Said Afrite
Salomé Mathe
Samantha Saint-Lary
Odile Schaeffer
5TS1

IPSA

Projet Master IPSA

Robots pédagogiques Mbot2

M. Labsir & M. Ortega
2025 - 2026

Table des matières

Remerciements

Nous tenons tout d'abord à remercier sincèrement M. Ortega et M. Labsir pour l'opportunité qu'ils nous ont offerte à travers ce projet, ainsi que pour leur accompagnement, leurs conseils et leur disponibilité tout au long de son déroulement. Leur encadrement a été essentiel à la bonne réalisation de notre travail et à notre progression, tant sur le plan technique que méthodologique.

Nous souhaitons également remercier Mme Florent pour sa réactivité, son efficacité et sa disponibilité dans l'organisation et la réservation des salles, ce qui a grandement facilité la planification de nos séances de travail et la bonne avancée du projet.

Nous adressons aussi nos remerciements à l'ensemble des collaborateurs et au corps pédagogique pour leur soutien, leur engagement et leur contribution à la mise en place de ce projet. Grâce à eux, nous avons pu bénéficier de matériel de qualité, de ressources adaptées et d'un environnement propice à l'expérimentation, à l'apprentissage et au développement de nos compétences.

Ce projet a représenté une expérience enrichissante, tant sur le plan académique que personnel, et nous sommes reconnaissants à toutes les personnes ayant contribué à sa réussite.

Introduction

Contextualisation

Dans le cadre de notre cursus d'ingénieur à l'IPSA, le Projet Master IPSA (PMI) constitue une étape essentielle de notre formation. Il représente l'aboutissement d'un enseignement approfondi, en se proposant comme un projet technique concret, mené avec un haut niveau d'autonomie. Encadré par M. Ortega et M. Labsir, ce projet s'articule autour de la mise en place et de l'exploitation pédagogique d'une plateforme robotique éducative, fondée principalement sur l'utilisation du robot mBot2.

Ce projet n'est pas qu'un simple exercice académique de programmation ou de prise en main matérielle. Il vise également à répondre à un besoin pédagogique identifié au sien de l'établissement, en proposant la conception de Travaux Pratiques destinés à être intégrés durablement dans le cursus des promotions futures.

L'enjeu central de ce projet réside dans la confrontation entre les modèles théoriques étudiés au cours du cursus et leur implémentation sur un système physique réel. L'utilisation du robot mBot2 permet de matérialiser des notions fondamentales d'automatique, de cinématique et de traitement du signal, généralement abordées de manière abstraite ou à travers des simulations numériques. Le passage au réel met en évidence des phénomènes absents des modèles idéalisés, tels que les frottements mécaniques, le bruit des capteurs, les délais de communication ou encore les incertitudes de mesure, confrontant ainsi les étudiants aux contraintes concrètes des systèmes embarqués.

Mené dans un cadre autonome, ce projet implique également une gestion rigoureuse du temps, des ressources matérielles et des contraintes techniques. À terme, il a pour ambition de transformer une flotte de robots éducatifs en un véritable laboratoire mobile, permettant l'apprentissage expérimental de la robotique mobile et constituant une base de ressources pédagogiques pérennes pour l'IPSA.

Objectifs

L'objectif principal de ce Projet Master IPSA est de concevoir et de déployer une plateforme robotique pédagogique, constituée d'un ensemble de cinq robots mBot2, exploitable dans le cadre de Travaux Pratiques de niveau ingénieur.

Afin d'atteindre cet objectif global, le projet vise en premier lieu à assurer une prise en main complète du robot mBot2, tant sur le plan matériel que logiciel. Cela implique la compréhension de son architecture générale, de ses capteurs, de ses actionneurs, ainsi que de ses capacités de programmation.

Dans un second temps, le projet a pour ambition de développer et de tester plusieurs applications robotiques représentatives, telles que le suivi de ligne, l'évitement d'obstacles ou encore la détection de couleurs. Ces applications permettent de valider le comportement du robot dans des situations concrètes et réalistes, proches de celles rencontrées lors de travaux pratiques.

Une partie essentielle du projet concerne l'exploitation des capteurs embarqués. Une attention particulière a été portée à l'acquisition, au traitement et à l'analyse des données issues du robot. À terme, l'objectif est de traiter ces données sous Matlab afin de les exploiter dans le cadre de Travaux Pratiques, notamment pour des applications de traitement de trajectoire ou d'analyse de performances.

L'utilisation des robots mBot2 est justifiée par leur adéquation en tant qu'outil d'introduction au domaine de la mécatronique. À plus long terme, ce projet pourra servir de base à l'utilisation de plateformes robotiques plus complexes, permettant le développement de codes plus avancés et de nouvelles applications, telles que la navigation GNSS. Les principaux avantages des robots mBot2 résident dans leur facilité de prise en main, leur simplicité de montage et leur caractère pédagogique.

Enfin, ce travail vise à formaliser l'ensemble de ces expérimentations sous forme de supports pédagogiques structurés, incluant des séances de Travaux Pratiques, des documents explicatifs et des démonstrations visuelles, tout en identifiant les limites et les difficultés liées à l'utilisation de robots réels dans un cadre d'enseignement.

I. Etat de l'art

Pour comprendre l'intérêt du mBot2, et donc le but de notre PMI, il faut d'abord comprendre comment la robotique est enseignée aujourd'hui et pourquoi ce robot constitue une évolution.

A. La robotique éducative dans la formation d'ingénieur

L'intégration de la robotique dans l'enseignement supérieur marque une avancée avec l'apprentissage purement algorithmique pour s'ancrer dans le domaine de l'ingénierie des systèmes. Dans ce contexte, le robot mBot2 n'est plus un simple support de programmation, mais devient une plateforme de validation physique pour des concepts théoriques. Pour un étudiant ingénieur de troisième année, le robot introduit le lien entre le modèle mathématique (la théorie) et le comportement dynamique réel (la pratique).

Cette approche est cruciale pour l'apprentissage de la mécatronique. Là où une simulation logicielle ignore souvent les non-linéarités, le robot physique impose les réalités du terrain. En manipulant le mBot2, les étudiants ne se contenterons pas de vérifier si leur code "fonctionne", mais plutôt de se rendre compte de comment il interagit réellement avec le monde qui l'entoure. Par exemple, l'implémentation d'un asservissement de position ne se résume plus à une courbe sur un écran, mais se traduit par la fluidité ou l'instabilité du mouvement du robot.

Le premier apport de cette approche physique réside dans la gestion des phénomènes stochastiques et des bruits de mesure. En simulation, les conditions sont idéales : un capteur renvoie une distance parfaite. Dans la réalité, les capteurs du mBot2 sont soumis à des échos, à des variations de température ou encore à l'absorption du signal par les matériaux. Cette confrontation les oblige à entrer dans le domaine du traitement du signal. Il faut alors concevoir et implémenter des filtres numériques pour extraire une information fiable d'un environnement bruité.

Au-delà du signal, c'est la mécanique du monde réel qui s'impose à l'utilisateur. Le passage au système physique révèle des non-linéarités souvent ignorées lors des calculs sur papier. Des concepts tels que le frottement statique ou l'inertie des moteurs deviennent des obstacles. On observe que la commande envoyée n'est jamais strictement égale au mouvement réalisé car les roues peuvent patiner sur une surface lisse ou le robot peut dévier à cause d'un léger déséquilibre de masse. Ces imprévus forcent l'apprentissage de l'asservissement en boucle fermée. Le robot n'est plus piloté à l'aveugle par une durée de fonctionnement, mais par une comparaison constante entre sa position réelle, mesurée par ses capteurs, et sa position cible.

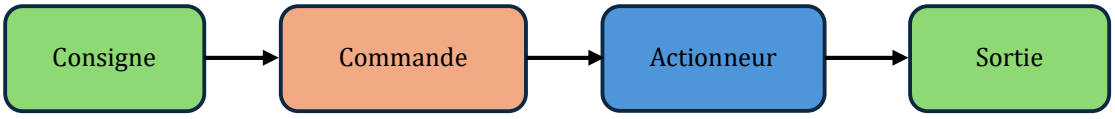


Figure 1 : Boucle ouverte

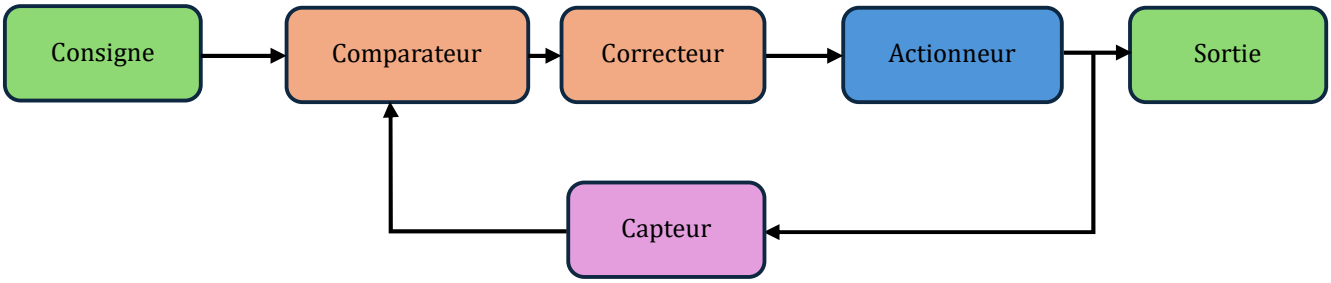


Figure 2 : Boucle fermée

Cette interaction constante entre le calcul et l'action introduit également la notion de discrétisation du temps. Alors que les équations physiques évoluent de manière continue, le processeur du mBot2 fonctionne par cycles (discret). L'étudiant doit alors appréhender l'importance de la fréquence d'échantillonnage, un concept fondamental des systèmes embarqués qu'ils approfondiront dans la suite de leur cursus.

Ainsi, ce projet ne se contente pas de valider des acquis, il prépare les futures promotions à la complexité des systèmes qu'ils devront concevoir en fin de cycle ingénieur. En tant qu'étudiants ayant déjà effectué ce lien théorique- pratique dans notre cursus, nous trouvons que l'introduire par un TP sur le mBot2 permettrais une vision plus globale et simplifiée avant d'entrer dans les détails de la mécatronique, de l'automatique, des systèmes embarqués et des télécommunications.

B. Rupture technologique : Pourquoi le mBot2

Le choix du mBot2 nous a été imposé pour ce Projet Master IPSA. Néanmoins, en se renseignant, on remarque qu'il marque une transition majeure. On passe d'un robot d'initiation (mBot v1.1), conçu pour apprendre les bases du code, à une véritable plateforme d'ingénierie système.

Pour comprendre l'intérêt de ce choix, il faut s'attarder sur les technologies qui font du mBot2 un outil professionnel :

Tout d'abord, concernant la puissance de calcul, le mBot2 utilise une carte appelée CyberPi, équipée d'un processeur double cœur.

En robotique, la puissance ne sert pas juste à aller plus vite, elle sert à gérer le multitâche. Un robot doit pouvoir analyser la distance d'un mur, calculer la vitesse de ses roues et communiquer en Wi-Fi et ce presque en même temps. Avec un processeur classique (simple cœur), le robot doit mettre ses calculs "en pause" pour envoyer une donnée Wi-Fi. Avec le mBot2, les tâches sont réparties : un cœur gère la sécurité et les mouvements, l'autre gère la communication. C'est ce qu'on appelle la gestion des systèmes embarqués complexes.

Un autre point crucial pour nos futurs ingénieurs est la notion de temps réel. En effet, en ingénierie, on se doit d'avoir une rigueur dans les codes afin qu'ils soient optimisés pour que la réaction soit la plus immédiate. En utilisant le mBot2, ils apprennent à programmer des systèmes

où la latence est faible. C'est exactement ce que l'on demande à un ingénieur travaillant sur les commandes de vol d'un avion.

Le mBot2 intègre aussi un écran couleur (CyberOS) de diagnostic qui permet au robot d'afficher l'état interne en direct. De plus cela permet une facilité de débogage directement sur l'interface utilisateur.

C. Précision et contrôle : les encodeurs optiques

L'une des limites majeures des robots éducatifs d'entrée de gamme est l'imprécision du mouvement (contrôle en boucle ouverte) qui se contente d'envoyer une tension aux moteurs en espérant que le robot avance droit. Le mBot2 résout cette problématique par l'utilisation de moteurs à encodeurs optiques intégrés.

Un encodeur optique est un capteur qui compte le nombre de graduations passant devant un faisceau lumineux lors de la rotation du moteur. Pour un étudiant de troisième année, cela permet de transformer un mouvement mécanique flou en une donnée numérique exacte. On ne parle plus en pourcentage de puissance moteur mais en degrés de rotation ou en distance parcourue.

L'un des défis majeurs est de faire avancer un robot parfaitement droit. Dans la réalité, deux moteurs ne sont jamais strictement identiques (différences de fabrication, usure, adhérence du sol). Sans encodeurs, le robot finit par dévier. Les encodeurs permettent une synchronisation : le système compare en permanence la vitesse réelle de la roue gauche par rapport à la roue droite et ajuste les tensions pour corriger les dérives. Cette rigueur dans le contrôle du mouvement est une introduction directe aux problématiques de guidage que l'on retrouve sur des systèmes de transport automatisés ou des rovers d'exploration.

D. Programmations multi-niveaux

L'un des atouts du mBot2 réside dans sa flexibilité logicielle. Pour un étudiant en ingénierie à l'IPSA, la programmation ne doit pas être un obstacle, mais un outil de résolution de problèmes. Le mBot2 permet une progression fluide, appelée "programmation multi-niveaux", qui accompagne l'étudiant de la logique de base vers une expertise professionnelle en Python.

L'étudiant peut utiliser la programmation par blocs. Elle permet de valider rapidement la logique d'un algorithme sans se soucier des erreurs de syntaxe ainsi que d'introduire le fonctionnement des capteurs du robot.

Néanmoins, pour les futurs ingénieurs de l'IPSA, la maîtrise de Python est indispensable. C'est le langage le plus utilisé aujourd'hui pour l'intelligence artificielle, l'analyse de données et le prototypage industriel. Le mBot2 embarque une version spécifique appelée MicroPython, optimisée pour les systèmes électroniques.

Le codage Python permet de manipuler directement les registres du processeur. On peut donc régler précisément la fréquence des moteurs ou la sensibilité des capteurs. En orientant nos TP vers le codage Python, nous préparons les étudiants aux réalités du marché du travail.

E. IoT et Intelligence Artificielle

Il est aussi essentiel d'aborder la dimension connectée du mBot2. Le mBot2 intègre des technologies qui permettent d'initier les étudiants à deux piliers de l'ingénierie contemporaine que sont l'Internet des Objets (IoT) et l'Intelligence Artificielle (IA).

Le mBot2 intègre des capacités de communication sans fil, notamment via le Wi-Fi ou le Bluetooth, qui permettent d'aborder concrètement les principes de l'Internet des Objets (IoT). Dans un cadre pédagogique, cette connectivité offre la possibilité de mettre en place des scénarios où le robot transmet des informations issues de ses capteurs vers une interface distante, ou reçoit des commandes externes.

Dans le cadre du mBot2, les capacités de calcul embarquées permettent d'illustrer ces principes à travers des algorithmes simples de prise de décision. Ces comportements constituent une première approche de l'intelligence artificielle embarquée, fondée sur des règles logiques ou des algorithmes déterministes.

Il est important de souligner que le mBot2 ne vise pas à reproduire des systèmes d'intelligence artificielle avancés tels que ceux utilisés dans la robotique industrielle ou les véhicules autonomes. En revanche, il constitue une plateforme pertinente pour introduire les concepts fondamentaux qui sous-tendent ces technologies.

Inclure l'IoT et l'IA permet de montrer aux étudiants que la robotique ne s'arrête pas à la mécanique et à l'électronique. C'est une discipline qui touche au réseau, à l'analyse des données et à l'intelligence logicielle.

II. Prise en main des robots

A. Découverte du matériel

Nous avons récupéré les robots ainsi que l'ordinateur sur lequel nous allions coder au début de ce semestre, afin de commencer ce tout nouveau projet. Dans un premier temps, l'objectif était de découvrir les cinq robots à notre disposition et, bien évidemment, de les monter. Nous avons donc commencé par assembler l'un des robots afin de comprendre le processus de montage, pour pouvoir, dans un second temps, réaliser une vidéo explicative destinée à faciliter le montage pour les prochains utilisateurs de ces robots.

Figure 3 : Kit de montage du mBot2

Le montage est relativement simple et rapide, étant donné que le mBot2 est un robot éducatif, conçu pour être utilisable par tous. Dans sa boîte, nous pouvons trouver l'ensemble des pièces qui le composent ainsi que le nécessaire à son assemblage. Nous avons alors pris connaissance des différentes parties du robot, de ses différents capteurs, ainsi que de ses principales fonctionnalités. Le robot mBot2 est donc composé en globalité de :

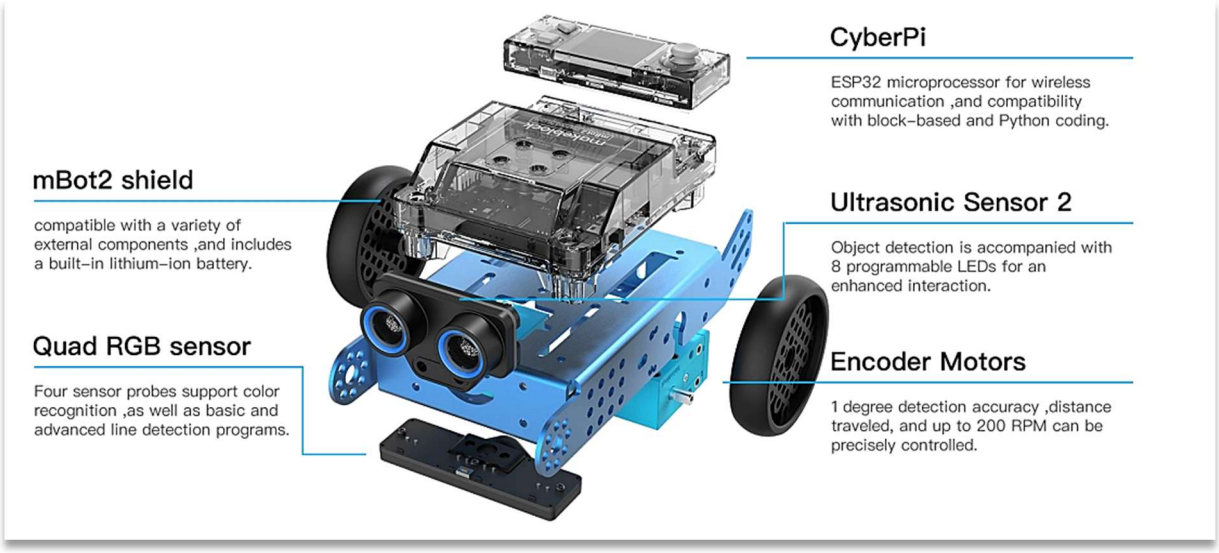


Figure 4 : Composition globale du mBot2

Voici également une liste détaillée des différents composants d'un robot mBot2 :

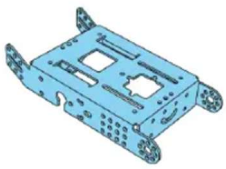
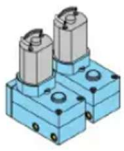
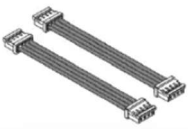
Liste des pièces détachées			
CyberPi		mBot2 Shield	
Capteur ultrason 2		Capteur RGB quadruple	
Slick Tyre		Roues	
Châssis		CyberPi Mini roue	
Encodeur moteur		Câble moteur	
Câble de connexion (10 cm)		Tournevis double tête	
Câble de connexion (20 cm)		Ensemble de vis à plusieurs dimensions	
Carte suiveuse de ligne		Câble USB	

Figure 5 : Pièces détachées du mBot2

B. Réalisation d'une vidéo explicative

Comme dit précédemment, nous avons eu l'idée de réaliser une vidéo du montage du robot afin de rendre cette étape plus éducative et ludique, mais aussi pour expliquer de manière générale ce qu'il est possible de faire avec les robots mBot2. Cette vidéo a été entièrement réalisée et montée par nos soins.

Pour le tournage, nous avons d'abord réalisé une première version sans trépied, mais les vidéos n'étaient pas assez stables. Étant donné que nous souhaitions obtenir une bonne qualité pour ce mini-projet, en complément des différents codes développés, nous avons décidé de refilmer l'ensemble des plans en utilisant cette fois-ci un trépied. Ensuite, nous avons procédé au montage de la vidéo sans expérience préalable dans ce domaine. Toutefois, cette étape nous a permis de découvrir un nouveau domaine et s'est révélée être une expérience formatrice.

Figure 6 : Screen vidéo

C. Prise en main des fonctionnalités

Après avoir réussi à monter les cinq robots, désormais nommés Bart, Lisa, Marge, Homer et Maggy (un moyen simple de les différencier) nous avons installé le logiciel mBlock sur l'ordinateur dédié au projet. À cette étape, nous avons commencé à nous intéresser à la documentation disponible concernant les robots ainsi que leurs différents composants.

Nous avons alors découvert que les robots disposent déjà, dans leur mémoire interne, de plusieurs programmes préinstallés, tels qu'un suiveur de ligne ou un détecteur d'obstacles. Souhaitant approfondir notre compréhension des capteurs, qui allaient être au cœur de nos différents codes, nous avons également recherché de la documentation spécifique à leur sujet. Toutefois, celle-ci s'est révélée globalement limitée et peu exploitable.

C’est ensuite en ouvrant le logiciel mBlock et en connectant un robot à l’ordinateur que nous avons pu commencer à développer nos premiers petits programmes. Dans un premier temps, nous avons utilisé l’interface Scratch, pour sa simplicité de prise en main et compréhension pour le début. Nous étions alors très enthousiastes à l’idée de tester les différentes fonctionnalités des robots, comme la synthèse vocale ou l’utilisation des capteurs. Ainsi, nos premiers codes consistaient à faire parler les robots, à les faire se déplacer, et à exploiter les capteurs à ultrasons et RGB sur des exemples simples, tels que la détection d’obstacles ou de lignes de différentes couleurs.

III. Codes et réalisations

Cette phase constitue le cœur du projet, où les concepts théoriques d'automatique et de programmation sont appliqués au système. L'objectif est de transformer le mBot2 d'un simple objet éducatif en une véritable plateforme d'expérimentation, capable de servir au développement et au test de différents codes et méthodes.

Pour programmer les robots mBot2, nous avons utilisé le logiciel mBlock. Celui-ci permet de coder de deux manières : soit à l'aide de blocs Scratch, soit avec le langage Python. Pour l'ensemble des codes présentés par la suite, nous avons choisi d'utiliser Python, un langage que nous maîtrisons bien et qui offre une grande liberté et flexibilité dans le développement des programmes.

Afin de répondre aux exigences du PMI, nous avons également entrepris de comprendre certaines fonctions issues des librairies *mbot* et *CyberPi* pour les retranscrire en MicroPython. Ces fonctions sont répertoriées et disponibles dans l'Annexe 1, 2 et 3. Cette démarche vise à encourager les futurs étudiants à mieux comprendre la logique interne du code et son fonctionnement, plutôt que de se reposer uniquement sur des fonctions prédéfinies.

L'ensemble des codes développés dans le cadre de cette phase est disponible sur le dépôt GitHub du projet : https://github.com/PMIRobot et pour chaque code décrit, le fichier de référence est indiqué.

Bien que certaines librairies simplifient fortement le développement et soient utiles pour une première prise en main, une compréhension approfondie des actions liées aux capteurs et aux moteurs reste essentielle. Cette étape permet notamment un contrôle plus précis du processeur CyberPi, comme l'ajustement fin de la fréquence des moteurs ou de la sensibilité des capteurs.

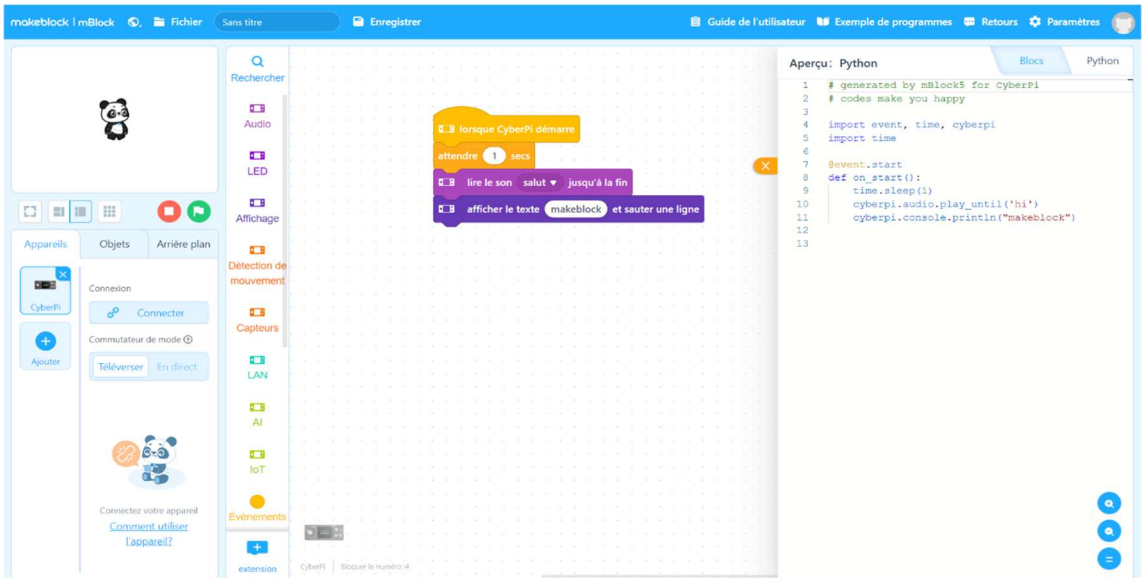


Figure 7 : Interface Mblock

En ce qui concerne le mBot2 et ses différents composants, le schéma ci-dessous synthétise leurs principales fonctionnalités.

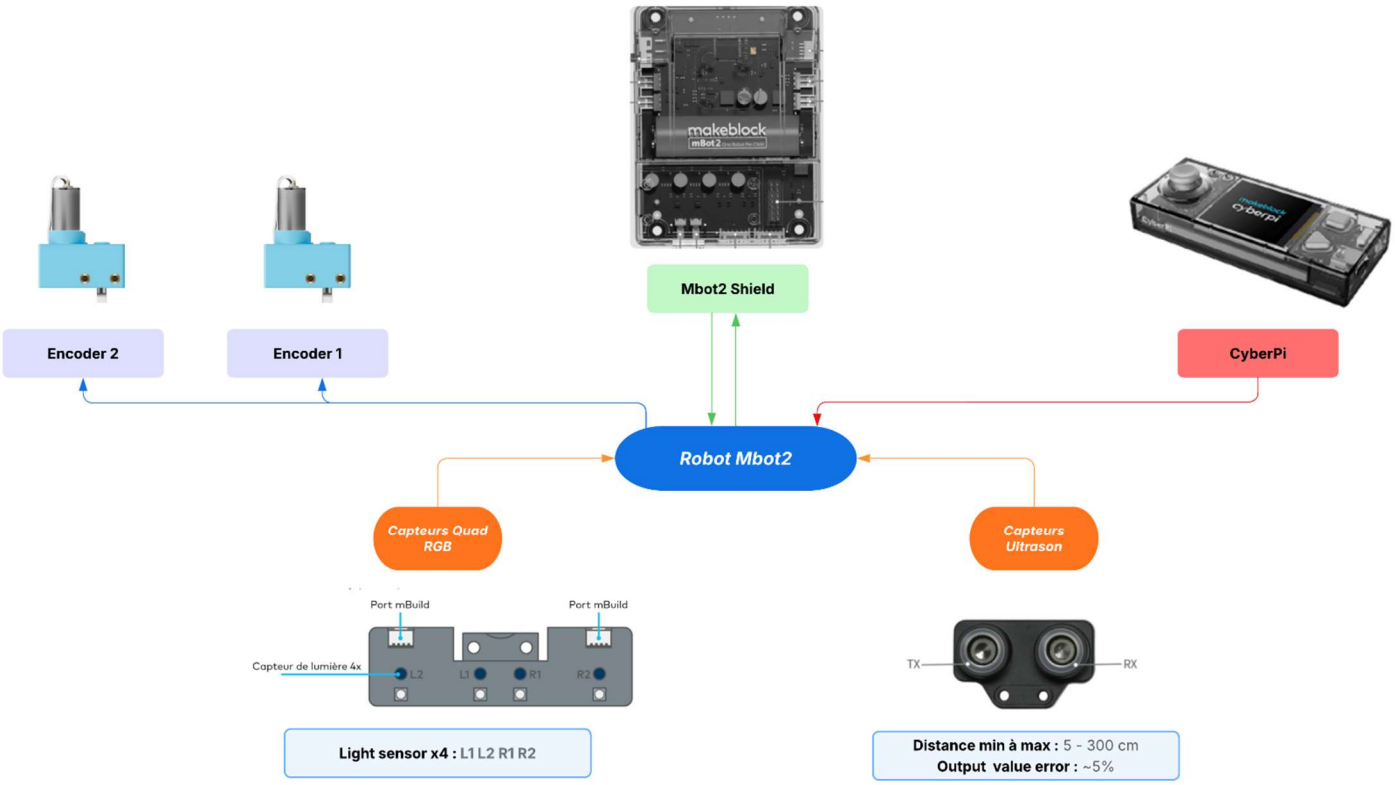


Figure 8 : Graphes des composants

Ici, sont représentés les différents composants en liaisons avec le Mbot2. Parmi eux nous retrouvons notamment la CyberPi et le Shield. Voici une liste des différentes fonctionnalités de ses derniers :

CyberPi :

Joystick	Capteur lumière	Microphone
Bouton A et B	Home bouton	Ecran Full-color

Shield :

Power switch	mBuild port	Multifonction port S1 et S2	Servo port S3 et S4
CyberPi port	DC motor port M1 et M2	Encoder motor port EM1 et EM2	

A. Implémentation des premiers codes

Dans un premier temps, nous avons utilisé l'interface simplifié en blocks de mBlock pour rapidement valider la logique algorithmique et le bon branchement des actionneurs/capteurs du robot. Une fois que l'ensemble des éléments et des branchements du robot ont été vérifiés, nous avons pu commencer par implémenter des codes simples utilisant les données que le robot peut recevoir.

1. Code : Suivi de ligne

Fichier associé : mblock_suivie_de_ligne.py

But : Suivre une ligne prédéfinie d'une certaine couleur, par exemple celle fournie dans le kit du mBot2.

Eléments utilisés : Capteurs RGB, encodeurs moteurs.

Schéma de fonctionnement :

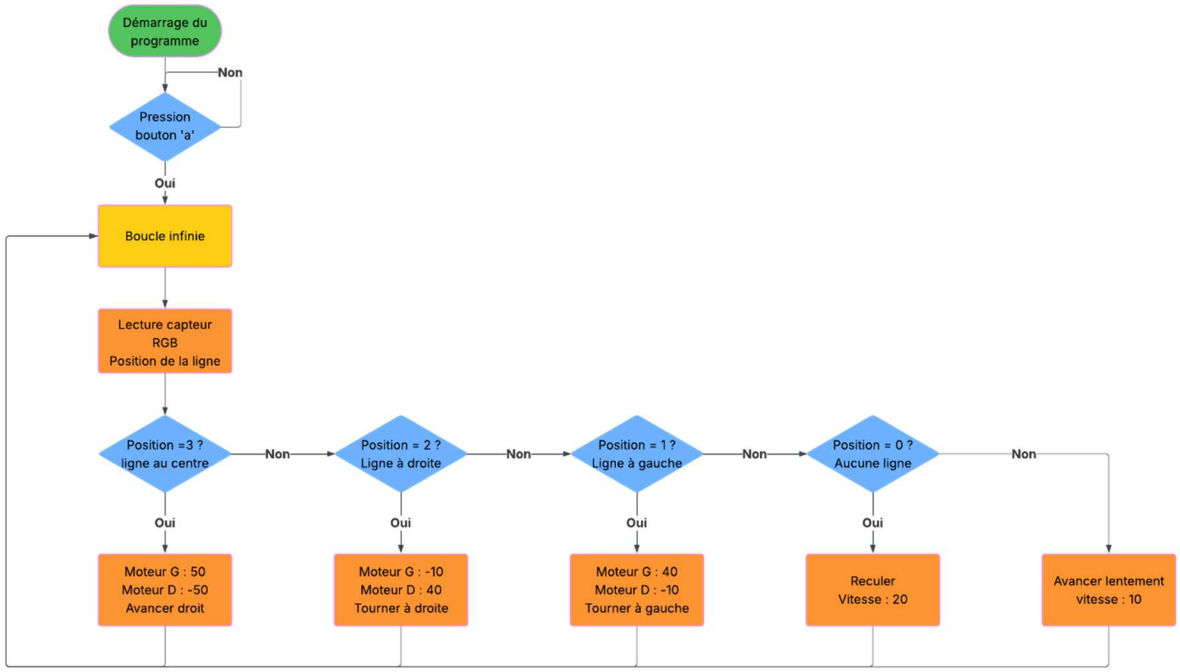


Figure 9 : Schéma du code de suivi de ligne

Explications : Ce code utilise le capteur RGB quadruple pour détecter le contraste et ajuster la vitesse différentielle des moteurs. L'exécution du suivi démarre lorsque l'utilisateur appuie sur le bouton « A », grâce au système d'événements. Une boucle infinie permet ensuite d'analyser en continu la position de la ligne sous le robot.

La fonction principale get_line_sta() du capteur quad RGB détecte la position de la ligne (centre, gauche, droite ou absente). En fonction de cette information, le programme ajuste la vitesse des moteurs gauche et droit via EM_set_speed() afin de corriger la trajectoire. Des commandes de déplacement simples comme forward() et backward() sont utilisées comme comportements par défaut.

Ainsi, le robot adapte dynamiquement sa vitesse et sa direction pour rester aligné avec la ligne, en combinant la lecture en temps réel des capteurs et un contrôle basique mais efficace des moteurs.

Observations : Il existe différentes manières d'imposer un critère sur le capteur RGB. Par exemple, au lieu de prendre des modes déjà définis, on peut imposer un pourcentage de coloris du capteur pour la correction, afin de d'être plus fluide et précis

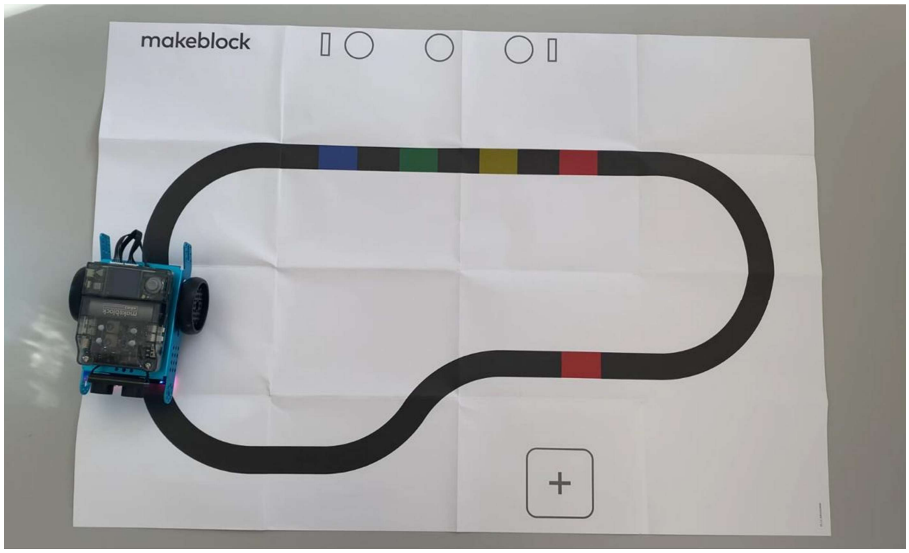


Figure 10 : Exemple de trajectoire pour le suivi de ligne

2. Code : Eviter un obstacle

Fichier associé : mblock_obstacle_avoidance.py

But : Eviter les collisions avec des obstacles.

Eléments utilisés : Capteurs ultrasons, encodeurs moteurs.

Schéma de fonctionnement : Voir Annexe 4

Explications : Ce programme exploite les capacités de la bibliothèque *cyberpi* pour gérer l'interface utilisateur, notamment via *cyberpi.console.println*() qui affiche les instructions et *cyberpi.controller.is_press()* qui conditionne le départ du robot à une action physique. La logique de navigation repose sur le capteur de distance via *mbuild.ultrasonic2.get(),* créant un système réactif où le robot avance avec *mbot2.forward()* tant que la voie est libre. Dès qu'un obstacle est détecté sous le seuil des 8 cm, le code utilise la fonction *random.randint()* pour choisir aléatoirement une direction, puis exécute un pivotement avec *mbot2.turn_left()* ou *turn_right().* L'intégration de la bibliothèque *time* avec la fonction *time.sleep(1)* est ici fondamentale, car elle permet de maintenir le mouvement de rotation pendant une seconde complète, laissant au châssis le temps nécessaire pour s'orienter vers un nouvel angle avant que le capteur ne reprenne son analyse de l'environnement.

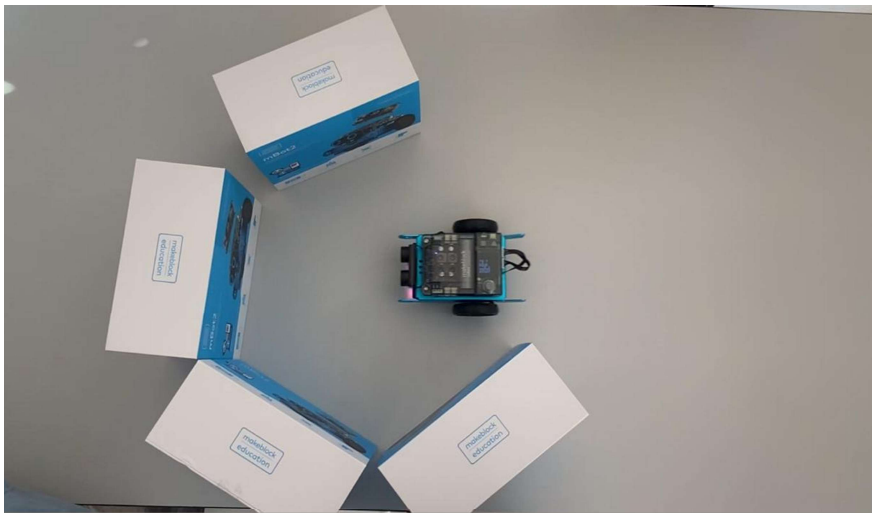


Figure 11 : Exemple évitement d'obstacle

Observations : Les capteurs ont un cône de vision de 30° donc le robot n'est pas capable de détecter les obstacles à l'extérieur de ce cône ce qui peut mener à des collisions sur les extrémités du robot.

3. Code : RGB et LEDs

Fichier associé : mblock_RGB.py

But : Lier les données des capteurs RGB à une action lumineuse sur la CyberPi. Par exemple si du rouge est détecté par le capteur RGB, la même couleur sera affichée au dos du robot.

Eléments utilisés : Capteurs RGB, ruban LED de la CyberPi

Explications : Ce programme utilise les bibliothèques *cyberpi* et *mbuild* pour transformer le robot en un analyseur de couleurs en temps réel capable de synchroniser sa perception sensorielle avec ses indicateurs visuels.

Le script entre dans une boucle infinie où le capteur *mbuild.quad_rgb_sensor.get_red()* (ainsi que ses variantes pour le vert et le bleu) extrait les composantes colorimétriques précises de la sonde R1. Ces données brutes sont ensuite formatées en une chaîne de caractères pour être projetées

sur l'écran via *cyberpi.display.show_label()*, permettant un monitoring direct des valeurs RVB. Enfin, le code établit un lien matériel direct grâce à *cyberpi.led.on()*, qui injecte les variables de couleur récupérées par le capteur dans les LED embarquées du CyberPi, créant ainsi une boucle de rétroaction visuelle entre la surface scannée et l'éclairage du robot.

Observations : Le robot dispose de quatre capteurs RGB qui sont de gauche à droite L2, L1, R1,R2. Chacun de ces capteurs est indépendant et peut relever une mesure différente (niveau de gris, niveau de rouge…).

B. Intercommunication et gestion de flotte

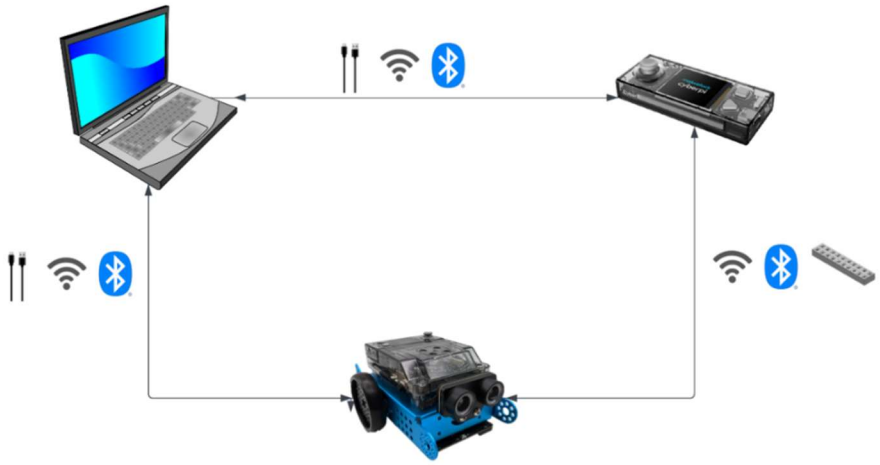


Figure 12 : Graphe de liaisons

Pour la suite, dans chaque code, on peut considérer différentes liaisons. Pour téléverser on privilégiera la liaison câbles USB-C et dans le cadre d'intercommunication ou d'extraction de données, on utilisera le Wi-Fi en considérant qu'il existe l'alternative du Bluetooth. La télécommande peut, elle, être directement fixée sur le mBot2 par une broche de connecteurs.

1. Utilisation du Wi-Fi

Dans un second temps, nous avons concentré nos efforts sur l'utilisation du Wi-Fi intégré à la carte CyberPi qui a permis d'explorer des scénarios de systèmes distribués. L'intérêt de ces outils de communication sans fils est de pouvoir récupérer les données en temps réel ou en batch des différents capteurs, sans s'encombrer de câbles. Puis dans un second temps de pouvoir faire communiquer les robots entre eux. La communication inter robots se fait à travers un canal Wi-Fi privé, où ces derniers peuvent s'échanger des messages. Ces messages sont des chaines de charactère qui peuvent être identifiés et liés à des action précises de mouvements, sons, ou réponses.

Une des méthodes d'utilisations est la suivante :

Dans un premier temps connectez obligatoirement votre ordinateur et les robots au même réseau Wi-Fi. Dans le code du logiciel mBlock en langage Python qui va sur la CyberPi, utilisez l'appel de réseau comme ci-dessous :

```
1
2   # Wi-Fi credentials
3
4   WIFI_SSID = "Nom_du_réseau"
5
6   WIFI_PASSWORD = "Mot_de_passe_du_réseau"
7
8   # Connect to Wi-Fi
9
10  cyberpi.display.show_label("Connecting WiFi", 16)
11  cyberpi.wifi.connect(WIFI_SSID, WIFI_PASSWORD)
12
13  while not cyberpi.wifi.is_connected():
14      time.sleep(0.5)
15
16  cyberpi.display.show_label("WiFi Connected", 16)
17  time.sleep(1)
18
```

Figure 13 : Exemple de connexion Wi-Fi

2. Code : Une télécommande plusieurs robots

Fichier associé : Robot_telecommande.py , telecommande.py

Objectif : Commander les déplacements de plusieurs robots à l'aide d'une seule télécommande CyberPi.

Eléments utilisés : CyberPi, plusieurs robots, communication Wi-Fi

Schéma de fonctionnement : Voir Annexe 5 et Annexe 6

Explications : Cet ensemble de codes illustre un système de communication sans fil utilisant la bibliothèque *cyberpi* pour créer une télécommande à distance. Le premier script configure un CyberPi comme récepteur en utilisant *cyberpi.wifi.connect()* pour établir la liaison réseau, puis reste à l'écoute via l'événement *cyberpi.event.mesh_broadcast()*, qui déclenche les mouvements du mbot2 et change la couleur des *cyberpi.led* selon le signal reçu. En parallèle, le second script transforme un autre appareil en émetteur, utilisant *cyberpi.controller.is_press()* pour détecter les inclinaisons du joystick et les traduire en messages envoyés via *cyberpi.mesh_broadcast.set()*. L'utilisation conjointe de la fonction *cyberpi.mesh_broadcast.get()* côté robot et de la temporisation *time.sleep()* côté manette permet un contrôle fluide et réactif, sans inonder le réseaux d'instructions, où chaque instruction logicielle est acheminée pour piloter les moteurs en temps réel.

Observations : Nous avons constaté des délais de communication (latence) lors de l'envoi simultané de commandes, illustrant les limites de la bande passante et les enjeux de la synchronisation en temps réel. De plus, Ces essais ont mis en évidence la nécessité de définir des règles d'arbitrage ou des priorités de commande pour éviter des erreurs du robot en cas de requêtes contradictoires.

3. Code : Un robot plusieurs télécommandes

Fichier associé : Robot_telecommande.py , telecommande.py

But : Commander un robot avec plusieurs télécommandes afin de tester la robustesse et la résilience du système.

Eléments utilisés : Plusieurs CyberPi, communication Wi-Fi

Schéma de fonctionnement : Voir Annexe 5 et Annexe 6

Explications : Au lieu d'une seule télécommande qui pilote un robot, plusieurs CyberPi envoie des instructions à un seul robot.

Observations : Lors d'instructions contradictoire on remarque des mouvements erratiques du robot. De plus, la latence entre les inputs et les mouvements est plus importante. On remarque aussi qu'il n'y a pas de priorité de connexion entre les télécommandes mais que tout se fait en même temps dans le canal. Il serait donc intéressant d'imposer des priorités.

4. Code : Communication en chaînes

Fichier associé : mblock_Envoie_message.py, mblock_Recois_message.py

But : Communiquer entre les différents robots afin de transmettre des informations ou des instructions.

Eléments utilisés : Communication Wi-Fi, Capteur RGB (optionnel)

Schéma de fonctionnement : Voir Annexe 7 et 8

Explications : Ce programme met en place une interaction réseaux entre deux robots via le protocole *WiFi Broadcast*, où le premier sert de détecteur et le second d'exécuteur. Le premier robot utilise *cyberpi.wifi.connect()* pour s'enregistrer sur le réseau, puis surveille son environnement grâce à *mbuild.quad_rgb_sensor.get_gray()*, une fonction qui mesure l'intensité lumineuse pour identifier une ligne noire au sol. Dès que le seuil de luminosité est franchi, le script déclenche *cyberpi.wifi_broadcast.set()* pour envoyer une commande spécifique sur un canal nommé "topic". Le second robot, constamment à l'écoute via *cyberpi.wifi_broadcast.get()*, analyse le message reçu et exécute l'action associée : il entame une marche de trois secondes à la réception du signal "Avance !" ou s'immobilise si le signal est "Stop !". Cette architecture, cadencée par des fonctions *time.sleep()* pour stabiliser les échanges réseau, permet de synchroniser les mouvements d'une flotte de robots en fonction des obstacles ou des marquages au sol rencontrés.

Observations : Les délais de latence sont considérables quand les messages sont envoyés en continu. Pour pallier ce problème on implémente une pause de 100ms dans le code.

C. Extraction et exploitation des données

L'intérêt de cette partie est de pouvoir utiliser les outils Wi-Fi des robots dans le but d'exploiter les données des capteurs en temps réel ou en les stockant dans une banque de donnée.

1. Utilisation d'un serveur en ligne

```python
#computer IP and port

SERVER_IP = "10.254.141.2"  # <-- Your computer IP
SERVER_PORT = 5000

# Create TCP socket

sock = socket.socket()
sock.connect((SERVER_IP, SERVER_PORT))

cyberpi.display.show_label("Connected to PC", 16)
time.sleep(1)

```

```
C:\Users\saman>ipconfig

Carte réseau sans fil Wi-Fi :

   Suffixe DNS propre à la connexion. . . :
   Adresse IPv6 de liaison locale. . . . .:
   Adresse IPv4. . . . . . . . . . . . . .:
   Masque de sous-réseau. . . . . . . . . :
   Passerelle par défaut. . . . . . . . . :
```

Figure 14 : Codes python et Matlab pour une connexion au serveur

La première solution que nous avions développé pour récupérer les données des capteurs était d'héberger une page web locale, à l'aide d'un code python où les robots pourront exporter les valeurs des capteurs en temps réel qui sont ensuite affiché dans une interface web simplifiée. Après un certain temps, ces données sont stockées dans une matrice qui est ensuite exporté vers Matlab pour exploitation. Cette première solution, bien que fonctionnelle, pose plusieurs limites : tout d'abord, il y'a une latence évidente entre le moment de la prise de mesure et son affichage. Ensuite, la prise en main du processus n'est pas optimale, il faut un code python pour le robot, un ordinateur qui servira d'hôte pour héberger le serveur avec un autre code python, un accès à la page web, et enfin une exportation des données sur Matlab. Ajouté à cela que la mise en place du système n'est pas intuitive, étant donné qu'il faut manuellement insérer l'adresse IP du serveur hôte et les informations du réseau Wi-Fi. L'ensemble fait que le code est propre à l'ordinateur utilisé et le réseau Wi-Fi connecte. Il faut l'adapter selon l'appareil et le réseau Wi-Fi de l'hôte. Nous avons alors réfléchi à des alternatives plus simples.

2. Utilisation d'un serveur local Matlab

Dans l'effort de simplifier le système, et après plusieurs recherches, nous avons réussi à simplifier la récupération des données, à un seul code Matlab qui héberge le serveur local (à l'aide d'une toolbox Matlab nommée « TCP toolbox »), récupère les données, les affiche en temps réel et les stocks. Cela permet de grandement simplifier le processus étant donné qu'on ne travaille qu'avec un code python sur le logiciel mBlock et un code Matlab sur l'ordinateur. Au lieu de 2 codes python, un code Matlab et une page web. Cela permet également de réduire la latence et d'avoir des valeurs plus précises en évitant les erreurs de mesures qui s'accumulent.

```
port = 5000;
adresseIP = "0.0.0.0"; % mettre adresse IPV4 de l'ordi
```

Figure 15 : Serveur local Matlab

3. Code : Extraction de distance

Fichier associé : mblock_envoie_distance_matlab.py , serveur_matlab_PMI.m

But : Extraire les données des capteurs ultrasons sur Matlab par wifi, et les stocker pour de futurs usages.

Eléments utilisés : Capteurs ultrasons, Wi-Fi.

Explications : Cette architecture de communication établit un pont de données entre le robot et un ordinateur via le protocole *TCP/IP*, permettant l'analyse des mesures de distances en temps réel. Le premier script utilise la bibliothèque *cyberpi* pour établir une connexion réseau avec *cyberpi.wifi.connect()*, puis initialise un connecteur réseau standard avec *socket.socket()* pour cibler l'adresse IP de l'ordinateur. Une boucle itérative récupère les mesures de distance via *cyberpi.ultrasonic2.get()* et les transmet au serveur sous forme de flux textuel encodé. Côté réception, le script *MATLAB* configure un point d'écoute avec *tcpserver*, où la fonction *readline()* intercepte les paquets entrants pour les convertir en valeurs numériques. Ce couplage permet de stocker les mesures dans des matrices temporelles (*distance_data* et *time_data*) et d'utiliser la puissance de calcul de MATLAB pour traiter les informations télémétriques envoyées par le mBot2 à une fréquence de 10 Hz, stabilisée par la fonction *time.sleep(0.1)*.

Observations : Le code s'exécute comme attendu, en général les latences sont minimes car lors du mouvement les variations de mesures sont progressives. On note néanmoins une latence importante quand les mesures de distances fluctuent grandement.

4. Code : Extraction RGB et distance

Fichier associé : logiciel_temps_réel_distance_et_rgb.py,

Matlab_temps_réel_distance_et_rgb.m

But : L'objectif ici est le même que le code précédent. L'ajout des données RGB sert à tester la robustesse du système quand plusieurs données sont transmises à la fois.

Eléments utilisés : Capteurs ultrasons, capteurs RGB, Wi-Fi

Explications : Similairement au code précédent, ce programme synchronise le mBot2 et l'environnement de calcul *MATLAB* pour réaliser une analyse multisensorielle en temps réel. Le script Python utilise *cyberpi.wifi.connect()* et la bibliothèque *socket* pour établir un flux de données constant, où il concatène les mesures de *cyberpi.ultrasonic2.get()* et les composantes chromatiques de *mbuild.quad_rgb_sensor()* dans une chaîne formatée envoyée via TCP/IP. Côté station de travail, le serveur MATLAB intercepte ces paquets avec *readline()* et utilise *strsplit* pour segmenter les valeurs de distance et de couleur. L'originalité de ce code réside dans son interface

de visualisation : MATLAB génère simultanément des graphiques temporels, une distribution pour l'espace colorimétrique et un aperçu visuel de la couleur détectée, tout en archivant l'intégralité de l'expérience dans des fichiers *.mat* et *.csv* pour un post-traitement. L'ensemble est cadencé par *time.sleep(0.1)* sur le robot et *drawnow* sur l'ordinateur, avec une fréquence de 10 Hz entre la perception et sa représentation numérique.

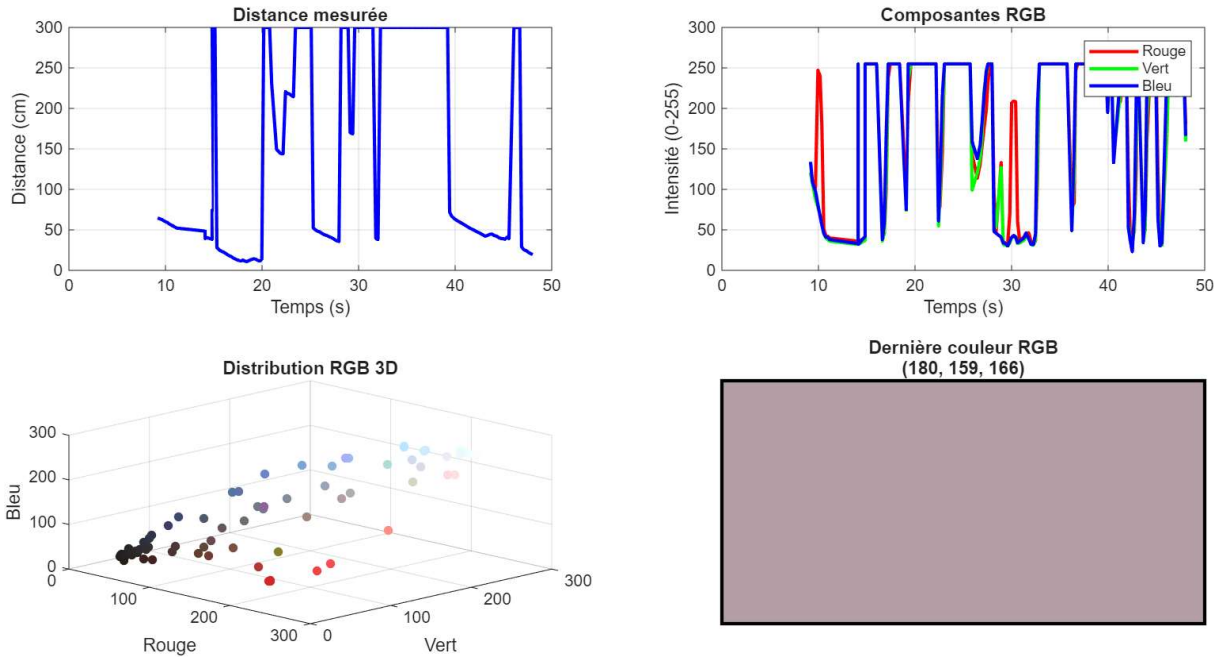


Figure 16 : Graphique temps réel distance et couleurs RGB

Observations : Il faut noter que le démarrage du code Matlab doit se faire avant de téléverser le code sur le robot obligatoirement. Nous avons voulu passer par des codes intermédiaires comme celui-ci qui rassemblait différentes composantes (ici le RGB et la distance) et les exploitait sur Matlab avant d'implémenter en plus une action guidée du robot.

5. Code : Suivi de ligne et extraction RGB et distance

Fichier associé : mblock_SuiviLigne_RGB_Distance.py, matlab_SuiviTraj_RGB_Distance.m

But : Toujours dans le but de tester la robustesse et la résilience du système, on ajoute plus de contraintes au code précédent en faisant un suivi de ligne en simultané de l'envoie de donnée.

Eléments utilisés : Capteurs RGB, capteurs ultrasons, Wi-Fi, encodeurs moteurs

Explications : Ce programme couple le suivi de ligne autonome du *mBot2* avec une analyse graphique temps réel sous *MATLAB* via une connexion *TCP/IP.* Le script Python gère le mouvement du robot en utilisant *mbuild.quad_rgb_sensor.get_line_sta()* pour détecter la position de la ligne et ajuster les moteurs avec *mbot2.EM_set_speed()*, tout en collectant simultanément la distance via *cyberpi.ultrasonic2.get()* et les couleurs via *mbuild.quad_rgb_sensor.get_red()*. Ces données sont envoyées par le biais de *sock.send()* vers l'ordinateur, où le serveur *MATLAB* les réceptionne avec *readline()* et les traite avec *strsplit().* L'interface *MATLAB* utilise alors *plot(), scatter3()* et

drawnow() pour transformer ces flux bruts en graphiques dynamiques, illustrant les mesures sensorielles du robot avant de compiler les résultats finaux dans des fichiers grâce aux fonctions *save()* et *writetable().* La synchronisation est assurée par *time.sleep()* côté robot pour éviter de surcharger le canal de messages.

Observations : Le robot est capable de cumuler les différentes taches données, mais il est nécessaire d'établir un ordre dans l'exécution du code.

6. Code : Estimation de trajectoire avec joystick

Fichier associé : matlab_retrace_traj_pilote.m, Robot_telecommande.py, telecommande_envoie_traj.py

But : Estimer la trajectoire du robot en temps réel sur Matlab dans une situation ou les mouvements sont piloté en utilisant le code de contrôle télécommandé à distance.

Eléments utilisés : Télécommande CyberPi, Wi-Fi, encodeurs moteurs.

Schéma de fonctionnement : Voir Annexe 9

Explications : Dans ce code, un contrôleur central pilote un robot tout en transmettant sa trajectoire à un serveur de calcul. Le premier script Python transforme un CyberPi en émetteur : il utilise *cyberpi.controller.is_press()* pour détecter les commandes du joystick, puis les diffuse simultanément via *cyberpi.mesh_broadcast.set()* vers le robot et via *sock.send()* vers l'ordinateur grâce à une connexion *socket*. Le second script, dédié au *mbot2*, intercepte ces signaux avec l'événement *cyberpi.event.mesh_broadcast()* pour activer les moteurs via *mbot2.forward()* ou *mbot2.turn_left()*. Enfin, le script *MATLAB* agit comme une station odométrique : il réceptionne les messages textuels avec *readline()*, puis applique un modèle de cinématique différentielle pour mettre à jour les coordonnées cartésiennes et l'angle d'orientation. Grâce aux fonctions *addpoints()*, *quiver()* et *drawnow()*, *MATLAB* génère une ligne animée représentant la trajectoire spatiale du robot en temps réel, offrant ainsi une reconstruction visuelle des déplacements effectués sur le terrain.

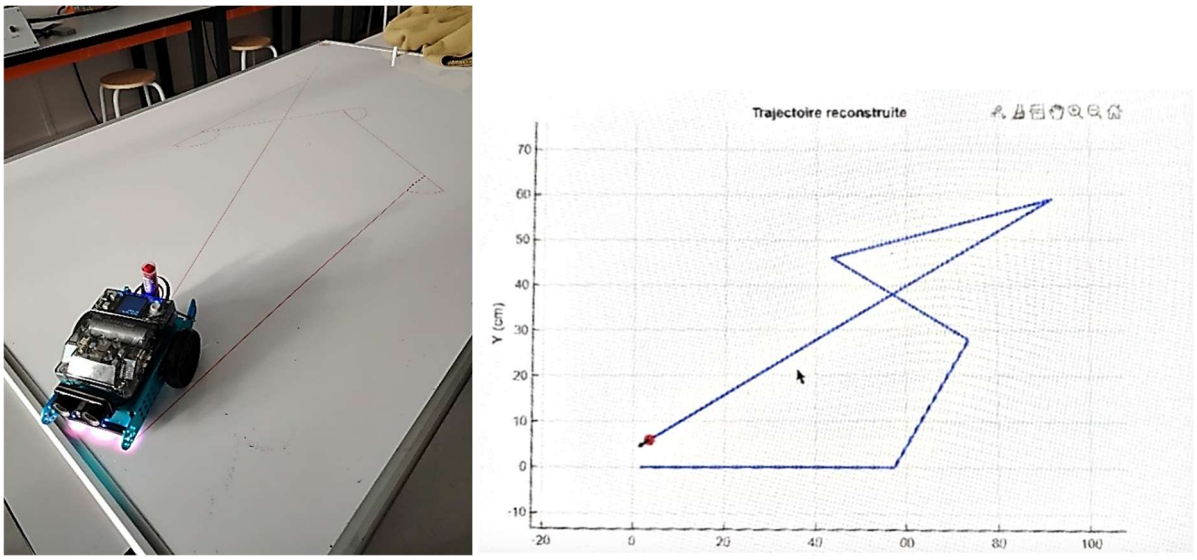


Figure 17 : Exemple de résultat entre trajectoire réelle et estimée

Observations : Lors de l'implémentation de ce code, l'introduction d'une phase de calibrage s'est révélée indispensable, en raison des écarts constatés à la fois sur la distance estimée et sur l'angle de rotation. Ces écarts sont principalement imputables aux limitations inhérentes au système de communication sans fil utilisé. En effet, le canal Wi-Fi introduit des délais variables (latence) susceptibles d'affecter la synchronisation entre l'émission et la réception des données. Par ailleurs, bien que le pas de temps d'acquisition soit choisi suffisamment faible afin de limiter ces effets, l'échantillonnage reste discret et ne peut donc être assimilé à une acquisition continue, ce qui induit une perte d'information et une approximation dans l'intégration des grandeurs cinématiques.

Ces phénomènes ont pour conséquence une accumulation d'erreurs au cours du temps, notamment lors de l'estimation de la position et de l'orientation, erreurs qui deviennent significatives lors de mouvements prolongés ou de changements rapides de direction. Afin de compenser ces dérives et d'améliorer la fidélité du modèle, une procédure de calibrage a été mise en place. Celle-ci repose sur l'exploitation des plans expérimentaux présentés ci-dessous, à partir desquels les valeurs réelles de vitesse linéaire et de vitesse angulaire ont été extraites manuellement. Ces valeurs ont ensuite été introduites dans l'environnement Matlab afin d'ajuster les paramètres du modèle et de réduire l'écart entre le comportement théorique simulé et le comportement réel observé. Cette étape de calibrage permet ainsi d'améliorer la robustesse et la précision globale du système.

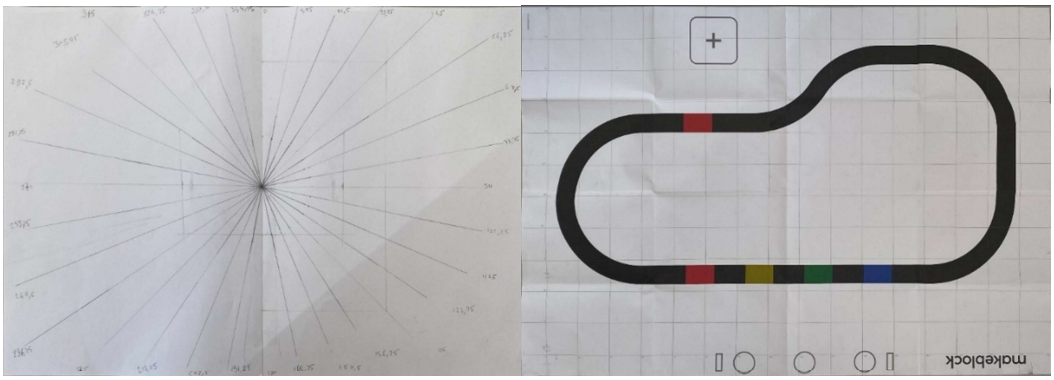


Figure 18 : Feuilles de calibrage

Après avoir essayé de tracer la trajectoire sur un tableau blanc en accrochant approximativement un stylo à l'arrière du robot, nous avons constaté que cette solution manquait de précision. Nous avons donc pris l'initiative de concevoir une pièce sur Onshape (logiciel de CAO) afin de fixer correctement un stylo à l'arrière du robot, en veillant à ce qu'il soit bien centré pour permettre un tracé plus propre de la trajectoire.

Le principal inconvénient de cette pièce est qu'elle ne permet pas de positionner le stylo exactement au centre de rotation du robot. Il est donc nécessaire de prendre en compte ce décalage lors de l'analyse de la trajectoire finale. Malgré cette limitation, cette solution nous offre une vision globale satisfaisante du parcours du robot.

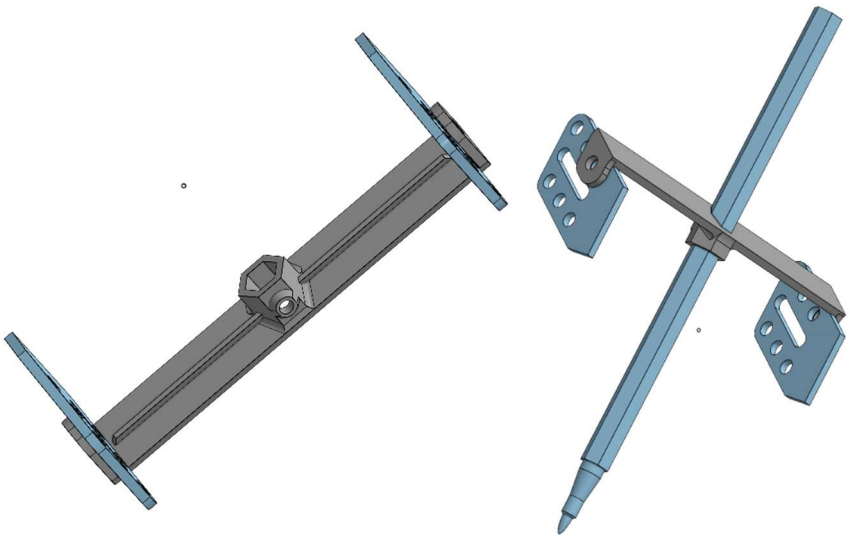


Figure 19 : Vues de la pièce support

Pour améliorer ce dispositif, il serait pertinent de concevoir un support offrant un espace plus large ou un design différent, afin de pouvoir accueillir plusieurs types de stylos. En effet, dans sa version actuelle, seul un crayon papier peut être utilisé.

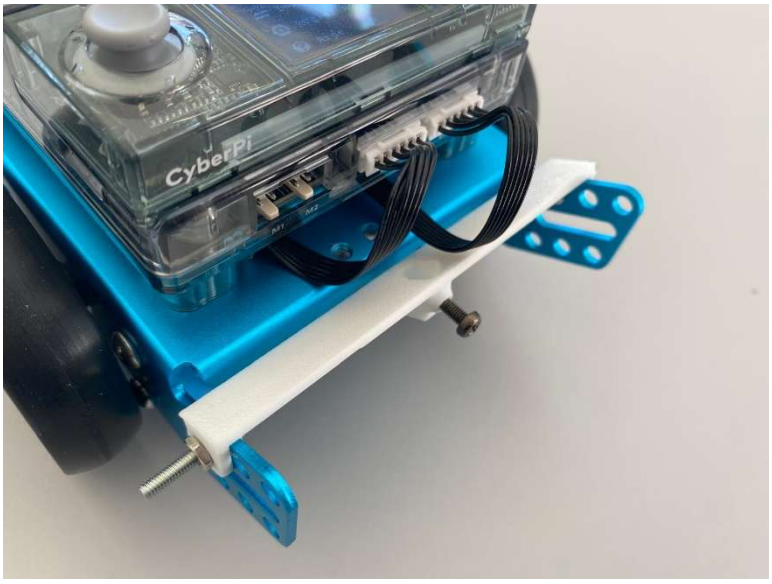


Figure 20 : Support crayon sur le robot mBot2

IV. Difficultés rencontrées

A. Vidéo explicative

Durant la création, nous avons rencontré quelques difficultés lors de la réalisation et du montage. Tout d'abord, pour la réalisation, nous avons produit une première version sans utiliser de trépied, mais les plans ne nous convenaient pas. Nous avons donc décidé de filmer une seconde fois l'intégralité de la vidéo.

Ensuite, en ce qui concerne le montage, nous n'avions aucune connaissance dans ce domaine. Il a donc fallu apprendre à monter, à enregistrer des voix off, et à utiliser les outils de montage. Cependant, cette expérience a été très formatrice et nous a permis de développer de nouvelles compétences.

B. Logistique

L'une des principales difficultés rencontrées concerne l'organisation du travail. En effet, la nécessité d'être présents à l'IPSA, dans une salle commune avec l'ensemble du groupe, limite les possibilités de travail individuel ou en petits groupes. Cette contrainte ne permet pas de poursuivre le travail à domicile, en dehors de la rédaction du rapport, ce qui réduit la flexibilité dans la gestion du temps et peut ralentir l'avancement du projet.

C. Extraction des données

Une difficulté importante a été rencontrée lors de l'extraction des données du robot mBot2. Dans un premier temps, une phase d'apprentissage a été nécessaire afin de comprendre le fonctionnement général du robot, son environnement de programmation ainsi que les différents capteurs disponibles. Cette étape, bien qu'indispensable, a été relativement chronophage et a retardé le début des travaux liés à l'exploitation des données.

Par la suite, l'équipe a constaté un manque significatif de ressources et de documentation concernant l'extraction et l'utilisation des données issues du mBot2. En effet, la majorité des supports disponibles se concentre principalement sur l'implémentation de consignes simples ou de comportements prédéfinis du robot, plutôt que sur la récupération, le traitement et l'analyse des données générées par ses capteurs. Ce manque d'informations a nécessité des phases de recherche et d'expérimentation supplémentaires, augmentant ainsi la complexité du travail et le temps requis pour parvenir à des résultats exploitables.

V. Améliorations possibles

Plusieurs pistes d'amélioration peuvent être envisagées afin d'optimiser le projet et d'enrichir les résultats obtenus. Tout d'abord, l'intégration d'un capteur de vitesse dédié permettrait de mesurer plus précisément les déplacements du robot. Cela offrirait la possibilité d'obtenir des données plus fiables sur la vitesse, l'accélération et les performances globales du mBot2, facilitant ainsi l'analyse et l'exploitation des données collectées.

Par ailleurs, l'utilisation de modules complémentaires compatibles avec le mBot2 constitue une amélioration intéressante. En particulier, l'ajout d'une caméra intelligente, pouvant être connectée au robot, permettrait d'introduire des fonctionnalités de vision artificielle. Ce type de capteur ouvrirait la voie à des applications plus avancées, telles que la reconnaissance d'objets, le suivi de trajectoire ou l'analyse de l'environnement en temps réel, rendant le projet plus complet et plus proche de situations réelles.

Dans le cadre des codes déjà implémentés, il serait utile de les optimiser notamment pour l'estimation de trajectoire grâce au joystick ou il faudrait affiner les méthodes de calibrage en trouvant une relation entre les différentes vitesses que l'on pourrait imposer au robot. De plus des codes en cours de développement pourraient être exploités par les élèves succédant à ce PMI comme un code qui détecte des landmarks et ou Matlab pourrait interférer dans les données Wi-Fi pour créer comme un correcteur de trajectoire.

Enfin, une amélioration pourrait également concerner l'accès à une documentation plus détaillée ou à des exemples spécifiques portant sur l'extraction et le traitement des données. Cela permettrait de réduire le temps consacré à la phase de recherche et d'expérimentation, tout en facilitant la prise en main des outils nécessaires. Une meilleure anticipation de ces aspects techniques dès le début du projet contribuerait à une organisation plus efficace et à des résultats plus approfondis.

Conclusion

Ce Projet Master IPSA a permis de transformer une plateforme robotique éducative en un outil d'expérimentation et de pédagogie. À travers la prise en main du mBot2, le développement de codes variés et l'exploitation des données issues des capteurs, nous avons pu mettre en pratique des notions essentielles en robotique, automatique, programmation et systèmes embarqués.

Ce projet nous a permis de mieux réaliser à quel point les contraintes du réel peuvent être un obstacle. Nous avons notamment pu l'observer en mettant en évidence les limites des modèles théoriques face aux phénomènes physiques tels que le bruit des capteurs, les frottements, la latence des communications ou encore les incertitudes de mesure. Cette confrontation entre théorie et pratique a constitué l'un des apports majeurs du projet, renforçant notre compréhension des enjeux de l'ingénierie appliquée.

Au-delà des aspects techniques, ce projet nous a également permit de développer des compétences transversales essentielles : autonomie, organisation, gestion du temps, travail en équipe, et capacité à résoudre des problèmes dans un contexte peu documenté. Malgré certaines difficultés, notamment en logistique, en documentation et en extraction de données, ces obstacles se sont révélés formateurs et ont contribué à enrichir notre expérience

D'un point de vue technique et académique, nous avons trouvé ce projet particulièrement stimulant et formateur. Il nous a permis de travailler sur un support concret, motivant et plus proche des problématiques rencontrées en milieu industriel, que ce sur quoi nous avons l'habitude de travailler. Le mBot2 s'est révélé être une plateforme pertinente pour initier des Travaux Pratiques en ingénierie, en offrant un bon compromis entre accessibilité, complexité et potentiel d'évolution.

Enfin, avec ce projet nous avons posé des bases solides pour les futures promotions, en proposant des codes exploitables et des perspectives d'amélioration intéressantes, notamment dans les domaines de la communication inter-robots et de l'exploitation avancée des données. Nous considérons ce PMI comme une expérience enrichissante, utile et représentative des exigences du métier d'ingénieur, et nous espérons qu'il contribuera durablement à l'enseignement de la robotique au sein de l'IPSA.

L'équipe PMI

Résumé

Ce rapport documente la prise en main et l'exploitation pédagogique du robot mBot2 dans le cadre de notre Projet Master IPSA (PMI). L'étude souligne l'évolution de nos travaux représentée par le mBot2, qui est une plateforme robotique basique contenant, pour les éléments majeurs, un capteur ultrasons, quatre capteur RGB, deux moteurs encodeurs, et une Cyber Pi compatible avec les programmes micro-python et Arduino.

Le travail présenté détaille plusieurs étapes clés : la découverte du matériel, la programmation de fonctionnalités fondamentales (suivi de ligne, évitement d'obstacles, gestion des LEDs RGB) et l'exploration de fonctions avancées. Une part importante du rapport est consacrée à l'intercommunication entre robots via Wi-Fi et à l'extraction de données vers des serveurs en ligne ou locaux, dans le but de pouvoir exploiter les données mesurées par les robots. Le document se conclut par des annexes répertoriant les fonctions des bibliothèques Python mbot2 et mbuild utilisées pour le contrôle du robot, et des Flowchart détaillant certains codes contenus dans ce rapport.

Abstract

This report documents the practical implementation and educational use of the mBot2 robot as part of our IPSA Master's Project (PMI). The study highlights the evolution of our work through the mBot2, a versatile robotic platform whose core components include an ultrasonic sensor, four RGB sensors, two encoder motors, and a CyberPi microcontroller compatible with MicroPython and Arduino environments.

The work presented details several key stages: hardware discovery, programming of fundamental features (such as line following, obstacle avoidance, and RGB LED management), and the exploration of advanced functionalities. A significant portion of the report is dedicated to inter-robot communication via Wi-Fi and data extraction to online or local servers, aimed at analyzing and processing the data measured by the robots. The document concludes with appendices listing the functions of the *mbot2* and mbuild Python libraries used for control, as well as flowcharts detailing the logic of specific programs included in this report.

Webographie

A4Téléchargement. (2023, janvier). *VISIO-presentation mBot2*.

https://www.a4telechargement.fr/MB-/VISIO-presentation_mBot2_janv2023.pdf

A4Téléchargement. (2021, juin). *Prise en main mBot2, CyberPi, mBuild avec mBlock5*.

https://www.a4telechargement.fr/mBlock/Prise_en_main_mBot2_CyberPi_mBuild_avec_mBlock5_Juin2021.pdf

A4Téléchargement. *mBot2 Getting Started Activities* [PDF].

https://www.a4telechargement.fr/MB-/MB-P1010132-mBot2_Getting_Started_Activities_FR.pdf

ArduiBlog. (2021, 10 mai). *mBot2*.

https://arduiblog.com/2021/05/10/mbot2/

Collège (Académie de Lille). *D-DB_092023* [Guide mBot2].

https://serveur.0622273j.clg.ac-lille.fr/techno.techno/mBot2/DOCUMENT/D-DB_092023.pdf

ENT2D (Académie de Bordeaux). (2023, janvier). *Notice d'utilisation mBot V2*.

https://ent2d.ac-bordeaux.fr/disciplines/sti-college/wp-content/uploads/sites/63/2023/01/Notice-dutilisation-mBot-V2.pdf

Pédagogie (Académie de Toulouse). (2021, novembre). *Install & Mise en route mBot2 App*.

https://pedagogie.ac-toulouse.fr/sii/system/files/2021-11/Install-MiseEnRoute-mBot2-App.pdf

Capytale. *Documentation Cyberpi*.

https://capytale.forge.apps.education.fr/documentation/Cyberpi/

Évolukid. *3 démonstrations à faire avec le mBot 2*.

https://www.evolukid.com/blog/3-demonstrations-a-faire-avec-le-mbot-2

Annexes

Annexe 1 : Tableau des fonctions de la bibliothèque CyperPi

Bibliothèque CyberPI	Arguments utilisés	Description
Affichage & Interface		
cyberpi.display.show_label()	text, size, x, y, ...	Affiche du texte sur l'écran du CyberPi. Les arguments définissent la taille de la police et la position (x,y).
cyberpi.display.clear()	Aucun	Efface tout ce qui est affiché sur l'écran.
cyberpi.console.println()	text	Affiche une ligne de texte dans la console (mode débuggage ou affichage rapide liste).
cyberpi.console.clear()	Aucun	Efface le contenu de la console.
cyberpi.console.set_font()	size	Change la taille de la police d'écriture de la console.
cyberpi.led.on()	r, g, b, id	Allume les LEDs RGB embarquées. r,g,b (0-255) définissent la couleur. id='all' allume toutes les LEDs.
Communication (Wi-Fi & Broadcast)		
cyberpi.wifi.connect()	ssid, password	Connecte le CyberPi à un réseau Wi-Fi spécifique.
cyberpi.wifi.is_connect()	Aucun	Vérifie si le Wi-Fi est connecté (retourne Vrai/Faux). *Note : Parfois noté is_connected() selon les versions*.
cyberpi.wifi_broadcast.set()	topic, message	Envoie un message à tous les robots écoutant le même "topic" via le Cloud/Wi-Fi.
cyberpi.wifi_broadcast.get()	topic	Récupère le dernier message reçu sur un "topic" spécifique.
cyberpi.mesh_broadcast.set()	message_id, value	Envoie un message en réseau local (Mesh/LAN) sans passer forcément par un routeur internet.
cyberpi.mesh_broadcast.get()	message_id	Lit la valeur d'un message reçu via le réseau Mesh.
Contrôle & Capteurs Internes		
cyberpi.controller.is_press()	'a', 'b', 'up', ...	Détecte si un bouton (A, B) ou le joystick (up, down, etc.) est pressé.
cyberpi.ultrasonic2.get()	Aucun	Récupère la distance en cm mesurée par le capteur ultrason (branché sur le shield du mBot2).
Cloud & IA		
cyberpi.cloud.tts()	langue, texte	"Text-to-Speech" : Le robot prononce le texte indiqué via son haut-parleur (nécessite Wi-Fi).

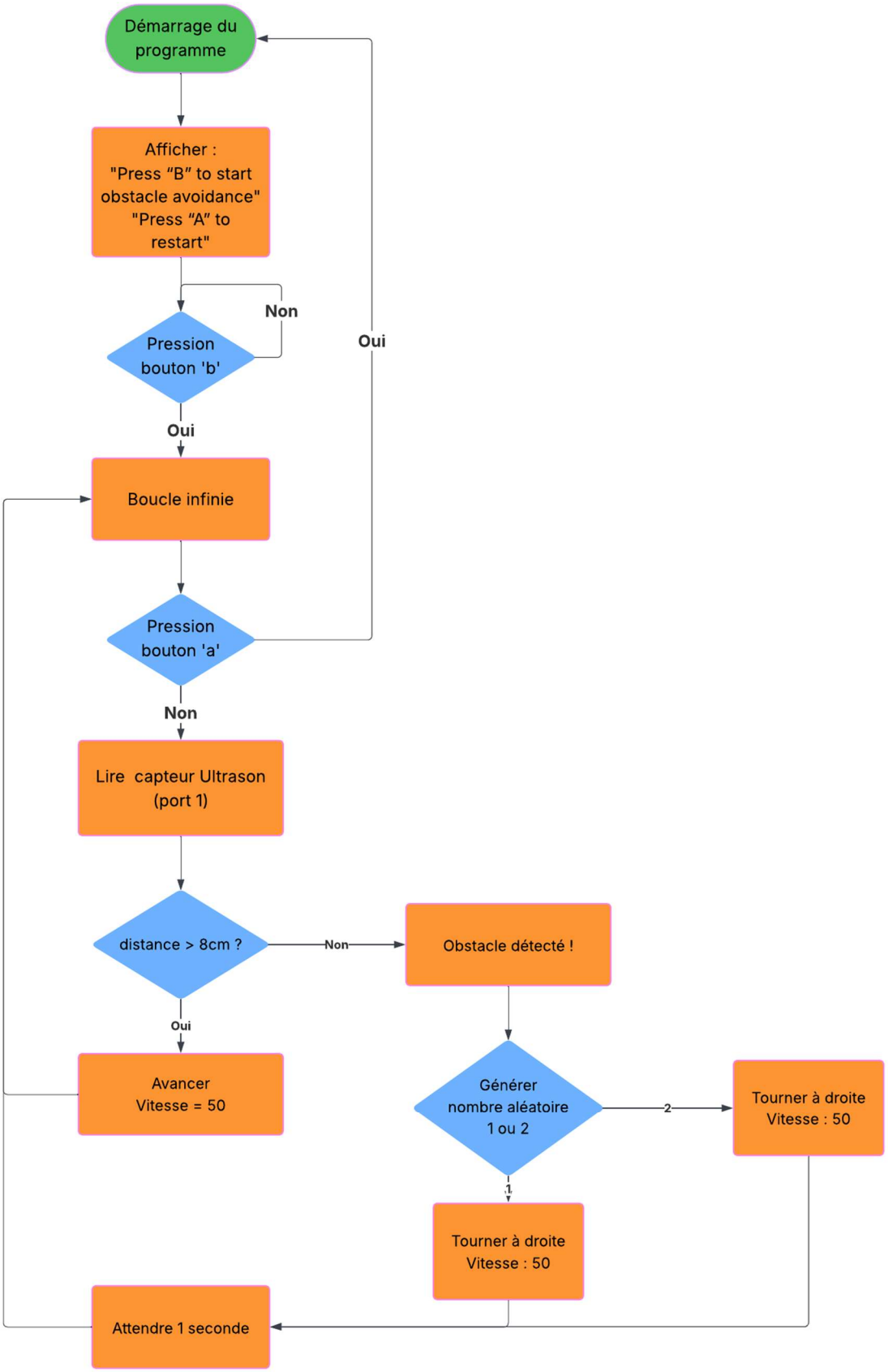
Annexe 2 : Tableau des fonctions de la bibliothèque mBot2

Bibliothèque mbot2	Arguments utilisés	Description
mbot2.drive_speed()	vitesse_gauche, vitesse_droite	Fait tourner les deux moteurs à des vitesses indépendantes (valeur négative pour reculer).
mbot2.forward()	vitesse, [temps]	Fait avancer le robot tout droit à une certaine vitesse. Peut prendre une durée optionnelle.
mbot2.backward()	vitesse	Fait reculer le robot tout droit.
mbot2.turn_left()	vitesse	Fait tourner le robot sur lui-même vers la gauche.
mbot2.turn_right()	vitesse	Fait tourner le robot sur lui-même vers la droite.
mbot2.EM_set_speed()	vitesse, port	Contrôle précis d'un moteur encodeur spécifique (ex: "EM1" ou "EM2").
mbot2.EM_stop()	"ALL" ou port	Arrête net les moteurs indiqués (ou tous).

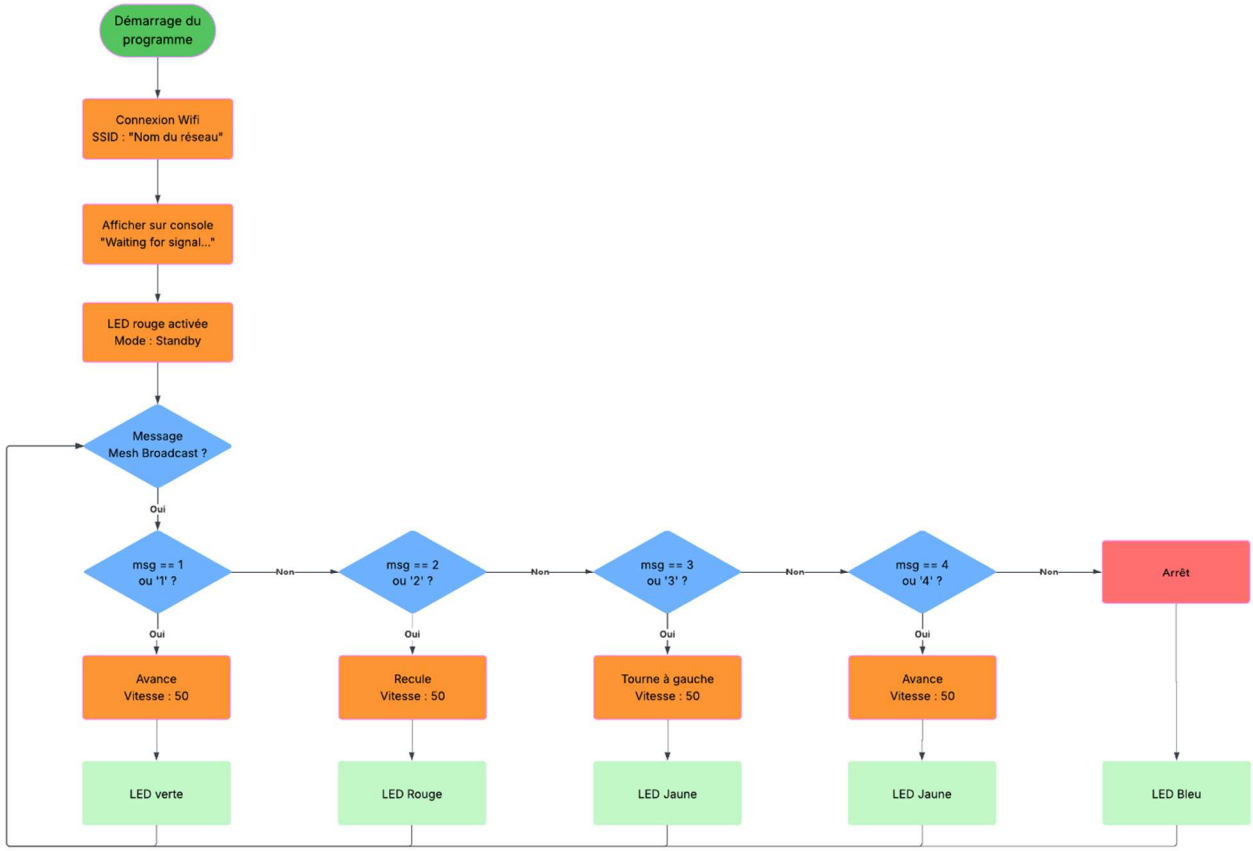
Annexe 3 : Tableau des fonctions de la bibliothèque mBuild

Bibliothèque mbuild	Arguments utilisés	Description
quad_rgb_sensor.get_gray()	index (ex: "L2")	Retourne la valeur de niveau de gris vue par un capteur spécifique (utile pour détecter le noir/blanc).
quad_rgb_sensor.get_line_sta()	position, channel	Retourne l'état de la ligne détectée (ex: 0=blanc, 1=noir) pour un capteur donné. Utilisé pour le suivi de ligne.
quad_rgb_sensor.get_red()	index	Retourne l'intensité de la composante **Rouge** vue par le capteur (0-255).
quad_rgb_sensor.get_green()	index	Retourne l'intensité de la composante **Verte** vue par le capteur.
quad_rgb_sensor.get_blue()	index	Retourne l'intensité de la composante **Bleue** vue par le capteur.

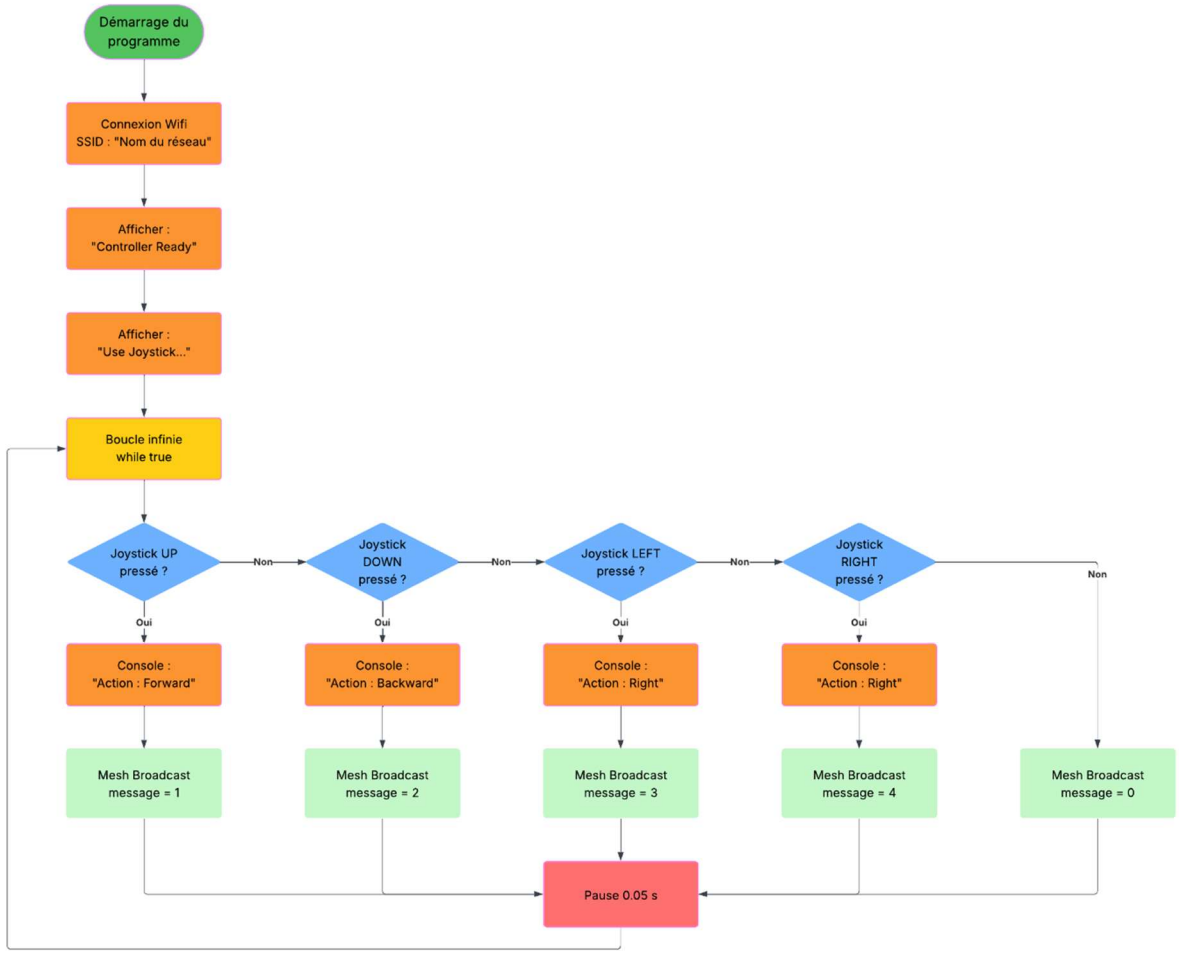
Annexe 4 : Schéma du code Eviter un obstacle



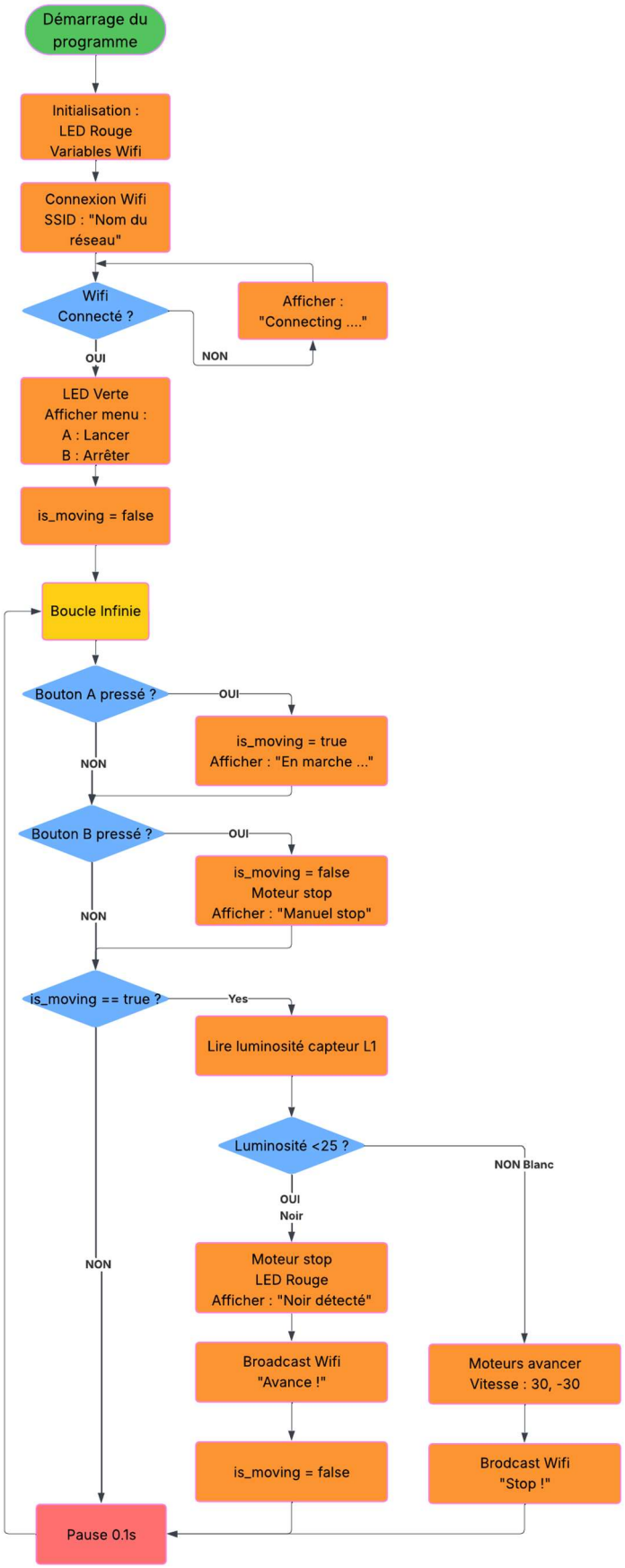
Annexe 5 : Schéma du code Robot télécommandé



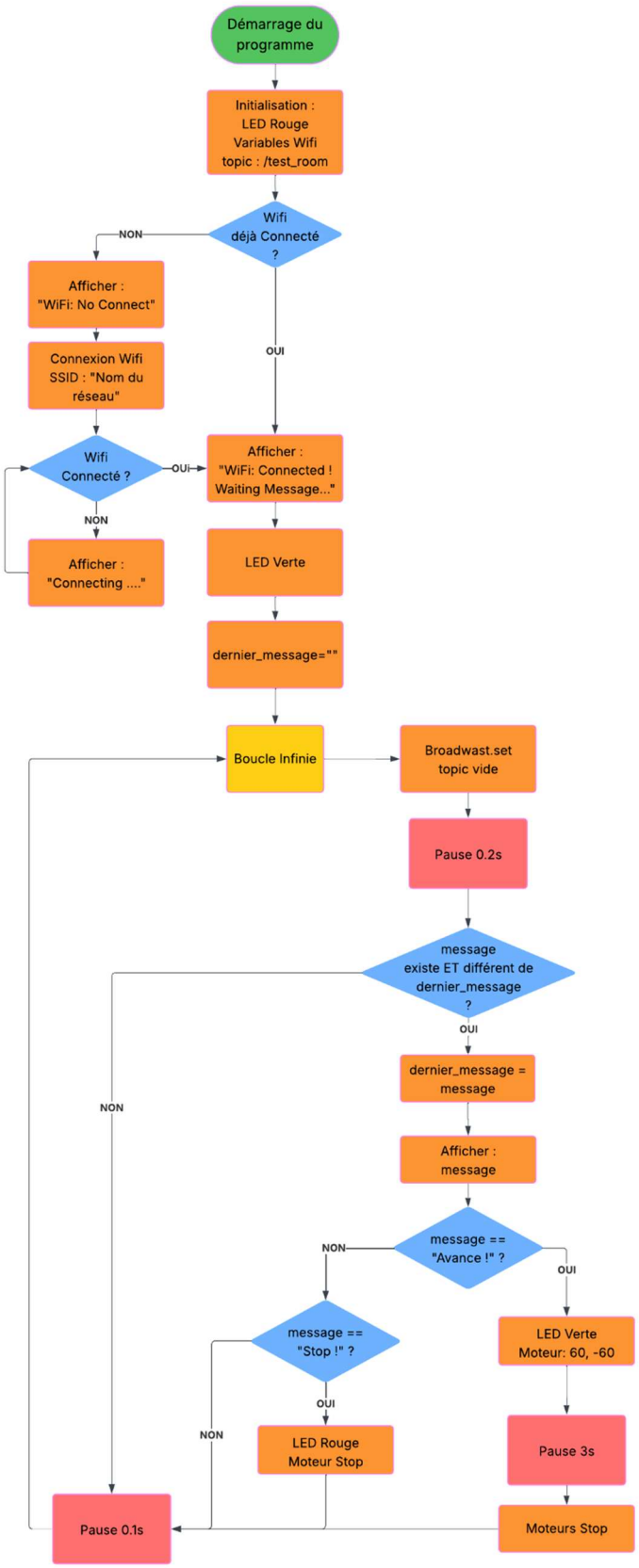
Annexe 6 : Schéma du code télécommande



Annexe 7 : Schéma du code communication en chaîne de l'envoi



Annexe 8 : Schéma du code communication en chaîne de la réception



Annexe 9 : Schéma du code tracé de trajectoire avec joystick