des périphériques, acquisition et filtrage des données de l'IMU, envoi des angles d'orientation vers la carte réceptrice via ESP-NOW puis mise à jour de l'affichage OLED.



Réalisation de la commande du robot

Ce code est basé sur le programme récepteur de la communication point à point, légèrement modifié afin de restreindre la réception des signaux uniquement à notre émetteur grâce à son adresse MAC.

Ci-dessous, le code permettant le filtrage par adresse MAC.

```
// Fonction appelée à la réception d'un message ESP-NOW
void OnDataRecv(const esp_now_recv_info_t *recv_info, const uint8_t *data, int len)
{
    // Adresse MAC autorisée à envoyer des commandes
    uint8_t allowedMAC[] = {0x58, 0x8c, 0x81, 0x9f, 0xaa, 0x1c};

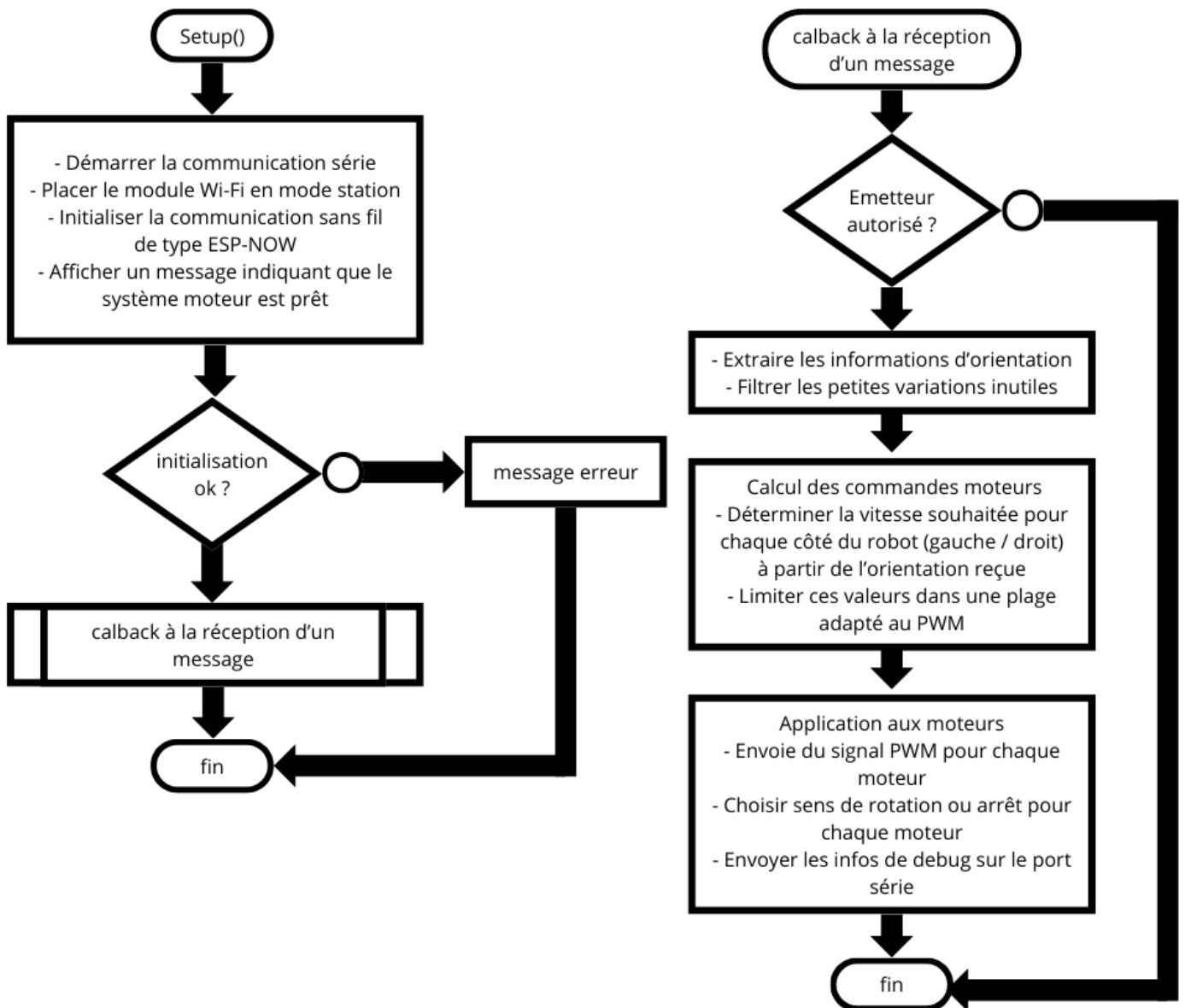
    // message ignorer si l'adresse mac émettrice n'est pas la bonne
    if (memcmp(recv_info->src_addr, allowedMAC, 6) != 0) {
        Serial.println("Message ignoré : MAC non autorisée.");
        return;
    }
}
```

À ce code est ajouté le code du livrable 6 qui permet la commande des moteurs à partir des orientations reçues via la communication point à point.

L'organigramme présenté à la page suivante illustre le déroulement complet du code récepteur, depuis l'initialisation jusqu'à la commande des moteurs, et le code complet est fourni en annexe G du rapport.

Organigramme du code :

Cet organigramme met en évidence la chaîne complète côté récepteur : réception des angles envoyés par l'IMU, calcul des vitesses gauche et droite du robot, génération des signaux PWM et commande du pont en H L298N pour piloter les moteurs.



Conclusion et perspectives

Ce projet a abouti à la réalisation d'une chaîne complète de télécommande inertielle d'un robot mobile : la centrale IMU ICM-20948 mesure l'orientation de la main, le filtre de Mahony fournit des angles exploitables en temps réel, la liaison ESP-NOW transmet ces informations à un second ESP32-C3 qui génère les signaux PWM de commande des moteurs via un pont en H.

Les essais ont montré que le robot réagit de manière fluide aux mouvements de la main, avec une latence faible et un comportement globalement stable, validant ainsi l'architecture matérielle et logicielle proposée.

Sur le plan des apprentissages, ce travail a permis d'approfondir la compréhension du fonctionnement interne d'une IMU MEMS (accéléromètre, gyroscope, magnétomètre) et de ses principales sources d'erreurs, ainsi que de découvrir concrètement un AHRS à travers le filtre de Mahony et la fusion de capteurs 9 axes.

Il a également renforcé les compétences en conception de systèmes embarqués : choix et justification des composants, dimensionnement énergétique, réalisation de PCB, mise en œuvre d'une communication radio point à point et développement d'une commande de moteurs adaptée à un pilotage gestuel.

Malgré ces résultats, plusieurs limites ont été mises en évidence. Le déplacement diagonal du robot reste moins précis que les mouvements purement avant/arrière ou gauche/droite, ce qui suggère un travail complémentaire sur la combinaison lacet/roulis et le réglage des zones mortes de plus par moment il a été constaté une latence entre la commande et le déplacement du robot sans raison apparente.

La gestion énergétique de la télécommande pourrait également être améliorée, notamment par l'ajout d'un bouton de mise sous/hors tension et par une optimisation plus fine des modes de veille de l'ESP32-C3.

Enfin, ce prototype ouvre plusieurs perspectives d'évolution centrées sur l'IMU. La détection de gestes (tap, double tap, inclinaisons rapides) permettrait d'enrichir la palette de commandes au-delà de la simple orientation continue, tandis qu'une commande « dynamique » basée sur l'intensité de l'accélération offrirait une correspondance plus naturelle entre la vivacité du mouvement de la main et l'accélération du robot. La mise en place d'un mode d'enregistrement, dans lequel l'IMU mémorise une séquence de mouvements que le robot rejoue ensuite en autonomie ouvrant ainsi la voie à des scénarios d'utilisation plus avancés.

Annexes

Annexes A : Code communication HTTP

Code serveur :

```
#include <WiFi.h>

const char* ssid = "ESP32_AP";

const char* password = "12345678";

WiFiServer server(80);

void setup() {

  Serial.begin(115200);

  WiFi.softAP(ssid, password); // Créer le point d'accès Wi-Fi

  Serial.print("Point d'accès créé : ");

  Serial.println(ssid);

  IPAddress IP = WiFi.softAPIP();

  Serial.print("Adresse IP AP : ");

  Serial.println(IP);

  server.begin();

  Serial.println("Serveur TCP démarré");

}

void loop() {

  WiFiClient client = server.available(); // écoute si un client se connecte

  if (client) {

    //Serial.println("Client connecté");

    while (client.connected()) {

      if (client.available()) {

        String data = client.readStringUntil('\n');

        Serial.print("Reçu : ");

        Serial.println(data); }}}}
```

Code client :

```
#include <WiFi.h>

const char* ssid = "ESP32_AP";

const char* password = "12345678";

const char* host = "192.168.4.1"; // IP par défaut du point d'accès ESP32
```

```
void setup() {  
  Serial.begin(115200);  
  WiFi.begin(ssid, password);  
  Serial.print("Connexion au réseau Wi-Fi...");  
  while (WiFi.status() != WL_CONNECTED) {  
    delay(500);  
    Serial.print(".");  
  }  
  Serial.println("Connecté au Wi-Fi !");  
  Serial.print("Adresse IP locale : ");  
  Serial.println(WiFi.localIP()); }  
void loop() {  
  WiFiClient client;  
  if (!client.connect(host, 80)) {  
    Serial.println("Connexion au serveur échouée");  
    delay(2000);  
    return;  
  }  
  client.println("data");  
  Serial.println("Message envoyé");  
  delay(20); }
```

Annexes B : Code communication Point à Point

Code receveur :

```
#include <esp_now.h>
#include <WiFi.h>
// Structure example to receive data
// Must match the sender structure
typedef struct struct_message {
    char a[32];
    int b;
    float c;
    bool d;
} struct_message;
// Create a struct_message called myData
struct_message myData;
// callback function that will be executed when data is received
void OnDataRecv(const esp_now_recv_info_t *recv_info, const uint8_t *data, int len) {
    memcpy(&myData, data, sizeof(myData));
    Serial.print("Bytes received: ");
    Serial.println(len);
    Serial.print("Char: ");
    Serial.println(myData.a);
    Serial.print("Int: ");
    Serial.println(myData.b);
    Serial.print("Float: ");
    Serial.println(myData.c);
    Serial.print("Bool: ");
    Serial.println(myData.d);
    Serial.println();
}
void setup() {
    // Initialize Serial Monitor
    Serial.begin(115200);

    // Set device as a Wi-Fi Station
    WiFi.mode(WIFI_STA);
    // Init ESP-NOW
    if (esp_now_init() != ESP_OK) {
        Serial.println("Error initializing ESP-NOW");
        return;
    }
    // Once ESPNow is successfully Init, we will register for recv CB to
    // get recv packer info
    esp_now_register_recv_cb(esp_now_recv_cb_t(OnDataRecv));
}
void loop() {
}
```

Code émetteur :

```
#include <esp_now.h>
#include <WiFi.h>
// REPLACE WITH YOUR RECEIVER MAC Address
uint8_t broadcastAddress[] = { 0x58, 0x8c, 0x81, 0xa0, 0x34, 0xfc };
// Structure example to send data
// Must match the receiver structure
typedef struct struct_message {
```

```

char a[32];
int b;
float c;
bool d;
} struct_message;
// Create a struct_message called myData
struct_message myData;
esp_now_peer_info_t peerInfo;
/*
// callback when data is sent
void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status) {
    Serial.print("\r\nLast Packet Send Status:\t");
    Serial.println(status == ESP_NOW_SEND_SUCCESS ? "Delivery Success" : "Delivery Fail");
}
*/

/*
static void OnDataSent(const esp_now_send_info_t *tx_info, esp_now_send_status_t status)
{
    Serial.print("\r\nLast Packet Send Status:\t");
    example_espnow_event_t evt;
    example_espnow_event_send_cb_t *send_cb = &evt.info.send_cb;
    if (tx_info == NULL) {
        ESP_LOGE(TAG, "Send cb arg error");
        return;
    }
    evt.id = EXAMPLE_ESPNOW_SEND_CB;
    memcpy(send_cb->mac_addr, tx_info->des_addr, ESP_NOW_ETH_ALEN);
    send_cb->status = status;
    if (xQueueSend(s_example_espnow_queue, &evt, ESPNOW_MAXDELAY) != pdTRUE) {
        ESP_LOGW(TAG, "Send send queue fail");
    }
}.
*/

static void OnDataSent(const esp_now_send_info_t *tx_info, esp_now_send_status_t status) {
    Serial.print("\r\nLast Packet Send Status:\t");
    Serial.println(status == ESP_NOW_SEND_SUCCESS ? "Delivery Success to remote" : "Delivery Fail to remote");
}

void setup() {
    // Init Serial Monitor
    Serial.begin(115200);
    // Set device as a Wi-Fi Station
    WiFi.mode(WIFI_STA);
    // Init ESP-NOW
    if (esp_now_init() != ESP_OK) {
        Serial.println("Error initializing ESP-NOW");
        return;
    }
    // Once ESPNow is successfully Init, we will register for Send CB to
    // get the status of Transmitted packet
    //Serial.println(type(OnDataSent));
    esp_now_register_send_cb(OnDataSent);
    // Register peer
    memcpy(peerInfo.peer_addr, broadcastAddress, 6);
    peerInfo.channel = 0;
    peerInfo.encrypt = false;
    // Add peer
    if (esp_now_add_peer(&peerInfo) != ESP_OK) {
        Serial.println("Failed to add peer");
        return;
    }
}

```

```
}  
void loop() {  
  // Set values to send  
  strcpy(myData.a, "dimitri");  
  myData.b = 15;  
  myData.c = 0;  
  myData.d = false;  
  // Send message via ESP-NOW  
  esp_err_t result = esp_now_send(broadcastAddress, (uint8_t *)&myData, sizeof(myData));  
  if (result == ESP_OK) {  
    Serial.println("Sent with success to the transmitter");  
  } else {  
    Serial.println("Error sending the data to the transmitter");  
  }  
  delay(2000);  
}
```

Annexes C : Code HTML cube 3D

```
<!DOCTYPE html>
<html lang="fr">
<head>
<meta charset="UTF-8">
<title>Cube 3D</title>
//<-- Importation de la librairie p5.js -->
<script src="https://cdnjs.cloudflare.com/ajax/libs/p5.js/1.7.0/p5.min.js"></script>
</head>
<body>
<script>
// Variables qui contiendront les angles envoyés en série
let roll = 0, pitch = 0, yaw = 0;
// Objet représentant le port série sélectionné par l'utilisateur
let port;
function setup() {
  // Création du canvas 3D (WEBGL = moteur graphique 3D de p5.js)
  createCanvas(600, 600, WEBGL);
  // Création d'un bouton pour lancer la connexion au port série
  let button = createButton('Connecter au port série');
  button.position(10, 10); // position du bouton sur la page
  button.mousePressed(connectSerial); // action au clic : demande d'accès au port série
}
function draw() {
  background(200); // couleur de fond du canvas
  lights(); // active un éclairage par défaut
  // Application des rotations selon les valeurs reçues
  rotateX(radians(pitch)); // rotation autour de l'axe X
  rotateY(radians(yaw)); // rotation autour de l'axe Y
  rotateZ(radians(roll)); // rotation autour de l'axe Z
  fill(150, 0, 200); // couleur du cube
  box(200); // dessine un cube de 200 px
}
// Fonction appelée lorsque l'utilisateur clique sur "Connecter au port série"
// Elle demande l'accès à un port série puis commence la lecture.
async function connectSerial() {
  try {
    // Ouvre une fenêtre permettant à l'utilisateur de choisir un port série
    port = await navigator.serial.requestPort();
    // Ouverture du port à 115200 bauds
    await port.open({ baudRate: 115200 });
    // Lancement du flux de lecture du port
    readSerial();
  } catch (err) {
    console.log('Erreur:', err); // si la connexion échoue
  }
}
// Fonction de lecture du port série en continu
// Elle utilise un TextDecoderStream pour transformer les octets reçus en texte.
async function readSerial() {
  // Décodeur de texte pour transformer les données binaires en chaîne de caractères
  const decoder = new TextDecoderStream();
  // On connecte la sortie du port série à l'entrée du décodeur
  port.readable.pipeTo(decoder.writable);
  // Création d'un lecteur de flux
  const reader = decoder.readable.getReader();
  while (true) {
    // Lecture des données en continu
    const { value, done } = await reader.read();
    if (done) break; // si le flux est terminé, on arrête
  }
}
```

```

    if (value) parseData(value); // sinon, on envoie les données à parseData()
  }
}
// Buffer servant à stocker les morceaux de lignes arrivant en plusieurs fragments
let buffer = "";
// Fonction chargée d'analyser les données série reçues.
// Les données arrivent parfois coupées, donc on reconstruit les lignes complètes.
function parseData(data) {
  buffer += data; // ajoute le dernier fragment reçu au buffer
  let lines = buffer.split('\n'); // découpe le buffer en lignes
  buffer = lines.pop(); // conserve la dernière ligne si elle est incomplète
  // Parcours de toutes les lignes complètes
  for (let line of lines) {
    // Nettoyage et découpe par virgule → format attendu : "roll,pitch,yaw"
    let vals = line.trim().split(',');
    // On vérifie qu'on a bien 3 valeurs
    if (vals.length == 3) {
      roll = parseFloat(vals[0]);
      pitch = parseFloat(vals[1]);
      yaw = parseFloat(vals[2]);
      // Affichage dans la console pour debug
      console.log("Parsed:", roll, pitch, yaw);
    }
  }
}
</script>
</body>
</html><!DOCTYPE html>

```

Annexes D : Code émetteur cube 3D

```
#include <Wire.h>
#include <math.h>
// =====
// --- CONFIGURATION GÉNÉRALE
// =====
// --- ICM20948 (IMU 9 axes)
// =====
#define ICM20948_ADDR 0x68
#define WHO_AM_I_REG 0x00
#define PWR_MGMT_1 0x06
#define USER_CTRL 0x03
#define INT_PIN_CFG 0x0F
#define ACCEL_XOUT_H 0x2D
#define INT_ENABLE_1 0x11
// =====
// --- AK09916 (magnétomètre)
// =====
#define AK09916_ADDR 0x0C
#define AK09916_WHO_AM_I 0x01
#define AK09916_CNTL1 0x31
#define AK09916_ASAX 0x10
#define AK09916_ST1 0x18
#define AK09916_XOUT_L 0x11
// Broche d'interruption DRDY de l'ICM20948 vers l'ESP32
#define INT_PIN 3
volatile bool dataReady = false;
// =====
// --- VARIABLES GLOBALES
// =====
// bouton pour la calibration
#define BUTTON_PIN 4
bool buttonPressed = false;
unsigned long lastDebounceTime = 0;
const unsigned long debounceDelay = 50;
// Offsets de calibration
float accelOffset[3] = {0.0f, 0.0f, 0.0f};
float gyroOffset[3] = {0.0f, 0.0f, 0.0f};
float magCalibration[3] = {1.0f, 1.0f, 1.0f};
// Paramètres Mahony
float twoKp = 1.0f;
float twoKi = 0.4f;
float integralFBx = 0.0f, integralFBy = 0.0f, integralFBz = 0.0f;
// Quaternion orientation
float q0 = 1.0f, q1 = 0.0f, q2 = 0.0f, q3 = 0.0f;
// Temps
unsigned long lastUpdate = 0;
// =====
// --- FONCTIONS BAS NIVEAU
// =====
// utilitaires I2C
void writeRegister(uint8_t addr, uint8_t reg, uint8_t data) {
    Wire.beginTransmission(addr);
    Wire.write(reg);
    Wire.write(data);
    Wire.endTransmission();
}
uint8_t readRegister(uint8_t addr, uint8_t reg) {
    Wire.beginTransmission(addr);
    Wire.write(reg);
    Wire.endTransmission(false);
    Wire.requestFrom(addr, (uint8_t)1);
    return Wire.read();
}
void readBytes(uint8_t addr, uint8_t reg, uint8_t count, uint8_t *dest) {
    Wire.beginTransmission(addr);
    Wire.write(reg);
    Wire.endTransmission(false);
```

```

Wire.requestFrom(addr, count);
for (int i = 0; i < count; i++) {
    dest[i] = Wire.read();
}
}
// Sélection de banque de registres ICM20948
void setBank(uint8_t bank) {
    writeRegister(ICM20948_ADDR, 0x7F, bank & 0x0F);
}
// Interruption DRDY de l'IMU
void IRAM_ATTR onDataReady() {
    dataReady = true;
}
// =====
// --- CAPTEUR ICM20948/AK09916
// =====
// Lecture accel + gyro bruts
void readAccelGyro(int16_t *accel, int16_t *gyro) {
    setBank(0);
    uint8_t buf[12];
    readBytes(ICM20948_ADDR, ACCEL_XOUT_H, 12, buf);
    accel[0] = ((int16_t)buf[0] << 8) | buf[1];
    accel[1] = ((int16_t)buf[2] << 8) | buf[3];
    accel[2] = ((int16_t)buf[4] << 8) | buf[5];
    gyro[0] = ((int16_t)buf[6] << 8) | buf[7];
    gyro[1] = ((int16_t)buf[8] << 8) | buf[9];
    gyro[2] = ((int16_t)buf[10] << 8) | buf[11];
}
// Init magnétomètre (lecture des facteurs d'ajustement + mode continu)
void initAK09916() {
    uint8_t raw[3];
    readBytes(AK09916_ADDR, AK09916_ASAX, 3, raw);
    for (int i = 0; i < 3; i++) {
        magCalibration[i] = ((raw[i] - 128) / 256.0f) + 1.0f;
    }
    // Mode 100 Hz
    writeRegister(AK09916_ADDR, AK09916_CNTL1, 0x10);
    delay(10);
}
// Lecture magnétomètre
void readMag(int16_t *mag) {
    uint8_t st1 = readRegister(AK09916_ADDR, AK09916_ST1);
    if (st1 & 0x01) { // Données prêtes
        uint8_t buf[7];
        readBytes(AK09916_ADDR, AK09916_XOUT_L, 7, buf);
        mag[0] = ((int16_t)buf[1] << 8) | buf[0];
        mag[1] = ((int16_t)buf[3] << 8) | buf[2];
        mag[2] = ((int16_t)buf[5] << 8) | buf[4];
    }
}
// Calibration accéléro + gyro (moyenne sur N échantillons)
void calibrateSensor() {
    long accelSum[3] = {0, 0, 0};
    long gyroSum[3] = {0, 0, 0};
    const int samples = 2000; // à réduire si tu veux un démarrage plus rapide
    Serial.println("Calibration accel/gyro...");
    for (int i = 0; i < samples; i++) {
        int16_t a[3], g[3];
        readAccelGyro(a, g);
        for (int j = 0; j < 3; j++) {
            accelSum[j] += a[j];
            gyroSum[j] += g[j];
        }
        delay(1);
    }
    for (int j = 0; j < 3; j++) {
        accelOffset[j] = (accelSum[j] / (float)samples) / 16384.0f;
        gyroOffset[j] = (gyroSum[j] / (float)samples) / 131.0f;
    }
    // Retirer la gravité sur Z
    accelOffset[2] -= 1.0f;
}

```

```

Serial.println("Calibration terminee !");
}
// =====
// --- FILTRE MAHONY
// =====
void MahonyUpdate(float gx, float gy, float gz, float ax, float ay, float az, float mx, float my, float mz, float dt) {
    float recipNorm;
    float hx, hy, bx, bz;
    float vx, vy, vz, wx, wy, wz;
    float ex, ey, ez;
    // Normalisation accéléromètre
    float normAcc = sqrtf(ax * ax + ay * ay + az * az);
    if (normAcc < 1e-6f) return;
    recipNorm = 1.0f / normAcc;
    ax *= recipNorm;
    ay *= recipNorm;
    az *= recipNorm;
    // Normalisation magnétomètre
    float normMag = sqrtf(mx * mx + my * my + mz * mz);
    if (normMag < 1e-6f) return;
    recipNorm = 1.0f / normMag;
    mx *= recipNorm;
    my *= recipNorm;
    mz *= recipNorm;
    // Référence magnétique
    hx = 2.0f * mx * (0.5f - q2 * q2 - q3 * q3) + 2.0f * my * (q1 * q2 - q0 * q3) + 2.0f * mz * (q1 * q3 + q0 * q2);
    hy = 2.0f * mx * (q1 * q2 + q0 * q3) + 2.0f * my * (0.5f - q1 * q1 - q3 * q3) + 2.0f * mz * (q2 * q3 - q0 * q1);
    bx = sqrtf(hx * hx + hy * hy);
    bz = 2.0f * mx * (q1 * q3 - q0 * q2) + 2.0f * my * (q2 * q3 + q0 * q1) + 2.0f * mz * (0.5f - q1 * q1 - q2 * q2);
    // Direction estimée gravité + champ magnétique
    vx = 2.0f * (q1 * q3 - q0 * q2);
    vy = 2.0f * (q0 * q1 + q2 * q3);
    vz = q0 * q0 - q1 * q1 - q2 * q2 + q3 * q3;
    wx = 2.0f * bx * (0.5f - q2 * q2 - q3 * q3) + 2.0f * bz * (q1 * q3 - q0 * q2);
    wy = 2.0f * bx * (q1 * q2 - q0 * q3) + 2.0f * bz * (q0 * q1 + q2 * q3);
    wz = 2.0f * bx * (q0 * q2 + q1 * q3) + 2.0f * bz * (0.5f - q1 * q1 - q2 * q2);
    // Erreur
    ex = (ay * vz - az * vy) + (my * wz - mz * wy);
    ey = (az * vx - ax * vz) + (mz * wx - mx * wz);
    ez = (ax * vy - ay * vx) + (mx * wy - my * wx);
    // Terme intégral
    if (twoKi > 0.0f) {
        integralFBx += twoKi * ex * dt;
        integralFBy += twoKi * ey * dt;
        integralFBz += twoKi * ez * dt;
        gx += integralFBx;
        gy += integralFBy;
        gz += integralFBz;
    } else {
        integralFBx = 0.0f;
        integralFBy = 0.0f;
        integralFBz = 0.0f;
    }
    // Terme proportionnel
    gx += twoKp * ex;
    gy += twoKp * ey;
    gz += twoKp * ez;
    // Intégration quaternion
    float qa = q0, qb = q1, qc = q2;
    q0 += (-qb * gx - qc * gy - q3 * gz) * (0.5f * dt);
    q1 += (qa * gx + qc * gz - q3 * gy) * (0.5f * dt);
    q2 += (qa * gy - qb * gz + q3 * gx) * (0.5f * dt);
    q3 += (qa * gz + qb * gy - qc * gx) * (0.5f * dt);
    // Normalisation quaternion
    float normQ = sqrtf(q0 * q0 + q1 * q1 + q2 * q2 + q3 * q3);
    if (normQ < 1e-6f) return;
    recipNorm = 1.0f / normQ;
    q0 *= recipNorm;
    q1 *= recipNorm;
    q2 *= recipNorm;
    q3 *= recipNorm;
}

```

```

}
// =====
// --- SETUP
// =====
void setup() {
  Serial.begin(115200);
  Wire.begin();
  Wire.setClock(400000);
  // Bouton
  pinMode(BUTTON_PIN, INPUT_PULLUP);
  // Réveil IMU
  setBank(0);
  writeRegister(ICM20948_ADDR, PWR_MGMT_1, 0x01);
  delay(10);
  // Bypass I2C pour magnétomètre
  setBank(0);
  writeRegister(ICM20948_ADDR, INT_PIN_CFG, 0x22);
  // Interruption DRDY côté ESP32
  pinMode(INT_PIN, INPUT);
  attachInterrupt(digitalPinToInterrupt(INT_PIN), onDataReady, RISING);
  // Init magnétomètre
  initAK09916();
  // Calibration initiale
  calibrateSensor();
  lastUpdate = micros();
  // Configuration gyro / accel (Bank 2)
  setBank(2);
  // Gyro : ±250 dps, DLPF Off, ~9 kHz
  writeRegister(ICM20948_ADDR, 0x01, 0x01); //gyro 9khz
  writeRegister(ICM20948_ADDR, 0x00, 0x00); // Diviseur gyro ignoré car 9khz
  // Accel : ±2 g, DLPF OFF, ~4.5 kHz
  writeRegister(ICM20948_ADDR, 0x14, 0x01); //accel 4.5khz
  writeRegister(ICM20948_ADDR, 0x10, 0x00); // Diviseur accel MSB (ignoré)
  writeRegister(ICM20948_ADDR, 0x11, 0x00); // Diviseur accel LSB (ignoré)
  // Retour Bank 0 + INT DRDY
  setBank(0);
  writeRegister(ICM20948_ADDR, INT_PIN_CFG, 0x10); // latch until read
  writeRegister(ICM20948_ADDR, INT_ENABLE_1, 0x01); // DRDY accel/gyro
}
// =====
// --- LOOP
// =====
void loop() {
  int16_t accelRaw[3], gyroRaw[3], magRaw[3];
  float ax, ay, az, gx, gy, gz, mx, my, mz;
  // Gestion bouton (calibration à la demande)
  int buttonState = digitalRead(BUTTON_PIN);
  if (buttonState == LOW && !buttonPressed && (millis() - lastDebounceTime) > debounceDelay) { //antirebond
    buttonPressed = true;
    lastDebounceTime = millis();
    Serial.print("Calibration...");
    calibrateSensor();
  }
  if (buttonState == HIGH && buttonPressed && (millis() - lastDebounceTime) > debounceDelay) { //antirebond
    buttonPressed = false;
    lastDebounceTime = millis();
  }
  // Attente DRDY
  if (!dataReady) return;
  dataReady = false;
  // Lecture accel / gyro
  readAccelGyro(accelRaw, gyroRaw);
  // Conversion en unités physiques (g, °/s)
  ax = (accelRaw[0] / 16384.0f) - accelOffset[0];
  ay = (accelRaw[1] / 16384.0f) - accelOffset[1];
  az = (accelRaw[2] / 16384.0f) - accelOffset[2];
  gx = (gyroRaw[0] / 131.0f) - gyroOffset[0];
  gy = (gyroRaw[1] / 131.0f) - gyroOffset[1];
  gz = (gyroRaw[2] / 131.0f) - gyroOffset[2];
  // Passage en rad/s
  gx *= (M_PI / 180.0f);

```

```

gy *= (M_PI / 180.0f);
gz *= (M_PI / 180.0f);
// Magnétomètre
readMag(magRaw);
mx = magRaw[0] * 0.15f * magCalibration[0];
my = magRaw[1] * 0.15f * magCalibration[1];
mz = magRaw[2] * 0.15f * magCalibration[2];
// dt
unsigned long now = micros();
float dt = (now - lastUpdate) / 1000000.0f;
lastUpdate = now;
// Orientation
MahonyUpdate(gx, gy, gz, ax, ay, az, mx, my, mz, dt);
// Quaternion -> Euler (deg)
float roll = atan2f(2.0f * (q0 * q1 + q2 * q3), 1.0f - 2.0f * (q1 * q1 + q2 * q2)) * 180.0f / M_PI;
float pitch = asinf(2.0f * (q0 * q2 - q3 * q1)) * 180.0f / M_PI;
float yaw = atan2f(2.0f * (q0 * q3 + q1 * q2), 1.0f - 2.0f * (q2 * q2 + q3 * q3)) * 180.0f / M_PI;
// Série
Serial.print(round(roll));
Serial.print(",");
Serial.print(round(pitch));
Serial.print(",");
Serial.println(round(yaw));
}

```

Annexes E : Code IHM

```
// =====
//          HUD
// =====
int centreX = 84;
int centreY = 30;

int stopX = 20;
int stopY = 26;
int stopLargeur = 28;
int stopHauteur = 12;

void dessinerHUD() {
    ecran.drawRect(0, 0, 128, 64, SSD1306_WHITE);
    ecran.fillRect(0, 51, 128, 13, SSD1306_WHITE);

    ecran.drawCircle(centreX, centreY, 16, SSD1306_WHITE);
    ecran.drawCircle(centreX, centreY, 18, SSD1306_WHITE);
}

// =====
//          LOGO WIFI
// =====
void dessinerLogoWifi(int x, int y) {
    // Point central
    ecran.fillCircle(x, y, (1/2), SSD1306_BLACK);

    // Arc 1 (petit)
    ecran.drawLine(x - 2, y - 2, x - 1, y - 3, SSD1306_BLACK);
    ecran.drawLine(x - 1, y - 3, x + 1, y - 3, SSD1306_BLACK);
    ecran.drawLine(x + 1, y - 3, x + 2, y - 2, SSD1306_BLACK);

    // Arc 2 (moyen)
    ecran.drawLine(x - 4, y - 4, x - 3, y - 6, SSD1306_BLACK);
    ecran.drawLine(x - 3, y - 6, x - 1, y - 7, SSD1306_BLACK);
    ecran.drawLine(x - 1, y - 7, x + 1, y - 7, SSD1306_BLACK);
    ecran.drawLine(x + 1, y - 7, x + 3, y - 6, SSD1306_BLACK);
    ecran.drawLine(x + 3, y - 6, x + 4, y - 4, SSD1306_BLACK);

    // Arc 3 (grand)
    ecran.drawLine(x - 6, y - 5, x - 5, y - 8, SSD1306_BLACK);
    ecran.drawLine(x - 5, y - 8, x - 2, y - 10, SSD1306_BLACK);
    ecran.drawLine(x - 2, y - 10, x + 2, y - 10, SSD1306_BLACK);
    ecran.drawLine(x + 2, y - 10, x + 5, y - 8, SSD1306_BLACK);
    ecran.drawLine(x + 5, y - 8, x + 6, y - 5, SSD1306_BLACK);
}

// =====
//          BANDEAU + AFFICHAGE
// =====
void afficherBandeau(int r, int t) {

    ecran.fillRect(0, 0, 128, 10, SSD1306_WHITE);
    ecran.setCursor(3, 2);
    ecran.setTextColor(SSD1306_BLACK);
    ecran.print("R:");
    ecran.print(r);
    ecran.print(" T:");
    ecran.print(t);

    // Texte état connexion
    ecran.setCursor(3, 55);
    ecran.setTextColor(SSD1306_BLACK);
    if (wifiOK) ecran.print("Connecte");
    else ecran.print("Deconnecte");

    // *** LOGO WIFI SEULEMENT SI CONNECTÉ ***
    if (wifiOK) {
```

```

dessinerLogoWifi(60, 62); // Position du logo dans le bandeau
}
}

// =====
//      FLECHES DE DIRECTION
// =====

void flecheHaut() {
    ecran.drawLine(centreX, centreY - 8, centreX, centreY - 2, SSD1306_WHITE);
    ecran.fillTriangle(centreX, centreY - 12, centreX - 3, centreY - 6, centreX + 3, centreY - 6, SSD1306_WHITE);
}

void flecheBas() {
    ecran.drawLine(centreX, centreY + 2, centreX, centreY + 8, SSD1306_WHITE);
    ecran.fillTriangle(centreX, centreY + 12, centreX - 3, centreY + 6, centreX + 3, centreY + 6, SSD1306_WHITE);
}

void flecheGauche() {
    ecran.drawLine(centreX - 8, centreY, centreX - 2, centreY, SSD1306_WHITE);
    ecran.fillTriangle(centreX - 12, centreY, centreX - 6, centreY - 3, centreX - 6, centreY + 3, SSD1306_WHITE);
}

void flecheDroite() {
    ecran.drawLine(centreX + 2, centreY, centreX + 8, centreY, SSD1306_WHITE);
    ecran.fillTriangle(centreX + 12, centreY, centreX + 6, centreY - 3, centreX + 6, centreY + 3, SSD1306_WHITE);
}

void flecheHautDroite() {
    ecran.drawLine(centreX, centreY, centreX + 6, centreY - 6, SSD1306_WHITE);
    ecran.fillTriangle(centreX + 10, centreY - 10, centreX + 5, centreY - 7, centreX + 7, centreY - 5, SSD1306_WHITE);
}

void flecheHautGauche() {
    ecran.drawLine(centreX, centreY, centreX - 6, centreY - 6, SSD1306_WHITE);
    ecran.fillTriangle(centreX - 10, centreY - 10, centreX - 5, centreY - 7, centreX - 7, centreY - 5, SSD1306_WHITE);
}

void flecheBasDroite() {
    ecran.drawLine(centreX, centreY, centreX + 6, centreY + 6, SSD1306_WHITE);
    ecran.fillTriangle(centreX + 10, centreY + 10, centreX + 5, centreY + 7, centreX + 7, centreY + 5, SSD1306_WHITE);
}

void flecheBasGauche() {
    ecran.drawLine(centreX, centreY, centreX - 6, centreY + 6, SSD1306_WHITE);
    ecran.fillTriangle(centreX - 10, centreY + 10, centreX - 5, centreY + 7, centreX - 7, centreY + 5, SSD1306_WHITE);
}

```

Annexes F : Code émetteur Final

--

Annexes G : Code récepteur Final

Bibliographie

Bibliographie sur les IMU

Information relative au fonctionnement

- « Qu'est-ce qu'une IMU MEMS ? », GuideNav <https://guidenav.com/fr/qu-39-est-ce-qu-39-un-mems-imu/>
- « Choisir un IMU : FOG vs MEMS », Canal Geomatics <https://canalgeomatics.com/fr/knowledgebase/choosing-imu-fog-vs-mems-imus/>
- Alan Petrillo, « Simulation des accéléromètres et des gyroscopes MEMS utilisés dans les unités de mesure inertielle », Comsol <https://www.comsol.fr/blogs/simulating-mems-accelerometers-and-gyroscopes-used-in-inertial-measurement-units> [24 mars 2022]
- Agustín L. Herrera-May, « Développement de microcapteurs à champ magnétique résonant : défis et applications futures », <https://www.intechopen.com/chapters/15851> [9 juin 2011]
- M. Dejan « Qu'est-ce que le MEMS ? Accéléromètre, Gyroscope & Magnétomètre avec Arduino », <https://howtomechatronics.com/how-it-works/electrical-engineering/mems-accelerometer-gyroscope-magnetometer-arduino/>

Information sur les outils de mesures de l'IMU

- « Piezoelectric Rate Gyroscope », Comsol <https://www.comsol.com/model/piezoelectric-rate-gyroscope-21471>
- « Surface Micromachined Accelerometer », Comsol <https://www.comsol.com/model/surface-micromachined-accelerometer-17325>
- Charles MAUC, « Développement d'un magnétomètre miniature exploitant un résonateur MEMS et une structure multicouche de matériaux magnétiques », https://theses.hal.science/tel-03955933v1/file/114105_MAUC_2022_archivage.pdf [9 décembre 2022]

Fiches techniques différentes IMU

- « World's Lowest Power 9-Axis MEMS MotionTracking™ Device », TDK InvenSens https://product.tdk.com/system/files/dam/doc/product/sensor/motion-inertial/imu/data_sheet/ds-000189-icm-20948-v1.5.pdf [2021]
- « MPU-9250 Product Specification Revision 1.1 », InvenSens https://product.tdk.com/system/files/dam/doc/product/sensor/motion-inertial/imu/data_sheet/ps-mpu-9250a-01-v1.1.pdf

- « BNO055 Intelligent 9-axis absolute orientation sensor », Bosch Sensortec
<https://www.robot-electronics.co.uk/files/BNO055.pdf> [juin 2016]

Bibliographie sur les microcontrôleurs

- « ESP32-C3 Series Datasheet », Espressif Systems
https://www.espressif.com/sites/default/files/documentation/esp32-c3_datasheet_en.pdf
- « Arduino Pro Mini Technical Datasheet », Arduino,
<https://docs.arduino.cc/resources/datasheets/Arduino-Pro-Mini-datasheet.pdf>
- « STM32F103x8 / STM32F103xB Datasheet », STMicroelectronics,
<https://www.st.com/resource/en/datasheet/stm32f103c8.pdf>

Bibliographie sur le filtre Mahony

- « ICM_20948-AHRS/ Mahony_AHRS/ICM_20948_Mahony.ino », GitHub
https://github.com/jremington/ICM_20948-AHRS/blob/main/Mahony_AHRS/ICM_20948_Mahony.ino
- « Comparison of Attitude and Heading Reference Systems using Foot Mounted MIMU Sensor Data: Basic, Madgwick and Mahony »,
<https://web.cs.ndsu.nodak.edu/~siludwig/Publish/papers/SPIE20181.pdf>
- « Explication du filtre Mahony de l'IMU », Medium
<https://medium.com/%40k66115704/imu-mahony-filter-explanation-1ae75bf033ab> [22 avril 2024]
- « Filtre d'orientation Mahony »
<https://ahrs.readthedocs.io/en/latest/filters/mahony.html>
- « Modèle mathématique d'une centrale inertielle »
<https://nininjsanket.github.io/tutorials/attitudeest/mahony.html>

Quatrième de couverture

Ce rapport présente la conception et la réalisation d'un système de télécommande inertielle d'un robot mobile à quatre roues à l'aide d'une centrale IMU ICM-20948 et de deux microcontrôleurs ESP32-C3. L'orientation de la main de l'utilisateur est estimée en temps réel grâce à un filtre de Mahony, puis transmise sans fil via ESP-NOW au robot qui génère, à partir de ces données, les signaux PWM nécessaires au pilotage des moteurs.

Le projet couvre l'ensemble de la chaîne des systèmes embarqués : étude des capteurs inertiels MEMS, fusion de capteurs pour l'estimation d'attitude, communication radio point à point, conception de cartes électroniques et développement d'une commande gestuelle du robot. Il met en évidence les performances et les limites de l'IMU dans cette application, et ouvre plusieurs perspectives, notamment la détection de gestes, la commande dynamique et la programmation par démonstration

This report presents the design and implementation of an inertial remote-control system for a four-wheeled mobile robot using an ICM-20948 IMU and two ESP32-C3 microcontrollers. The user's hand orientation is estimated in real time with a Mahony filter and wirelessly transmitted via ESP-NOW to the robot, which converts these data into PWM signals to drive the motors.

The project addresses the complete embedded systems chain: analysis of MEMS inertial sensors, sensor fusion for attitude estimation, point-to-point radio communication, electronic board design, and gesture-based robot control. It highlights both the performance and the limitations of the IMU in this context, and suggests several extensions such as gesture recognition, dynamic control, and learning-by-demonstration programming