
Rapport de Projet : Conception et Programmation d'un Robot Autonome de Labyrinthe

Équipe : Loïc PORTOIS et Damien DINE-BULLOT

Module : Projet Robotique Mobile (I2)

Sommaire

1. Introduction et Cahier des charges
 2. Description matérielle (Hardware)
 3. Conception 3D, Assemblage et Câblage
 4. Programmation et Logique de navigation (Software)
 5. Résultats, Difficultés et Améliorations
 6. Conclusion
-

1. Introduction et Cahier des charges détaillé

1.1. Contexte général et enjeux du projet

Dans le cadre de notre formation en cycle préparatoire (année I2), le module de **Robotique Mobile** nous confronte à un défi technique majeur : la conception, l'assemblage et la programmation intégrale d'un système embarqué autonome. Ce projet a pour vocation de nous faire passer de la théorie à la pratique.

Pour mener à bien ce projet, notre équipe, composée de Loïc PORTOIS et Damien DINE-BULLOT, a dû mobiliser des compétences :

- **La mécanique et la CAO (Conception Assistée par Ordinateur) :** pour l'assemblage du châssis et la modélisation de pièces sur mesure.
- **L'électronique :** pour la gestion de l'alimentation, le câblage, et le développement d'une carte d'interface (Shield) visant à interconnecter les différents composants.
- **L'informatique et l'algorithmique :** pour le traitement des données en temps réel et la mise en place d'une intelligence artificielle basique capable de prendre des décisions de navigation (langage C++).

1.2. L'objectif principal : L'exploration autonome

L'objectif final de notre travail est de concevoir un robot capable d'évoluer de manière 100 % autonome à l'intérieur d'un circuit fermé délimité par des murs en bois, s'apparentant à un labyrinthe (bien que celui-ci ne comporte pas de pièges ou d'impasses).

La mission du robot est simple en apparence mais complexe dans son exécution : **il doit réaliser un tour complet de la piste le plus rapidement possible, sans jamais entrer en collision avec les parois.** Pour y parvenir, le robot doit être capable de "voir" son environnement, de comprendre sa position spatiale par rapport aux murs (trop près à gauche, trop près à droite, ou face à un obstacle), et d'adapter instantanément la vitesse de ses roues pour corriger sa trajectoire ou effectuer des virages à 90°.

1.3. Le Cahier des charges et contraintes techniques

Pour que notre projet soit validé, notre robot devait respecter un cahier des charges imposant plusieurs contraintes matérielles et fonctionnelles :

- **Autonomie totale** : Il s'agit de la règle d'or du projet. Une fois le robot mis sous tension et placé sur la piste de départ, aucune intervention humaine, aucune correction manuelle ni aucun pilotage à distance ne sont autorisés. Le robot doit s'appuyer uniquement sur son code et ses capteurs.
- **Base matérielle imposée** : Le projet nous impose de partir d'une base commune comprenant un microcontrôleur spécifique (Arduino Nano), un type précis de capteurs de distance (des capteurs laser de type VL6180X fonctionnant via le bus I2C) et des servomoteurs à rotation continue (FS90R).
- **Liberté d'optimisation et d'innovation** : Bien que la base matérielle soit fournie, le cahier des charges nous encourage fortement à innover pour optimiser les performances. Nous étions libres de créer des pièces en impression 3D pour améliorer l'ergonomie, de concevoir des cartes électroniques (Shield) pour fiabiliser le câblage, ou encore d'ajouter des interfaces de communication (écrans, alertes sonores).
- **Fiabilité et robustesse** : Le robot va subir de nombreuses phases de tests, d'arrêts et de redémarrages. Le cahier des charges exige donc un assemblage robuste, évitant les fils volants ou les composants mal fixés qui pourraient causer des courts-circuits ou des erreurs de lecture.
- **Performance chronométrique** : Le but n'est pas seulement de finir le parcours, mais de le faire avec la plus grande fluidité possible. Les saccades ou les collisions (même légères) ralentissant le robot, l'algorithme doit être optimisé pour anticiper les virages et maintenir une vitesse de croisière.

1.4. Organisation du rapport

Afin de rendre compte de notre démarche, ce rapport s'articulera de la manière suivante : nous présenterons tout d'abord l'architecture matérielle (Hardware) de notre robot en détaillant les composants utilisés. Ensuite, nous aborderons la conception 3D et le câblage de notre carte électronique sur mesure. Nous détaillerons par la suite le cœur de notre travail : la logique logicielle (Software) et l'algorithme de navigation que nous avons développé. Enfin, nous conclurons sur les résultats chronométriques obtenus en piste, les difficultés que nous avons dû surmonter, ainsi que les axes d'amélioration envisageables.

2. Description matérielle (Hardware)

Pour répondre aux exigences de réactivité, de précision et d'autonomie imposées par le cahier des charges, le choix et l'agencement des composants matériels sont cruciaux. L'architecture matérielle de notre robot a été pensée autour de cinq points : le traitement de l'information (microcontrôleur), l'acquisition de données (capteurs), l'actionnement (moteurs), l'interaction utilisateur (IHM) et l'énergie.

2.1. Le “Cerveau” : Le Microcontrôleur Arduino Nano Maker

Le rôle d'orchestrateur de l'ensemble du système est confié à une carte **Arduino Nano Maker**. Contrairement à une carte Arduino Uno classique, très encombrante, le format “Nano” a été privilégié pour sa compacité exceptionnelle, un atout indispensable pour l'intégration dans un châssis de robotique mobile. La version “Maker” offre plusieurs avantages notables pour le prototypage :

- **Capacité de traitement** : Basée sur le microcontrôleur ATmega328P, elle dispose de la puissance de calcul nécessaire pour lire simultanément plusieurs capteurs et ajuster les signaux moteurs en temps réel.
- **Connectivité** : Elle offre un nombre suffisant de broches d'entrées/sorties (GPIO) pour gérer à la fois les signaux PWM (Pulse Width Modulation) destinés aux moteurs, et le bus de communication I2C indispensable pour nos capteurs et notre écran.
- **Diagnostic visuel** : Les cartes de la gamme Maker intègrent souvent des LED témoins sur les broches, ce qui facilite grandement le débogage matériel lors de l'envoi des signaux.

2.2. Les “Yeux” : Capteurs de distance laser VL6180X (Pololu)

Pour permettre au robot de voir les murs, nous avons **3 capteurs Pololu VL6180X**, disposés stratégiquement : un à gauche, un au centre (face à la route) et un à droite.

Ces composants sont des bijoux de technologie basés sur le principe du **Temps de Vol (*Time-of-Flight* ou ToF)**.

- **Fonctionnement** : Le capteur émet une impulsion laser invisible et extrêmement rapide. Il mesure ensuite précisément le temps que met la lumière pour rebondir sur l'obstacle (le mur) et revenir à son récepteur.
- **Précision** : Cette technologie laser garantit une mesure millimétrique insensible à la couleur ou à la texture des murs (contrairement à l'infrarouge classique). Dans notre configuration, nous avons limité électroniquement la lecture maximale à 200 mm (20 cm), distance largement suffisante pour anticiper les virages dans le labyrinthe.
- **Protocole I2C** : Ces capteurs communiquent numériquement via le bus I2C (broches SDA et SCL). Cela garantit un transfert de données rapide et sans perte vers l'Arduino.

2.3. Les “Jambes” : La propulsion par Servomoteurs FS90R

Pour la motricité du robot, nous utilisons **deux servomoteurs à rotation continue de type Feetech FS90R** (les célèbres petits servos bleus). Contrairement à un moteur à courant continu (DC) classique qui nécessite un lourd pont en H (driver moteur) pour fonctionner, ou à un servomoteur standard dont la rotation est bridée à 180 degrés, le FS90R présente le compromis idéal :

- **Simplicité de commande** : Il se pilote directement via une broche de l'Arduino grâce à un signal PWM.
- **Logique de rotation** : La commande s'effectue en envoyant un angle théorique. Un angle de 90° ordonne l'arrêt total du moteur. Une valeur inférieure à 90° (ex: 75°) fait tourner le moteur dans un sens, et une valeur supérieure (ex: 105°) le fait tourner dans le sens inverse. La vitesse de rotation dépend de l'écart par rapport au point mort (90°).
- **Légèreté** : Ne pesant que 9 grammes chacun, ils n'alourdissent pas le châssis, préservant ainsi l'autonomie de la batterie.

2.4. L'Interface Homme-Machine (IHM) : Écran OLED et Buzzer

Afin de rendre notre robot “bavard” et de faciliter nos phases de tests sans avoir à brancher un câble USB en permanence, nous lui avons greffé une IHM :

- **L'Écran OLED (128x64 pixels)** : Connecté lui aussi en I2C (adresse 0x3C), cet écran est un véritable tableau de bord embarqué. Il affiche en temps réel le statut du robot (“START !”) au démarrage, puis indique les valeurs millimétriques lues par les capteurs Gauche, Centre et Droit (G: -- | C: 45 | D: --). Si un capteur ne voit rien (valeur > 200 mm), l'écran affiche “—”.
- **Le Buzzer (Broche 8)** : Un avertisseur sonore a été ajouté pour créer un *feedback* (retour) auditif immédiat. Lorsqu'un obstacle frontal critique est détecté à moins de 75 mm, le buzzer émet un son (1000 Hz), nous confirmant que le robot a bien vu le mur et qu'il amorce sa procédure de virage.

2.5. L'Énergie et la Structure Mécanique

- **Alimentation 5V** : L'intégralité du système (Arduino, 3 capteurs laser, 2 moteurs, 1 écran) est particulièrement énergivore et exige une tension parfaitement stable. Nous avons donc intégré une **batterie externe (Powerbank)** délivrant un courant constant de 5V. Cela évite les redémarrages intempestifs de l'Arduino qui surviennent souvent lorsque les moteurs pompent trop de courant sur une alimentation faible (comme une pile 9V classique).
- **Le Châssis** : La structure porteuse principale est constituée d'une base robuste imprimée en 3D (plastique PLA) qui nous a été fournie. Elle permet de fixer les moteurs et de loger la batterie en son centre pour abaisser le centre de gravité du robot, garantissant une meilleure adhérence des roues sur la piste.

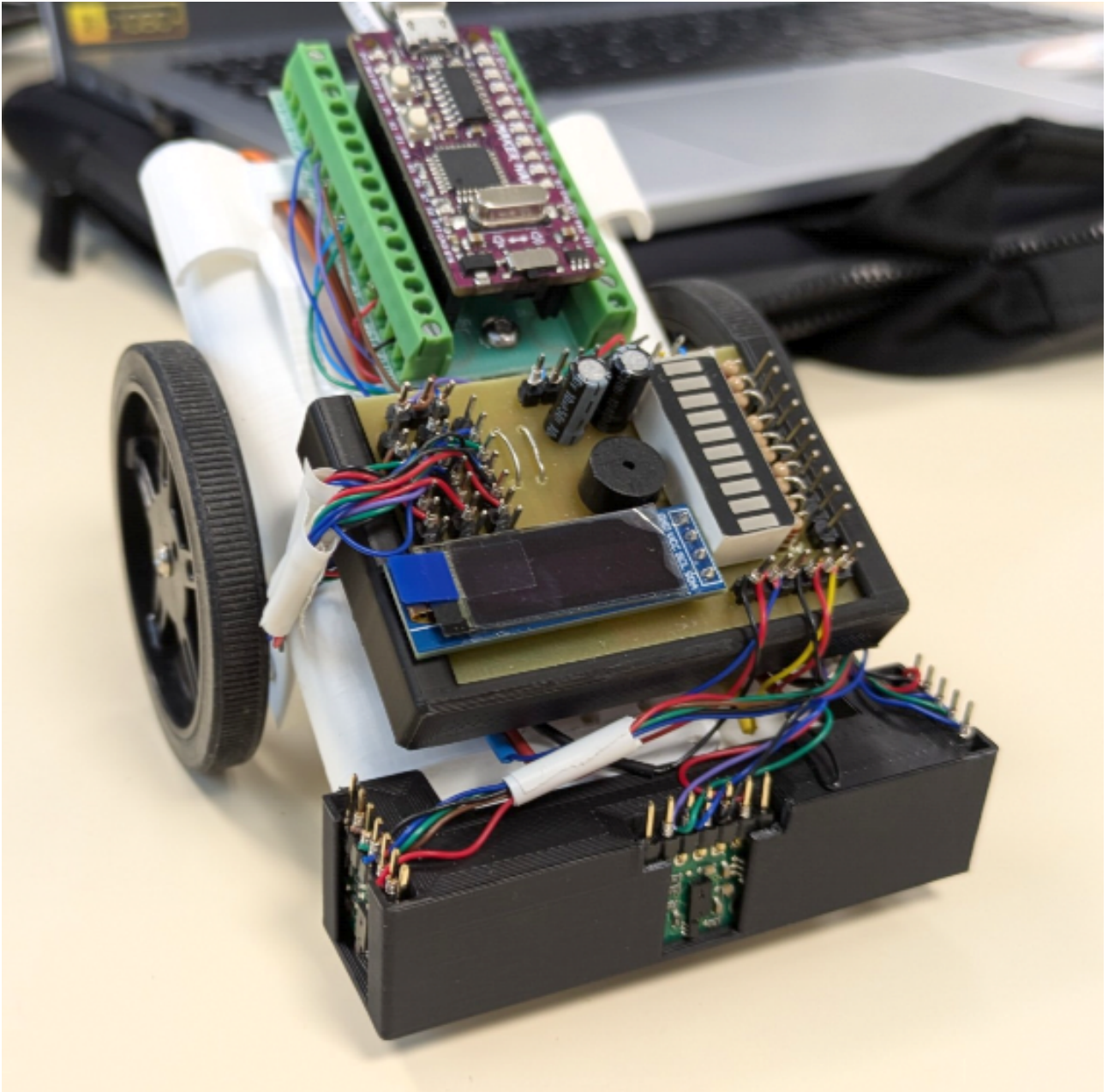


Figure 1: image robot

3. Conception 3D, Assemblage et Câblage

La fiabilité d'un robot autonome ne repose pas uniquement sur son code algorithmique, mais tout autant sur la robustesse de son assemblage mécanique et la qualité de son architecture électrique. Dans un environnement dynamique (mouvements rapides, vibrations, freinages), un simple faux contact peut s'avérer fatal. C'est pourquoi nous avons accordé une attention toute particulière à la conception 3D et à la réalisation d'une carte d'interface électronique sur mesure (Shield).

3.1. Mécanique et Impression 3D : Le support capteurs sur mesure

Le châssis principal du robot, imprimé en PLA (Acide Polylactique), nous a été fourni. S'il remplissait parfaitement son rôle de support structurel pour loger la batterie et fixer les servomoteurs, il ne permettait pas une disposition optimale de nos capteurs de distance.

Dans notre logique de navigation, l'angle de vision du robot est le paramètre physique le plus critique. Nous avons donc décidé de concevoir de toutes pièces un module de fixation dédié :

- **Modélisation CAO** : À l'aide d'un logiciel de Conception Assistée par Ordinateur (CAO), nous avons dessiné un support sur mesure s'emboîtant parfaitement sur le châssis existant.
- **Angles stratégiques** : Cette pièce a été millimétrée pour accueillir les 3 capteurs VL6180X de manière rigoureuse : un capteur visant strictement à 0° (axe frontal), et deux capteurs visant à -90° et $+90^\circ$ (axes latéraux perpendiculaires). Cette orthogonalité parfaite est indispensable pour que nos algorithmes de calcul de distance ne soient pas faussés par des angles de lecture hasardeux.
- **Fabrication** : La pièce a ensuite été exportée au format STL et fabriquée via une imprimante 3D. Ce choix technologique nous a permis un prototypage rapide et une intégration parfaite.

3.2. Câblage et Fiabilité : La conception du Shield d'Interface

Le recours à un câblage volant (les fameux fils "jumpers" sur une *breadboard*) est la première cause de défaillance en robotique mobile (déconnexions liées aux vibrations, faux contacts). Pour pallier ce problème et centraliser nos systèmes, nous avons conçu **une carte électronique d'interface sur mesure (un "Shield")**, au format extrêmement compact (35 x 55 mm), venant s'enficher directement sur notre Arduino Nano.

Ce shield remplit plusieurs rôles fondamentaux pour notre robot :

A. Résolution du conflit d'adresses I2C (Le multiplexage logiciel) C'était l'un des plus grands défis matériels du projet : nos trois capteurs VL6180X communiquent via le bus I2C et possèdent par défaut **la même adresse matérielle**. Les brancher directement en parallèle aurait rendu toute communication impossible (conflit de bus). Pour contourner ce problème matériellement, notre shield a été conçu avec trois connecteurs à 7 broches (J1, J2, J3). Si

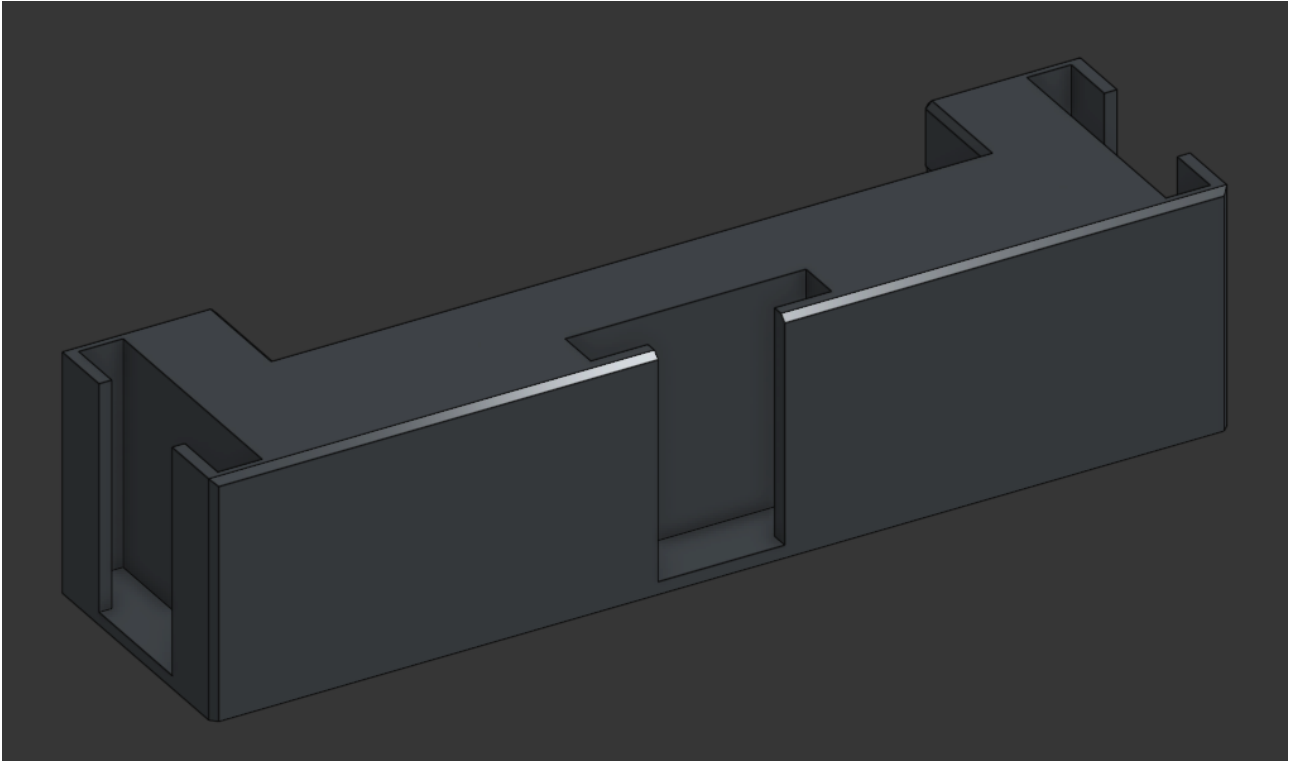


Figure 2: image de la cao

les lignes d'alimentation (VCC/GND) et de communication (SDA/SCL) sont communes, **la broche d'activation (GPIO/CE) de chaque capteur a été routée individuellement vers des broches analogiques de l'Arduino (A0, A1, A2)**. Cela nous permet, au démarrage du robot, de maintenir les capteurs éteints, puis de les allumer séquentiellement un par un pour leur attribuer une nouvelle adresse logicielle unique (0x30, 0x31, 0x32).

B. Intégration de l'Interface Homme-Machine (IHM) Le shield ne se contente pas de gérer les capteurs, il centralise toute l'interface visuelle et sonore du robot :

- **Routage de l'écran OLED** : Connecté sur le même bus I2C que les capteurs pour afficher les données de *debug* en direct.
- **Intégration du Buzzer (BZ1)** : Routé sur la broche D8 (pour éviter les broches dédiées au PWM des moteurs), il est prêt à émettre nos alarmes de collision.
- **Bargraphe à LED** : La carte a également été pensée pour intégrer un afficheur bargraphe à 10 segments avec son réseau de 10 résistances (R1 à R10), offrant la possibilité d'avoir une indication visuelle immédiate de la distance sous forme de jauge matérielle.

C. Filtrage de l'alimentation et Routage (PCB Design) Les moteurs et le buzzer génèrent des pics de consommation (appels de courant) qui peuvent créer des micro-chutes de tension, risquant de faire redémarrer le microcontrôleur ou de fausser les lasers.

- **Filtrage capacitif** : Nous avons intégré au routage de la carte deux condensateurs polarisés (C1, C2) sur la ligne d'alimentation principale. Ils agissent comme des réservoirs d'énergie, lissant la tension et absorbant les parasites.

Pour le développement, nous avons fait le choix de délaisser l’IDE Arduino classique (souvent limité pour les gros projets) au profit de l’environnement **Visual Studio Code (VSCode) couplé à l’extension PlatformIO**. Ce choix nous a offert un cadre de développement professionnel (autocomplétion avancée, gestionnaire de bibliothèques intégré, compilation plus rapide) tout en conservant l’usage du langage C++ et du framework Arduino.

Notre architecture logicielle repose sur des bibliothèques spécifiques (<Wire.h> pour l’I2C, <VL6180X.h> pour les lasers, <Adafruit_SSD1306.h> pour l’OLED et <Servo.h> pour les moteurs) et se divise en plusieurs phases critiques que nous détaillons ci-dessous.

4.1. Phase d’Initialisation et Multiplexage Logiciel (Le `setup`)

Au démarrage du robot, la fonction `setup()` exécute une séquence cruciale pour la stabilité du système. La première action consiste à forcer l’arrêt physique des moteurs en envoyant les valeurs d’arrêt pré-calibrées. Ensuite, le code résout le conflit matériel des adresses I2C évoqué dans la partie matérielle :

1. **Extinction générale** : Les broches d’alimentation logique des trois capteurs (A0, A1, A2) sont forcées à l’état bas (`LOW`). Les capteurs sont donc éteints et invisibles sur le bus I2C.
2. **Allumage séquentiel** : Le programme allume le capteur gauche (`HIGH`), initialise sa configuration par défaut et modifie son adresse logicielle de base pour lui assigner `0x30`.
3. **Répétition** : Il procède de la même manière pour le capteur central (qui devient `0x31`) et le capteur droit (qui devient `0x32`).
4. **Lancement** : L’écran OLED est initialisé et affiche le message “START !” pour nous confirmer visuellement que le multiplexage s’est déroulé sans erreur.

4.2. Le Calibrage Asymétrique des Servomoteurs

L’une des particularités – et difficultés – de la commande de servomoteurs à rotation continue low-cost (FS90R) est leur forte asymétrie de fonctionnement. Mathématiquement, l’angle 90° devrait stopper le moteur, tandis que des valeurs comme 80° et 100° devraient offrir la même vitesse dans des sens opposés. En pratique, la mécanique interne des engrenages fausse ces valeurs.

Pour garantir une trajectoire rectiligne et des virages maîtrisés, nous avons dû réaliser des tests empiriques approfondis pour définir des constantes “en dur” dans le code :

- **Le point mort (Arrêt)** : Le moteur droit s’arrête bien à 90° , mais le moteur gauche nécessite une valeur de 89 (sans quoi il “rampe” lentement).
- **La vitesse de croisière (Ligne droite)** : Pour avancer droit de manière fluide, le moteur gauche reçoit la valeur 91 (légère marche avant), tandis que le droit nécessite 85.
- **La vitesse de correction (Poussée)** : Lors des manœuvres, le moteur gauche est poussé à 93 et le droit à 78. Ces constantes (`vG_croisiere`, `vD_pousse`, etc.) nous permettent d’appeler des comportements fiables dans la boucle principale sans nous soucier des dérives matérielles à chaque itération.

4.3. Acquisition et Filtrage des Données en temps réel

Dans la boucle principale (`loop()`), le robot lit en permanence les distances mesurées par les capteurs Gauche (`dG`), Centre (`dC`) et Droit (`dD`). Cependant, les données brutes peuvent contenir des erreurs (un capteur pointant vers le vide absolu ou renvoyant un *timeout*). Nous avons donc implémenté un **filtre numérique** : Si la distance mesurée dépasse notre constante `DIST_MAX` (200 mm) ou si le capteur ne répond pas dans les temps, le code force la variable à la valeur 255. Cette valeur 255 devient, dans notre logique, l'indicateur universel signifiant : "Il n'y a pas de mur de ce côté, la voie est totalement dégagée".

4.4. L'Algorithme Décisionnel : Le Cerveau Réactif

Une fois les données filtrées, le robot applique une série de conditions (structures conditionnelles `if / else if / else`) pour décider de son comportement. Le robot cherche à se maintenir à une distance cible (`CIBLE = 75 mm`) des murs, avec une marge de tolérance (`TOLERANCE = 25 mm`) pour éviter de faire des zigzags intempestifs.

L'algorithme traite les situations par ordre de priorité :

Priorité 1 : Évitement d'obstacle frontal (Virage à 90°)

- **Condition** : Si le capteur central détecte un mur à moins de 75 mm (`dC <= CIBLE`).
- **Action** : Le buzzer retentit immédiatement (signal à 1000 Hz). Le robot compare l'espace disponible à gauche et à droite. S'il y a plus d'espace à gauche (`dG >= dD`), il stoppe net la roue gauche et donne une impulsion sur la roue droite pour faire pivoter le châssis vers la gauche. Si l'espace est plus grand à droite, il exécute la manœuvre inverse.

Priorité 2 : Navigation en Couloir (Se centrer entre deux murs)

- **Condition** : La voie est libre devant, et le robot détecte un mur de chaque côté (`dG != 255` et `dD != 255`).
- **Action** : Il compare son écartement. S'il est désaxé et se rapproche dangereusement du mur droit (différence supérieure à la tolérance), il stoppe momentanément la roue gauche pour se déporter vers la gauche. Dès qu'il est centré, il reprend sa vitesse de croisière.

Priorité 3 : Suivi de mur simple (Longeur de paroi)

- **Condition** : La voie est libre devant, mais le robot ne détecte qu'un seul mur (par exemple, uniquement à droite : `dD != 255` et `dG == 255`).
- **Action** : C'est le cas typique après la sortie d'un virage. L'algorithme se comporte alors comme un régulateur de suivi de ligne, mais appliqué à une paroi invisible. S'il s'écarte à plus de 100 mm ($75 + 25$) du mur droit, il corrige vers la droite. S'il s'en approche à moins de 50 mm ($75 - 25$), il corrige vers la gauche.

Priorité 4 : Mode Exploration (Aucun mur)

- **Condition** : Tous les capteurs renvoient 255.
- **Action** : Le robot avance en ligne droite à sa vitesse de croisière en attendant de détecter une nouvelle surface.

4.5. Interface de Débogage non-bloquante (L'Écran OLED)

Pour vérifier l'état des capteurs lors des essais sans ralentir le robot, nous mettons à jour l'écran OLED. En robotique, utiliser la fonction `delay()` est proscrit car elle fige littéralement le microcontrôleur : un robot aveugle pendant 100 ms à pleine vitesse percutera un mur. Pour résoudre cela, nous avons utilisé une approche non-bloquante basée sur la fonction `millis()`. L'écran n'est rafraîchi que toutes les 200 millisecondes. Ainsi, la boucle de décision des moteurs tourne à une vitesse fulgurante (plusieurs centaines de fois par seconde), garantissant des réflexes mécaniques quasi immédiats, tout en offrant une interface visuelle parfaitement lisible pour l'opérateur.

5. Résultats, Difficultés et Axes d'amélioration

Concevoir un système embarqué sur le papier est une chose, le confronter aux lois de la physique en conditions réelles en est une autre. Cette phase de validation sur la piste nous a permis de mesurer l'efficacité de nos choix architecturaux et algorithmiques, de relever les limites de notre matériel, et d'envisager des évolutions concrètes.

5.1. Résultats et Analyse des performances

Le bilan global de notre robot explorateur est extrêmement positif. Lors de l'épreuve de validation, **notre robot a réussi à effectuer un tour complet du labyrinthe de manière 100 % autonome, franchissant la ligne d'arrivée en seulement 58 secondes.**

Ce temps chronométré valide directement plusieurs de nos choix techniques :

- **Fiabilité des Capteurs et du Shield** : Le multiplexage I2C conçu sur notre carte d'interface a fonctionné sans aucune défaillance. Le robot n'a souffert d'aucun "gel" (*freeze*) de communication. Les capteurs laser VL6180X ont prouvé leur supériorité face aux capteurs à ultrasons, détectant les angles des murs avec une précision redoutable, sans être perturbés par les rebonds d'ondes.
- **Efficacité de l'algorithme** : La logique de navigation réactive (priorité à l'évitement frontal, puis au centrage en couloir) a permis au robot de ne jamais se retrouver bloqué (dans une boucle infinie ou contre un mur). Les virages à 90° ont été exécutés avec réactivité dès que le seuil critique des 75 mm était franchi.

5.2. Difficultés techniques rencontrées : La limite mécanique

Si le résultat chronométrique est très satisfaisant, l’observation du comportement dynamique du robot sur la piste met en évidence notre principale difficulté : **le contrôle de la trajectoire et l’asymétrie des servomoteurs FS90R.**

Notre algorithme ordonne au robot d’aller “tout droit”. Or, comme évoqué dans la partie programmation, envoyer la même consigne de vitesse théorique aux deux moteurs ne produit jamais la même vitesse de rotation réelle (*RPM*). Cette asymétrie mécanique engendre un phénomène de **dérive cinématique** : le robot “tire” naturellement vers la droite ou la gauche.

Conséquences sur le comportement : Puisque le robot dérive, il se rapproche inévitablement d’un mur. Notre algorithme s’en rend compte (grâce à la tolérance de 25 mm) et envoie une impulsion de correction pour le recentrer. Le robot a donc effectué son parcours de 58 secondes en “zigzagait” légèrement dans les couloirs droits, avançant par à-coups (principe de la régulation dite “Bang-Bang” ou “Tout ou Rien”). Le temps perdu lors de ces micro-corrections nous a empêchés de réaliser un temps encore plus court. Nous avons dû passer de très nombreuses heures à affiner empiriquement les valeurs PWM “en dur” dans le code pour minimiser ce phénomène, une tâche fastidieuse et peu pérenne puisque la tension de la batterie influe sur la vitesse des moteurs à mesure qu’elle se décharge.

5.3. Axes d’amélioration et perspectives

Le recul sur cette expérience nous permet d’identifier précisément les modifications qui transformeraient ce prototype fonctionnel en un système de course hautement performant. Si nous disposions d’un mois supplémentaire et d’un budget illimité, notre travail se concentrerait sur deux axes majeurs :

A. Amélioration Matérielle (*Hardware*) : Moteurs asservis Le talon d’Achille de notre robot est sa propulsion en boucle ouverte (le microcontrôleur donne un ordre au moteur mais ne sait pas à quelle vitesse il tourne réellement). Nous remplacerions les servomoteurs à rotation continue par des **moteurs à courant continu (DC) équipés d’encodeurs rotatifs en quadrature**, pilotés par un pont en H (ex: L298N ou TB6612FNG). Les encodeurs permettraient à l’Arduino de compter précisément le nombre de tours de chaque roue (odométrie). Ainsi, le microcontrôleur pourrait ajuster électriquement la puissance de la roue la plus lente pour garantir mathématiquement une ligne droite parfaite.

B. Amélioration Logicielle (*Software*) : Le Régulateur PID Actuellement, notre robot fonctionne par seuils stricts (If/Else). Pour exploiter des moteurs de meilleure qualité, nous abandonnerions cette logique conditionnelle au profit d’une approche d’automatique classique : **le Régulateur PID (Proportionnel - Intégral - Dérivé).**

- **Proportionnel** : Plus le robot est proche du mur, plus il corrige fort. S’il est très légèrement désaxé, il corrige en douceur.
- **Intégral** : Permet de supprimer l’erreur statique (la fameuse dérive asymétrique des moteurs) en la compensant dans le temps.
- **Dérivé** : Anticipe les variations brutales pour empêcher le robot d’osciller ou de “zigzaguer” entre les deux murs. Avec une telle régulation, le mouvement du robot ne serait

plus saccadé mais parfaitement lissé et fluide, lui permettant d'augmenter drastiquement sa vitesse de pointe dans les lignes droites.

6. Conclusion Générale

Ce projet de Robotique Mobile s'est révélé être une expérience fondatrice dans notre parcours d'ingénieurs. Il ne s'agissait pas seulement de compiler des lignes de code ou de visser des composants, mais bien de gérer l'intégration complexe d'un système mêlant mécanique, électronique et informatique.

Le cahier des charges a été entièrement respecté et validé par un temps de parcours de 58 secondes. Au-delà de cette réussite finale, ce sont les étapes intermédiaires qui constituent notre plus grande victoire : la modélisation et l'impression 3D d'un support stratégique pour optimiser la vision du robot, la conception minutieuse d'un Shield électronique pour éradiquer les problèmes de faux contacts et maîtriser le multiplexage I2C, ainsi que l'utilisation d'un environnement de développement professionnel (VSCode/PlatformIO) couplé à une programmation non-bloquante.

Ce projet a mis en lumière la réalité du terrain : une excellente logique logicielle restera toujours limitée par les contraintes et les imperfections du monde physique (les moteurs dans notre cas). Comprendre comment le code doit compenser la mécanique est l'essence même de l'ingénierie des systèmes embarqués. Forts de cette expérience, nous sommes aujourd'hui outillés pour aborder des systèmes robotiques plus complexes impliquant de l'odométrie et des asservissements dynamiques.

ANNEXES :

```
#include <Wire.h>
#include <VL6180X.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <Servo.h>

// =====
// VITESSES CALIBRÉES POUR CE ROBOT
// =====
// 1. POINT D'ARRÊT (CORRIGÉ)
int STOP_G = 90; // Arrêt total de la roue gauche
int STOP_D = 90; // Arrêt total de la roue droite

// 2. VITESSE DE CROISIÈRE (Droit ralenti)
int vG_croisiere = 95;
int vD_croisiere = 85;
```

```

// 3. VITESSE DE VIRAGE / CORRECTION
int vG_pousse = 93;
int vD_pousse = 87;
// =====

// Configuration Écran
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define OLED_ADDR 0x3C
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);

// Capteurs et Moteurs
VL6180X sensorG, sensorC, sensorD;
Servo servoGauche, servoDroit;

// Broches
const int pinG = 9;
const int pinD = 10;
const int pinBuzzer = 8;
const int PIN_S_G = A0;
const int PIN_S_C = A2;
const int PIN_S_D = A1;

// Paramètres de navigation
const int CIBLE = 75;
const int TOLERANCE = 25;
const int DIST_MAX = 200;

unsigned long chronoEcran = 0;

void setup() {
  Wire.begin();
  Wire.setClock(400000);

  servoGauche.attach(pinG);
  servoDroit.attach(pinD);
  pinMode(pinBuzzer, OUTPUT);

  // Maintien à l'arrêt au démarrage
  servoGauche.write(STOP_G);
  servoDroit.write(STOP_D);

  display.begin(SSD1306_SWITCHCAPVCC, OLED_ADDR);

  // Initialisation matérielle des capteurs
  pinMode(PIN_S_G, OUTPUT); pinMode(PIN_S_C, OUTPUT); pinMode(PIN_S_D, OUTPUT);
  digitalWrite(PIN_S_G, LOW); digitalWrite(PIN_S_C, LOW); digitalWrite(PIN_S_D, LOW);
  delay(100);

  // Allumage et configuration capteur Gauche

```

```

digitalWrite(PIN_S_G, HIGH); delay(50);
sensorG.init(); sensorG.configureDefault(); sensorG.setAddress(0x30);
sensorG.setTimeout(200);

// Allumage et configuration capteur Centre
digitalWrite(PIN_S_C, HIGH); delay(50);
sensorC.init(); sensorC.configureDefault(); sensorC.setAddress(0x31);
sensorC.setTimeout(200);

// Allumage et configuration capteur Droit
digitalWrite(PIN_S_D, HIGH); delay(50);
sensorD.init(); sensorD.configureDefault(); sensorD.setAddress(0x32);
sensorD.setTimeout(200);

// Message de démarrage
display.clearDisplay();
display.setTextSize(2);
display.setTextColor(WHITE);
display.setCursor(10,20);
display.println("START !");
display.display();
delay(1500);
}

void loop() {
  // Lecture des capteurs
  uint8_t dG = sensorG.readRangeSingleMillimeters();
  uint8_t dC = sensorC.readRangeSingleMillimeters();
  uint8_t dD = sensorD.readRangeSingleMillimeters();

  // Filtre > 200mm ou Erreur = on considère que c'est vide (255)
  if (dG > DIST_MAX || sensorG.timeoutOccurred()) dG = 255;
  if (dC > DIST_MAX || sensorC.timeoutOccurred()) dC = 255;
  if (dD > DIST_MAX || sensorD.timeoutOccurred()) dD = 255;

  // =====
  // LOGIQUE 1 : MUR EN FACE (ON TOURNE)
  // =====
  if (dC <= CIBLE) {
    tone(pinBuzzer, 1000);
    if (dG >= dD) {
      // Si plus de place à gauche : On bloque Gauche, on pousse Droite
      servoGauche.write(STOP_G);
      servoDroit.write(vD_pousse);
    } else {
      // Si plus de place à droite : On bloque Droite, on pousse Gauche
      servoGauche.write(vG_pousse);
      servoDroit.write(STOP_D);
    }
  }
}

```

```

// =====
// LOGIQUE 2 : VOIE LIBRE (ON SE CENTRE / ON SUIT LE MUR)
// =====
else {
    noTone(pinBuzzer);

    // CAS A : Il y a un mur de chaque côté (On se centre)
    if (dG != 255 && dD != 255) {
        if (dD < (dG - TOLERANCE)) {
            // Trop près de droite -> On s'écarte à GAUCHE (Bloque G, pousse D)
            servoGauche.write(STOP_G); servoDroit.write(vD_pousse);
        }
        else if (dG < (dD - TOLERANCE)) {
            // Trop près de gauche -> On s'écarte à DROITE (Pousse G, bloque D)
            servoGauche.write(vG_pousse); servoDroit.write(STOP_D);
        }
        else {
            // Bien centré -> Tout droit (vitesse ralentie)
            servoGauche.write(vG_croisiere); servoDroit.write(vD_croisiere);
        }
    }

    // CAS B : Il n'y a qu'un mur à DROITE (On longe à 75mm)
    else if (dD != 255 && dG == 255) {
        if (dD < (CIBLE - TOLERANCE)) {
            // Trop près -> On s'écarte à GAUCHE
            servoGauche.write(STOP_G); servoDroit.write(vD_pousse);
        }
        else if (dD > (CIBLE + TOLERANCE)) {
            // Trop loin -> On se rapproche à DROITE
            servoGauche.write(vG_pousse); servoDroit.write(STOP_D);
        }
        else {
            servoGauche.write(vG_croisiere); servoDroit.write(vD_croisiere);
        }
    }

    // CAS C : Il n'y a qu'un mur à GAUCHE (On longe à 75mm)
    else if (dG != 255 && dD == 255) {
        if (dG < (CIBLE - TOLERANCE)) {
            // Trop près -> On s'écarte à DROITE
            servoGauche.write(vG_pousse); servoDroit.write(STOP_D);
        }
        else if (dG > (CIBLE + TOLERANCE)) {
            // Trop loin -> On se rapproche à GAUCHE
            servoGauche.write(STOP_G); servoDroit.write(vD_pousse);
        }
        else {
            servoGauche.write(vG_croisiere); servoDroit.write(vD_croisiere);
        }
    }
}

```

```

    }
}

// CAS D : Aucun mur détecté
else {
    // Tout droit
    servoGauche.write(vG_croisiere); servoDroit.write(vD_croisiere);
}
}

// =====
// 3. AFFICHAGE OLED
// =====
if (millis() - chronoEcran > 200) {
    chronoEcran = millis();
    display.clearDisplay();
    display.setTextSize(2);

    // Affichage avec "---" si le capteur ne voit rien (valeur 255)
    display.setCursor(0,0);
    display.print("G: "); if(dG == 255) display.print("---"); else display.print(dG);

    display.setCursor(0,24);
    display.print("C: "); if(dC == 255) display.print("---"); else display.print(dC);

    display.setCursor(0,48);
    display.print("D: "); if(dD == 255) display.print("---"); else display.print(dD);

    display.display();
}
}

```