ENSTA, campus de Brest
2, rue François Verny
29806 BREST cedex
FRANCE
Tel : +33 (0)2 98 34 88 00
https://www.ensta.fr/

Rapport de stage

MASTER ROB 2026

Mars – Septembre 2026

Cartographie de la maison à l'aide d'un robot et positionnement d'objets

Brunelle JOSSE

brunelle.josse@ensta.fr

Tuteur entreprise : Loïc AVENEL

Tuteur école : Luc Jaulin

1 Remerciements

Avant de présenter ce travail, je tiens à exprimer ma sincère reconnaissance à toutes les personnes qui ont contribué, de près ou de loin, à la réalisation de ce projet de fin d'études.

Je remercie tout d'abord mon encadrant académique, Luc Jaulin, pour sa disponibilité, et son accompagnement tout au long de ce projet et de cette année scolaire.

Mes remerciements s'adressent également à Loïc Avenel, mon maître de stage au sein d'Orange Innovation, pour m'avoir accueilli et intégré au sein de l'équipe, pour la confiance et la liberté qu'il m'a accordées ainsi que pour ses précieux conseils tout au long de cette expérience.

Je souhaite aussi remercier l'ensemble de l'équipe de NUX pour leur accueil chaleureux, leur disponibilité et leur soutien technique, qui ont grandement facilité mon intégration et l'avancement de mon travail. Et tout particulièrement à Mariano Belaunde, pour tous ses conseils et réponses à mes nombreuses questions.

Je tiens également à exprimer ma gratitude envers l'ensemble du corps enseignant de l'ENSTA Campus de Brest, pour la qualité de la formation dispensée durant ces années d'études, qui m'a permis d'acquérir les compétences nécessaires à la réalisation de ce projet.

Enfin, je remercie ma famille et mes proches pour leur soutien indéfectible et leurs encouragements tout au long de ce parcours.

À toutes ces personnes, je présente mes remerciements les plus sincères.

Table des matières

2 Introduction

2.1 Présentation de l'entreprise

Orange est l'un des principaux opérateurs de télécommunications au monde, présent dans une trentaine de pays et comptant plusieurs centaines de millions de clients. Le groupe propose des services de téléphonie mobile et fixe, d'accès à Internet, ainsi que des solutions numériques destinées aux particuliers et aux entreprises.

Orange est l'un des principaux opérateurs de télécommunications au monde, présent dans une trentaine de pays et comptant plusieurs centaines de millions de clients. Le groupe propose des services de téléphonie mobile et fixe, d'accès à Internet, ainsi que des solutions numériques destinées aux particuliers et aux entreprises (cloud, cybersécurité, objets connectés, etc.).
Au-delà de son activité d'opérateur, Orange investit massivement dans la recherche et l'innovation à travers son entité Orange Innovation, qui regroupe les activités de RD du groupe. L'innovation chez Orange représente 612 millions d'euros investis, 8400 collaborateurs et plus de 11 000 brevets déposés mondialement. Cette entité a pour mission d'anticiper les usages de demain et de développer les technologies qui façonneront les futurs services du groupe.
Le site de Lannion, en Bretagne, est historiquement l'un des principaux centres de RD du groupe : implanté dans cette ville depuis 1962, Orange y a développé une expertise dans des domaines aussi variés que la cybersécurité, la 5G ou encore le big data.
Ce stage se déroule au sein de l'équipe NUX (New User eXperience), chargée de développer les solutions du futur dans l'amélioration de l'expérience utilisateur.

2.2 Présentation du sujet

Le sujet de ce projet de fin d'études porte sur la "cartographie de la maison à l'aide d'un robot et positionnement d'objets".
Ce travail s'appuie sur des robots aspirateurs connectés Roborock S6 Pure, équipés d'un système de cartographie par LiDAR.
Ces robots reposent nativement sur un cloud propriétaire chinois, limitant fortement les possibilités d'accès aux données et de personnalisation. Pour s'affranchir de cette contrainte, ils ont été déverrouillés à l'aide de Valetudo [Val26], une surcouche logicielle open source qui permet de piloter le robot entièrement en local, sans dépendance au cloud constructeur. Cette solution offre ainsi la possibilité d'implémenter et de tester nos propres développements directement sur les robots.

FIGURE 1 – Roborock S6 Pure blanc et sa base.

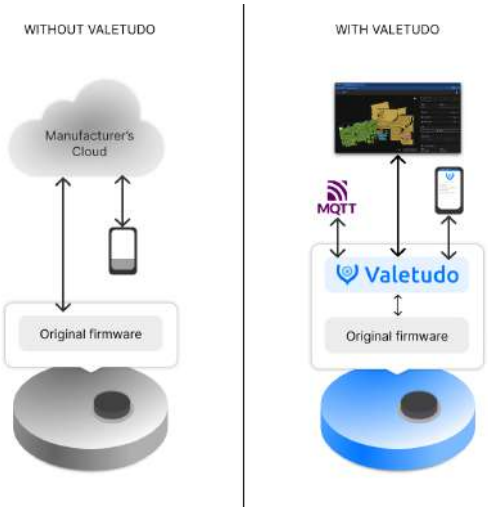

FIGURE 2 – Schéma du fonctionnement de Valetudo.

La première partie de ce stage consiste à reprendre les travaux déjà engagés sur ce sujet et à améliorer l'existant, dans l'objectif d'obtenir une solution de cartographie rapide, alliant précision, répétabilité et reproductibilité.

La seconde partie de ce stage s'est orientée vers une problématique différente de la deuxième partie initiale du sujet : il ne s'agira plus de positionnement d'objets, mais d'apprentissage de modèle appliqué à la robotique, sur le bras robotique SO-101 (plateforme open-source du projet LeRobot). L'objectif confié par l'entreprise était de faire réaliser à ce bras une tâche complexe. Cette mission présentait une difficulté particulière : aucun dataset ni modèle pré-existant n'était disponible pour la tâche que nous avons choisie, ce qui impliquait de concevoir l'ensemble du processus, de la collecte des données d'entraînement jusqu'à l'apprentissage et l'évaluation du modèle, sans pouvoir s'appuyer sur des travaux similaires déjà réalisés.

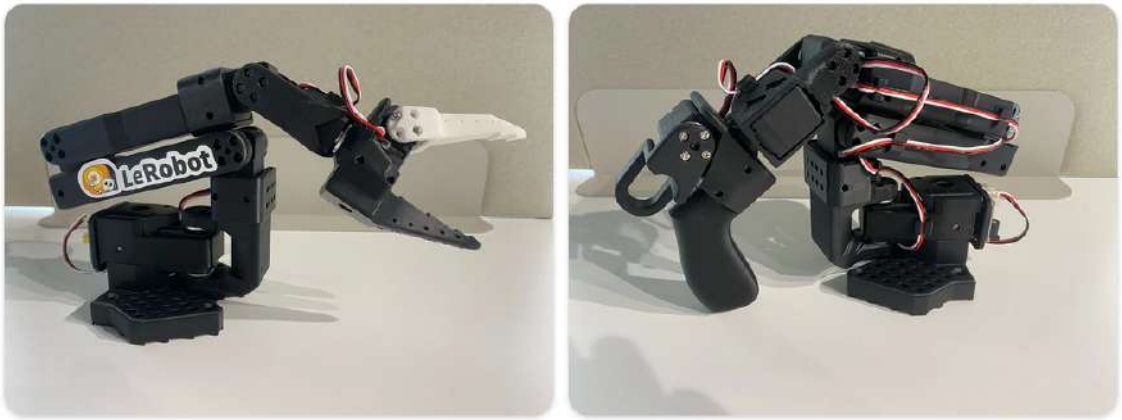

FIGURE 3 – Bras LeRobot SO-101, follower à gauche, leader à droite

2.3 État de l'existant

Lors de mon arrivée chez Orange, j'ai dans un premier temps pris le temps de lire et de comprendre les codes existants, ainsi que leur utilisation et l'environnement de travail dans lequel s'inscrit le projet. Les robots aspirateurs sont actuellement équipés de plusieurs briques logicielles développées par Orange : un système de détection rapide, un dispositif de captation du signal et du débit Wi-Fi, ainsi qu'une intelligence artificielle de conseil destinée à l'utilisateur. Ces robots sont déployés à titre expérimental chez des employés d'Orange, afin de réaliser des tests de cartographie, mais surtout d'évaluer l'IA de conseil, qui a pour rôle d'indiquer à l'utilisateur la qualité de sa couverture Wi-Fi et de lui proposer des pistes d'amélioration.
Une interface a également été développée (Figure 22, Figure 23) pour permettre à l'utilisateur d'interagir avec le système : elle donne accès à la création de carte en temps réel, aux résultats de l'IA d'analyse, à un historique des cartes générées, incluant les cartes de signal et de débit, ainsi qu'à un mode de contrôle manuel du robot. Cette interface est exécutée sur une Raspberry Pi 4, fixée directement sur le robot, qui assure également la réalisation des mesures Wi-Fi.

FIGURE 4 – Interface d'accueil

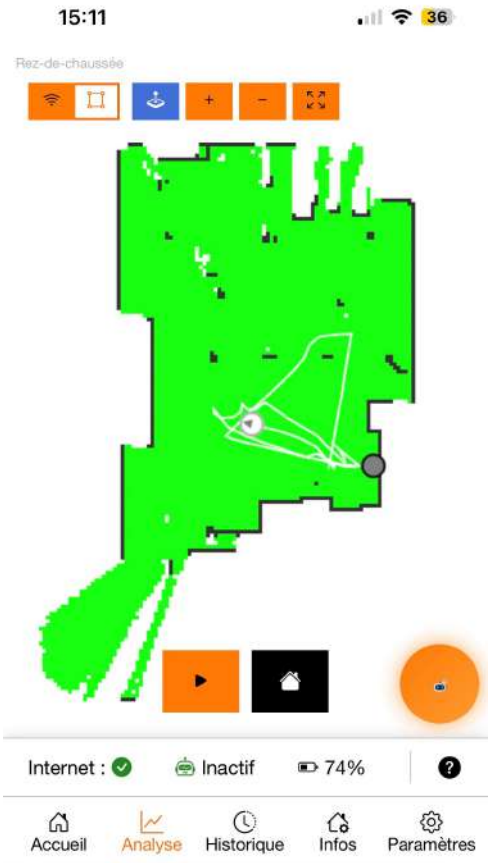

FIGURE 5 – Interface d'analyse

Comme expliqué plus tôt, la communication entre le robot aspirateur, la Raspberry Pi et l'utilisateur se fait en réseau local grâce à Valetudo. Le schéma de fonctionnement est le suivant :

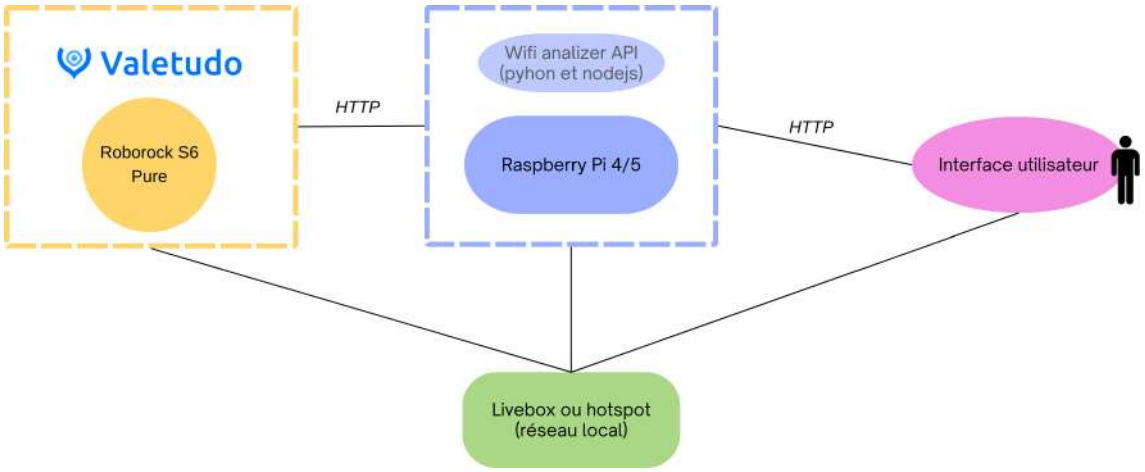

FIGURE 6 – Schéma de communication du robot d'exploration rapide.

Les codes existants n'ayant pas été documentés, j'ai entrepris de réaliser différents diagrammes et logigrammes afin de les formaliser et de les transmettre au reste de l'équipe (voir annexe A).
Deux approches principales coexistaient : celle fondée sur une détection de pièces, et celle fondée sur une détection de frontières. Ces deux approches seront détaillées plus loin dans ce rapport.
Après avoir lu, décrit et compris ces différents codes, le travail a consisté à reprendre celui de Mr.Christian Aubry, afin de l'améliorer, que ce soit par un ajustement des paramètres (tuning) ou par des modifications

plus profondes du code.

2.4 État de l'art : découverte rapide d'environnements par LiDAR

La découverte autonome d'un environnement inconnu consiste à construire progressivement une représentation de l'espace tout en choisissant les déplacements permettant d'acquérir rapidement de nouvelles informations. Pour un véhicule autonome terrestre équipé d'un LiDAR, cette problématique couple ainsi la perception et la cartographie, généralement au travers d'un système de SLAM (Simultaneous Localization and Mapping), avec une stratégie de planification visant à maximiser la couverture de l'environnement tout en limitant la distance parcourue. Les approches d'exploration peuvent être regroupées en trois grandes familles : les méthodes fondées sur les *frontiers*, les méthodes par échantillonnage de points de vue (*Next-Best-View*) et les approches hybrides combinant ces deux principes [Hua+23]. Une quatrième voie, plus récente, consiste à apprendre directement une politique d'exploration par apprentissage profond, mais son coût d'entraînement et sa généralisation à de nouveaux environnements limitent encore son intérêt pour une plateforme terrestre contrainte en calcul.

Les méthodes *frontier-based*, introduites notamment par Yamauchi [Yam98], constituent une approche classique de l'exploration autonome. Elles reposent sur l'identification des frontières séparant les régions connues des régions encore inconnues, puis sur la sélection d'une frontière comme prochaine zone à explorer. Une stratégie particulièrement simple consiste à sélectionner la frontière la plus proche du robot. Cette approche présente l'avantage d'être peu coûteuse en calcul elle est ainsi particulièrement adaptée aux systèmes embarqués devant réagir rapidement aux nouvelles observations du LiDAR.

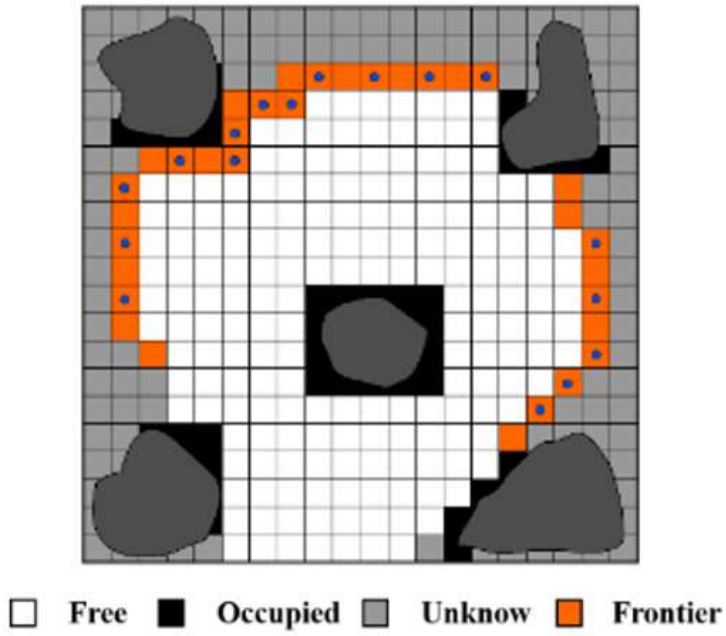

FIGURE 7 – Schéma explicatif d'une frontière proposé par [Zha+25].

La principale limite de cette stratégie réside dans son caractère local : la distance à la frontière constitue un critère simple, mais ne renseigne pas directement sur la quantité d'information que cette frontière permettra d'obtenir ni sur les possibilités d'exploration qu'elle peut ouvrir. Dans certaines configurations, une frontière plus éloignée peut ainsi présenter un intérêt supérieur en termes de gain d'information ou de progression dans l'environnement. Cette question a motivé le développement de méthodes intégrant davantage d'informations dans la sélection des objectifs d'exploration.

Les méthodes par échantillonnage, notamment les approches NBVP (*Next-Best-View Planning*), évaluent plusieurs points de vue candidats en fonction de leur gain d'information et de leur accessibilité. Elles permettent ainsi de prendre en compte des critères dépassant la seule proximité, au prix d'une complexité algorithmique plus importante. Des approches hybrides, telles que FAEL (Fast Autonomous Exploration for Large-scale Environments), combinent également la détection des frontières avec la géné-

ration de points de vue et l'optimisation de trajectoire. FAEL montre notamment l'intérêt d'une sélection incrémentale des frontières et des points de vue afin de conserver une fréquence de planification élevée dans les environnements de grande taille.

Dans ce contexte, la sélection de la frontière la plus proche reste un compromis intéressant entre simplicité, réactivité et coût computationnel. L'enjeu n'est donc pas nécessairement de remplacer cette stratégie par une méthode plus complexe, mais d'étudier dans quelle mesure le critère de sélection peut être enrichi tout en préservant les avantages d'une approche *frontier-based*. Cette problématique constitue le point de départ du présent travail.

2.5 Travaux initiaux

Tout d'abord, il a fallu comprendre les documents existants ; le sujet de mon stage ayant déjà été travaillé par plusieurs personnes, je bénéficiais donc d'un environnement de travail complet. J'ai ainsi commencé par prendre en main le robot et ses codes, qui comportaient deux versions de l'exploration, celle d'Antoine Sauvenay et celle de Christian Aubry. Par la suite, on utilisera seulement la version de Mr.Aubry, mais il m'a été demandé de créer de la documentation sur les deux versions. Cette documentation était nécessaire, d'une part, parce qu'elle n'avait pas été réalisée auparavant et qu'elle permettrait à toute personne souhaitant utiliser les codes de les comprendre sans avoir à lire l'intégralité des fichiers Python. D'autre part, elle permettrait de mieux comprendre les différentes approches qui avaient été utilisées pour traiter le problème de la cartographie rapide.

La principale différence entre les deux codes était la stratégie d'approche de la cartographie. Les codes de Mr.Sauvenay sont basés sur une approche d'action en fonction de l'endroit où le robot se trouve (pièce, couloir) tandis que les codes de Mr.Aubry sont basés sur une approche de frontière.
Comme vu dans l'état de l'art, l'approche par frontières est quelque chose de largement utilisé pour la cartographie, c'est ainsi la version de code qui fonctionne actuellement le mieux sur le robot.

Pour résumer, les deux différentes approches sont les suivantes :
- **Antoine** : cette approche repose sur la reconnaissance des murs afin de déterminer la situation du robot (pièce ou couloir) et d'adapter son comportement en conséquence. Dans une pièce, le robot quadrille l'espace et se déplace vers chaque zone, en privilégiant les cases dont l'état évolue le moins, jusqu'à ce qu'un critère de satisfaction soit atteint, après quoi il quitte la pièce. Dans un couloir, le robot cherche à identifier une entrée vers une nouvelle pièce.
- **Christian** : cette approche repose sur la détection de *frontières*, c'est-à-dire les zones de transition entre un espace connu libre et un espace inconnu. Ces frontières sont regroupées en clusters, que le robot rejoint successivement afin d'étendre progressivement la carte.

Les prochaines sections expliquent plus en détail les deux stratégies.

Dans la suite de ce document, l'ensemble des unités des axes des cartes générées par le robot sont des coordonnées définies par le système du Roborock S6 Pure.

2.5.1 Codes d'Antoine Sauvenay

Les codes d'Antoine s'organisent autour de la détection de la situation du robot. Lors de sa progression, le robot trouve tous les murs validés et potentiels autour de lui, puis en fonction du nombre de murs qu'il voit il détermine sa prochaine action. Quatre choix sont alors possibles :
- Aucun mur n'a été détecté ou seulement une partie d'un mur : dans ce cas le robot va dans une direction ouverte pour essayer de découvrir plus de surface et de murs ;
- Deux murs ont été détectés et validés comme des murs de couloir : le robot va se déplacer dans la direction de celui-ci et cherche à trouver la sortie/entrée vers une pièce la plus proche.
- Le robot a détecté assez de murs ayant les bonnes caractéristiques pour s'assurer qu'il est dans une pièce ; le robot créé alors une grille sur la surface de la pièce, puis à chaque case de cette grille est assigné un critère de poids qui correspond à : sa distance au robot, sa surface inexplorée, la présence du robot dans cette case et l'évolution de la couverture du robot. Le robot est ensuite envoyé au point le plus intéressant dans la pièce suivant l'évolution des poids jusqu'à ce qu'aucun point ne soit plus intéressant. Le robot est ensuite guidé hors de la pièce.

Une fois que le robot a détecté toutes les pièces et les a visitées, il considère l'exploration finie.
Cette méthode basée sur une approche intelligente de type de région et quadrillage de zone ne s'est pas

montrée aussi performante que celle de Mr.Aubry, et pouvait parfois tourner en boucle entre des point très proches dans une pièce sans jamais en sortir.

2.5.2 Codes de Christian Aubry

Lors de son exécution, le code de Mr.Aubry, qu'on appellera code d'origine, fonctionne de la sorte : Sur l'interface utilisateur, une carte apparaît avec un cercle qui représente le robot, une surface verte qui représente la surface libre vue par le robot et des pixels noirs qui représentent des obstacles (objets, personnes, murs...). Ensuite, des broches de localisation apparaissent et un trajet en pointillé partant du robot jusqu'à cette broche est tracé, c'est le point objectif du robot et le trajet calculé pour s'y rendre. Ce trajet peut changer au fur et à mesure que le robot découvre son environnement.
Du point de vue de ce qui se passe sur la pi lors du lancement du code, une carte apparaît aussi, avec un cercle bleu qui représente le robot, des pixels blancs pour la surface libre vue par le robot, des pixels noirs pour les obstacles et des pixels gris pour la surface qui est encore inconnue.
A chaque fois que le robot s'arrête, il calcule les frontières, ce sont les limites entre un pixel connu libre (blanc) et un pixel inconnu (gris). Ces frontières sont ensuite regroupées en clusters en fonction de leur éloignement puis le centroïde de chaque cluster est sauvegardé comme étant un point d'intérêt. Ces points d'intérêt sont ensuite donnés au robot sous forme d'ordre "go to target" et celui-ci s'y rend, puis recalcule les frontières jusqu'à ce qu'elles aient toutes été vues.

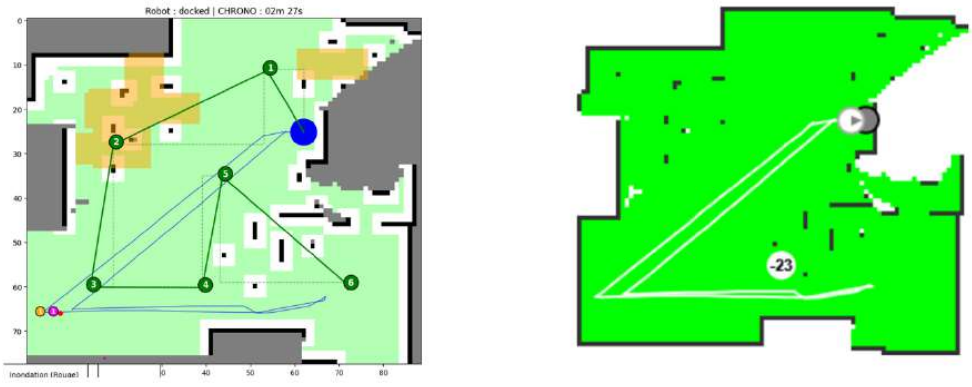

FIGURE 8 – A gauche la carte observée sur la pi, à droite la carte observée sur l'API.

A la fin du script, le robot rentre à sa base et crée une carte pour la prochaine étape, étape de la prise de mesures Wi-Fi. Cette carte contient les informations suivantes : zone de danger, zone inaccessible, zone libre sûre et points de mesure en plus de la cartographie elle-même.

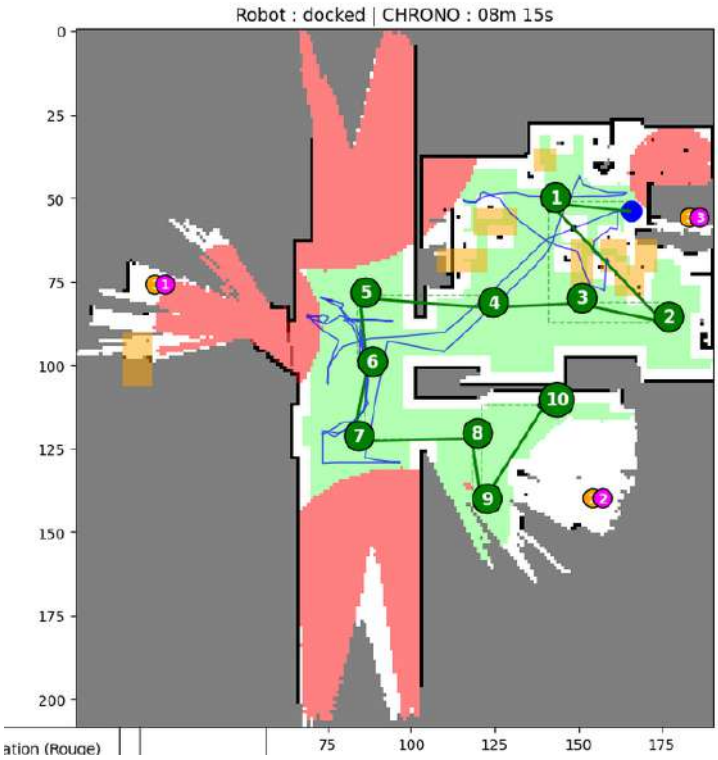

FIGURE 9 – Carte générée à la fin de l'exploration. En vert la zone libre et sûre, en orange les zones à risque, en rouge les zones inaccessibles, les chiffres vert foncé sont les points de mesure Wi-Fi et leur numéro indique l'ordre dans lequel le robot s'y rendra (ici de 1 à 10).

Le robot part ensuite faire les mesures Wi-Fi aux points indiqués.

2.6 Méthodologie

Afin de répondre au mieux aux objectifs du sujet on procède par étapes.
Premièrement, observer le comportement du robot avec le code actuel et noter les problèmes vus ou rapportés par des utilisateurs. Par la suite, une liste des paramètres pouvant être ajustés a été établie, en tenant compte des options accessibles via l'interface Valetudo, tout en reconnaissant que certains paramètres restent inaccessibles. Enfin, améliorer ce qui est possible de l'être avec ces informations. Cela peut paraître simple, mais la tâche n'a pas été facile, de nombreux changements ont été apportés puis supprimés, des paramètres ont été modifiés puis remis à leur valeur de base, plusieurs solutions ont été envisagées pour résoudre les problèmes identifiés, mais leur mise en œuvre a été limitée par des accès restreints aux fonctionnalités du robot, ce qui a complexifié le processus d'amélioration.

3 Problèmes et solutions

La partie à suivre traite des changements effectués dans le code qui ont fonctionné et qui sont désormais implémentés sur le robot.

3.1 Démarrage du robot

La première action du robot lorsqu'il est lancé est la sortie de sa base.
Avec le code d'origine, le robot démarre de la sorte : il lance ses moteurs d'aspiration, une voix indique qu'il lance le nettoyage, le robot recule d'environ 20 cm, tourne de 90° dans le sens direct, commence à tourner dans l'autre sens puis s'arrête avant de calculer les premiers points d'intérêt. Ce démarrage entraîne souvent un « retournement » de la carte créée. En effet, l'orientation de la carte affichée sur la Raspberry Pi peut varier d'un lancement à l'autre, ce qui paraît surprenant puisque le robot est systématiquement lancé depuis le même point. Malgré cette variation d'orientation, les points proposés

par la suite semblent globalement se situer dans les mêmes zones d'un démarrage à l'autre, bien que leur correspondance reste approximative.

La plainte la plus récurrente des utilisateurs étant le bruit, on tente d'abord de supprimer l'aspiration au démarrage, malheureusement même en forçant les moteurs en 'off' le robot aspire et aucune solution n'a été trouvée. Cela fait partie des paramètres inaccessibles via Valetudo. Néanmoins le robot fait moins de bruit qu'avant car il n'avait pas de contrainte d'intensité et restait sur les paramètre du dernier usager et est maintenant obligé d'être en minimum à chaque début d'exploration.
Ci-dessous des captures d'écran des cartes créées après le cycle de démarrage, à l'arrêt, juste avant le calcul des premiers points d'intérêt.

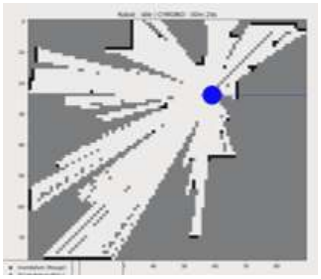

FIGURE 10 – Démarrage 1

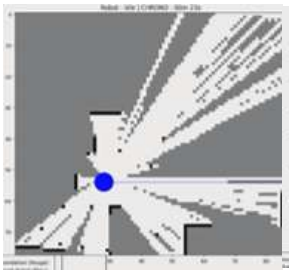

FIGURE 11 – Démarrage 2

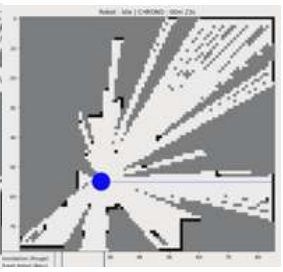

FIGURE 12 – Démarrage 3

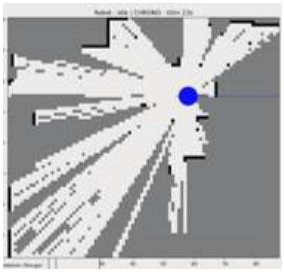

FIGURE 13 – Démarrage 4

FIGURE 14 – Quatre démarrages dans des conditions identiques avec le code d'origine

On peut voir sur ces quatre démarrages que deux sont dans un sens et deux dans l'autre, on peut aussi voir que la surface blanche présente des rayons gris même sur des espaces qui sont connus comme libres (réellement aucun obstacles). On peut se demander si c'est dû à la portée du LiDAR, car si les rayons ne reviennent pas au robot, la surface dans cette direction sera déclarée comme inconnue. Seulement les rais ne sont pas au même endroit à chaque fois et la distance parcourue par les rayons du LiDAR est inférieure à la portée maximale, on cherche donc à comprendre. L'arrêt brutal du robot lors de sa rotation nous indique tout de même que le robot n'a peut-être pas le temps de percevoir tout ce qu'il y a autour de lui. Afin de découvrir plus de surface au démarrage, et dans l'optique de partir directement au bon endroit, et non vers un point considéré d'intérêt mais qui pourrait être vu au début et éviterait de perdre du temps à s'y rendre, on implémente un nouveau départ qui consiste à faire reculer le robot d'environ 20 cm puis tourner d'un peu plus que 360° sur lui même pour ensuite s'arrêter et calculer les premiers points. Ce nouveau démarrage a permis de supprimer les retournements de cartes, celles-ci sont désormais pratiquement tout le temps dans le même sens dans des conditions identiques, mais aussi de rendre les premiers points reproductibles quasiment à chaque lancement. En effet la surface explorée étant plus grande et toujours la même car le robot a le temps d'envoyer ses rayons partout, la logique de placement des points effectue toujours le même choix.
Pour 4 démarrages lancés avec le nouveau code on a les surfaces explorés suivantes :

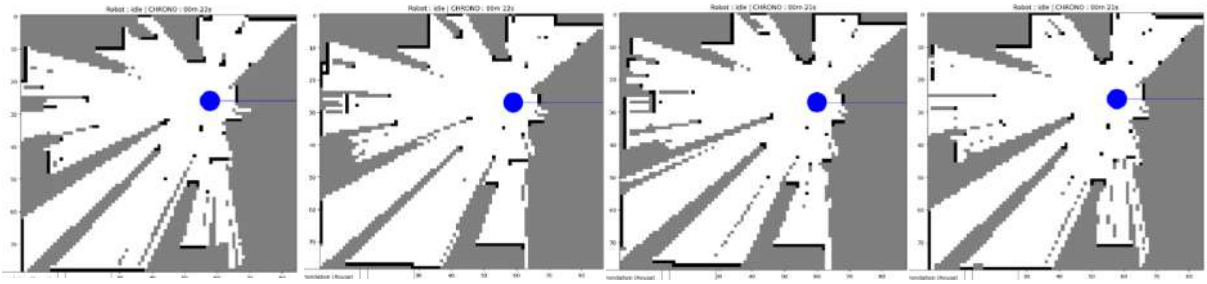

FIGURE 15 – Démarrage 1FIGURE 16 – Démarrage 2FIGURE 17 – Démarrage 3FIGURE 18 – Démarrage 4

FIGURE 19 – Quatre démarrages dans des conditions identiques avec le nouveau code.

On remarque que toutes les cartes sont dans le même sens, que les surfaces de pixels blancs sont plus nettes, qu'il n'y a presque plus de rais de pixels gris sauf derrière des obstacles et que la surface totale

explorée au démarrage est plus grande.

Voyons maintenant la répartition des premiers points d'intérêt proposés par les deux codes sur six cartes obtenues dans des conditions identiques : (le premier point auquel le robot se rendra est indiqué par un cercle orange à contour noir)

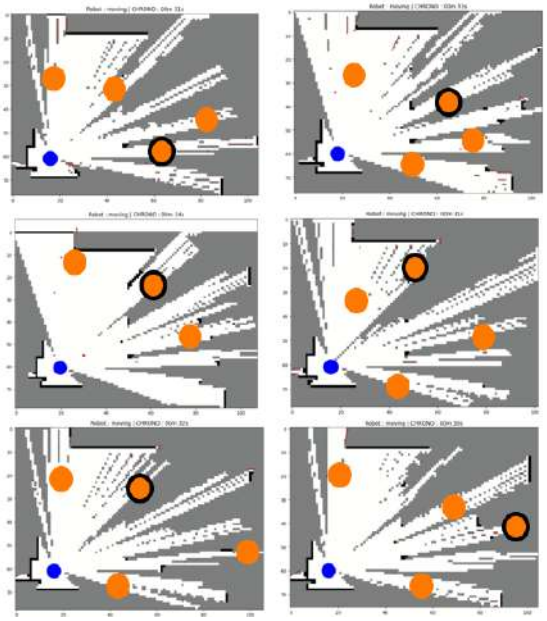

FIGURE 20 – Graphe de répartition du code d'origine.

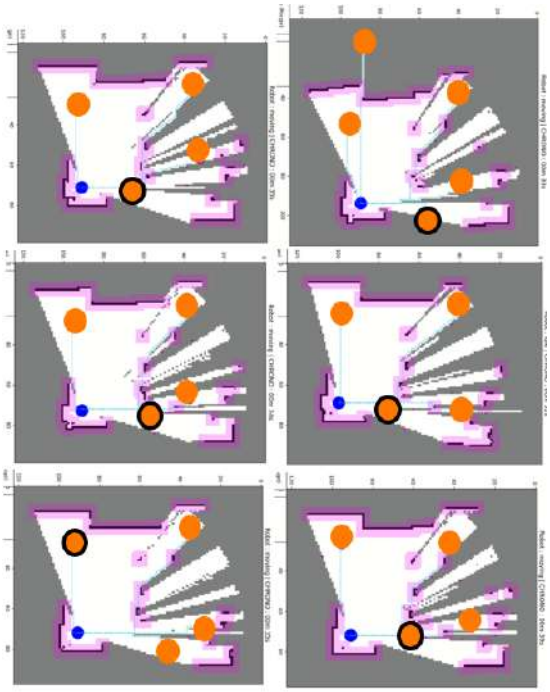

FIGURE 21 – Graphe de répartition du nouveau code.

On peut voir que pour 6 démarrages en conditions identiques, le code d'origine créé des cartes qui se ressemblent peu, les points sont dans les même zones mais aucun motif régulier n'est observé, de plus le premier point d'intérêt auquel le robot se rendra est le même seulement une fois sur trois ou moins. Le nouveau code propose quant à lui des cartes avec beaucoup de similitudes, sauf la deuxième où un faux rayon est tracé vers le haut - le rayon est impossible car il y a un mur à cet endroit - l'erreur vient donc du capteur en lui même. De plus, les points ont l'air d'être tous dans les mêmes zones et le premier point d'intérêt choisi est toujours le même, sauf lors du premier test.
Plus concrètement, si on regarde la répartition de tous les premiers points pour les deux codes on a les graphes suivants (avec en coordonnées x et y les coordonnées fournies par le robot) :

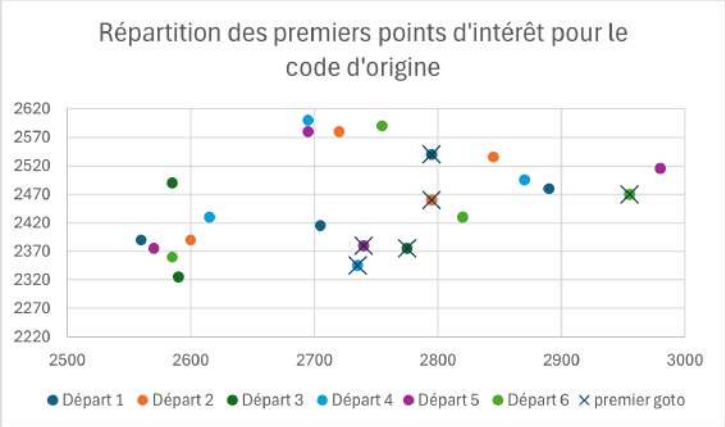

FIGURE 22 – Premiers points d'intérêt proposés avec le code d'origine, une croix noire indique le premier point auquel le robot se rendra.

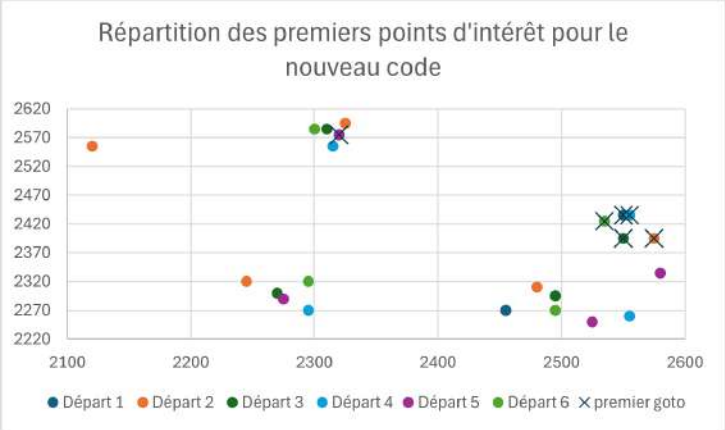

FIGURE 23 – Premiers points d'intérêt proposés avec le nouveau code, une croix noire indique le premier point auquel le robot se rendra.

On constate alors que les points générés par le nouveau code sont globalement répartis de manière similaire, à l'exception d'un point résultant d'une erreur de capteur. À l'inverse, les points issus de l'ancien code sont beaucoup plus « éparpillés », sans pour autant qu'un regroupement particulier ne se dégage. De plus, les premiers points auxquels le robot se rendra sont quasiment tous regroupé au même endroit pour le nouveau code, à la différence de l'ancien code qui générait des points beaucoup plus dispersés.

On peut donc dire que la solution du nouveau départ rend les choix du robot plus réguliers et plus stables.

3.2 Orientation de la carte générée

Malgré les changements apportés au départ du robot permettant à la carte générée d'être dans le bon sens, on constate que celle-ci n'est pas systématiquement alignée dans l'orientation souhaitée. Elle peut être penchée et les murs générés sont donc irréguliers (Figure 24) ce qui diminue la qualité du résultat final. Cela peut aussi provoquer des erreurs telles que des chemins pour rejoindre un point qui traversent des murs, car ils ne sont pas fermés à cause des espaces entre les pixels noirs.

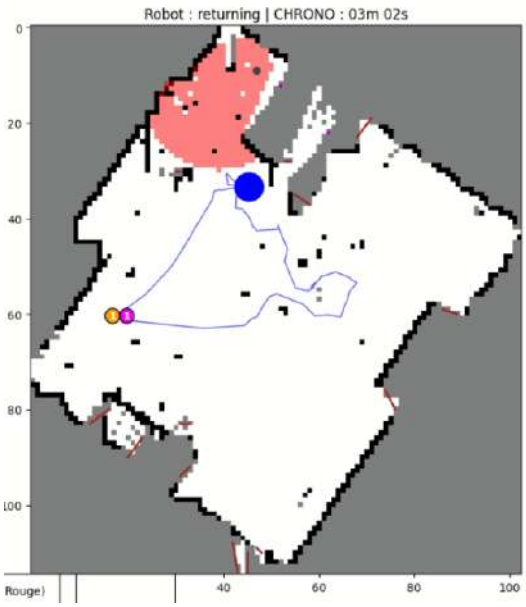

FIGURE 24 – Carte générée penchée, avec des murs en diagonale.

Cependant, lors de tests dans mon logement personnel j'ai réalisé que ce comportement n'était pas aléatoire. Le logement comportait ainsi un mur penché par rapport aux autres, et lorsque le robot était lancé contre celui-ci, la carte qui en résultait était elle aussi penchée. A l'inverse, s'il était lancé contre un autre mur (parallèle ou perpendiculaire à tous les autres), alors la carte était d'une qualité bien meilleure. Cela observé, des tests avec des lancements orientés différemment nous ont permis de faire un lien entre position du robot au départ et orientation de la carte. Ce lien n'ayant pas été fait auparavant, nous pensions que la carte était générée en fonction des premiers murs que le robot voyait.
On observe ci-dessous deux cartes obtenues dans le même logement, mais avec le robot lancé au départ contre des murs différents, le résultat est sans appel, la carte est bien meilleure lorsque le robot démarre contre un mur qui n'est pas penché par rapport à la majorité des autres.

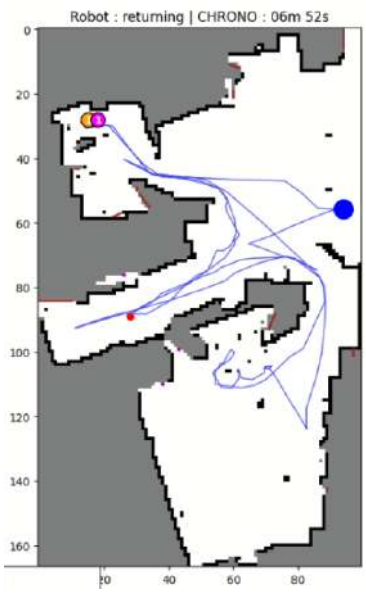

FIGURE 25 – Carte obtenue avec le robot lancé contre le mur penché.

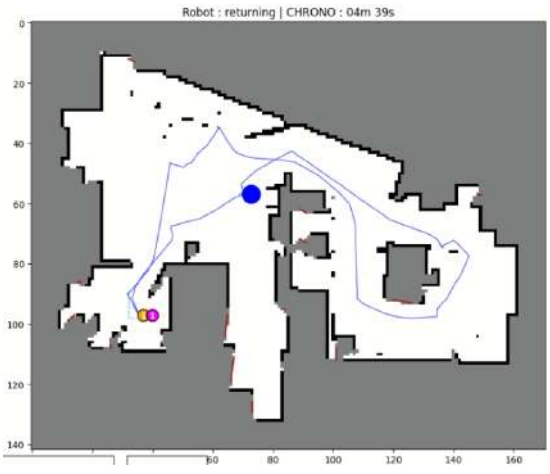

FIGURE 26 – Carte obtenue avec le robot lancé contre un autre mur.

Il s'avère que le robot contient une boussole électronique qui lui permet de s'orienter et donc d'orienter

la carte. Cette idée que la position du robot au départ a un impact sur l'orientation de la carte a été vérifié au laboratoire d'Orange, en lançant le robot dans la même pièce mais avec des orientations différentes.

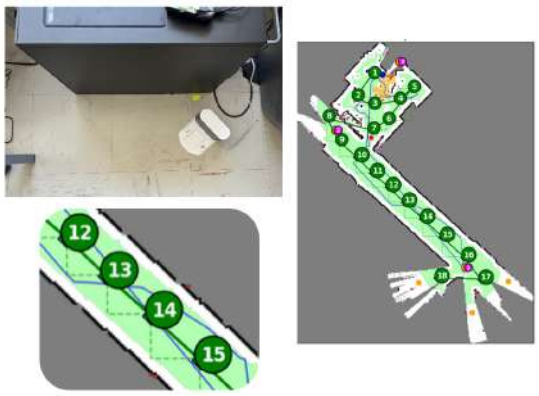

FIGURE 27 – Carte obtenue avec le robot lancé penché par rapport aux murs de la pièce.

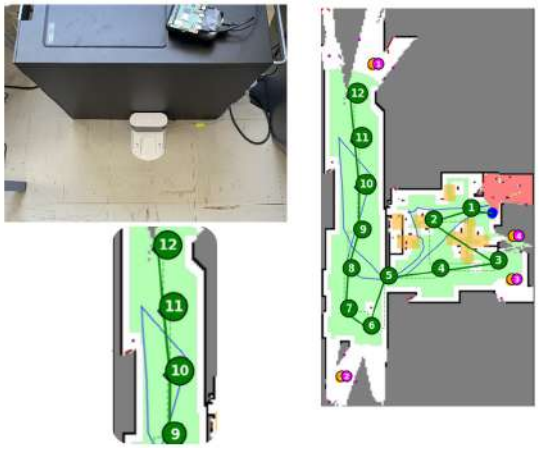

FIGURE 28 – Carte obtenue avec le robot lancé aligné par rapport aux murs de la pièce.

FIGURE 29 – Cartes de l'appartement orientées différemment

Comme montré ci-dessus, on peut voir que l'angle de départ va influer fortement sur les cartes obtenues *in fine*.

Afin d'obtenir à chaque lancement la carte la plus nette et la plus fiable possible, on ajoute alors une indication à l'utilisateur sur le placement de la base du robot au départ. Jusqu'à présent, l'indication était de placer la base contre un meuble ou un mur, ce qu'on change maintenant en "placer la base du robot contre un mur, de préférence aligné avec le plus de murs de la maison". Ainsi, l'orientation de la carte sera contrôlée, même si parfois elle n'est pas tout à fait droite, car le robot ne s'est pas rangé parfaitement dans sa base et qu'un petit décalage d'angle peut avoir un gros impact si la carte est grande.

3.3 Des allers-retours chronophages

Lors de plusieurs lancements avec le code d'origine, on remarque que parfois le robot fait des allers-retours entre les pièces sans vraiment savoir pourquoi. En effet, on peut par exemple voir le robot rentrer dans une pièce, en ressortir pour atteindre un autre point d'intérêt puis retourner dans la pièce en rejoindre un autre. On pourrait penser que c'est à cause d'une frontière qui s'est déclarée au moment où le robot sortait de la pièce et qui amène le robot à retourner d'où il vient car il n'avait pas tout vu, seulement celle-ci y était déjà en rentrant dedans. La question est donc de savoir pourquoi le robot n'y va-t-il pas.
Ci-dessous des trajets observés dans un appartement, on y voit bien les allers-retours faits par le robot.

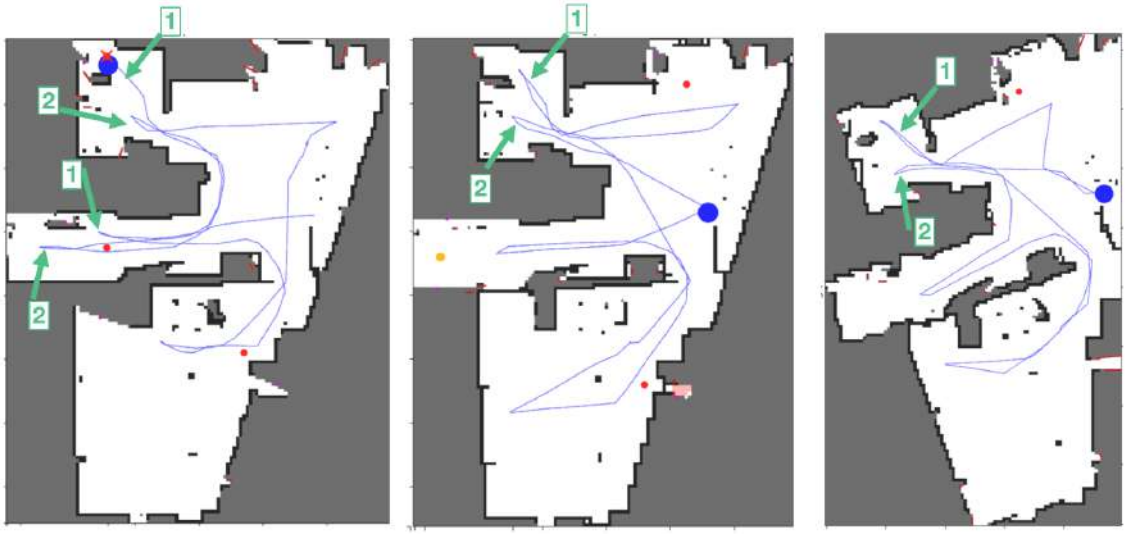

FIGURE 30 – Trajet du robot (bleu) et numérotation d'allers-retours (vert) pour trois cartes différentes.

En regardant en détail l'intelligence derrière le choix du prochain point d'intérêt où le robot va se rendre, on s'aperçoit qu'on utilise un pathplanning géré avec le TSP : Travelling Salesman Problem ou problème du voyageur de commerce. Ce problème consiste à déterminer le plus court chemin passant par chaque points d'un ensemble en une seule fois, or dans notre cas tous les points ne sont pas connus au départ, le problème évolue à chaque déplacement. En effet, les points d'intérêt à un instant t ne sont pas les mêmes à l'instant t+1, programmer un trajet en se basant sur l'ensemble de points à l'instant t est donc inadéquat.

FIGURE 31 – Ensemble de points (à gauche) et le trajet les reliant calculé par le TSP (à droite) [Wik26]

Ce qui causait les allers-retours était donc une stratégie de gestion du prochain point d'intérêt inadaptée à notre mission de cartographie rapide basée sur des frontières.
Cette intelligence de path planning a été modifiée, dorénavant le robot choisit le point le plus proche de lui-même en distance réelle parcourue.

Lors des tests menés après modification de la stratégie, on ne voit plus d'allers-retours inexpliqués, le seul cas particulier où l'on peut en voir un est celui où le robot rentre dans la pièce, en ressort car il n'y a plus de points d'intérêt dans celle-ci, mais dois y retourner plus tard car il a vu des frontières lorsqu'il sortait de la pièce qu'il n'avait pas vues auparavant.

3.4 Le robot n'explore pas toute la zone

Dans certains cas, on peut voir le robot ne pas finir l'exploration et présenter un plan incomplet (cf figure 32), ce qui pose un problème majeur à l'ensemble du dispositif, car l'exploration doit être relancée si l'on veut une carte entière.

Après une nouvelle analyse du choix fait par le robot pour son prochain point d'intérêt, on sait que lorsqu'un point est considéré comme trop proche du robot ou du trajet déjà réalisé, il est exclu de la liste des candidates. Cette condition est censée éviter que le robot retourne en boucle au même endroit si il n'y arrive pas ou qu'il fasse de tout petits déplacements. Or dans des cas particuliers comme celui Figure 32 ces points considérés trop proches sont ceux qui permettent de finir l'exploration.

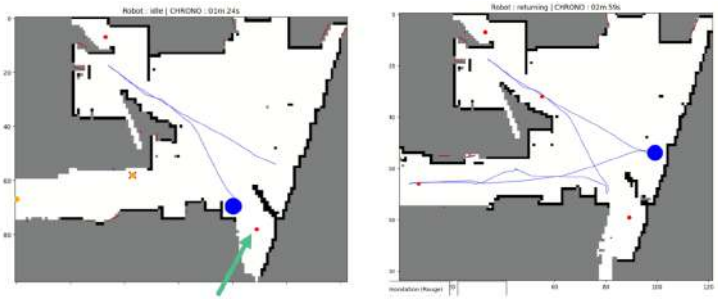

FIGURE 32 – A gauche, moment de décision et point d'intérêt trop proche exclu signalé par la flèche verte, à droite plan final produit.

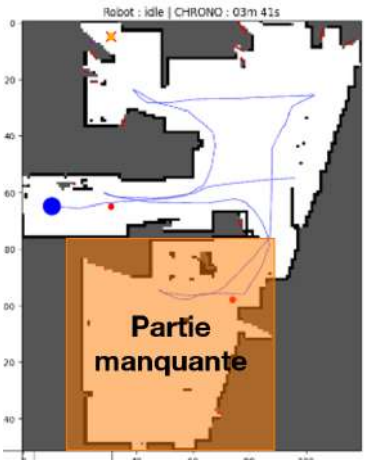

FIGURE 33 – Carte complète avec la partie manquante à l'autre carte.

La solution envisagée est donc de diminuer cette distance limite qui exclue trop de points d'intérêts. Le robot peut maintenant faire des "petits pas" mais il va partout comme dans le cas ci-dessous, Figure 34. Le robot ira maintenant à un point à moins d'un mètre de sa dernière position.

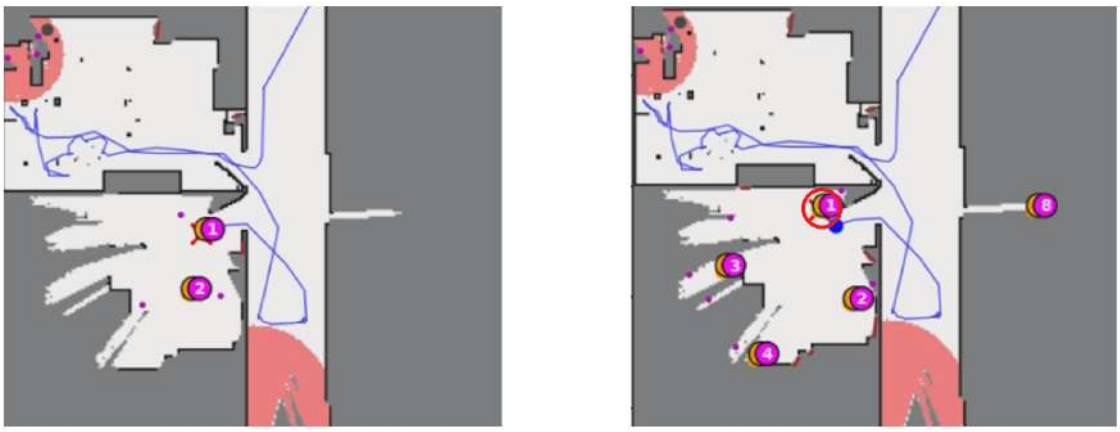

FIGURE 34 – Position du robot à l'instant t (à gauche), choix du prochain point d'intérêt signalé par une croix rouge (à droite).

3.5 Éléments manquants sur les bords de la carte générée pour les calculs

On observe parfois d'autre allers-retours qui ne sont pas causés par le TSP, ni par un point considéré trop proche, ceux-ci apparaissent souvent lorsqu'une seule pièce n'a pour le moment été visitée et les allers-retour se font vers les bords de la carte. Lorsqu'on compare les cartes générées par Valetudo et celle générée sur la Pi pour les calculs on observe une différence, il peut manquer des murs sur les bords de la carte générée sur la Pi (Figure 35).

Étant donné qu'une frontière est définie dans ce projet comme l'interface entre un pixel libre et un pixel inconnu à une distance sûre d'obstacles, les bords de carte où les murs n'apparaissent pas sont

considérés comme tels et un point d'intérêt y sera placé pour que le robot s'y rende. Lorsque le robot s'y rend, les murs ne sont toujours pas affichés, des frontières seront alors encore présentes et un point d'intérêt de nouveau placé.
Ci-dessous, le trajet bleu (à gauche) ou blanc (à droite) indique que le robot s'est rendu deux fois dans le coin inférieur gauche de la pièce. Sur la carte de gauche créée sur la Pi, les murs ne sont pas tracés, tandis qu'ils sont affichés sur Valetudo.

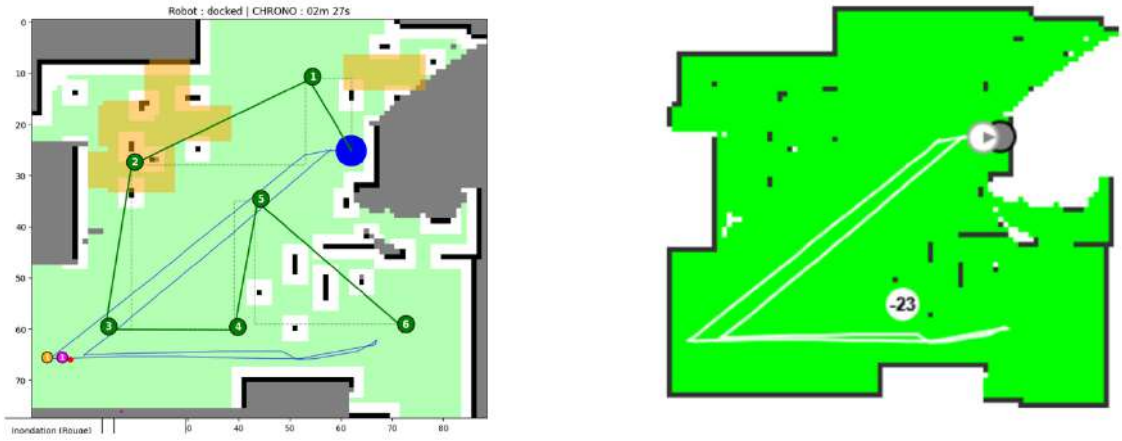

FIGURE 35 – Carte générée sur la Pi (à gauche), carte générée sur Valetudo (à droite).

Afin d'éviter au robot de se rendre à des lieux connus mais mal représentés, on ajoute donc une bande de pixels gris sur tous les bords de la carte générée sur la Pi, après cette modification les murs jusqu'à présent non apparents sont tracés et le robot ne s'y rend plus pour rien (Figure 36). Le problème venait donc de l'espace de création de la carte qui devait être trop petit au départ.

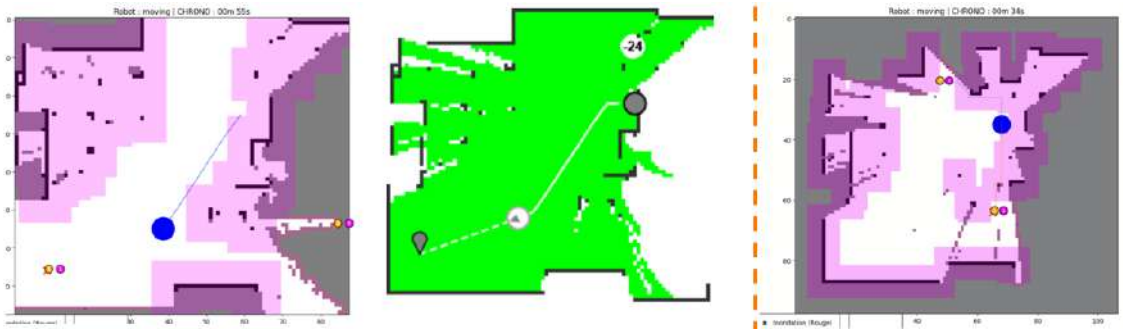

FIGURE 36 – En partant de la gauche, plan anciennement représenté sur la pi, plan représenté sur Valetudo, plan dorénavant représenté sur la Pi.

3.6 Points inaccessibles

A présent, on peut se demander quelle pourrait être une nouvelle cause d'allers-retours après tous ces réglages. Une des réponses possibles peut être qu'il y ait des points inaccessibles pour le robot, ce que nous allons détailler ensuite.
Parfois lors du choix du prochain point d'intérêt, même en demandant au robot d'aller au point le plus proche, celui-ci n'y allait pas, ce qui causait de nouveau des allers-retours et donc une perte de temps ou même encore une partie de la carte ignorée.
Avec le code actuel, un point d'intérêt est placé au centroïde d'un cluster de frontières, il s'avère que ce centroïde peut parfois atterrir sur un pixel gris. Or un pixel gris est considéré comme inaccessible par le robot et si les coordonnées de ce point sont envoyés par Valetudo sous forme d'ordre de déplacement, alors un message d'erreur apparaîtra et le robot ne s'y rendra pas.

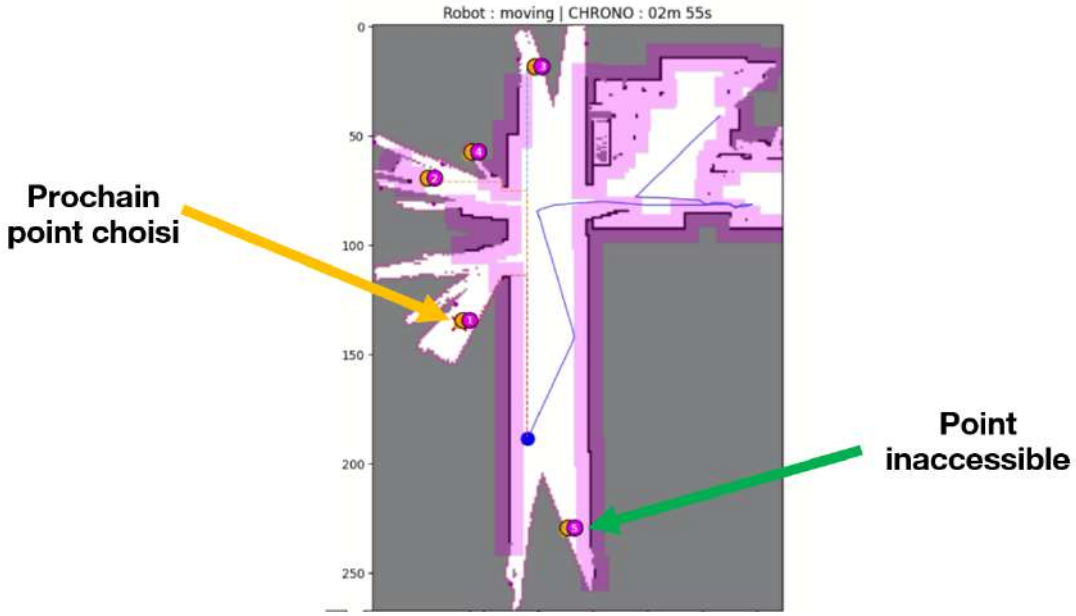

FIGURE 37 – Choix du prochain point d'intérêt, le pathfinding n'arrive pas à atteindre le point le plus proche du robot.

On implémente alors une logique de "snap to free", ce qui veut dire que si le centroïde est placé sur un pixel gris (inconnu) il sera déplacé au pixel blanc (libre) le plus proche. Les points proposés sont maintenant toujours sur un pixel libre, aucun ne sera donc ignoré.

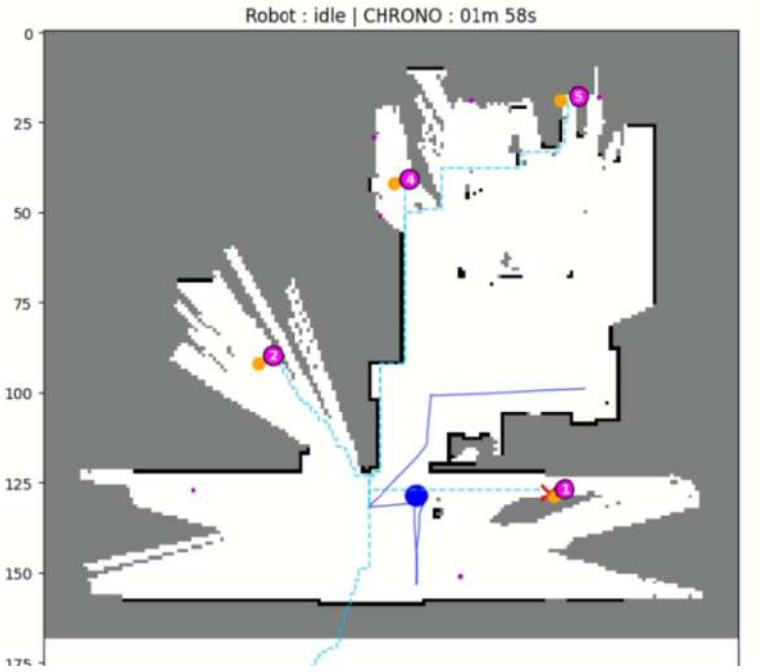

FIGURE 38 – Croix rouge légèrement décalée par rapport au cercle jaune de la frontière, indiquant que les coordonnée du prochain point ont été déplacées par rapport à celles du point d'origine

3.7 Mauvaise échelle

A la fin de l'exploration, un assistant IA donne des indications sur la qualité de la couverture Wi-Fi mais aussi sur la surface de la zone explorée. Lors d'un test dans mon logement personnel j'ai reçu une valeur presque deux fois inférieure à celle indiquée dans mon bail de location, ce qui m'a interpellée. En effet l'unité utilisée est celle donnée par Valetudo, un pixel ferait 5mm x 5mm. Or cette valeur ne donne

pas de résultats probant, on décide donc de redéfinir l'équivalent pixel/mm réels.
Pour ce faire on réalise plusieurs fois le protocole suivant : lancer le démarrage du robot, une fois sorti de sa base on récupère sa position (x, y), on envoie un 'goto' (ordre de déplacement) au point de coordonnées (x-20, y) et on mesure la distance réelle entre ces deux positions. Après 10 lancements on obtient le tableau suivant (39), qui nous permet d'obtenir en réalité qu'un pixel mesure 4,34 mm x 4,34 mm. Cette calibration permet d'utiliser le facteur de mise à l'échelle de 1,15 pour convertir les pixels en millimètres réels.

pixels	cm réels
20	85
	84,5
	82
	86,5
	93,5
	87
	88
	87
	87,5
	95

FIGURE 39 – Données réelles obtenues pour 10 tests.

pixels	cm réels
20	86,77777778
1	4,338888889

FIGURE 40 – Calcul du facteur d'échelle.

On constatera par la suite que ce facteur d'échelle n'est applicable que sur des petites surfaces et qu'au dessus de 50 m^2 il ne l'est plus. Il sera par la suite nécessaire de réaliser différents tests sur différentes tailles de surfaces afin de définir une formule générale.

Après avoir appliqué ce facteur sur toutes les conversions, la surface calculée à la fin de l'exploration paraît plus cohérente mais il manque tout de même quelques mètres carrés. On regarde alors quelle surface est renseignée par le programme et on voit que ça n'est pas la surface totale mais la surface considérée comme sûre à parcourir pour le robot, or ça n'est pas la même chose. Considérons le plan suivant :

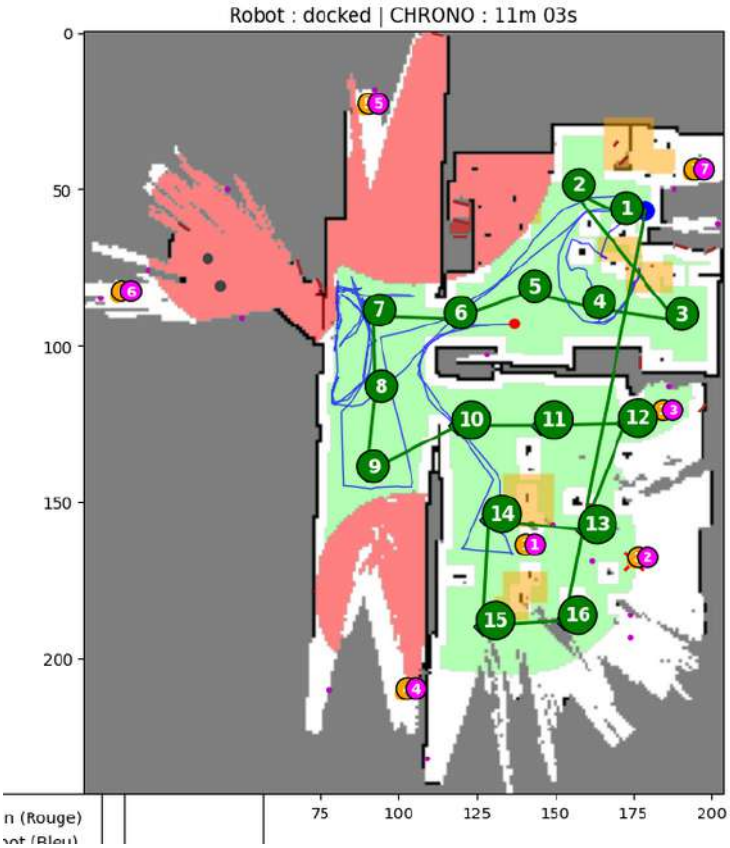

FIGURE 41 – Carte générée en fin d'exploration.

Lors de la création de la carte finale avec les points de mesure, la surface libre (blanche) de l'exploration est réduite car on dilate les pixels noirs (par une matrice 3x3) afin que les déplacements du robot évitent les obstacles, mais on place aussi des zones inaccessibles en rouge et à risque en jaune, qui empiètent sur la surface blanche. La surface verte et celle qui est pour le moment donnée à l'utilisateur comme la surface totale de sa maison/son appartement. On change alors juste ce qui est utilisé par l'assistant IA en la surface totale de pixels blancs.
On obtient alors une surface beaucoup plus cohérente, mais non vérifiable car les objets et meubles posés au sol occupent de la surface libre et ne sont pas facilement calculables, on se satisfait donc d'une surface cohérente "à vue d'oeil".

3.8 Pas de limite de distance à la base

Un problème soulevé par mon collègue Mr.Belaunde Mariano est le suivant, le robot n'a aucune sécurité quant à sa distance à sa base, c'est-à-dire que le robot peut partir aussi loin qu'il le peut sans rien qui l'en empêche tant que le code d'exploration tourne sur la Pi, ce qui pose un problème de sécurité.
La solution proposée est donc celle d'une limite de distance à la base qui permettrait au robot de ne pas dériver et pourquoi pas de sortir de la maison.
On implémente alors une distance maximale à la base qui filtre les points auxquels les robot peut se rendre et exclue ceux situés trop loin. Le robot peut ainsi dépasser cette limite pour se rendre à un point mais pas se rendre à un point qui dépasse cette limite.
Comme on le voit sur le plan ci-dessous (Figure 42), la croix orange est la base du robot, le cercle orange est la distance maximale définie au delà de laquelle les points d'intérêt ne seront pas retenus, le points d'intérêts (tous marqués par des points jaunes) ont différentes lettres accolées : un F noir indique un point "Too Far" ce qui veut dire qu'il est plus loin que la distance maximale à la base.

FIGURE 42 – Visualisation des la distance maximale à la base.

On peut donc contrôler l'éloignement à la base avec un seul paramètre et donc s'assurer que le robot reste en sécurité.

3.9 Durée maximum pour atteindre un point trop élevée

Il arrive que le robot mette du temps à atteindre sa cible ou qu'il n'y arrive pas du tout, dans certaines configurations en particulier les pièces avec beaucoup de chaises. Le robot peut mettre du temps à atteindre ses cibles car il ne le détecte pas bien les petits objets ou les pieds de meubles, il essaie donc de tout traverser et se cogne à de multiples reprises. Pour cela le code d'origine a prévu une limite de 120 secondes pour atteindre un point, mais cette limite fait perdre beaucoup de temps. Au cours de nombreuses expériences il a été relevé que si le robot n'atteignait pas sa cible en moins d'une minute, il n'y arriverait pas en 2 minutes. Ce temps maximal pour atteindre une cible a donc été réduit afin d'éviter de perdre du temps à atteindre une cible inaccessible.
Cette durée maximale a été définie suite à de nombreux tests avec différentes valeurs, à 50 secondes.

De plus, dans le code d'origine, lorsqu'un robot n'atteignait pas sa cible, une petite zone rouge était créée et considérée comme inaccessible. Cependant, il arrivait que le robot n'atteigne pas sa cible dans le temps imparti alors que celle-ci restait en réalité accessible en empruntant une trajectoire différente. La création de cette zone rouge pouvait donc être erronée et empêcher toute mesure du signal Wi-Fi à cet endroit, puisque les zones identifiées en rouge sont exclues des zones de mesure.

Prenons l'exemple de la Figure 43, un cercle rouge a été placé pour un point précédemment non atteint après le délai, or on voit le robot (cercle bleu) et son trajet qui traversent cette zone. C'est donc le signe qu'un point n'a pas été atteint à temps mais est accessible, c'est donc une erreur que de placer une zone rouge à cet endroit.

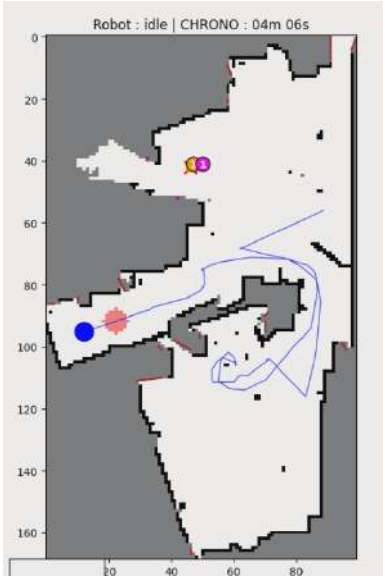

FIGURE 43 – Carte obtenue après qu'un point n'ai pas été atteint en 120 secondes.

Les changements apportés sont les suivants :
le robot a maintenant 50 secondes pour atteindre sa cible, si il n'y arrive pas il s'arrête et recalcule les frontières pour vérifier si le point d'origine est toujours pertinent. Ensuite se présentent plusieurs cas :

- Cas 1 : Le point d'intérêt précédemment suivi se trouve à plus de 2 mètres du robot. Une nouvelle cible située dans la même zone que l'ancien point est identifiée. Le robot continue alors de se diriger vers le même point d'intérêt.
- Cas 2 : Le point d'intérêt précédent se trouve à plus de 2 mètres du robot. Une nouvelle cible située dans la même zone que l'ancien point est détectée. Une autre cible, plus proche du robot, est également identifiée. Le robot se dirige alors vers cette cible la plus proche.
- Cas 3 : Le robot se trouve à moins de 2 mètres de la cible. Il recalcule alors ses cibles et reprend la procédure habituelle en excluant un nouveau point dans la zone de l'ancienne cible.

Dans tous les cas, aucune zone rouge n'est placée à la fin de la durée maximale.
Cette procédure a été mise en place de manière à perdre le moins de temps possible lors de l'exploration.

3.10 Pas d'indicateur de fiabilité de la carte pour l'utilisateur

Lors d'un test de 50 minutes dans les locaux d'Orange, tout semble bien se passer, le robot passe par tous les bureaux ouverts, perd une fois la connexion mais met à jour la carte lorsqu'il la retrouve, passe par dessus des câbles, doit retrouver sa position et y arrive, la carte produite à la fin semble complète Figure 44. Mais lors de la revue de la carte on remarque plusieurs problèmes : certaines ouvertures de porte sont barrées d'un mur alors que le robot y est passé et que physiquement les portes sont toutes ouvertes (cercles oranges), il est censé y avoir un espace entre le labo et le bureau de Loïc car un bureau était fermé, le robot a traversé 10 pièces mais 9 sont présentes sur la carte. Pour comparaison, la Figure 45 est une carte juste des locaux.
En effet, le robot a visiblement produit une carte fausse sans indiquer à l'utilisateur aucun message d'une erreur qui aurait donné lieu à une telle confusion.

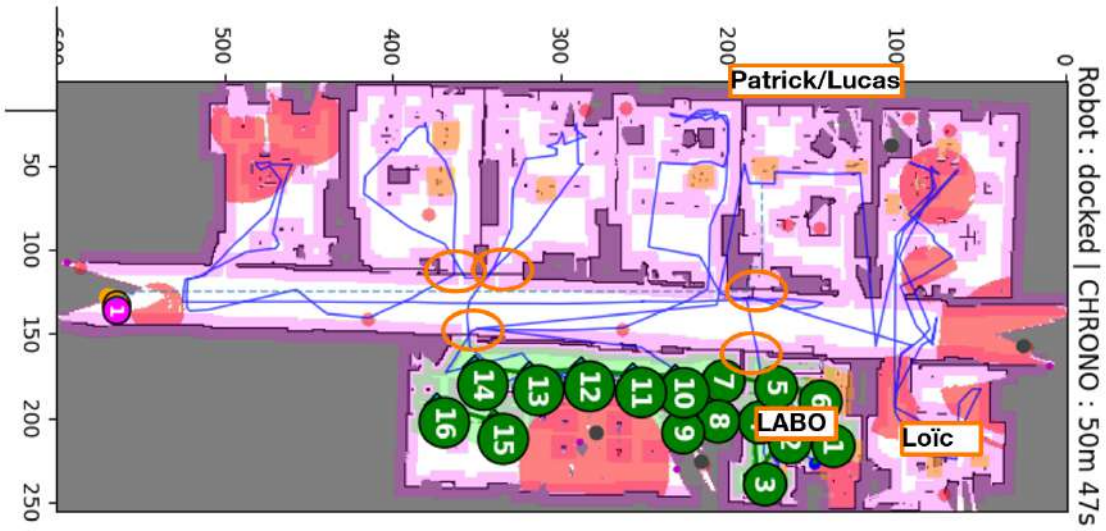

FIGURE 44 – Carte non valide des bureaux.

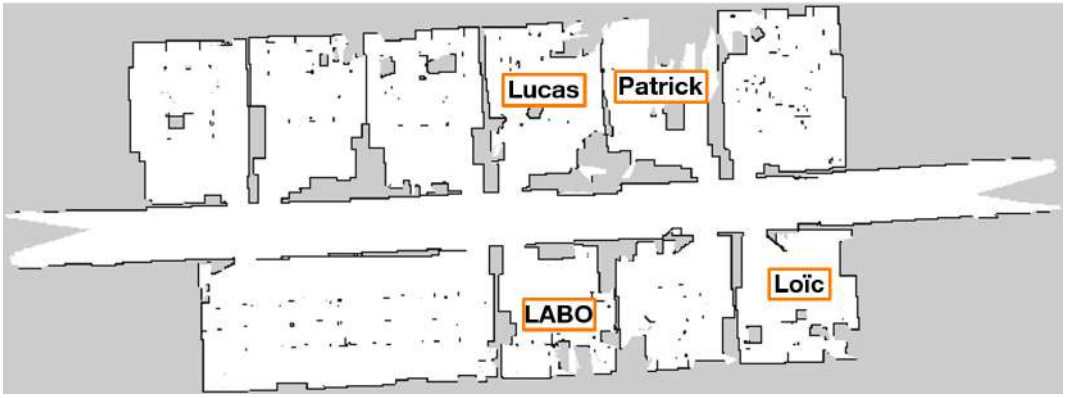

FIGURE 45 – Carte valide des bureaux.

Une nouvelle fonctionnalité implémentée est donc le comptage d'erreurs et leur type, ainsi que la création d'un fichier .txt qui pourra être transmis à l'assistant IA afin qu'il puisse prévenir d'une carte potentiellement non fiable et demander à l'utilisateur de vérifier.
La prochaine étape est de classer ces erreurs par ordre de la plus à la moins impactante sur la carte, seulement une même erreur n'a pas toujours le même effet. La perte de connexion lors du test qui a permis de mettre en lumière ce problème est ensuite réapparue lors d'un autre test, et pas moins de 68 fois sur 54 minutes, mais la carte était tout à fait juste. Lors d'un autre test le robot s'est retrouvé coincé dans des câbles et il a été déplacé à la main, a retrouvé son chemin mais les murs du bureau dans lequel il se trouvait ont été déplacés et un passage inexistant avec le bureau d'à côté a été crée, pourtant lors d'autres tests, un déplacement à la main n'a pas entraîné d'erreurs visibles de cartographie.
Après discussion avec les personnes à l'origine du code sur lequel on travaille, les plus gros problèmes de cartes ne sont pas liés à une perte de connexion, on peut donc mettre celle-ci de côté, par ailleurs lorsque le robot se perd et essaie de retrouver sa position, il arrive qu'il pense y arriver mais la carte est mauvaise.
Ce genre d'erreurs n'est malheureusement pas réellement recensée dans les logs du robot et est donc difficile à relever.
La classification des erreurs reste alors encore un sujet complexe.

3.11 Les points d'intérêt sont derrière des obstacles

Ce qui est très chronophage lors de l'exploration, ce sont les points d'intérêt placés derrières des obstacles, malheureusement le robot ne les voit pas tous mais concernant ceux qu'il voit il s'agirait de ne

pas y aller.
Ce qui est observable c'est que certains points sont placés dans des zones d'obstacles (Figure 46), ici sur la carte de gauche, lancée avec le code d'origine, les 4 points noirs autour du point d'intérêt numéro un sont les pieds d'une chaise, on aimerait éviter d'y aller. Cela est dû au fait que les obstacles ne sont pas assez dilatés, l'espace entre les pieds sera donc considéré sûr. Une solution est d'augmenter la dilatation, on passe donc d'une dilatation en (3, 3) à une dilatation en (7, 7), c'est ce qui a été considéré idéal pour le compromis d'éviter les obstacles mais de pouvoir s'en approcher pour pouvoir par exemple traverser des couloirs. Cette nouvelle dilatation est celle visible en rose sur la carte de droite de la Figure 46, on peut voir qu'il n'y a pas d'espace libre blanc entre les 4 pieds et donc le point d'intérêt de cette zone n'est pas placé au milieu d'obstacles.

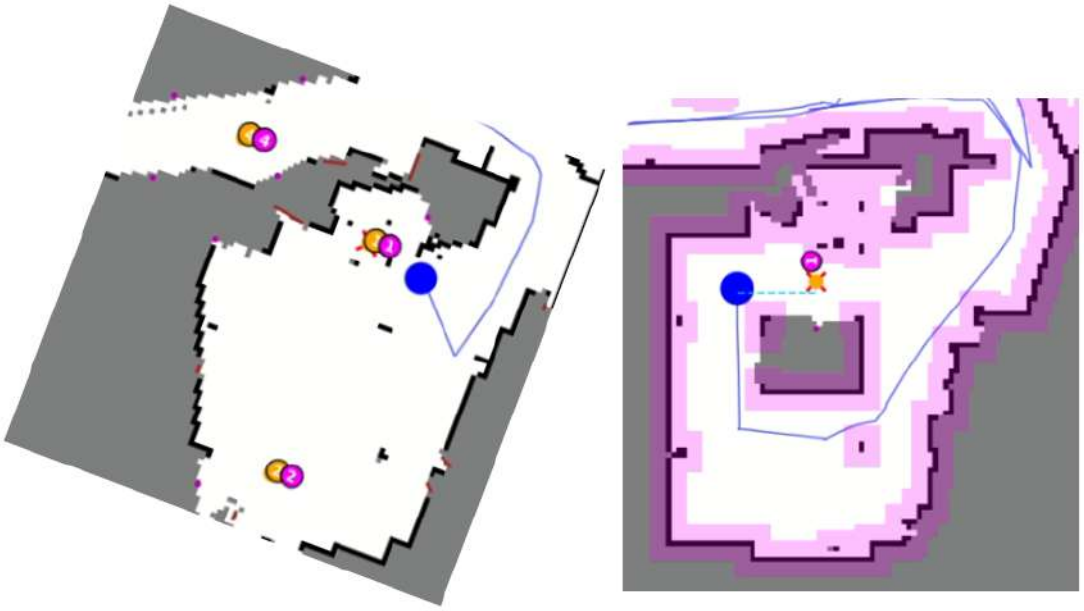

FIGURE 46 – A gauche, carte générée avec le code d'origine, à droite carte générée avec le nouveau code.

3.12 Les points d'intérêt sont dans une zone rouge

Les zones rouges sont placées lorsque le robot considère qu'une zone est inaccessible, il ne faut donc pas proposer de nouveaux points à cet endroit, pourtant dans certains cas c'était possible. Les zones rouges apparaissent à un moment t et ne couvrent que les pixels libres, si le robot découvre de nouveaux pixels blancs à t+1 ils ne seront pas couverts par la zone rouge et l'algorithme peut placer des points à l'intérieur.
Ci-dessous deux cartes générées avec le code d'origine, on y voit des zones rouges avec des point d'intérêt placés à l'intérieur Figure 47.

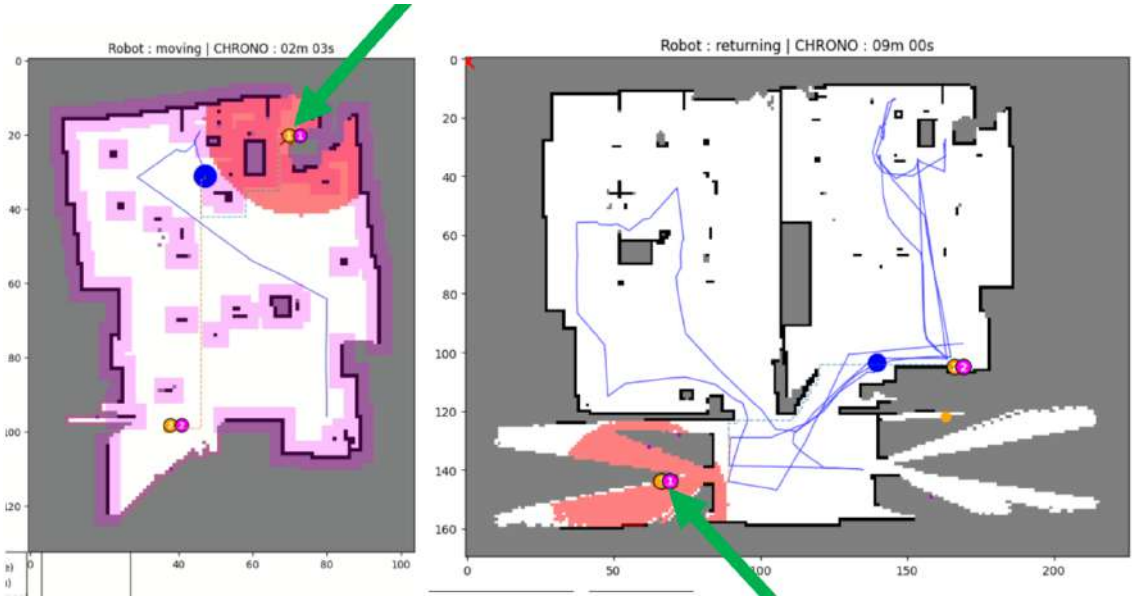

FIGURE 47 – Zones rouges et point d'intérêts placés à l'intérieur.

Une première idée a été de mettre un masque rouge sur les pixels blancs et gris afin qu'il ne puisse plus y avoir de pixels blancs découverts dans cette zone mais cette solution pouvait impacter l'accès à d'autres pièces, par exemple sur la Figure 48, l'accès en bas à droite de la carte est obstrué alors que le point inaccessible était dans une autre pièce.

FIGURE 48 – Zone rouge gênante pour l'accès à d'autres pièces.

La solution qu'on propose est de, tout comme pour les murs et obstacles, dilater la zone rouge dans la limite de son périmètre maximal, afin que si des pixels blancs sont découverts, ils seront sous la zone rouge dilatée. Si des points sont tout de même placés dans cette zone c'est qu'au final ils étaient accessibles, mais c'est un cas qui n'a pas encore été observé.
De plus, pour éviter que les zones rouges impactent d'autres pièces, on fait en sorte que lors de la création de la zone, celle-ci ne traverse pas les obstacles, ainsi les murs bloquent la propagation des zones inaccessibles et les restreignent.
La Figure 56 illustre bien les changements. On y voit en haut à droite, une zone rouge avec des zones rouges plus claires et des rais derrière les obstacles. Ces rais sont la preuve que les pixels noirs bloquent la propagation de la zone rouge et leur couleur rouge clair est la conséquence de la dilatation. La zone rouge au dessus du robot (point bleu) est issue d'un point inaccessible dans la pièce de gauche, et l'on

peut voir que celle-ci ne traverse pas les murs et ne prend donc pas de surface sur une autre pièce.

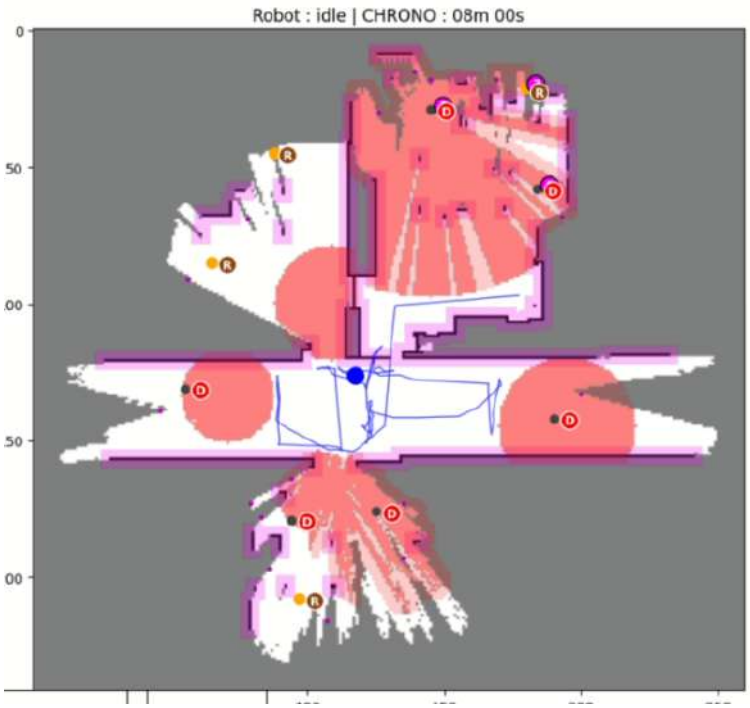

FIGURE 49 – Zones rouges crées avec le nouveau code.

3.13 Les trajets entre les points de mesure Wi-Fi passent à travers les murs

A la fin de l'exploration, une carte est générée contenant les point de mesure du débit et signal Wi-Fi ainsi que leur ordre de passage, cette carte comporte un problème, l'ordre des point fait faire des aller-retours entre des pièces au robot. En y regardant de plus près le trajet calculé entre les points de mesure traverse les murs ce qui donne lieu à quelque chose de non optimisé. Sur la Figure 50, on peut voir qu'entre les points 5 et 6 mais aussi 11 et 12, le trajet traverse un mur, le robot devra donc faire des allers-retours inutiles entre les deux pièces.

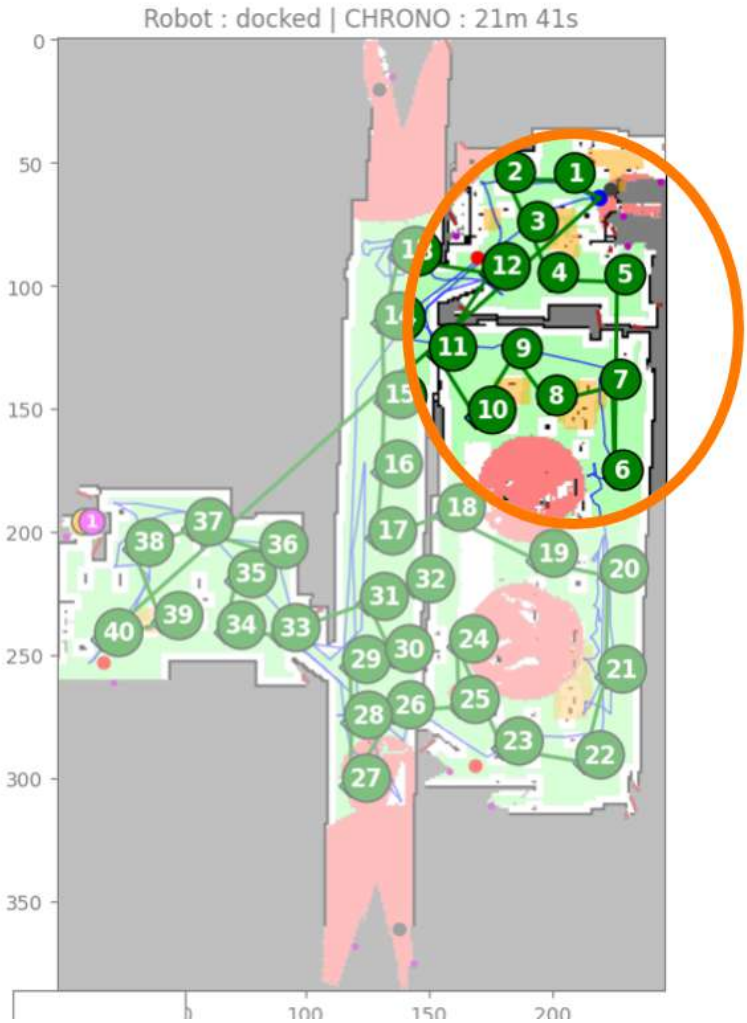

FIGURE 50 – Trajet des points de mesure avec le code d'origine.

En effet, afin de rendre les calculs et donc la création de carte plus rapides, un facteur de réduction est appliqué à la carte d'exploration pour placer les points et c'est ce qui donne lieu à des murs qui "disparaissent" et laissent passer les trajets. Ce facteur a été enlevé car les murs ne faisant qu'un pixel d'épaisseur, n'importe quel facteur de réduction supprimais ceux-ci. Les points sont maintenant reliés de manière logique et le trajet ne fait pas d'allers-retours (Figure 52).

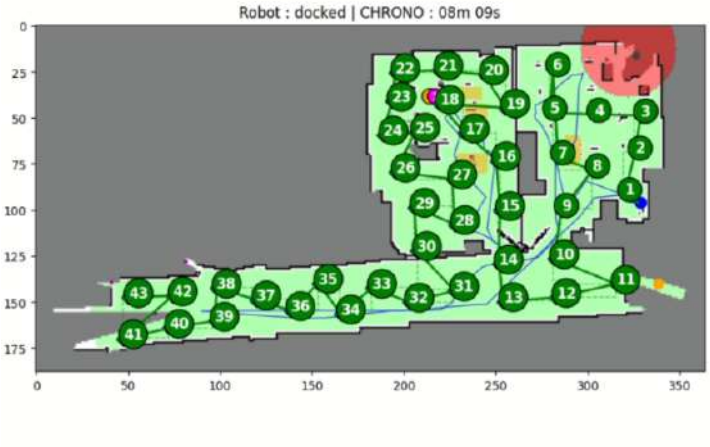

FIGURE 51 – Trajet des points de mesures avec le nouveau code.

Par la suite, on penchera aussi sur la question de densité des points de mesure Wi-Fi, car celle-ci ne semble pas être constante alors qu'elle prend en entrée une valeur fixe de densité de points par m^2.
Si on regarde la figure 52, on peut constater que dans les deux pièces en haut à droite, qui font presque la même surface, il y a 10 points dans la plus à droite contre 16 points dans la deuxième. De plus on observe parfois la même chose mais dans différentes parties d'un couloir alors qu'il fait la même largeur partout.
On réalise quelques essais avec une autre manière de répartir les points, une tessellation de Voronoï centroïdale (CVT) basée sur la distance géodésique, couplée à une distance aux murs plus importante qu'avant. Cette approche n'est pas encore au point mais permet déjà quelques résultats :

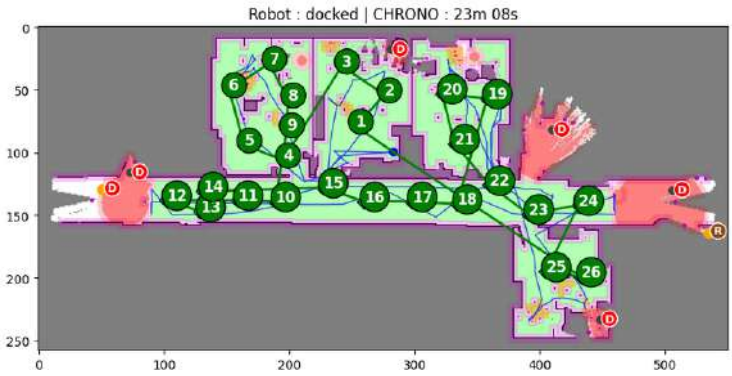

FIGURE 52 – Nouvelle répartition des points de mesure Wi-Fi.

On peut voir que les points de mesure sont parfois mieux placés, moins près des murs, mais il persiste en bas à gauche une grande densité comparé au reste de la carte.
Une solution sera implémentée avant la fin de ce stage.

4 Analyse des résultats

L'ensemble de ce travail d'amélioration a produit des résultats significatifs, tant du point de vue du gain de temps et de la reproductibilité que de la compréhension et de l'acceptation par l'utilisateur, ainsi que de la sécurité et de la qualité de résultat.

En effet, le nouveau code a permis un gain de temps notable. Un benchmark comparant la durée d'exploration entre le code d'origine et le nouveau code pour 11 départs a été réalisé dans des conditions identiques. Les résultats sont présentés ci-dessous :

Date	Heure	Durée	Bug (y/n)	Map penchée (y/n)
18/06/2026	09 :04 :17	05 :46	n	y
18/06/2026	09 :10 :20	05 :43	n	y
18/06/2026	09 :17 :44	07 :06	n	y
18/06/2026	09 :25 :06	05 :35	n	y
18/06/2026	09 :30 :56	03 :13	y	y
18/06/2026	09 :34 :25	06 :33	n	y
18/06/2026	09 :41 :20	06 :41	n	y
18/06/2026	09 :48 :16	06 :54	n	y
18/06/2026	09 :55 :24	05 :16	n	y
18/06/2026	10 :00 :57	06 :12	n	y
18/06/2026	10 :07 :24	06 :55	n	y

TABLE 1 – Résultats des différents lancements du robot avec l'ancien code.

Date	Heure	Durée	Bug (y/n)	Map penchée (y/n)
18/06/2026	10 :15 :58	07 :39	n	y
18/06/2026	10 :23 :58	02 :24	y	n
18/06/2026	10 :26 :43	05 :17	n	n
18/06/2026	10 :32 :28	05 :26	n	n
18/06/2026	10 :38 :09	05 :11	n	n
18/06/2026	10 :43 :36	05 :54	n	n
18/06/2026	10 :49 :46	05 :22	n	n
18/06/2026	10 :55 :23	04 :59	n	n
18/06/2026	11 :00 :37	05 :46	n	n
18/06/2026	11 :06 :41	07 :45	n	n
18/06/2026	11 :14 :41	06 :04	n	n

TABLE 2 – Résultats des différents lancements du robot avec le nouveau code.

La colonne 'Bug' indique si il y a eu un retour à la base avant d'avoir fini l'exploration et la colonne 'Map penchée' indique si la carte produite était alignée avec son mur de départ ou non.
On peut constater que les deux codes ont produit un test menant à une exploration non finie, cela peut s'expliquer à cause de la géométrie particulière de l'espace dans lequel ont été réalisés les tests qui amène à ce genre de problèmes.
Concernant l'orientation des cartes produites, toutes celles obtenues avec le code d'origine étaient penchées, et donc moins bonnes alors que sur onze tests, seulement une carte était penchée avec le nouveau code. Cette nette amélioration contribue à la qualité des résultats fournis à l'utilisateur.

Faisons maintenant la moyenne de durée nécessaire à l'exploration pour les deux codes, en enlevant les deux lancements finis abruptement qui pourraient fausser les résultats. On obtient pour l'ancien code une durée moyenne de 6 min 16 s, contre 5 min 56 s pour le nouveau code, soit une amélioration de 20 secondes par lancement, correspondant à une réduction de la durée moyenne d'environ 5,3 pourcent sur une surface de 30 m^2.
Ce résultat ne peut pas être étendu à d'autres surfaces car l'amélioration n'est pas supposée linéaire, il faudrait réaliser des tests pour cela, ce qui sera fait lorsque le nouveau code sera implémenté sur les robots mis à disposition chez des testeurs à Orange.

Concernant la reproductibilité, comme évoqué dans la partie 3.1, le nouveau code se montre nettement plus stable que le code d'origine dès les premiers points. Cette meilleure stabilité se retrouve également dans les points suivants. Toutefois, elle reste limitée par la précision du capteur LiDAR, qui introduit une part d'aléatoire dans les mesures et peut ainsi entraîner de légères variations d'un lancement à l'autre.

Le nouveau code se révèle également plus compréhensible pour l'utilisateur. Auparavant, les allers-retours ainsi que le choix de se positionner dans des zones encombrées d'obstacles suscitaient des interrogations chez les testeurs du robot. Ces derniers peinaient à saisir la logique du raisonnement et à adhérer aux choix effectués par le code, ce qui nuisait à leur perception de l'efficacité du processus de découverte. Le nouveau code adopte à présent un raisonnement plus intuitif et plus proche du comportement humain, un aspect essentiel pour les utilisateurs souhaitant un test rapide et cohérent. Le retour utilisateur devrait donc s'en trouver amélioré sur ce point.

En matière de sécurité, l'ajout d'une distance maximale par rapport à la base constitue une garantie minimale.

5 Deuxième partie du stage

L'intitulé de ce stage était : "cartographie de la maison à l'aide d'un robot et positionnement d'objets", nous avons vu la partie cartographie rapide, mais nous ne verrons pas la partie positionnement d'objets. En effet, le choix de travailler sur un autre projet a été évoqué et celui-ci a été retenu.
Ce sujet porte sur l'apprentissage d'une tâche à un bras téléopérable LeRobot SO-101.

Le diagramme de Gantt correspondant à cette deuxième partie de stage se trouve en annexe B

5.1 Description du sujet et état de l'existant

Dès le début de ce stage, Mr.Loïc Avenel le tuteur de ce stage avait évoqué le besoin de devoir monter des robots pour l'équipe lorsque ceux-ci arriveraient. Ces robots sont les Reachy Mini wireless et sont arrivés vers mi juillet. Comme convenu ces robots ont chacun été assemblés et pris en main pour la première fois par plusieurs personnes. Simultanément à cette tâche, Mr.Avenel a proposé de travailler sur les bras LeRobot SO-101, est alors venue l'idée d'associer le Reachy Mini et les bras SO-101.

FIGURE 53 – Reachy Mini wireless à gauche et paire de bras LeRobot SO-101 à droite.

En effet, le Reachy Mini est un robot compagnon open-source développé par Pollen Robotics [Pol25], en partenariat avec Hugging Face, conçu pour l'expérimentation en interaction homme-robot et en intelligence artificielle, il est équipé d'une tête et d'un buste motorisés, d'antennes animées pour l'expressivité, ainsi que de capacités de perception multimodale grâce à une caméra, des microphones et des haut-parleurs intégrés. De plus il est entièrement programmable via un SDK Python open-source.

Ce qui nous intéresse ici c'est son flux vidéo grand angle et sa capacité à orienter sa tête, nous allons voir pourquoi.

Le SO-101 est un bras robotique open-source et low-cost, développé par The Robot Studio en partenariat avec Hugging Face [Hug26], dans le cadre du projet LeRobot. Il s'agit d'un bras à 6 degrés de liberté, généralement fourni en paire : un bras "leader", manipulé à la main pour réaliser des démonstrations, et un bras "follower", qui reproduit ces mouvements en téléopération ou qui est piloté par un modèle entraîné lors de l'évaluation.

Le SO-101 est pensé pour l'apprentissage : il s'intègre directement à la bibliothèque open-source LeRobot, permettant de collecter des données de mouvement par téléopération et d'entraîner des modèles d'apprentissage par imitation exécutés directement sur le bras. LeRobot fournit ainsi les outils nécessaires à la collecte de données, à l'entraînement, au contrôle et à l'évaluation des politiques, avec un hub communautaire de datasets partagés.

Pour l'enregistrement de datasets et l'entraînement de modèles, le bras doit être équipé de caméras, plusieurs points de vue sont préconisés afin d'avoir une vue d'ensemble de la scène [You26b][You26a][You26c]. L'idée aurait alors été d'ultiliser le flux vidéo du Reachy Mini comme vue d'ensemble et d'ajouter une caméra dans la pince du bras
On dispose alors d'un Reachy Mini dont le flux vidéo est accessible et d'une paire de bras LeRobot SO-101.

Par la suite nous verrons ce qui a été réellement mis en place.

5.2 LeRobot SO-101

La documentation du SO-101 est très complète et la communauté active, il n'a donc pas été compliqué de trouver des exemples d'application du bras. La plus commune et d'ailleurs celle présentée comme exemple sur le site officiel est la suivante : un bol ou autre récipient et un cube coloré type lego sont placés devant le bras, il doit alors attraper le cube avec sa pince et le mettre dans le bol. Nous avons

décidé d'une autre tâche, plus complexe, qui est de saisir un jeton de puissance 4 et de le placer sur la grille.
Ci-après une photo de l'installation, on y place le bras follower (celui qui apprend), la grille de puissance 4 ainsi que des jetons, une caméra en face du jeu, une caméra sur la pince du bras, et une raspberry pi 500 équipée d'une clé USB 3.0. Nous expliquerons plus loin dans ce rapport cet équipement ainsi que leur placement.

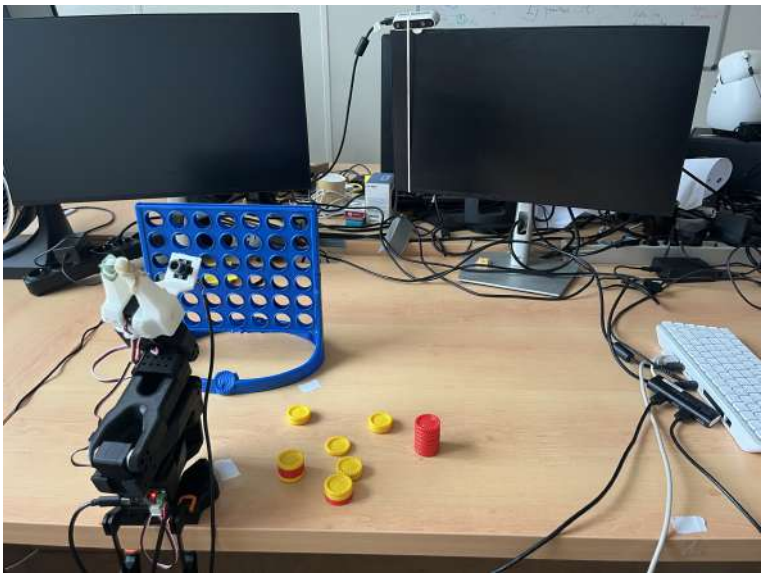

FIGURE 54 – Installation du projet.

Avec cette mise en place, on peut lancer les premières téléopérations, on peut donc manipuler le bras leader et le bras follower suit exactement chaque mouvement du premier. On a aussi accès au retour vidéo en direct sur un onglet rerun.io [Rer26].
On se lance alors dans les premiers essais de jeu de puissance 4.

5.2.1 Pince et préhension.

Le premier obstacle a été la préhension du jeton. La pince étant en impression 3D le contact avec les jetons de puissance 4 est glissant. La solution retenue, car rapide, a été d'ajouter des élastiques en caoutchouc autour des pinces, ce qui a rendu la tâche beaucoup plus simple.

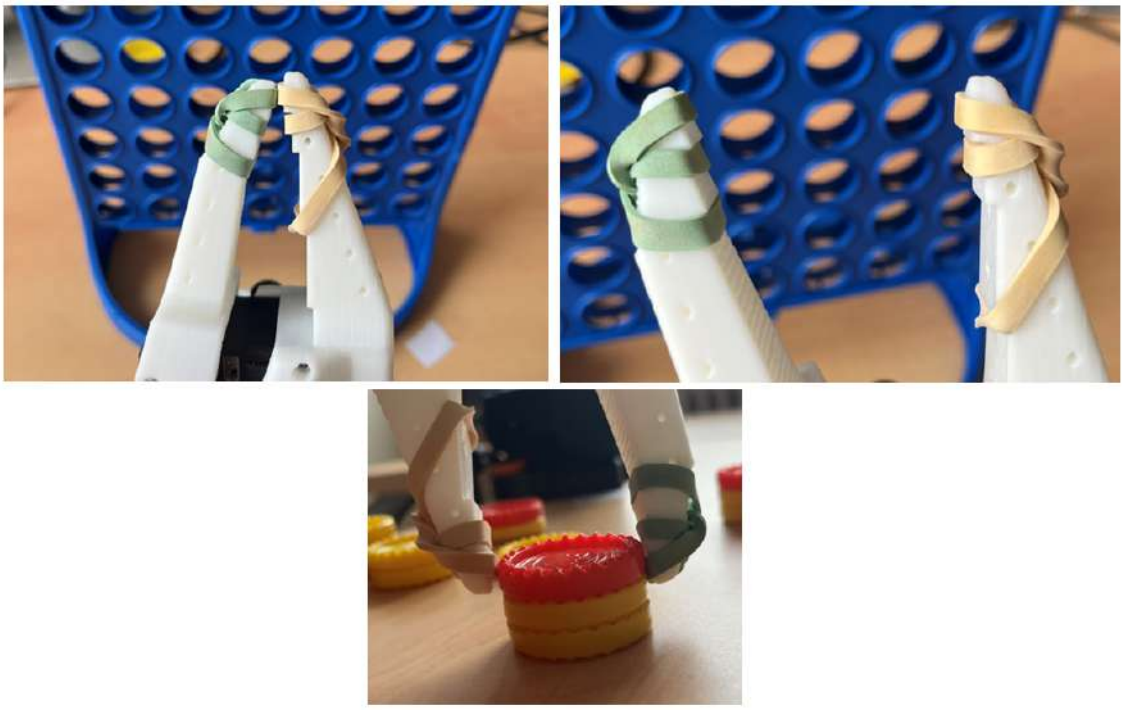

FIGURE 55 – Amélioration du contact pince jeton.

Cette solution n'est pas finale, quelque chose de plus définitif et durable sera installé dans le futur.

5.2.2 Caméras, flux vidéos et points de vue.

Il a ensuite fallu s'entraîner à téléopérer le bras leader pour réaliser la tâche avec le bras follower, afin que les enregistrements d'épisodes pour les datasets soient fluides. Pour ce faire il est recommandé de ne regarder que les retours caméra, car si un humain n'est pas capable de réaliser la tâche comme cela, alors le modèle n'y arrivera pas non plus. C'est à ce moment là que le flux du Reachy Mini a posé problème. Le flux du Reachy Mini était libéré et partagé via le réseau en protocole RTSP (Real-Time Streaming Protocol), or tout flux passant par le réseau est moins rapide qu'un flux direct comme celui avec la caméra placée sur la pince et connectée en USB sur la raspberry pi 500. Malgré les changements de propriété du flux (FPS (frames per second), dimensions), le retour caméra était trop lent et impossible de manipuler les bras dans ces conditions. La caméra du Reachy Mini et son grand angle ont alors été abandonnés et remplacés par une caméra RealSense D435if branchée directement sur la Pi.

Avec maintenant deux caméras et un retour presque instantané, l'étape d'après était de trouver le bon angle pour la caméra RealSense D435if. Dans les tutoriels de la communauté, la plupart ont soit une caméra en vue de dessus, soit une caméra en vue de face, soit les deux. Seulement dans notre cas, avec deux caméras dont une dans la pince, le robot doit être capable de voir la partie de Puissance 4, il a donc besoin d'un plan de face, mais aussi de situer la fente de la grille de jeu, donc d'avoir un plan de dessus ou de profil. La position finale de la caméra est donc légèrement en plongée et légèrement de profil. On peut voir les jetons déjà placés mais aussi situer l'emplacement et la profondeur de la fente du jeu.

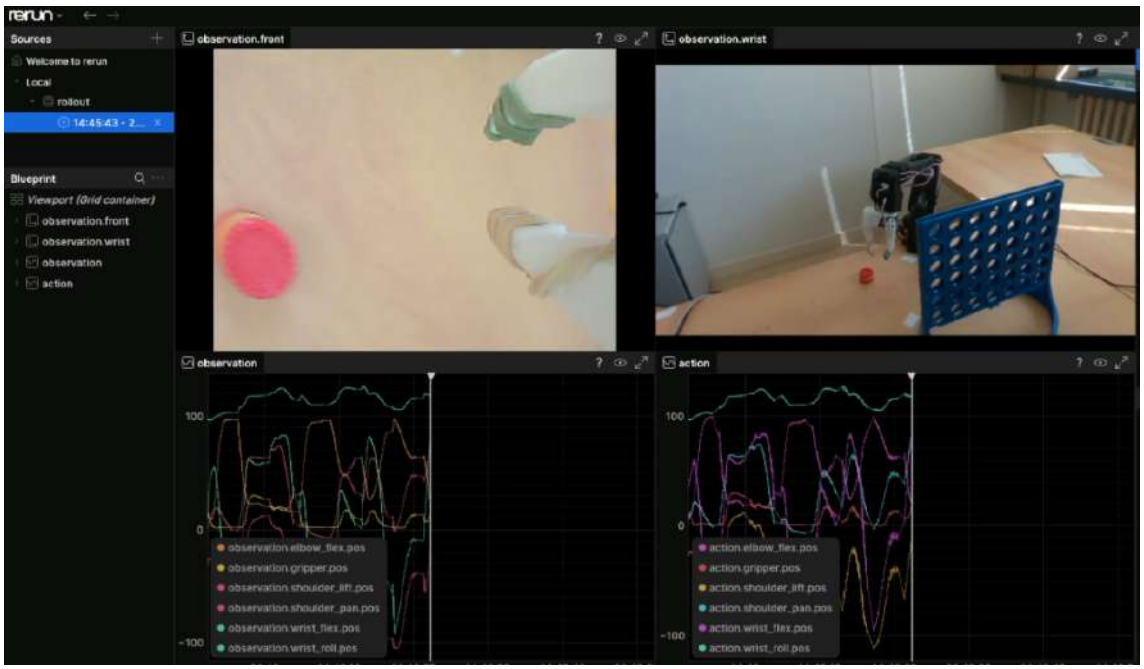

FIGURE 56 – Point de vue final des deux caméras en direct sur la fenêtre rerun.

Finalement, la position du bras et de la grille ont été marqués ainsi que les limites de la vue de la caméra RealSense D435if à droite du bras pour savoir où placer les jetons de jeu et qu'ils soient vus.

Nous passons donc maintenant à l'enregistrement de jeux de données.

5.2.3 Enregistrement de datasets.

Avant d'enregistrer un dataset, il faut définir les paramètres que l'on souhaite faire varier. Dans notre cas il y en a quatre :

— La couleur du jeton à saisir ;
— La hauteur du jeton à saisir ;
— La position du jeton à saisir ;
— La colonne dans laquelle placer le jeon une fois saisi.

Après différents essais, la couleur la plus visible sur le bureau était le rouge, la hauteur la plus facile à saisir était que le jeon ne soit pas posé seul sur la table, on met donc en place un tas de trois jetons posés dans cet ordre : rouge jaune rouge, pour imiter une partie, le tour de jouer étant au bras. La position du tas sur la table et la colonne dans laquelle placer le jeton sont deux paramètres amenés à varier au cours d'une partie normale. Pour simplifier la tâche de ce premier dataset, un seul de ces deux paramètres est défini comme pouvant varier, et c'est la position du jeton à saisir. Le tas de jetons peut donc se déplacer dans l'angle de vue de la caméra, mais le robot placera toujours celui-ci dans la même colonne.
Ce choix de fixer la colonne simplifie aussi l'enregistrement d'épisodes, car placer le jeton dans les colonnes les plus éloignées du centre réduisait le nombre d'épisodes réussis de 9/10 à 3/10. Cela est dû au fait que la grille est plane et que le robot tourne autour d'un axe, les extrémités ne sont donc pas alignées avec le bras. On fixe donc la colonne à celle du milieu de la grille de jeu

Le premier dataset enregistré était donc le suivant : à chaque début d'épisode le bras se déplie pour aller chercher un jeton sur la pile, il le prend et le place dans la colonne du milieu de la grille de jeu, puis revient à sa position de départ.

Après avoir enregistré 82 épisodes, nous passons à l'entraînement du premier modèle.

5.2.4 Entraînement du modèle.

Plusieurs solutions sont mises à dispositon par Hugging Face pour entraîner son modèle. La bibliothèque LeRobot propose un script dédié, LeRobot-train, prenant en entrée le jeu de données ainsi que le type de politique choisi (par exemple l'architecture ACT). Ce script s'adapte automatiquement à la configuration du robot utilisé, notamment au nombre de capteurs et de caméras présents dans le dataset, et permet de sélectionner le matériel de calcul souhaité, que ce soit un GPU Nvidia (CUDA) ou une puce Apple Silicon (MPS). Concernant les ressources de calcul, trois approches dsont disponibles : l'entraînement en local, lorsque la machine dispose d'un GPU suffisamment puissant ; l'utilisation de Google Colab, offrant un accès gratuit à un GPU distant via un notebook préconfiguré ; et enfin le recours à Hugging Face Jobs, un service de calcul cloud facturé à l'usage.
Dans un premier temps le notebook Google Colab a été utilisé, cela permettait l'accès à un GPU (non présent sur l'ordinateur sur lequel ce stage a été réalisé), et un entraînement gratuit.

Malheureusement cette solution a ses limites : la durée d'une session est plafonnée à 12 heures maximum et peut être interrompue plus tôt en cas d'inactivité, l'attribution d'un GPU n'est jamais garantie et dépend de la disponibilité du moment, et l'environnement de travail (fichiers, paquets installés) n'est pas persistant, étant réinitialisé à chaque déconnexion. Google ne publie par ailleurs aucun quota horaire fixe, ces limites variant dynamiquement selon la demande et l'historique d'utilisation du compte. À ces contraintes techniques s'ajoute un enjeu de confidentialité : les données et le code exécutés sur Colab transitent et sont traités sur l'infrastructure cloud de Google, sans garantie de confidentialité équivalente à un environnement local ou à une infrastructure interne maîtrisée, ce qui rend cette solution peu adaptée pour l'entraînement de modèles à partir de données sensibles ou propriétaires

Par la suite, nous n'utiliserons plus Google Colab mais une unité de calcul distante mise à disposition par Orange.

Le notebook fourni par Hugging Face, met en œuvre le protocole d'apprentissage par imitation de la politique ACT (Action Chunking Transformer), une architecture de type transformer conçue pour traiter conjointement les états du robot (positions articulaires issues des moteurs) et les flux d'images des caméras, afin de générer des séquences d'actions groupées ("chunks") plutôt que des actions isolées, ce qui améliore la fluidité des mouvements produits.
Le nombre de steps initial est fixé à 20 000, le batch size à 8 (adapté au GPU de Colab). Les datasets ainsi que les modèles entraînés seront tous enregistrés en privé sur HuggingFace.

5.3 Premiers résultats.

Le premier entraînement sur 82 épisodes donne lieu à un premier modèle qui est ensuite testé sur le bras. On reproduit alors la configuration des enregistrements et on observe.

Les premières conclusions sont les suivantes :
- Le bras n'attrape pas bien les jetons, il est soit à côté, soit sur le tas mais mal placé pour attraper donc il déplace le tas ou il le détruit en appuyant dessus ;
- Lorsque le bras réussit a attraper des jetons, il en attrape deux d'un coup ;
- Les jetons ne sont pas bien placés sur la grille, le bras force contre la grille et les jetons volent partout, on est obligé d'intervenir ;
- Le bras essaie d'attraper un jeton même quand il n'y a pas de tas, il se déplie et se déplace vers l'endroit où les tas peuvent être placés en permanence.

Cette liste de défauts nous amène à penser que : le dataset est mauvais, ou alors que la calibration du bras est mauvaise ou bien que le setup des caméras n'est pas assez bon. Nous essayons de trouver la cause réelle du mauvais modèle.

La première étape a été de compléter le dataset avec plus de cas différents. En effet, si le robot essaie d'attraper des jetons inexistants c'est parce qu'il n'a pas été entraîné à ne pas réagir lorsqu'il n'y en a pas. On en profitera pour lui apprendre à ne pas non plus réagir à des jetons jaunes.
On enregistre alors environ 25 épisodes où le bras ne bouge pas face à des tas de jetons jaunes, ou des tas de jetons rouge et jaune mais avec un jeton jaune sur le dessus, ou alors aucun tas.
On enregistre aussi des épisodes avec des hauteurs des tas différentes, car si le bras n'arrive pas à bien attraper les jetons c'est sans doute car il a appris sur une hauteur fixe, qui peut être mal apprise, et qu'il

essaie de reproduire cette position sans chercher à s'adapter à ce qu'il voit. On placera alors des tas de hauteurs différentes sur plusieurs enregistrements.
On enregistre aussi des épisodes avec des luminosités différentes, dans le cas où le premier modèle aurait appris avec la luminosité constante du premier dataset et que cela gênerait à la compréhension de la situation.

On passe donc de 82 à 142 épisodes sur le dataset, on décide aussi de l'entraîner cette fois sur 30 000 steps.

Le deuxième modèle entraîné, on le teste de nouveau sur le bras et, de nouveau, on observe.

Cette fois on voit :
— Le robot n'attrape pas bien les jetons, il y a un décalage, toujours le même, lorsqu'il s'approche ;
— Avec de l'aide, le bras peut attraper un jeton, ou deux ;
— Qu'il attrape un ou deux jetons, le jeton rouge est bien placé dans la grille et le bras ne force pas contre la structure ;
— Le bras ne fait plus rien lorsqu'il n'y a pas de jeton rouge placé.

Une nette amélioration est celle de la distinction entre le cas où le bras doit attraper un jeton et les cas où le bras ne doit rien faire. Une autre amélioration est le placement du jeton une fois attrapé, celui-ci rentre facilement dans la colonne du milieu sans chambouler le jeu.
Un problème persistant est celui du décalage lors de la préhension du jeton.

On décide de comparer les modèles pour différents cas. On trace alors le mouvement prédit des 6 moteurs pour les deux modèles, comparés aux déplacements réels.
L'épisode choisi ne fait pas partie des épisodes sur lesquels les modèles ont été entraînés.

Pour le cas où un tas de trois jetons tous rouges est placé à droite du bras, on obtient les courbes suivantes :

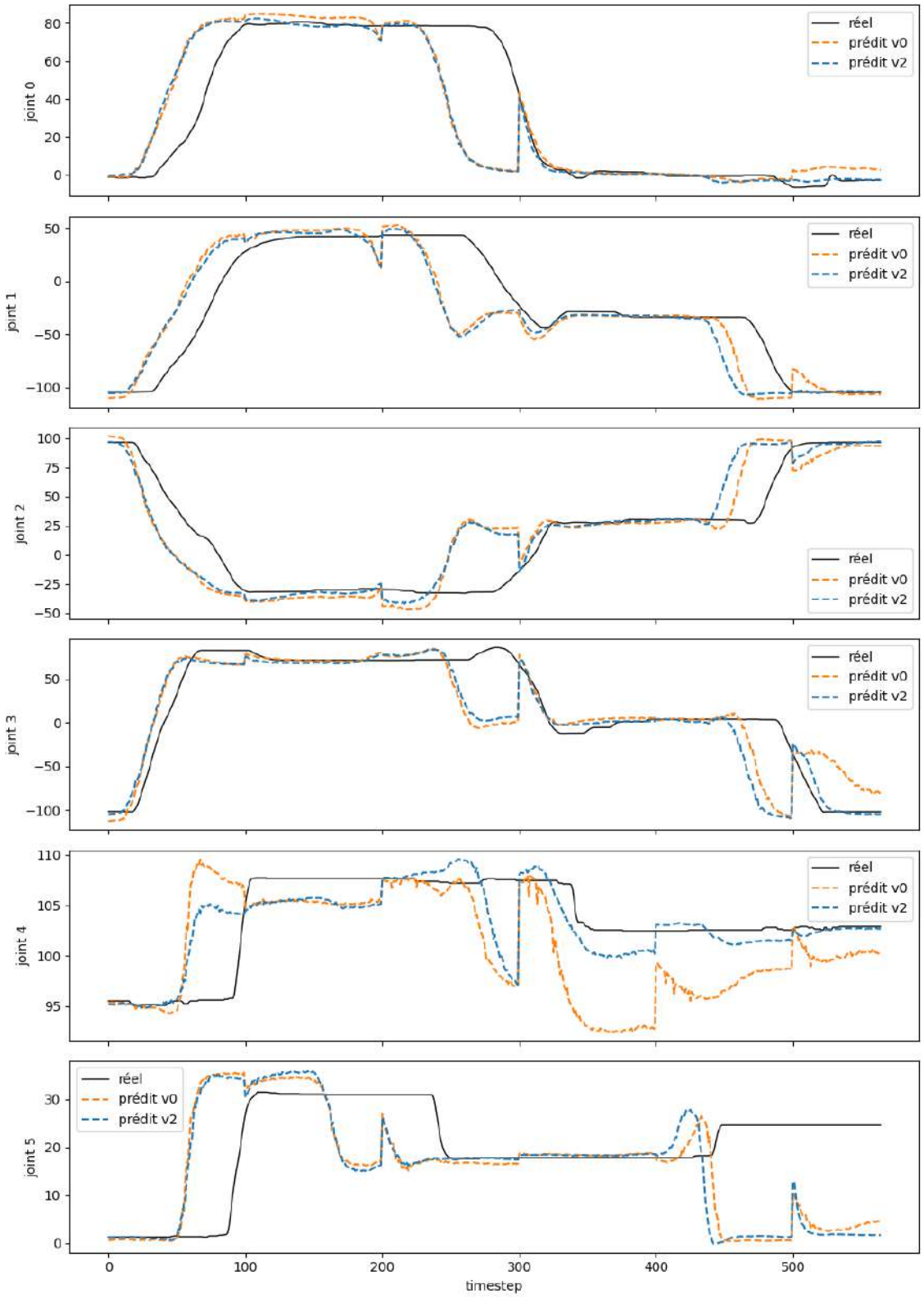

FIGURE 57 – Comparaison des modèles pour un épisode simple.

Chaque moteur a donc son déplacement lors de l'épisode, la prédiction du premier modèle est en orange (prédit v0) et la prédiction du deuxième modèle est en bleu (prédit v1). Les mouvements réels du robot sont en trait noir continu. On peut voir que les deux modèles ont des prédictions plutôt similaires. Le moteur 4 (joint 4), qui correspond au moteur qui fait tourner la pince sur elle même, est moins bien prédit que les cinq autres, même si le deuxième modèle le prédit un peu mieux sur la fin, et

enfin les deux modèles peinent à prédire les bons mouvements au début de l'épisode et ont des "sursauts".

Si on prend maintenant un épisode où le bras est censé ne rien faire ;

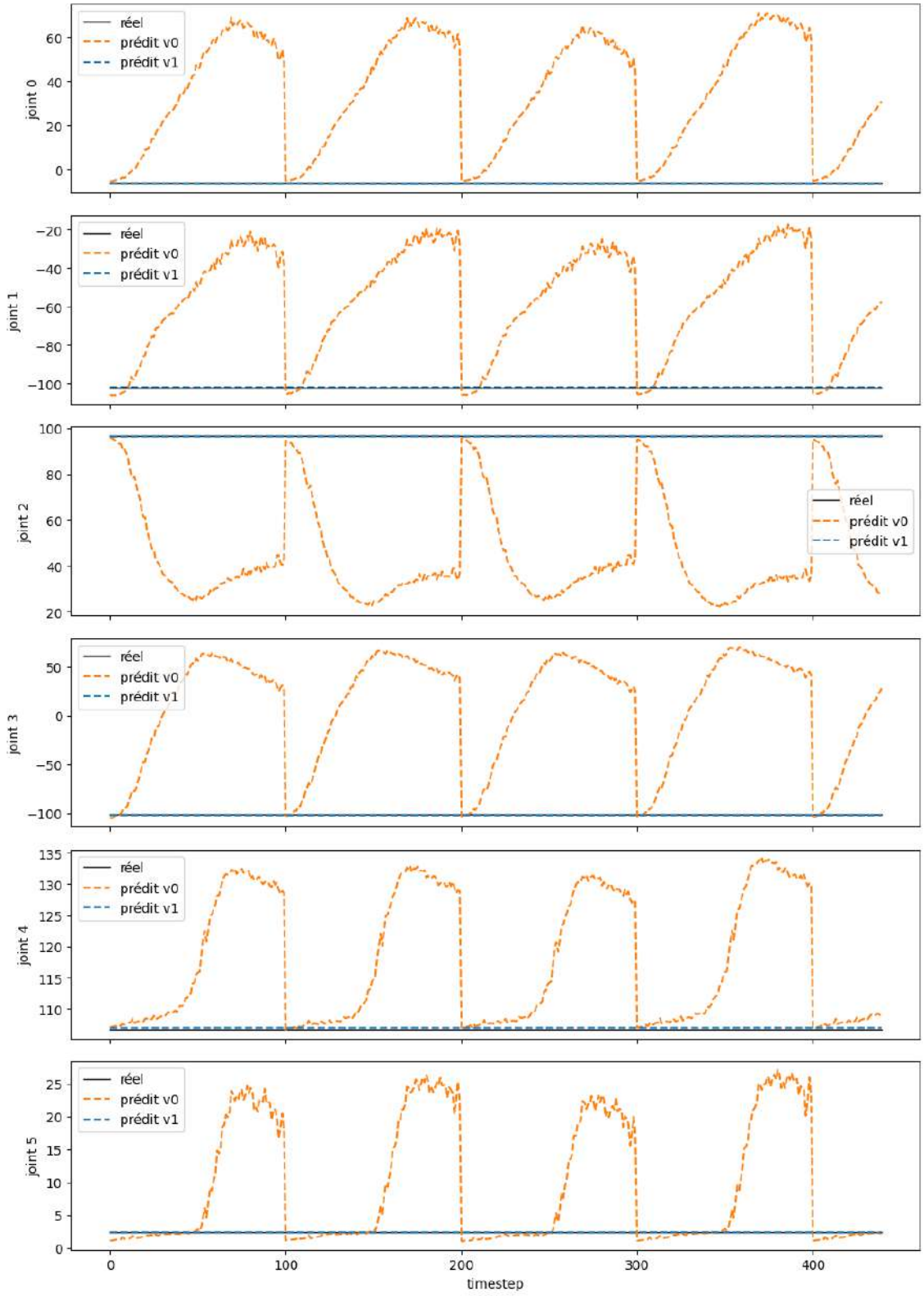

FIGURE 58 – Comparaison des modèles pour un épisode où le robot ne doit rien faire.

On peut clairement voir ici que le premier modèle tente en permanence de se déplacer, de rejoindre

une position où devrait se trouver le jeton alors qu'il n'y en a pas, le deuxième modèle comporte des petits sursauts mais à bien plus petite échelle que le premier.

On peut donc dire qu'il y a eu une amélioration entre le premier et le deuxième modèle, mais qu'il reste encore beaucoup de lacunes au dernier modèle, qui doivent être corrigées.

5.4 Futures améliorations

La suite de ce stage est tournée vers l'amélioration du modèle du bras LeRobot SO-101 afin qu'il puisse réaliser la tâche demandée.
Pour ce faire, les solutions à explorer sont :

- Entraîner le modèle sur plus de steps, cela règlera peut-être le problème du décalage ;
- Refaire la calibration des bras, afin de peut-être corriger le décalage observé lors de la préhension du jeton ;
- Compléter le dataset avec de nouveaux cas, notament plus de cas avec des hauteurs de piles différentes afin que le bras s'adapte plus facilement aux changements ;
- Compléter le dataset ou modifier les épisodes qui pourraient induire le décalage lors de la préhension. En effet les vidéos de la communauté montrent que parfois les biais humains sont appris et peuvent influencer le modèle, il est donc important de ne pas les laisser influencer le dataset.
- Apprendre au bras à placer les jetons aussi dans les autres colonnes, afin qu'il puisse réellement jouer au jeu ;
- Réaliser les mêmes tâches avec les jetons jaunes, pour que le bras puisse jouer les deux couleurs, une fonctionnalité qui donne le choix à l'utilisateur de la couleur qu'il veut jouer au lieu de la lui imposer.

Références

[Hua+23] Junlong Huang et al. “FAEL : Fast Autonomous Exploration for Large-scale Environments With a Mobile Robot”. In : *IEEE Robotics and Automation Letters* 8.3 (2023), p. 1667-1674. doi : 10.1109/LRA.2023.3236573.

[Hug26] Hugging Face. *SO-101*. https://huggingface.co/docs/lerobot/so101. Documentation LeRobot, consultée le 17 août 2026. 2026.

[Pol25] Pollen Robotics. *Reachy Mini*. https://pollen-robotics.com/reachy-mini/. Consulté le 17 août 2026. 2025.

[Rer26] Rerun. *LeRobot loader*. https://rerun.io/examples/integrations/lerobot_loader. Documentation Rerun, consultée le 17 août 2026. 2026.

[Val26] Valetudo. *Valetudo*. https://valetudo.cloud/. Documentation du projet, consultée le 17 août 2026. 2026.

[Wik26] Wikipedia contributors. *Travelling salesman problem*. 2026. url : https://en.wikipedia.org/wiki/Travelling_salesman_problem (visité le 17/08/2026).

[Yam98] Brian Yamauchi. “Frontier-Based Exploration Using Multiple Robots”. In : *Proceedings of the Second International Conference on Autonomous Agents (Agents ’98)*. Minneapolis, MN, USA : ACM, 1998, p. 47-53. doi : 10.1145/280765.280773.

[You26a] YouTube. *Running AI robotics experiments at home with LeRobot and SO-ARM100*. https://www.youtube.com/watch?v=DeBLc2D6bvg. Vidéo YouTube, consultée le 17 août 2026. 2026.

[You26b] YouTube. *This $150 Robot Arm Is The Best Way to Start With Advanced Robotics*. https://www.youtube.com/watch?v=59JTCvpG_Ec. Vidéo YouTube, consultée le 17 août 2026. 2026.

[You26c] YouTube. *Vidéo sur la robotique*. https://www.youtube.com/watch?v=-tkEMLOLEwo. Vidéo YouTube, consultée le 17 août 2026. 2026.

[Zha+25] Xu Zhang et al. “FAEM : Fast Autonomous Exploration for UAV in Large-Scale Unknown Environments Using LiDAR-Based Mapping”. In : *Drones* 9.6 (2025), p. 423. doi : 10.3390/drones9060423.

A Exemples d'un logigramme et d'un diagramme des codes d'Antoine Sauvenay

Voici un exemple d'un logigramme et d'un diagramme réalisés pour la compréhension des codes d'Antoine Sauvenay.

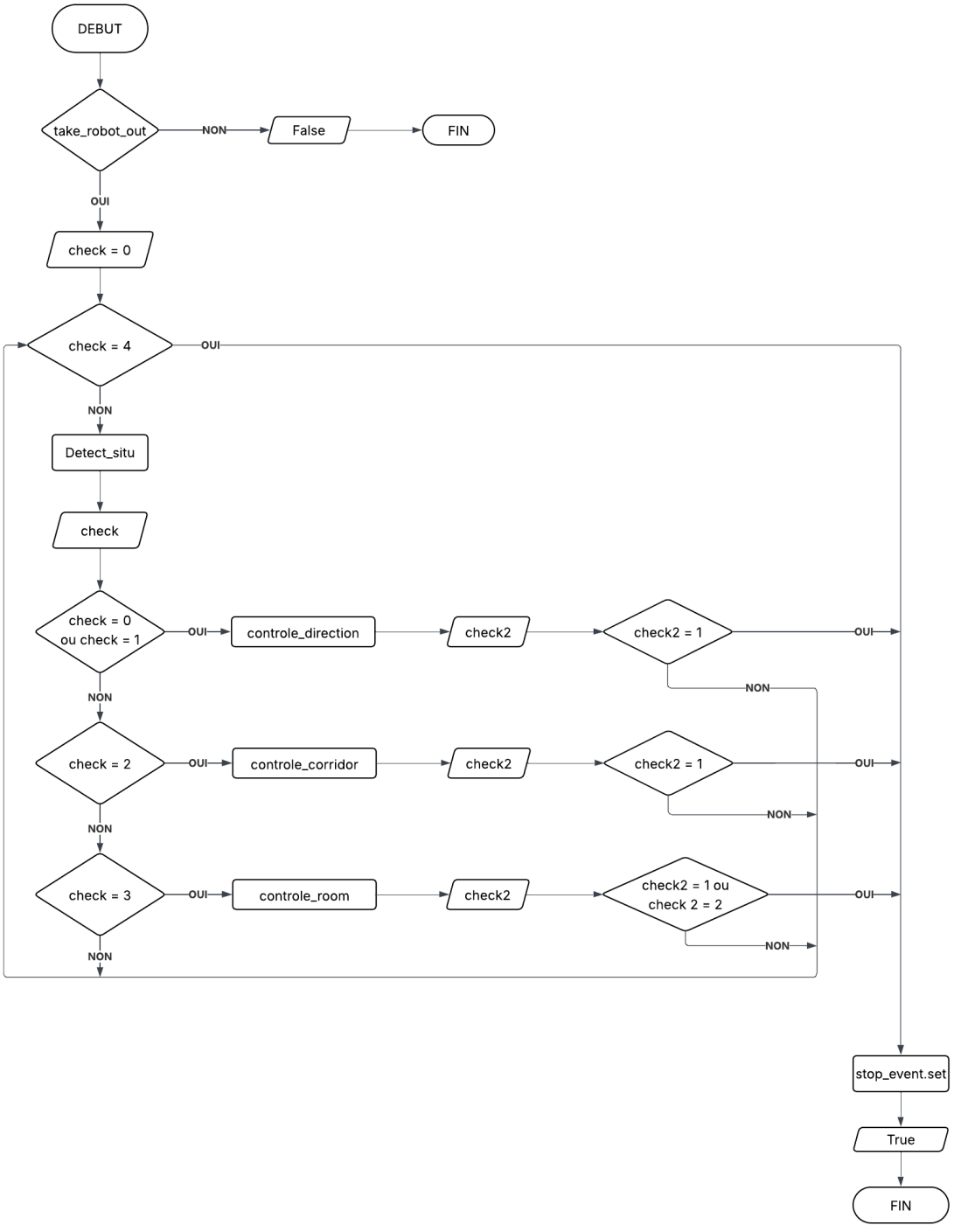

FIGURE 59 – Logigramme du code d'exploration.

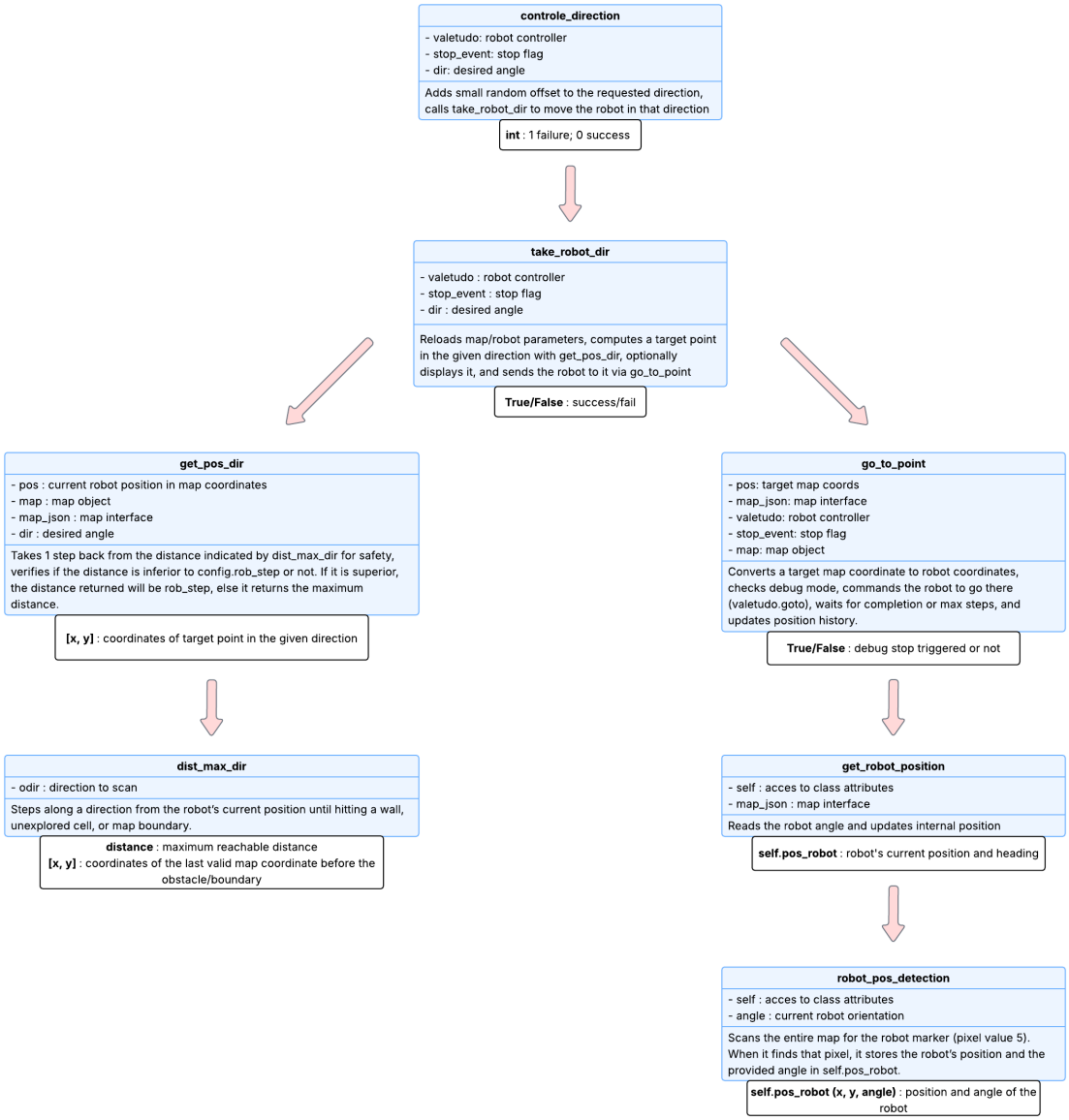

FIGURE 60 – Diagramme du code de direction.

B Diagramme de Gantt

Voici le diagramme de Gantt sur la période de la deuxième partie du stage. Les tâches qui ont du retard sont indiquées par une tâche du même nom mais avec "réel" ajouté et de couleur orange.

FIGURE 61 – Diagramme de Gantt deuxième période du stage.

Le reste des tâches reste à déterminer suite à l'évolution du troisième modèle.