REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
الجمهورية الجزائرية الديمقراطية الشعبية

MINISTRY OF HIGHER EDUCATION AND SCIENTIFIC RESEARCH

HIGHER SCHOOL IN APPLIED SCIENCES
--T L E M C E N--

وزارة التعليم العالي والبحث العلمي

المدرسة العليا في العلوم التطبيقية
-تلمسان-

المدرسة العليا في العلوم التطبيقية
École Supérieure en Sciences Appliquées

Mémoire de fin d'étude

Pour l'obtention du diplôme d'Ingénieur

Filière : Automatique
Spécialité : Automatique

Présenté par : BESSEGHIEUR Mohammed Hichem
ALLAK Mohamed Adel

Thème

Réalisation et commande d'un robot mobile à quatre roues basé sur ROS.

Soutenu publiquement, le 03 / 07 / 2024, devant le jury composé de :

M CHIALI Anisse	MCA	ESSA. Tlemcen	Président
M BESSEGHIEUR Khadir Lakhdar	MCB	EMP. Alger	Directeur de mémoire
M MEGNAFI Hicham	MCA	ESSA. Tlemcen	Directeur de mémoire
M ABDELLAOUI Ghouti	MCA	ESSA. Tlemcen	Examinateur 1
M MOKHTARI Rida	MCA	ESSA. Tlemcen	Examinateur 2
M KANOUN Ali	MRA	CDS. Oran	Partenaire socio-économique
M BOUZID Yasser	MCA	EMP. Alger	Invité 1

Année universitaire : 2023 /2024

Dédicaces

Je dédié ce travail

À ma chère mère, pour sa patience à m'élever depuis le jour de ma naissance jusqu'à aujourd'hui, pour ses sacrifices quotidiens, pour ses encouragements constants, pour tout. Ce travail est spécialement dédié à vous.

À mon cher père, je demande à Dieu Tout-Puissant de vous accorder une grande miséricorde et de nous réunir au Paradis, insha'Allah.

À mon cher frère, son soutien, ses encouragements et sa présence ont été une grande aide tout au long de ce parcours. Je te remercie sincèrement pour tout ce que tu as fait pour moi.

À mes chères sœurs, pour leur amour, leur soutien et leurs encouragements qui ont été une source inestimable de force pour moi. Que Dieu vous bénisse et vous protège toujours.

À tous les membres de ma famille, pour leur amour, leur soutien inconditionnel et leurs prières qui ont été déterminants pour ma réussite. Je vous suis infiniment reconnaissant pour tout ce que vous avez fait pour moi.

À mes amis, leur soutien, leur amitié et leurs encouragements ont été essentiels tout au long de ce parcours. Je vous suis profondément reconnaissant pour votre présence et votre aide constante.

À mon binôme, sa collaboration, son soutien moral, sa patience et sa compréhension tout au long de ce projet, ont été la clé pour la réussite de ce projet. Je te remercie pour ton engagement et ta camaraderie tout au long de cette aventure. Ce travail est dédié à toi, en signe de ma profonde gratitude et de notre accomplissement commun.

Merci d'être toujours là pour moi.

BESSEGHIEUR Mohammed Hichem

Dédicaces

Je dédié ce travail

À mes chers parents, pour tous leurs sacrifices, leur amour, leur tendresse, leur soutien et leurs prières tout au long de mes études, ce travail est spécialement dédié à vous.

À mes petites sœurs, ma grand-mère et ceux qui ont partagé avec moi tous les moments d'émotion lors de la réalisation de ce travail, ils m'ont chaleureusement soutenu et encouragé tout au long de mon parcours. Que Dieu continue de vous bénir et de veiller sur vous en tout temps.

À ma famille élargie, spécialement à mes oncles et leurs enfants, je vous suis reconnaissant pour votre soutien constant, vos encouragements et vos précieux conseils qui ont enrichi mon parcours académique.

À mon binôme, je te remercie infiniment pour ton engagement, ta camaraderie et ta patience tout au long de cette aventure académique. Ta collaboration précieuse a grandement enrichi ce projet.

À mes amis et camarades, pour leur amitié sincère et leur soutien inestimable.

ALLAK Mohamed Adel

Remerciements

Nous exprimons notre profonde gratitude à Dieu Tout-Puissant pour nous avoir guidés dans la réalisation de ce modeste travail.

Nous exprimons notre reconnaissance envers nos familles, particulièrement nos parents, pour leur soutien inconditionnel et leur encouragement tout au long de mes études.

Nous remercions, sincèrement notre encadrant **M. BESSEGHIEUR Lakhdar Khadir** pour son encadrement précieux, ses conseils éclairés et son soutien constant tout au long de ce projet. Ses orientations ont été essentielles pour surmonter les défis techniques rencontrés.

Nous exprimons également nos remerciements à notre encadrant **M. MEGNAFI Hicham** pour son soutien constructif et son engagement dans la réussite de ce travail.

Nous sommes profondément honorés par la participation de **M. CHIALI Anisse**, qui a présidé notre mémoire et a honoré notre jury de sa présence.

Nous souhaitons exprimer nos sincères remerciements à **M. ABDELLAOUI Ghouti** et à **M. MOKHTARI Rida** pour avoir accepté d'examiner ce mémoire et pour leurs observations pertinentes et constructives.

Nous exprimons notre profonde gratitude envers **M. ADJIM Ramz-eddine Abderrezak**, ingénieur de FABLAB à l'École Supérieure en Sciences Appliquées de Tlemcen, pour sa contribution précieuse dans la réalisation de notre travail.

Nous tenons également à remercier chaleureusement tous nos amis ainsi que toutes les personnes qui ont contribué de près ou de loin à la réalisation de ce travail. Leur soutien et leur collaboration sont grandement appréciés.

ملخص

يركز مشروع دراستنا النهائية في المدرسة العليا للعلوم التطبيقية بتلمسان على تطوير نظام رسم الخرائط لروبوت متنقل رباعي العجلات، باستخدام تقنية روس و ليدار. يثري هذا المشروع الأعمال السابقة من خلال دمج وظائف التنقل المتقدمة ورسم الخرائط وتحديد الموقع. تستكشف الأطروحة المفاهيم النظرية الأساسية للروبوتات المتنقلة وأنظمة الملاحة المستقلة، بالإضافة إلى تصميم وتنفيذ بنية الأجهزة والبرامج الخاصة بالروبوت. نقوم أيضًا بتفصيل تنفيذ استراتيجيات التنقل، بما في ذلك النمذجة الحركية و سلام و خوارزميات التحكم.

إن التقييم الشامل لأداء النظام في بيئات العالم الحقيقي يسلط الضوء على قدرته الفعالة على رسم خرائط للبيئات المعقدة. وتظهر النتائج التي تم الحصول عليها متانة النظام في ظل هذه الظروف. في الختام، نقوم بتقييم الأهداف التي تم تحقيقها والأداء العام للنظام المطور، وبالتالي المساهمة بشكل كبير في التقدم في مجال الروبوتات المتنقلة ورسم الخرائط المستقلة.

Abstract

Our final study project at the Higher School of Applied Sciences of Tlemcen focuses on the development of a mapping system for a four-wheeled mobile robot, using ROS and LiDAR technology. This project enriches previous work by integrating advanced navigation, mapping and localization functionalities. The dissertation explores the fundamental theoretical concepts of mobile robotics and autonomous navigation systems, as well as the design and implementation of the robot's hardware and software architecture. We also detail the implementation of navigation strategies, including kinematic modeling, control algorithms and SLAM.

A thorough evaluation of the system's performance in real-world environments highlights its effective ability to map complex environments. The results obtained demonstrate the robustness of the system under these conditions. In conclusion, we evaluate the objectives achieved and the overall performance of the developed system, thus contributing significantly to advances in mobile robotics and autonomous mapping.

Résumé

Notre projet de fin d'études à l'École Supérieure en Sciences Appliquées de Tlemcen se concentre sur le développement d'un système de cartographie pour un robot mobile à quatre roues, utilisant ROS et la technologie LiDAR. Ce projet enrichit des travaux précédents en intégrant des fonctionnalités avancées de navigation, de cartographie et de localisation. Le mémoire explore les concepts théoriques fondamentaux de la robotique mobile et des systèmes de navigation autonome, ainsi que la conception et l'implémentation de l'architecture matérielle et logicielle du robot. Nous détaillons également la mise en œuvre des stratégies de navigation, incluant la modélisation cinématique, les algorithmes de contrôle et le SLAM.

Une évaluation approfondie des performances du système dans des environnements réels met en lumière sa capacité efficace à cartographier des environnements complexes. Les résultats obtenus démontrent la robustesse du système dans ces conditions. En conclusion, nous évaluons les objectifs atteints et la performance globale du système développé, contribuant ainsi significativement aux avancées en robotique mobile et en cartographie autonome.

TABLE DES MATIÈRES

LISTE DES ABRÉVIATIONS

AMCL Adaptive Monte Carlo Localization.
AUV Autonomous Underwater Vehicle.
CNC Computer Numerical Control.
DWA Dynamic Window Approach.
GPS Global Positioning System.
Hz Hertz (used for frequency).
IMU Inertial Measurment Unit.
KF Kalman Filter.
LDS Laser Distance Sensor.
LTS Long Term Support.
LiDAR Light Detection And Ranging.
LiPo Lithium-Polymer.
ODE Open Dynamics Engine.
PID Proportional-Integral-Derivative.
PWM Pulse Width Modulation.
ROS Robot Operating System.
RViz Ros Visualization.
SLAM Simultaneous Localization And Mapping.
SSH Secure Socket Shell.
UAV Unmanned Aerial Vehicle.
UGV Unmanned Ground Vehicle.
USB Universal Serial Bus.
VNC Virtual Network Computing.
WiFi Wireless Fidelity.

NOMENCLATURES

λ	Longueur d'onde
ω	Vitesse angulaire
D	Distance
g	Fonction de transition d'état
GND	Ground
K_d	Coefficient de dérivé du correcteur
K_i	Coefficient d'intégrateur du correcteur
K_p	Coefficient de proportionnel du correcteur
L	Distance entre les roues
m	a carte de grille d'occupation
N	Nombre de tours
n_eff	Nombre effectif de particules
T	Période de temps
t	Temps
$u_{0:t}$	Mesures d'odométrie
v_d	Vitesse linéaire du coté droit
v_g	Vitesse linéaire du coté gauche
VCC	Alimentation positive
$x_{1:t}$	Trajectoires du robot
x_t^k	Etat de la particule
$z_{1:t}$	Observations du LiDAR
v	Vitesse de translation

TABLE DES FIGURES

LISTE DES TABLEAUX

INTRODUCTION GÉNÉRALE

La robotique est désormais un secteur essentiel, jouant un rôle important dans l'automatisation et l'innovation dans divers domaines. Les robots mobiles, en particulier, ont révolutionné de nombreuses industries grâce à leur capacité à se déplacer, à interagir avec leur environnement, à améliorer l'efficacité et à réduire les coûts de main-d'oeuvre. Dans notre projet, nous avons opté pour les robots à quatre roues à cause de leurs capacité à offrir une combinaison optimale de stabilité, de capacité de charge, d'adaptabilité à divers environnements et de facilité de contrôle. Ces caractéristiques nous permettent de répondre efficacement aux exigences spécifiques de notre projet, tout en assurant une performance fiable et une productivité accrue dans les applications envisagées.

La cartographie des environnements et la navigation autonome des robots posent un ensemble complexe de défis techniques et technologiques. La problématique centrale de ce projet est comment peut-on concevoir, réaliser et contrôler un robot mobile à quatre roues capable de construire et d'utiliser des cartes précises de son environnement pour naviguer de manière autonome. Ce défi englobe le développement d'une architecture matérielle et logicielle qui repose sur une combinaison de capteurs, d'algorithmes et de systèmes mécaniques afin d'assurer la commande du robot ainsi que l'implémentation d'un système de navigation basé sur ROS et la technologie LiDAR pour la cartographie permettant au robot de se localiser, planifier des trajectoires et de se déplacer de manière autonome dans son environnement.

Notre projet consiste principalement sur la réalisation et la commande d'un robot mobile à quatre roues capable de cartographier son environnement et de se déplacer de manière autonome. Tout d'abord, la première étape du projet est de réaliser la structure mécanique du robot mobile en abordant la conception puis la réalisation de chaque pièce constituant le robot. Pour la mise en marche du robot, nous proposons une architecture électronique basée sur trois calculateurs qui communiquent à travers un framewrok développé qui se base sur le système ROS. Après l'assemblage de l'architecture mécanique et électronique du robot, nous attaquons le développement et l'implémentation des stratégies de controle bas-niveau et haut-niveau pour notre robot. La première a pour tache d'assurer l'asservissement en vitesse du robot, tandis que la dernière se concentre sur l'implémentation des algorithmes de navigation autonome permettant au robot d'assurer des taches telles que : la localisation, cartographie, planification des chemins et exécution des chemins planifiés.

Afin de bien mener la présentation du travail réalisé dans ce projet, le présent mémoire est répartit sur quatre chapitres dont trois portant sur l'aspect implémentation pratique. Pour une exploration approfondie du sujet, la structure du manuscrit est organisée comme suit :

Le premier chapitre pose les bases théoriques nécessaires à la compréhension de la robotique mobile et des systèmes de navigation autonome. Il définit les applications variées de la robotique mobile, présente ses défis ainsi que les différents types des robots mobiles en détaillant les robots mobiles à quatre roues. En plus, ce chapitre détaille les concepts liés à la navigation autonome des robots, y compris les architectures de contrôle, les types de navigation ainsi que les taches principales du robot pour assurer une navigation autonome. Ce chapitre met en lumière les algorithmes de cartographie, de planification des chemins ainsi que de la navigation autonome.

Le deuxième chapitre présente la conception et la réalisation d'un robot mobile à quatre roues, en se concentrant sur quatre aspects principaux : la modélisation cinématique, la structure mécanique, l'architecture électronique et l'architecture logicielle. La modélisation cinématique, fondamentale pour un contrôle précis du robot, établit les équations de mouvement. La conception et la réalisation mécanique détaille le processus de construction du robot en utilisant des outils comme SolidWorks et des techniques avancées telles que le découpage CNC et l'impression 3D. L'architecture électronique est divisée en deux segments : haut-niveau et bas-niveau. Le premier gère les algorithmes de navigation et de guidage, tandis que le segment bas-niveau exécute les commandes et supervise les moteurs. Enfin, l'architecture logicielle, utilisant des outils tels qu'Ubuntu et ROS, assure le contrôle et la communication au sein du robot. Cette approche intégrée illustre la synergie entre la mécanique, l'électronique et le logiciel pour créer un robot fonctionnel et performant.

Le troisième chapitre explore en profondeur le développement d'une architecture logicielle basée sur ROS (Robot Operating System). Nous commençons par examiner les fondements de ROS, ses avantages, ses outils de visualisation, ainsi que ses principales applications dans la création de robots autonomes. Le chapitre aborde ensuite la mise en place de la communication entre les différents composants du système. Nous détaillons la commande bas-niveau du robot, en commençant par l'asservissement en vitesse d'un seul moteur à l'aide d'un contrôleur PID, puis en étendant ce contrôle à l'ensemble du robot. Une commande manuelle est également développée pour tester l'exactitude de l'asservissement en vitesse et le bon fonctionnement des liaisons de communication. Enfin, nous présentons l'architecture de commande haut-niveau basée sur ROS. Cela inclut l'implémentation de la cartographie de l'environnement à l'aide de l'algorithme SLAM , l'utilisation de l'algorithme AMCL pour la localisation précise du robot, et la planification des chemins pour permettre une navigation autonome.

Le dernier chapitre se concentre sur la présentation et l'analyse des résultats obtenus à chaque étape de notre projet. Nous commençons par simuler notre robot dans Gazebo pour observer son comportement dans des environnements virtuels. Ensuite, nous évaluons la précision et la fiabilité du système odométrique dans diverses conditions pour garantir son bon fonctionnement pendant la navigation. Par la suite, nous abordons les résultats de la cartographie de l'environnement en évaluant l'implémentation du SLAM sous ROS. Nous analysons ensuite la performance de la localisation du robot sur la carte générée. Enfin, nous présentons les résultats relatifs à la navigation autonome, mettant en lumière l'efficacité de notre système pour accomplir des tâches de navigation vers des objectifs spécifiques.

CHAPITRE

1

GÉNÉRALITÉS SUR LA ROBOTIQUE MOBILE

1.1 Introduction :

Ce premier chapitre, intitulé ”Généralités sur la robotique mobile”, vise à fournir une compréhension globale de ce domaine en présentant ses aspects fondamentaux et ses défis. Nous allons établir les bases nécessaires pour comprendre les développements ultérieurs et les applications spécifiques de la robotique mobile, offrant ainsi une vue d'ensemble essentielle pour le reste de ce mémoire.

Nous commencerons par définir la robotique mobile, en explorant ses différents types et les applications variées où elle est mise en oeuvre, ainsi que les défis technologiques et opérationnels qu'elle présente. Ensuite, nous nous pencherons plus spécifiquement sur les robots mobiles à quatre roues qui constituent le type de robot réalisé dans notre travail, en examinant leurs avantages et les défis associés à leur utilisation. Enfin, nous aborderons les principes généraux de la navigation autonome des robots mobiles, en soulignant les techniques et technologies utilisées pour permettre aux robots de se déplacer de manière efficace et sécurisée dans des environnements complexes.

1.2 Définition du robot

Le terme robot provient du mot ”robota” qui signifie en tchèque “travail”, “labeur”. La racine rob signifie ”esclave”. Le terme ”robota” trouve son origine dans une pièce de théâtre tchèque datant des années 1920. Le robot représente ainsi une machine qui assiste ou prend la place de l'être humain (ou même des animaux) dans des activités complexes et difficiles [42].

Donc un robot mobile désigne un dispositif intégrant des composantes mécaniques, électroniques et informatiques, qui interagit physiquement avec son environnement afin

d'accomplir une tâche spécifique qui lui est attribuée. Cette technologie polyvalente peut ajuster son fonctionnement en réponse à des variations dans son environnement opérationnel [26].

Les robots se divisent généralement en deux grandes catégories : robots mobiles et robots fixes. La principale différence entre les robots fixes et mobiles réside dans leur capacité de mouvement et la nature des tâches pour lesquelles ils sont conçus. Les robots fixes sont destinés à des tâches stationnaires et répétitives nécessitant une grande précision, tandis que les robots mobiles sont conçus pour se déplacer et interagir avec des environnements dynamiques. Nous nous intéressons dans ce travail aux robots mobiles grâce à leur multiples avantages et applications que nous allons exposés dans la section qui suit.

1.3 Applications de la robotique mobile

Le domaine de la robotique mobile est en développement continu et très rapide. Les robots trouvent des applications dans une multitude de domaines, offrant des solutions innovantes et efficaces. Voici quelques-uns des principaux domaines d'applications des robots :

1.3.1 Surveillance et sécurité militaire

Les robots mobiles présentés sur la Figure 1.1 peuvent patrouiller dans des zones sensibles, telles que les frontières ou les installations militaires, pour détecter les intrusions, surveiller les activités suspectes, renforcer la sécurité, le déminage, et même dans des opérations de combat réduisant ainsi les risques pour les soldats.

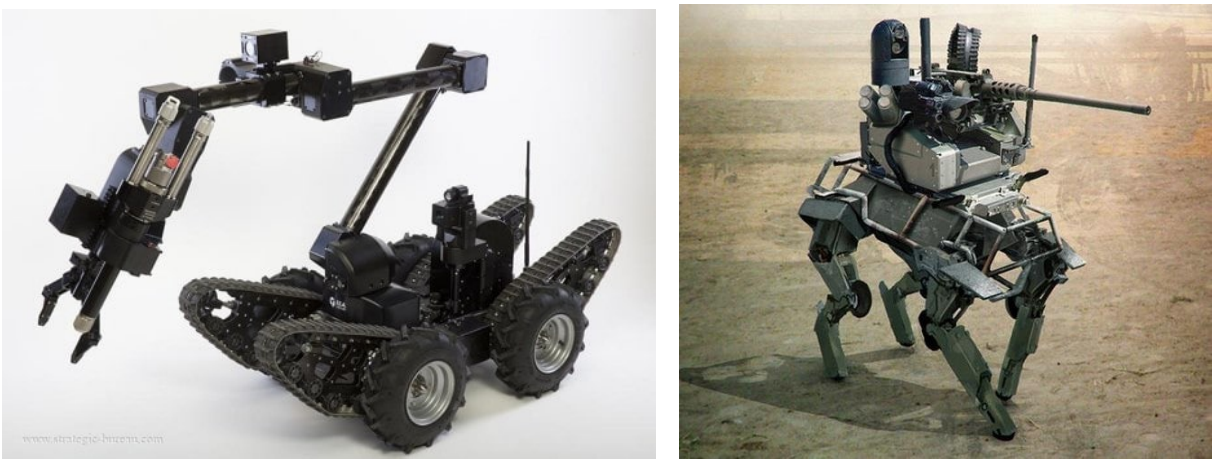


FIG. 1.1 – Robots militaires

1.3.2 Industrie

Les robots mobiles industriels sont des équipements automatisés conçus pour déplacer des charges au sein des installations de fabrication et de logistique. Equipés de systèmes de sécurité avancés, ces machines peuvent fonctionner en proximité des travailleurs. Ce type de robot trouve ses racines dans les chariots filoguidés largement utilisés dans les années 1980. Cependant, avec les progrès dans les technologies de navigation, de capteurs et

de contrôle, les robots mobiles industriels sont devenus des outils polyvalents dans divers secteurs industriels. Ils sont utilisés pour le transport de matériaux, le chargement et le déchargement des machines, pour des tâches d'assemblage et de montage, la soudure, la peinture et le contrôle qualité [19] comme le montre la figures 1.2.

FIG. 1.2 – Robots industriels de la marque KUKA [10]

1.3.3 Logistique et entreposage automatisé

L'intégration croissante des robots mobiles dans le domaine de la logistique constitue une avancée significative. Ces systèmes robotisés offrent une solution efficace pour exécuter diverses tâches logistiques avec précision et rapidité. Grâce à leur capacité à se déplacer de manière autonome et à s'adapter à leur environnement, ils peuvent optimiser les processus de manipulation, de stockage et de déplacement des marchandises dans les entrepôts. De plus, l'utilisation de robots mobiles permet de réduire les coûts opérationnels et d'améliorer la productivité en minimisant les temps d'arrêt. Voici un exemple d'un robot de déchargement dans la figure 1.3

FIG. 1.3 – Robot de déchargement [11]

1.3.4 Commerce

Dans le domaine du commerce, les robots mobiles sont de plus en plus utilisés pour assister les clients, effectuer des livraisons et gérer les stocks. Un exemple emblématique est celui d'Amazon sur la figure 1.4, où les stocks sont entièrement pris en charge par des robots mobiles [26].

FIG. 1.4 – Robot d'AMAZON

1.3.5 Agriculture de précision

La robotique agricole offre de nombreuses solutions aux agriculteurs en automatisant des tâches fastidieuses ou répétitives avec précision, et en intervenant dans des zones difficiles, comme les vignobles en forte pente, sans mettre les individus en danger. Elle est utilisée dans diverses opérations telles que le semis, le désherbage, la pulvérisation de pesticides (la figure 1.5), la surveillance des produits agricoles (la figure 1.6) et la récolte (la figure 1.7), ce qui contribue à augmenter l'efficacité et la productivité agricoles [15]. la récolte (la figure 1.7)

FIG. 1.5 – Robot de pulvérisation de pesticides

FIG. 1.6 – Surveillance des produits agricoles

FIG. 1.7 – Robot de récolte

1.3.6 Assistance médicale et de réhabilitation

La robotique médicale présente divers défis et opportunités dans le domaine de la santé. L'introduction des robots à l'hôpital a considérablement élargi le champ des possibilités pour l'amélioration de la santé humaine. Une gamme variée de robots médicaux est désormais utilisée, allant des robots d'assistance chirurgicale (la figure 1.8) qui sont couramment employés depuis plus d'une décennie, aux capsules endoscopiques miniaturisées récemment développées pour explorer le corps humain. En outre, les prothèses de membres, les systèmes robotiques pour la rééducation et l'assistance (la figure 1.9), les exosquelettes, les chaises roulantes robotisées et même les robots interactifs sociaux sont également devenus des domaines d'intérêt croissant. Ces avancées ouvrent de nouvelles perspectives pour le diagnostic, la thérapie et la réhabilitation dans des domaines variés, tels que la gestion de l'autisme et la stimulation cognitive chez les personnes âgées. [32]

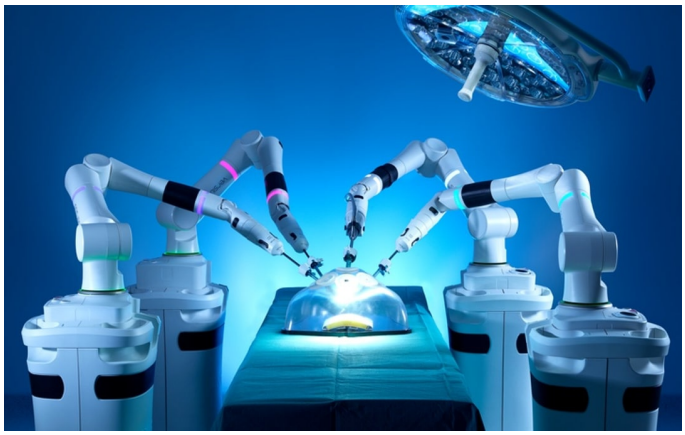


FIG. 1.8 – Robot chirurgien

FIG. 1.9 – Robot de réhabilitation

1.3.7 Livraison autonome

Les robots mobiles sont utilisés pour la livraison de colis et de produits dans des environnements urbains et péri-urbains (figures 1.10 et 1.11), offrant une solution de livraison rapide et efficace tout en réduisant les coûts et les émissions de carbone [44].

FIG. 1.10 – Robot de livraison autonome (AMAZON)

FIG. 1.11 – Robot de livraison bipède (Humanoïde)

1.3.8 Nettoyage et entretien

Les robots mobiles sont utilisés pour nettoyer et entretenir des espaces intérieurs et extérieurs, tels que les bureaux, les entrepôts, les centres commerciaux et les espaces publics, en effectuant des tâches de balayage et de lavage comme illustré dans la figure 1.12.

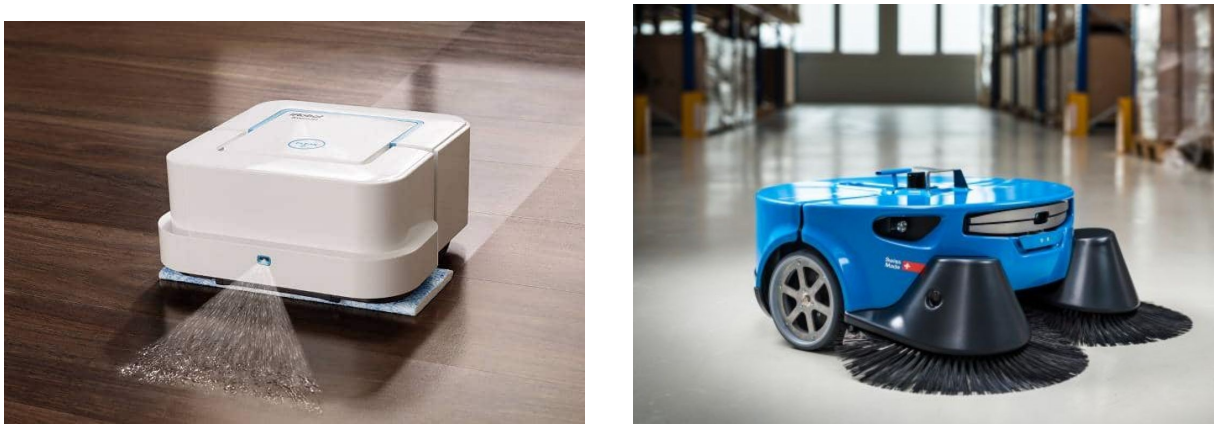


FIG. 1.12 – Robots de nettoyage et entretien

1.3.9 Éducation et divertissement

Les robots mobiles sont déployés dans les salles de classe, les amphithéâtres (comme illustré dans la figure 1.13), les musées et les parcs d'attractions pour offrir des expériences éducatives et de divertissement interactives (comme illustré dans la figure 1.14). Ils enseignent des concepts scientifiques et technologiques tout en proposant des spectacles et des jeux interactifs, enrichissant ainsi l'apprentissage et le divertissement des utilisateurs.

FIG. 1.13 – Robot d'éducation

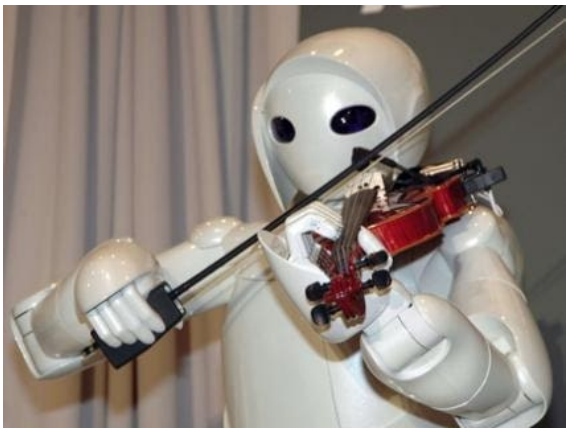


FIG. 1.14 – Robot de divertissement

1.4 Les défis de la robotique

Bien que le domaine de la robotique progresse rapidement et fasse preuve d'innovations constantes, il continue de rencontrer plusieurs défis significatifs :

— **Navigation autonome :** La capacité à naviguer de manière autonome dans des environnements complexes et dynamiques, en évitant les obstacles et en planifiant des trajectoires optimales, reste l'un des défis majeurs de la robotique mobile.

— **Perception de l'environnement :** Les robots mobiles doivent être capables de percevoir leur environnement de manière précise et fiable à l'aide de capteurs tels que des caméras, des LiDARs et des radars, afin de prendre des décisions éclairées et de s'adapter aux changements dans leur environnement.

— **Robustesse et fiabilité :** Les robots mobiles doivent être capables de fonctionner de manière fiable dans des conditions variées et parfois hostiles, telles que des températures extrêmes, des terrains accidentés ou des environnements bruyants.

— **Gestion de l'énergie :** L'autonomie énergétique reste un défi majeur pour les robots mobiles, en particulier pour ceux qui opèrent sur de longues périodes ou dans des environnements éloignés où l'accès à une source d'alimentation peut être limité.

— **Coopération et coordination :** Dans de nombreux scénarios, les robots mobiles doivent être capables de travailler en collaboration les uns avec les autres pour accomplir des tâches complexes. La coordination efficace entre les robots pose des défis en termes de planification, de communication et de gestion des conflits.

En surmontant ces défis, la robotique mobile peut réaliser son potentiel dans une variété d'applications, allant de la logistique et de la fabrication à l'exploration spatiale et à l'assistance à la personne.

1.5 Types des robots mobiles

Dans le domaine de la robotique, il existe trois grandes catégories des robots mobiles, notamment : les robots mobiles aériens (UAV), robots mobiles marins (AUV) et robots mobiles terrestres (UGV). Dans ce qui suit, nous allons présenté brièvement chaque type de ces robots en mettant l'accent sur les robots mobiles terrestres qui constituent le vif de notre travail dans ce projet.

1.5.1 UAV (Unmanned Aerial Vehicle)

Les UAV, également connus sous le nom de drones ou les robots volants, sont des véhicules aériens sans pilote qui peuvent être contrôlés à distance ou fonctionner de manière autonome grâce à des systèmes embarqués. Ils se distinguent par leur capacité à recueillir et transmettre des informations à l'aide de divers capteurs, leur autonomie et leur vitesse. Les UAV sont utilisés dans une variété d'applications civiles et militaires, telles que la surveillance, la photographie aérienne, l'inspection d'infrastructures, la livraison de colis, et la lutte contre les insurrections et le terrorisme. Dans le domaine civil, ils jouent également un rôle crucial dans la lutte contre les incendies et l'agriculture, où ils surveillent les champs agricoles et détectent des anomalies comme les plantes manquantes lors des récoltes. L'utilisation des drones nécessite le respect de certaines réglementations pour assurer leur sécurité et leur efficacité[38] [27].

Il existe actuellement plusieurs types de drones dans le monde, chacun ayant des applications spécifiques. Par exemple, le DJI Phantom 4 (la figure 1.15) est largement utilisé pour la photographie aérienne et la vidéographie grâce à sa qualité d'image exceptionnelle

et sa stabilité en vol. Le MQ-9 Reaper (la figure 1.16), quant à lui, est employé par l'armée américaine pour des missions de reconnaissance et des frappes aériennes, mettant en avant son endurance et sa capacité à transporter des charges lourdes. Dans le domaine civil, des drones comme ceux utilisés par Amazon Prime Air (la figure 1.17) pour la livraison de colis illustrent l'étendue des possibilités offertes par les UAV, transformant la logistique et la distribution grâce à leur efficacité et rapidité.

Fig. 1.15 – DJI Phantom 4

Fig. 1.16 – MQ-9 Reaper

Fig. 1.17 – Amazon Prime Air

1.5.2 AUV (Autonomous Underwater Vehicle)

Un Véhicule Sous-Marin Autonome (AUV) est un robot piloté par ordinateur, équipé de capteurs pour collecter des données océanographiques. Il peut suivre des trajectoires préprogrammées avec précision, réaliser des mesures spatiales et temporelles, et explorer des zones spécifiques en activant sa propulsion ou en stationnant passivement. Les AUV jouent un rôle crucial dans la cartographie des fonds marins, la surveillance environnementale, la recherche océanographique, et la collecte de données pour des applications telles que la météorologie, la navigation et la défense. Son endurance est limitée à 8 à 50 heures, avec une vitesse optimale d'environ 1,5 m/s. Les opérations semi-autonomes offrent des avantages en matière de surveillance et de redirection des missions par rapport à l'autonomie totale [46].

Il existe différents types de Véhicules Sous-Marins Autonomes (AUV) utilisés pour diverses applications. Par exemple, le Bluefin-21 (la figure 1.18) est utilisé pour des missions de recherche et de cartographie océanographique, notamment pour retrouver l'épave du vol MH370 de Malaysia Airlines. De plus, l'Iver3 (la figure 1.19), un AUV compact et

portable, est utilisé pour la cartographie sous-marine, l'inspection d'infrastructures et les applications militaires. Un autre exemple est l'OceanOne (la figure 1.20), un robot sous-marin humanoïde qui explore les fonds marins. Il combine les meilleures caractéristiques des véhicules télécommandés et des robots humanoïdes, disposant notamment d'une main robotique permettant de récupérer des objets comme le ferait un être humain.

FIG. 1.18 – Bluefin-21

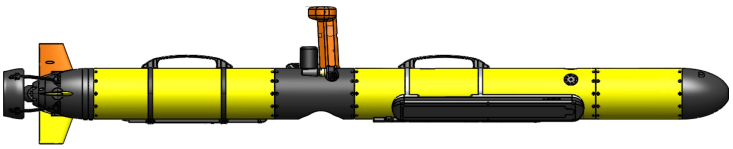


FIG. 1.19 – Iver3

FIG. 1.20 – Robot OceanOne

1.5.3 UGV (Unmanned Ground Vehicle)

Les UGV (véhicules terrestres sans pilote) sont conçus pour accomplir diverses tâches au sol. Ils sont utilisés dans des domaines tels que la surveillance, le déminage, le transport de marchandises, les opérations de secours, l'agriculture de précision et les missions militaires. Ces véhicules peuvent être contrôlés à distance ou fonctionner de manière autonome grâce à des systèmes avancés de navigation et de détection. Les UGV comprennent différents types de robots mobiles, notamment les robots rampants, à chenilles, marcheurs et à roues, chacun étant adapté à des environnements et des missions spécifiques.

1. **Les robots à pattes (ou marcheurs)**

   Les robots à pattes possèdent la capacité d'ajuster leur posture de marche afin de naviguer efficacement dans des espaces restreints, grâce à leur grande variété de degrés de liberté. Lorsque l'accès à un site devient difficile voire dangereux pour les humains, les robots marcheurs sont conçus pour relever ce défi [18].

   Les robots mobiles à pattes se déclinent en de nombreuses variantes, caractérisées par leur forme et le nombre de leurs membres. On distingue ainsi les monopodes (Toyota's One Legged Jumping Robot), les bipèdes (humanoïdes) à deux jambes, les quadrupèdes (de type cheval) à quatre pattes, ainsi que les hexapodes (de type araignée), les octopodes, et ceux possédant plusieurs pattes.

FIG. 1.21 – Robot monopode (TOYOTA)[40]

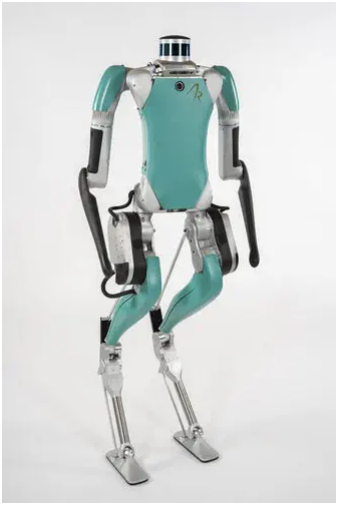


FIG. 1.22 – Robot bipède (type humanoïdes)[20]

FIG. 1.23 – Quadrupède (type cheval)

FIG. 1.24 – Hexapodes (type araignée)[21]

2. **Robots mobiles à chenilles**

   L'utilisation des chenilles présente de nombreux avantages, notamment une adhérence au sol optimale et une capacité à franchir des obstacles. Ces caractéristiques

en font un choix privilégié pour les terrains accidentés ou de mauvaise qualité, tels que ceux présentant de la boue ou de l'herbe. En comparaison, les roues rencontrent souvent des difficultés sur de tels terrains, ce qui peut entraîner une perte d'efficacité et des interactions indésirables avec le sol. Les chenilles protègent les robots à roues dans ces conditions en offrant une meilleure adhérence et en permettant de franchir les obstacles plus facilement. Il est important d'utiliser des matériaux de haute qualité pour les chenilles afin d'éviter tout glissement ou dérapage des véhicules équipés de ce système [38] [27] [36]. La figure 1.25 illustre ce type de robot.

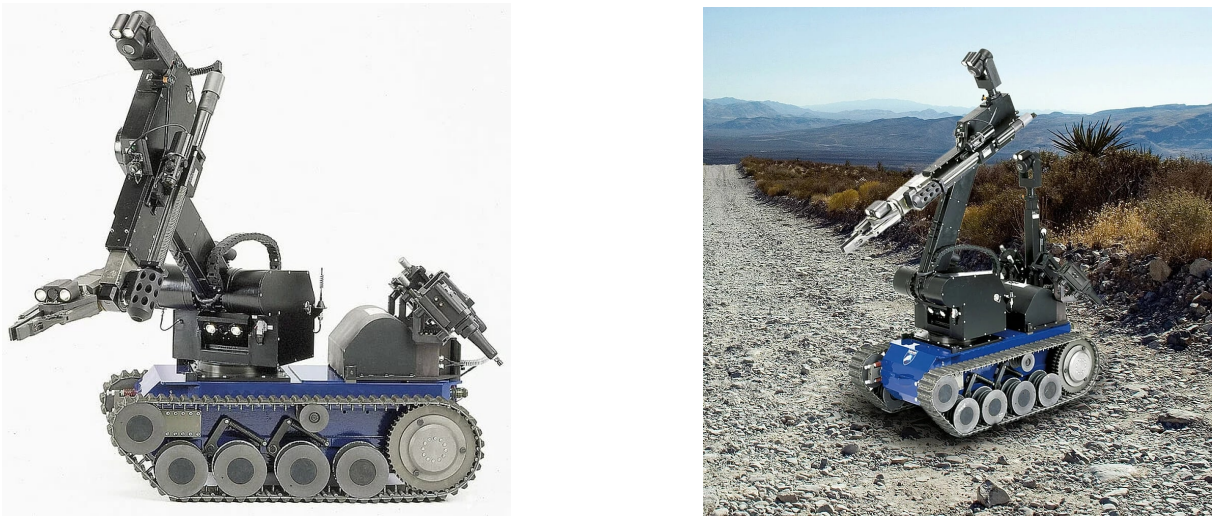


FIG. 1.25 – Robots mobiles à chenilles [3]

3. **Robots mobiles rampants**

Les robots rampants ou serpentiformes comme illustré dans la figure 1.26 sont des robots hautement articulés capables de coordonner leurs degrés de liberté internes pour réaliser diverses formes de locomotion, dépassant ainsi les capacités des robots traditionnels à pattes ou à roues. Leurs principaux avantages résident dans leur polyvalence, leur permettant d'adopter des comportements variés tels que grimper, ramper et nager [36].

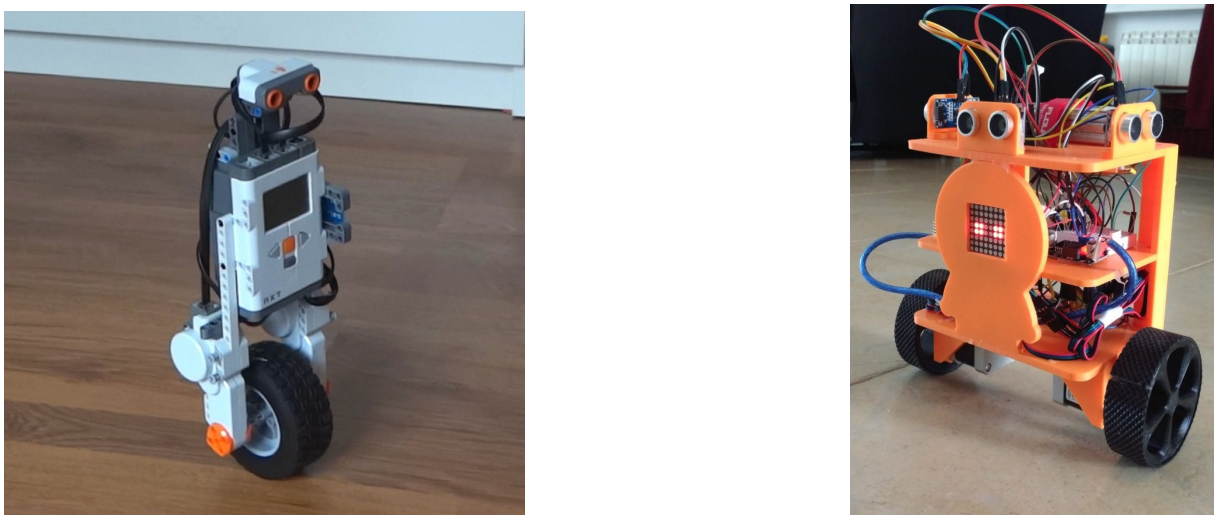
FIG. 1.26 – Robot mobile rampant (serpentiforme)

4. **Robot mobiles à roues**

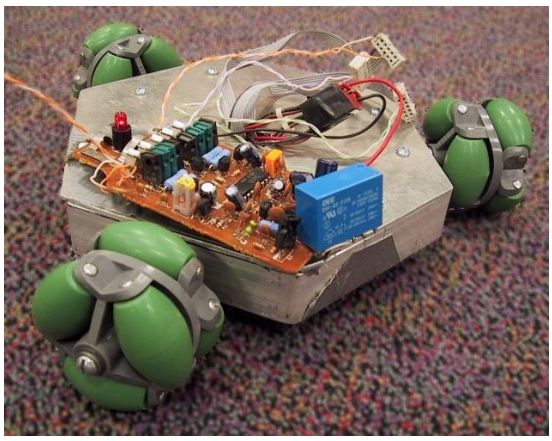
Les robots mobiles à roues offrent des avantages significatifs en matière de locomotion robotique, notamment dans des domaines tels que le transport, la logistique et la distribution. Ils sont plus simples à concevoir, construire et programmer par rapport à des alternatives telles que les chenilles ou les jambes, surtout sur des terrains plats et non accidentés. De plus, ils sont généralement plus abordables et causent moins d'usure sur les surfaces qu'ils parcourent. Leur contrôle est moins complexe, et ils offrent une meilleure stabilité car ils restent en contact avec le sol. Cependant, leur principal inconvénient réside dans leur capacité limitée à naviguer dans des obstacles tels que des terrains rocheux ou des surfaces à faible friction. Il existe plusieurs types des robots mobiles à roues selon le types des roues comme illustré dans la figure 1.27 (roue standard fixe, roue mecanum et roue sphérique) et aussi le nombres des roues comme illustré dans la figure 1.28 (uni-cycle, bicycle, tricycle, 4 roues de type voiture....) [36].

Standard fixe Roue mecanum Roue sphérique

FIG. 1.27 – Types des roues des robot mobiles



Uni-cycle Bicycle



Tricycle [41]

4 roues

FIG. 1.28 – Types des robot mobiles à roues

1.6 Robots mobiles à quatre roues

Notre travail porte sur la réalisation d'un robot mobile à 4 roues. Ces robots sont des systèmes robotiques équipés de quatre roues qui leur permettent de se déplacer de manière autonome dans leur environnement. Cette configuration offre une stabilité supérieure à leurs homologues à trois roues, grâce à une répartition uniforme du poids sur quatre points de contact avec le sol. Ils peuvent ainsi naviguer efficacement sur une variété de surfaces, qu'elles soient planes ou légèrement inclinées, et surmonter des obstacles de taille modérée. En outre, les roues de ces robots peuvent être dirigées de manière différentielle, fonctionner par paires (comme un mouvement de tank), ou adopter une configuration similaire à celle d'une voiture, ce qui améliore leur manoeuvrabilité et leur adaptabilité aux différentes conditions d'utilisation [36].

1.6.1 Avantages des robots mobiles à quatre roues

— **Stabilité :** Grâce à leurs quatre points de contact avec le sol, ces robots offrent une stabilité supérieure, réduisant le risque de basculement, même sur des terrains irréguliers.

— **Manoeuvrabilité :** Ils possèdent une bonne capacité de manoeuvre, notamment avec des configurations de direction avancées comme la direction à quatre roues, permettant des virages serrés et des mouvements plus fluides.

— **Simplicité de conception :** Comparés aux robots à pattes, les robots à roues sont généralement plus simples à concevoir et à construire, avec moins de pièces mobiles, ce qui réduit les coûts de maintenance et de fabrication.

— **Efficacité énergétique :** Les systèmes de propulsion à roues sont souvent plus efficaces sur des surfaces planes, permettant une consommation d'énergie optimisée pour des missions prolongées.

1.6.2 Défis des robots mobiles à quatre roues

— **Mobilité limitée sur terrains accidentés :** Bien qu'ils soient efficaces sur des surfaces planes, les robots à quatre roues peuvent rencontrer des difficultés sur des terrains très accidentés, sablonneux ou boueux, où les roues peuvent perdre de la traction ou s'enliser.

— **Obstacles et franchissement :** La capacité à franchir des obstacles verticaux, comme des marches ou des débris, est limitée par rapport aux robots à pattes, nécessitant parfois des adaptations supplémentaires pour ces environnements.

— **Adhérence et traction :** Sur des surfaces glissantes ou instables, maintenir une bonne adhérence peut être un défi, affectant la précision des mouvements et la capacité à effectuer des tâches de manière fiable.

— **Complexité de la navigation autonome :** Bien que les systèmes de navigation autonome aient progressé, les robots à roues doivent encore surmonter des défis liés à la détection et à l'évitement d'obstacles en temps réel dans des environnements dynamiques et imprévisibles.

1.7 Généralités sur la navigation autonome des robots mobiles

La navigation autonome constitue un défi majeur en robotique mobile, impliquant de fournir au robot la capacité de prendre des décisions de manière autonome. Son objectif est de permettre au robot de se déplacer de manière autonome à travers son environnement de travail, en utilisant exclusivement les informations captées par ses capteurs, sans intervention humaine. En d'autre termes, la navigation des robots mobiles vise à élaborer des méthodes permettant à un robot de se déplacer de manière autonome dans un environnement, qu'il soit intérieur ou extérieur, en trouvant un chemin libre entre un point de départ et un point d'arrivée. L'implémentation de ces méthodes qui permettent d'atteindre l'objectif de la navigation nécessite une architecture de contrôle qui sera détaillée dans la section qui suit.

1.7.1 Architecture de contrôle

Un robot mobile est un système complexe qui comprend divers composants matériels et logiciels. L'architecture de contrôle d'un robot (illustrée dans la figure 1.29) désigne l'organisation et la disposition des logiciels qui influencent son comportement et ses actions. Cette structure est élaborée dans le but de permettre au robot d'interagir avec son environnement, de prendre des décisions basées sur les informations sensorielles disponibles, et d'accomplir des tâches de navigation spécifiques de manière autonome ou semi-autonome. L'architecture de contrôle doit assurer toutes les taches qui permettent au robots de naviguer de façon autonome, notamment :

— **Perception :** Évaluer les informations environnementales collectées par ses capteurs pour comprendre le contexte dans lequel il opère ainsi que se localiser dans son environnement.

— **Décision :** Utiliser les données environnementales pour définir les actions à entreprendre, établissant ainsi des séquences d'actions appropriées. Lors de la navigation, cette étape est basée principalement sur le SLAM.

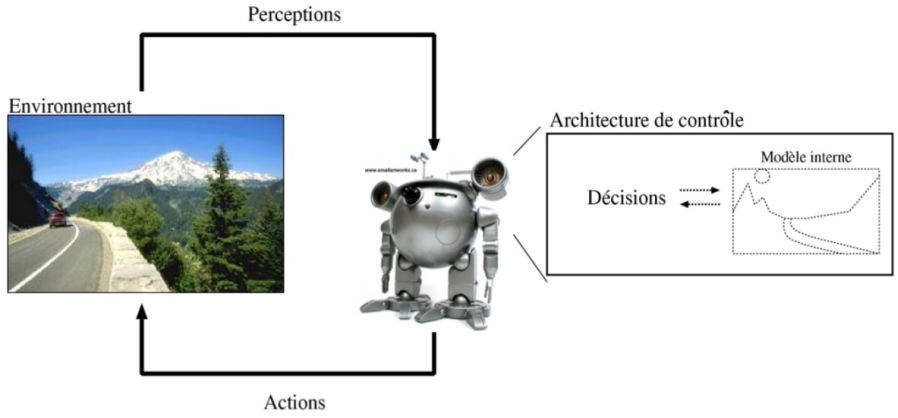


FIG. 1.29 – Architecture de contrôle [24]

— **Action :** Mettre en oeuvre des commandes ou des instructions aux actionneurs pour que le robot exécute les séquences de mouvements déterminées par les décisions prises.

Ces trois principales taches de l'architecture de contrôle peuvent être organisées en trois manières [13] :

1. **Les contrôleurs hiérarchiques :** Ils sont organisés en une hiérarchie de niveaux, chacun responsable d'un aspect spécifique de la tâche à accomplir. Ces contrôleurs sont basés sur trois étapes principales : la modélisation de l'environnement, la planification des actions et l'exécution du plan. Ces taches sont exécutés en série.

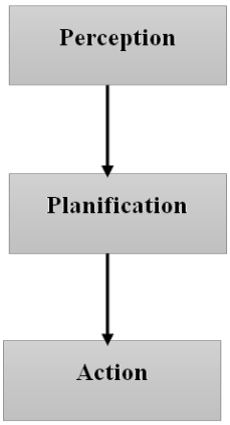


FIG. 1.30 – Illustration des contrôleurs hiérarchiques

2. **Les contrôleurs réactifs :** Selon BROOKS [17], les contrôleurs qu'il propose rendent la navigation comme à un ensemble de comportements réactifs de base. Le robot doit effectuer deux taches principales en parallèle : la perception et l'action.

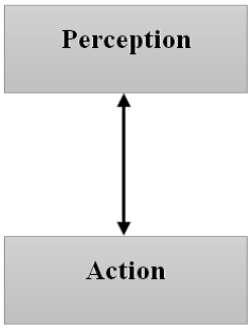


FIG. 1.31 – Illustration des contrôleurs réactifs

3. **Les contrôleurs hybrides :** Cette catégorie de contrôleurs intègre les deux architectures mentionnées précédemment, ce qui en fait la solution actuellement la plus répandue en raison de sa réactivité rapide et de sa capacité à effectuer une planification. C'est l'architecture adoptée dans notre travail.

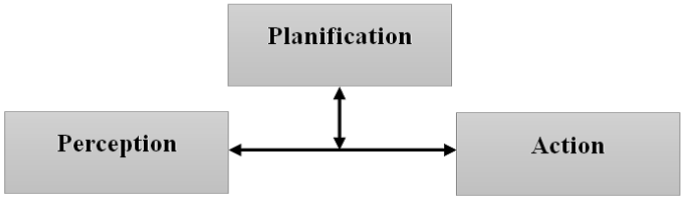


FIG. 1.32 – Illustration des contrôleurs hybrides

Dans ce qui suit nous allons détailler chacune de ces taches principales que le robot doit assurer afin de compléter la navigation autonome.

1.7.2 Perception :

Pour un robot, la perception est essentielle afin de se déplacer librement dans son environnement. Cela signifie qu'il doit être capable de repérer et de réagir aux différentes situations. Les capteurs jouent un rôle essentiel en lui fournissant une sorte de vision, lui permettant ainsi de déterminer sa position ou de comprendre comment il se déplace d'un point à un autre. Ces capteurs peuvent être regroupés en deux catégories principales :

— **Capteurs proprioceptifs :** Ils sont conçus pour mesurer les données internes du robot, comme la vitesse de rotation des roues ou l'angle d'inclinaison. Ils aident à comprendre et à contrôler l'état et les mouvements internes du robot, sans nécessiter de référence à l'environnement externe. Ces capteurs comprennent des accéléromètres, qui mesurent l'accélération du robot, des gyroscopes, qui détectent son orientation et sa rotation, ainsi que des encodeurs de roues, qui surveillent la distance parcourue. Dans notre cas, nous avons intégré des moteurs avec encodeurs sur le robot pour déterminer les grandeurs liées à son déplacement et à sa vitesse (La description des encodeurs est détaillée dans la partie 3.4.1).

— **Capteurs extéroceptifs :** Ils sont conçus pour fournir des informations sur l'environnement externe du robot. Ils comprennent des dispositifs tels que les caméras, les LiDARs ou les sonars, qui permettent de détecter et d'interagir avec le monde environnant du robot. Ces capteurs sont utilisés pour des tâches telles que la détection d'obstacles, la localisation et la cartographie de l'environnement. Pour notre projet, nous avons utilisé un LiDAR comme principal capteur extéroceptif (La description du LiDAR est détaillée dans la partie 2.4.1).

1.7.3 SLAM :

Le processus complet qui permet à un robot mobile d'analyser son environnement et de se situer dedans est désigné par l'acronyme SLAM. Ensuite, pour qu'un robot navigue de manière autonome, il doit être capable de se localiser précisément sur une carte. La cartographie implique la création d'une carte représentant la structure spatiale de l'environnement à partir des données collectées par les capteurs du robot (LiDAR). La localisation consiste quant à elle à déterminer la position du robot sur cette carte, ce qui correspond à sa position dans le monde réel. Alors, c'est quoi un SLAM ?

SLAM, qui signifie "Localisation et Cartographie Simultanées" en français, est un processus par lequel un robot autonome établit une carte d'un environnement inconnu tout en se localisant simultanément à l'intérieur de cette carte. Ainsi, nous pouvons dire que le SLAM implique deux tâches principales : la localisation et la cartographie. Ces deux tâches sont définies dans un processus qui est dynamique et itératif. Pendant que le robot se déplace, il met à jour sa carte et sa compréhension de l'environnement, tout en améliorant sa propre localisation [14].

— **Mécanismes Clés du SLAM :**

1. Collecte de données : Les robots utilisent divers capteurs comme les LiDARs, les caméras et les IMU pour recueillir en continu des données sur leur environnement. Ces données sont essentielles pour identifier des points de repère servant à créer la carte et à localiser le robot.
2. Estimation et prédiction : Le SLAM utilise des techniques avancées telles que les filtres de Kalman (KF) et les filtres de particules pour estimer la position actuelle du robot et prédire ses mouvements futurs. Ces estimations sont constamment ajustées à mesure que de nouvelles données sont intégrées.
3. Optimisation : Des méthodes d'optimisation sont utilisées pour résoudre les divergences entre les données enregistrées précédemment et les nouvelles observations. La méthode graphique d'optimisation, où les données sont représentées par des noeuds et leurs relations par des arêtes, est souvent employée pour affiner la carte et la localisation.
4. Cartographie : En parallèle avec la localisation, une représentation de l'environnement est créée en associant les estimations de position aux données des capteurs. Cette carte est utilisée pour la navigation et d'autres fonctions robotiques. Il est important de noter que tous les défis en cartographie ne sont pas égaux en termes de difficulté. La complexité de ces problèmes découle de divers facteurs, dont lesquels on cite : la taille de l'environnement, le bruit dans la perception et l'actionnement , l'ambiguïté perceptuelle (confusion des caractéristiques similaires) et les cycles (chemins qui bouclent sur eux-mêmes) [43].

— **Différents algorithmes SLAM**

Pour répondre aux défis de la localisation et de la cartographie simultanées, divers algorithmes de SLAM sont employés. Les principaux parmi eux incluent [43] :

1. Extended Kalman Filter SLAM (EKF) : Un des premiers algorithmes de SLAM qui utilise un filtre de Kalman étendu pour gérer l'incertitude. Il se base sur un modèle de prédiction pour estimer la trajectoire du robot et la position des repères dans son environnement.
2. FastSLAM : Cette approche emploie un filtre de particules pour estimer la trajectoire du robot, tandis que des filtres de Kalman distincts sont utilisés pour estimer les positions des repères individuels. Cette méthode s'avère efficace pour la gestion de grands environnements en séparant l'estimation de la pose et la cartographie.
3. Graph-based SLAM : Cet algorithme construit un graphe où les noeuds représentent les positions du robot à différents instants, et les arêtes représentent les contraintes de déplacement entre ces positions. Des techniques d'optimisation sont alors employées pour déterminer la configuration la plus probable du graphe, reflétant ainsi la trajectoire du robot et la carte.
4. Grid-based SLAM : Cette méthode utilise des grilles pour représenter la carte (Figure 1.33). Chaque cellule de la grille stocke une probabilité d'occupation. Les algorithmes d'actualisation de grille sont alors appliqués pour intégrer les mesures des capteurs et mettre à jour les croyances sur l'état de l'environnement [43].

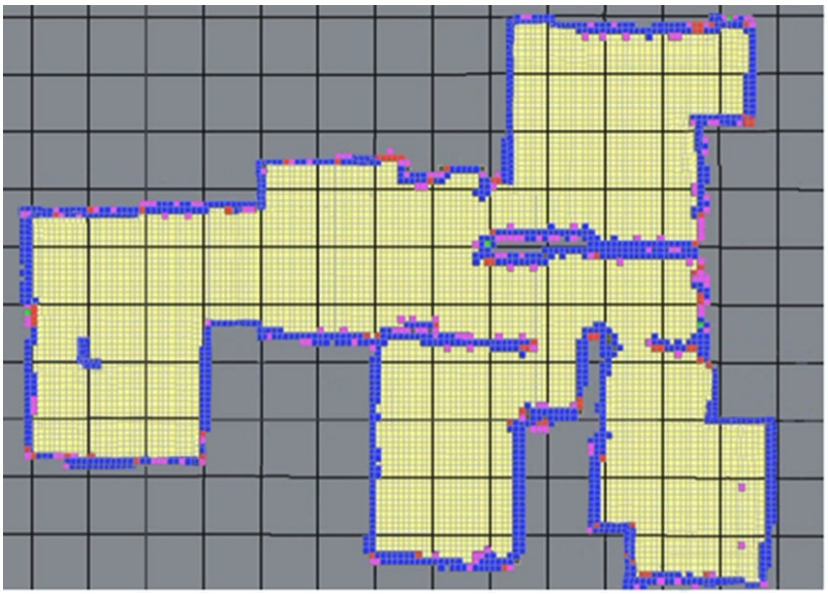


FIG. 1.33 – Carte en grille construite avec grid-based SLAM [30]

— **Principe de fonctionnement du Grid-based SLAM :**

Dans notre projet, la cartographie de l'environnement est réalisée grâce au grid-based SLAM, dont nous détaillerons le fonctionnement dans cette section. Ce système utilise un filtre particulaire pour fusionner les données d'odométrie avec celles fournies par le LiDAR. Chaque particule représente une trajectoire possible du robot et maintient sa propre carte de la grille. Lors de l'initialisation, chaque particule

est assignée à une position de départ avec une carte vide. Le principe fondamental du filtre particulaire pour le SLAM est d'estimer une distribution postérieure $p(x_{1:t}|z_{1:t}, u_{0:t})$ sur les trajectoires $x_{1:t}$ du robot basée sur ses observations $z_{1:t}$ réalisées avec le LiDAR et ses mesures d'odométrie $u_{0:t}$. Cette distribution postérieure est ensuite utilisée pour calculer une distribution postérieure sur les cartes et les trajectoires [31].

$$p(x_{1:t}, m|z_{1:t}, u_{0:t}) = p(m|x_{1:t}, z_{1:t})p(x_{1:t}|z_{1:t}, u_{0:t}) \tag{1.1}$$

Tel que m est la carte de grille d'occupation, exprimée dans l'équation 1.2

$$m = \sum_{i=1}^{N} m_i \tag{1.2}$$

Chaque cellule m_i de la carte m est attribuée d'une valeur binaire indiquant son occupation : "1" pour occupée et "0" pour libre. La décomposition de cette équation (équation 1.1) permet de décomposer le problème complexe en deux sous-problèmes plus simples. La première partie traite de la distribution postérieure sur les trajectoires s $x_{1:t}$, comme décrit dans l'équation suivante :

$$p(x_{1:t}|z_{1:t}, u_{0:t}) = \frac{p(z_t|x_t)p(x_{1:t}|z_{1:t-1}, u_{0:t})}{p(z_t|z_{1:t-1}, u_{0:t})} \tag{1.3}$$

1. **Étape de prédiction :** La prédiction de la position future du robot repose sur l'utilisation du modèle de mouvement basé sur les mesures d'odométrie :

   $$x_t^k = g(x_{t-1}^k, u_t) \tag{1.4}$$

   où x_t^k est l'état de la particule k à l'instant t, g est la fonction de transition de l'état basée sur l'odométrie.
   Pendant la phase de prédiction, nous utilisons le modèle de mouvement pour estimer la nouvelle position x_t à partir de la position précédente x_{t-1} et de la commande de mouvement u_t.Cette estimation est représentée par $p(x_{1:t}|u_{0:t})$ et $p(x_{1:t}|z_{1:t-1}, u_{0:t})$ est mise à jour en propageant les trajectoires des particules comme illustré dans l'équation 1.5.

   $$p(x_{1:t}|z_{1:t-1}, u_{0:t}) = p(x_t|x_{t-1}, u_t)p(x_{1:t-1}|z_{1:t-1}, u_{0:t}) \tag{1.5}$$

2. **Étape de mise à jour de l'observation (Correction de pose avec scan matching) :**La correction de pose utilise les observations du capteur pour ajuster la pose estimée du robot. Maximisons la vraisemblance

   $$x_t^k = \arg\max_x p(z_t|x, m_{t-1}^k) \tag{1.6}$$

   Où m_{t-1}^k est la carte maintenue par la particule k jusqu'à l'instant t-1.
   Lors de la correction, nous ajustons la probabilité de la trajectoire en fonction de la nouvelle observation z_t. Cela est représenté par le terme de vraisemblance : $p(z_t|x_t)$
   Nous fusionnons cette vraisemblance avec la distribution prédite des trajectoires pour obtenir la distribution postérieure mise à jour. Le dénominateur $p(z_t|z_{1:t-1}, u_{0:t})$ agit comme un facteur de normalisation et assure que la somme des probabilités reste égale à 1.

La deuxième partie, représentant la distribution postérieure de la carte compte tenu des trajectoires estimées du robot et des observations $p(m|x_{1:t}, z_{1:t})$,est calculée en mettant à jour la carte à chaque étape en fonction de chaque nouvelle estimation de pose et de l'observation correspondante. Chaque particule met à jour sa propre carte en fonction de la nouvelle estimation de pose et des nouvelles observations, comme illustré dans l'équation 1.7 :

$$m_t^k = \text{update\_map}(m_{t-1}^k, x_t^k, z_t) \tag{1.7}$$

Où update_map est la fonction de mise à jour de la carte basée sur les observations z_t et la pose estimée m_t^k
Après cette étape, nous calculons le poids d'importance de chaque particule pour refléter sa probabilité en tenant compte des nouvelles observations. Cela nous permet de déterminer quelles particules représentent le mieux l'état actuel du robot et de son environnement. Le poids d'importance peut être calculé à l'aide de l'équation 1.8 :

$$w_t^k = p(z_t|x_t^k, m_{t-1}^k) \tag{1.8}$$

Les poids d'importance sont normalisés afin de garantir qu'ils forment une distribution de probabilité valide, selon l'équation 1.9 :

$$w_t^k = \frac{w_t^k}{\sum_{i=1}^{N} w^i} \tag{1.9}$$

La normalisation garantit que la somme des poids est égale à 1, ce qui est essentiel pour interpréter les poids comme des probabilités, et prépare les poids pour l'étape de ré-échantillonnage, où les particules sont sélectionnées en fonction de leurs poids relatifs.
Dans la dernière étape, le ré-échantillonnage, les particules sont à nouveau sélectionnées en fonction de leurs poids d'importance pour éviter leur épuisement. Les particules ayant des poids plus élevés ont une probabilité accrue d'être choisies pour la génération suivante. Le ré-échantillonnage élimine les particules avec des poids faibles, qui sont moins représentatives de l'état actuel du robot. En privilégiant les particules avec des poids élevés, l'algorithme concentre ses ressources de calcul sur les particules les plus prometteuses, améliorant ainsi la précision et l'efficacité.
Cette représentation en grille simplifie le traitement des données en réduisant l'espace des états possibles à des cellules discrètes. Cela conduit à un traitement plus rapide et plus gérable, même sur des ordinateurs moins puissants [31]. Après la création de la carte, le robot doit être capable de se repérer et de se localiser de manière autonome, sans nécessiter d'intervention humaine.

— **Localisation :** La localisation désigne la capacité à déterminer avec précision la position du robot dans un environnement donné. On peut la diviser en trois catégories :

1. Localisation relative : Elle repose sur l'utilisation de capteurs proprioceptifs. Son but est d'évaluer la position, la vitesse et l'accélération du robot mobile. Cette évaluation se fait en intégrant les informations fournies par les capteurs proprioceptifs et la pose à l'instant t-1. Cependant, un inconvénient majeur de cette méthode est l'accumulation d'erreur.

2. Localisation absolue : C'est une méthode permettant à un robot de se situer dans un repère global, lié à l'environnement. Parmi les capteurs utilisés dans ce type de localisation, on compte le GPS , les caméras et les télémètres laser.
3. Localisation hybride : cette approche intègre des données provenant à la fois de capteurs extéroceptifs et proprioceptifs pour déterminer la position du robot. Son objectif est de combiner les avantages des localisations absolue et relative, tout en surmontant leurs limitations, afin d'améliorer la précision et la fiabilité de la localisation du robot. Les informations recueillies par les différents capteurs sont fusionnées à l'aide de techniques de fusion telles que le filtre de Kalman (KF), le filtre de Kalman étendu (EKF) ou le filtre à particules, pour fournir une estimation plus précise et fiable de la pose du robot.

Dans notre projet, nous avons opté pour l'algorithme d'AMCL, qui repose sur une approche de localisation hybride. L'AMCL combine la localisation relative par odométrie avec les données de perception du robot pour estimer avec précision sa position. Il utilise des filtres de particules pour fusionner les informations de mouvement fournies par l'odométrie avec les observations du capteur externe (LiDAR), permettant ainsi une localisation précise. L'AMCL maintient une mise à jour continue de la pose relative du robot en utilisant les données d'odométrie. Parallèlement, les observations des capteurs corrigent les erreurs accumulées, assurant ainsi une localisation précise en se référant à une carte préétablie. La figure 3.7 illustre le fonctionnement de l'AMCL.

L'implémentation de cet algorithme dans notre projet sera détaillée dans le troisième Chapitre.

1.7.4 Planification du chemin

La planification du chemin permet au robot mobile de trouver le meilleur chemin entre son point de départ et sa destination. Elle se divise en deux méthodes : la planification locale et la planification globale. La Figure 1.35 explique le principe de génération du trajectoire.

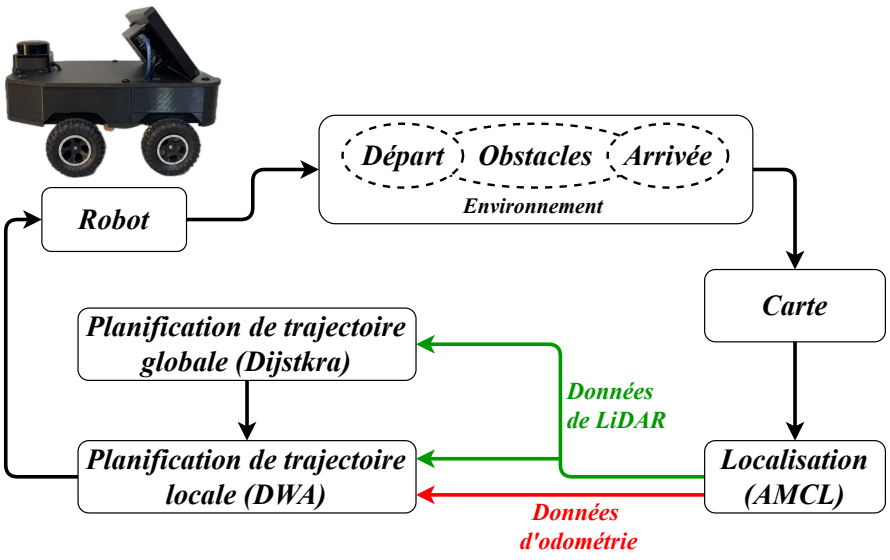


Fig. 1.35 – Principe de génération du trajectoire

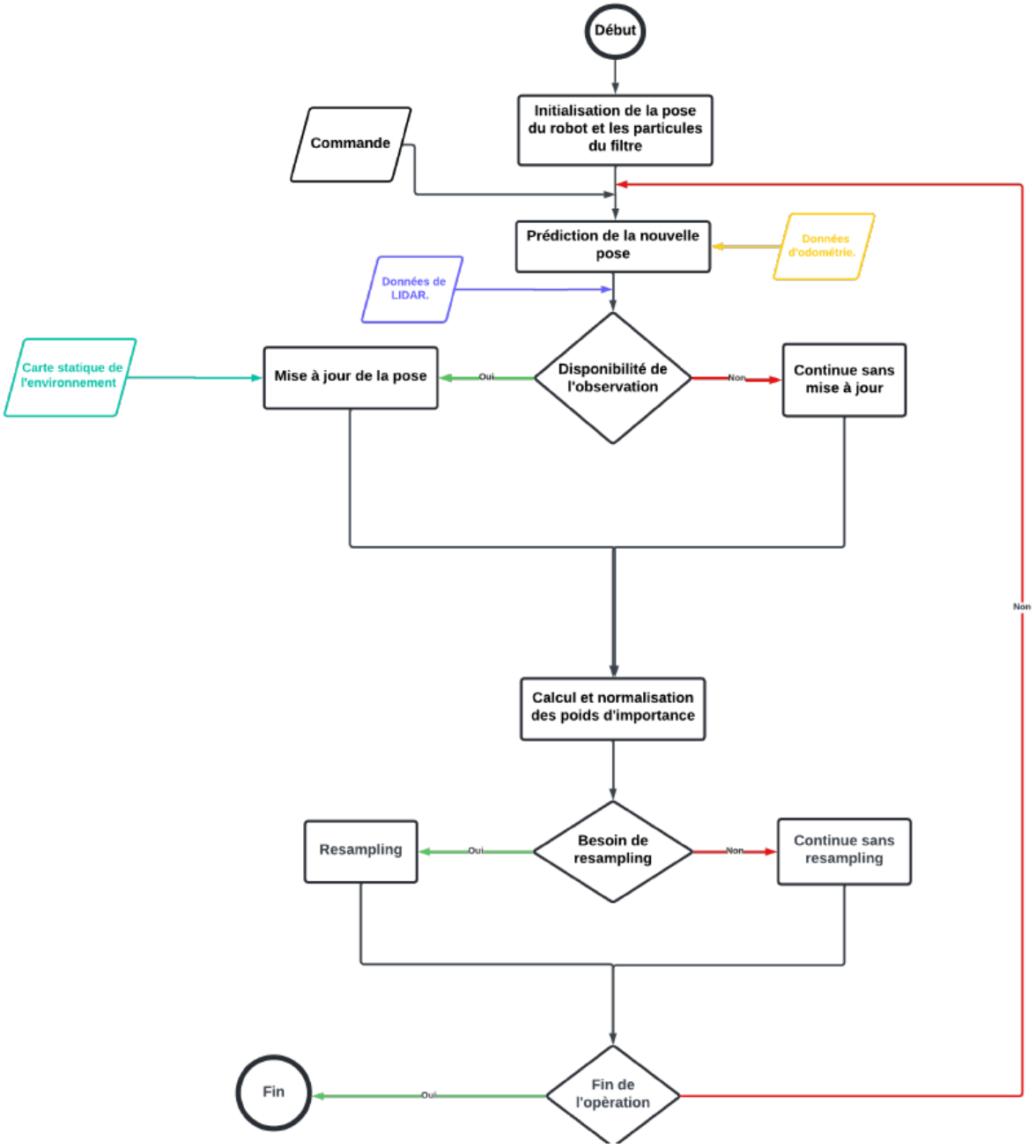


FIG. 1.34 – Le principe du fonctionnement de l'AMCL [31]

1. **Planification locale :** Créer une trajectoire locale en se basant sur les données environnementales disponibles afin de permettre au robot d'éviter efficacement les obstacles proches. Les planificateurs de trajectoire locaux sont très répandus car les informations fournies par le système de perception évoluent en temps réel dans des environnements dynamiques [16]. Des méthodes telles que la méthode de DWA, le champ de potentiel et l'histogramme de champ vectoriel se démarquent par leur souplesse, car elles peuvent être ajustées pour fonctionner avec des cartes locales dynamiques ou même sans carte du tout. Leur adaptabilité découle de leur capacité à traiter des données en temps réel pour l'évitement d'obstacles et la navigation directe, ce qui les rend utilisables dans divers contextes de navigation, qu'une carte préexistante soit disponible ou non.

Dans notre projet, nous avons utilisé la méthode DWA car c'est l'une des algorithmes les plus prisés pour la planification locale. Cette méthode repose sur l'optimisation de la trajectoire d'un robot mobile en échantillonnant et en évaluant diverses combinaisons de vitesses linéaires et angulaires, tout en respectant les contraintes cinématiques du robot. L'illustration présentée dans la Figure 1.36 détaille le processus de l'évitement d'obstacles à l'aide de la méthode DWA. Nous allons voir l'implémentation de cette méthode dans la section 3.8.1.

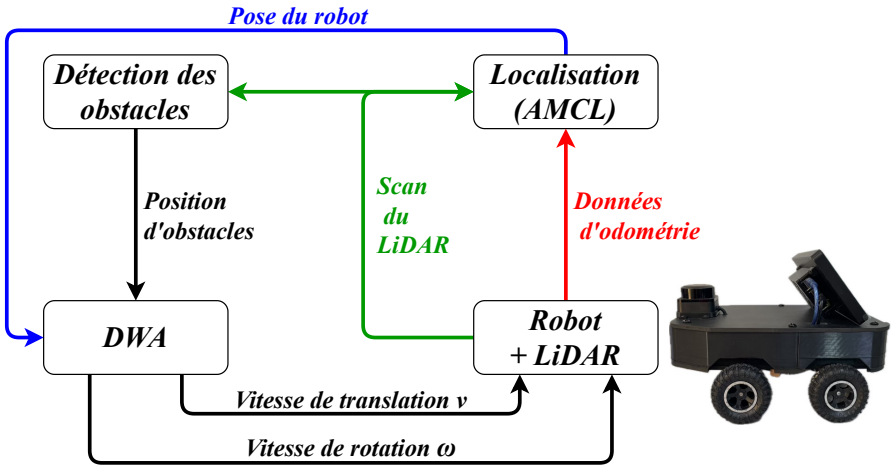


FIG. 1.36 – Architecture de système de navigation locale

2. **Planification globale :** La planification globale focalise sur la création d'une trajectoire complète, allant du point de départ au point d'arrivée, avant que le robot ne commence son déplacement. Elle requiert des cartes détaillées et souvent préalablement établies de l'ensemble de l'environnement. Parmi les algorithmes les plus couramment employés pour cette planification de trajectoire à l'échelle globale, on trouve l'algorithme de Dijkstra. Cet algorithme maintient une liste mise à jour des distances les plus courtes depuis un noeud de départ vers tous les autres noeuds du graphe. Au début, la distance jusqu'au noeud de départ est définie à zéro, tandis qu'elle est considérée comme infinie pour tous les autres noeuds. L'algorithme utilise un ensemble, souvent appelé la "frontière", qui contient les noeuds pour lesquels les distances minimales sont déjà connues. À chaque étape, Dijkstra sélectionne le noeud avec la plus petite distance dans cet ensemble, met à jour les distances de ses voisins si nécessaire, et l'ajoute à l'ensemble des noeuds traités. Ce processus se répète jusqu'à ce que la destination soit atteinte ou que tous les noeuds accessibles aient été traités [23].

   Dans notre projet, nous avons mis en oeuvre le planificateur globale qui repose sur l'algorithme Dijkstra. Cette décision découle du fait que notre environnement à explorer est une salle intérieure de taille moyenne, nécessitant une exploration approfondie. Dans la section 3.8.1, nous allons détailler l'implémentation de cet algorithme sur notre robot mobile réalisé afin de planifier le chemin pour une tache de navigation autonome.

1.7.5 Navigation autonome

Après avoir établit la carte de son environnement et assuré sa capacité à se localiser dans son environnement ainsi que la planification du chemin, le robot peut désormais aborder une tache de navigation autonome telle que l'Allez au but tout en assurant l'évitement d'obstacles présents sur la carte construite. Selon Trullier, la navigation autonome des robots est classifiée en deux grandes catégories résumées sur la Figure 1.37 :

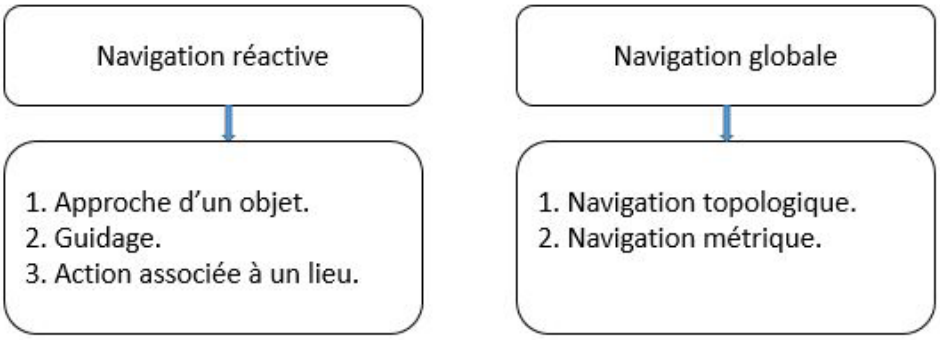


FIG. 1.37 – Types de navigation [45]

• **Navigation réactive :** La navigation réactive, appelée aussi navigation sans carte, est quand un robot se déplace sans utiliser de cartes préétablies de son environnement. Au lieu de cela, il se fie à ses capteurs, comme des caméras et des LiDARs, pour voir ce qui l'entoure et décider où aller. C'est très utile à l'extérieur ou dans des endroits qui changent souvent, où les cartes pourraient ne pas être précises. Mais ça peut être difficile car le robot doit prendre des décisions rapides en fonction de ce qu'il voit en temps réel.

1. **Approche d'un objet :** elle autorise le déplacement du robot mobile vers un objectif visible depuis sa position actuelle (Figure 1.38).

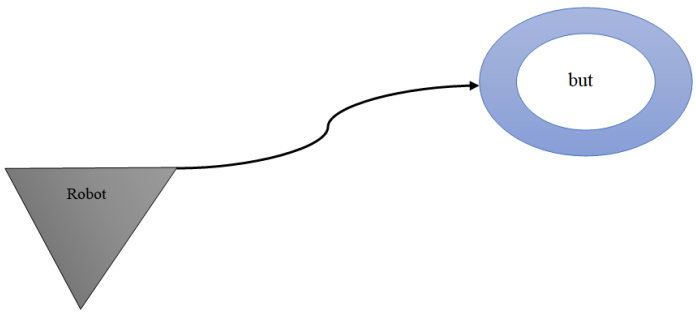


FIG. 1.38 – Approche d'un objet

2. **Guidage :** Le robot vise à atteindre un objectif qui n'est pas visible physiquement, mais plutôt caractérisé par un ensemble de repères ou d'amers (Figure 1.39).

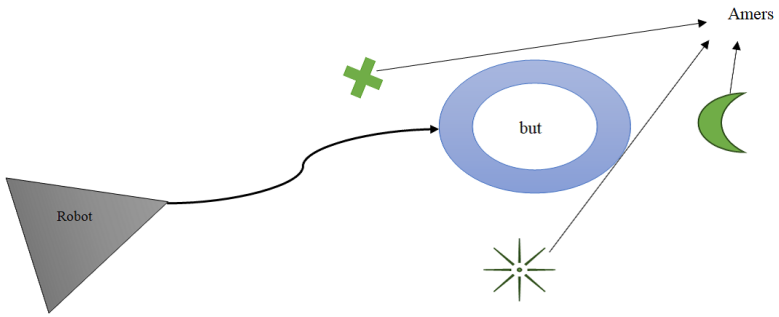


FIG. 1.39 – Guidage

3. **Action associée à un lieu :** Cette stratégie permet d'atteindre un objectif non visible en utilisant des repères également invisibles, en se basant sur la mémorisation de sa position (Figure 1.40).

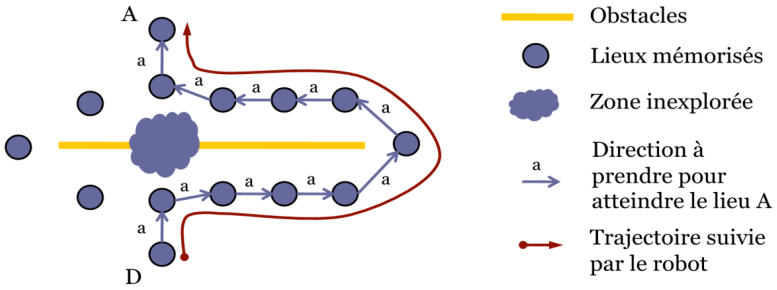


FIG. 1.40 – Action associée à un lieu [13]

• **Navigation globale :** La navigation globale, appelée aussi navigation basée sur une carte, pour un robot mobile est une méthode autonome où le robot utilise des cartes préexistantes de l'environnement pour planifier son chemin. Cependant, elle dépend fortement de la précision des cartes et peut rencontrer des difficultés dans des environnements changeants ou dynamiques.

1. **Navigation topologique :** elle repose sur la mémorisation des parcours possibles en utilisant des passages connus pour indiquer la possibilité de se déplacer d'un point à un autre sans objectif défini (Figure 1.41).

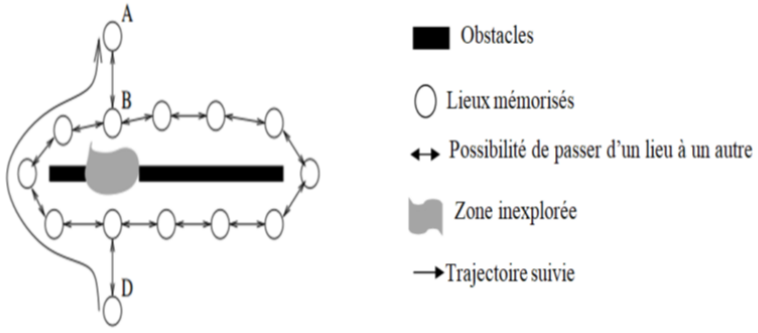


FIG. 1.41 – Navigation topologique [24]

2. **Navigation métrique :** elle permet de mémoriser les chemins et les positions des lieux possibles afin de calculer le trajet le plus court entre deux points (Figure 1.42).

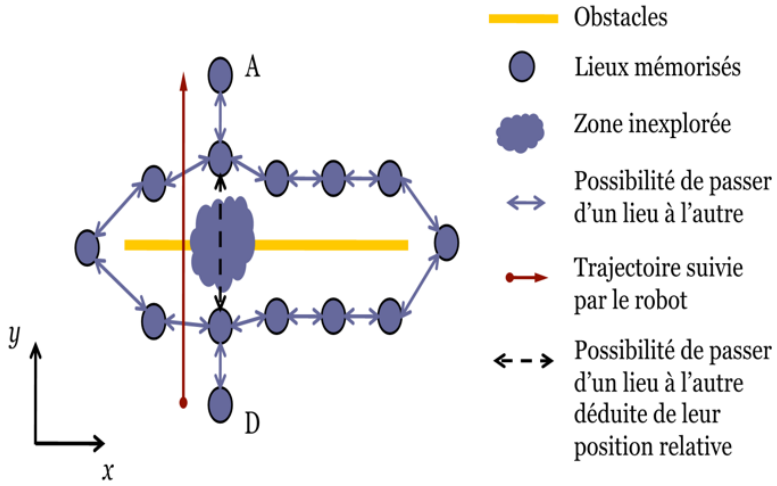


FIG. 1.42 – Navigation métrique [13]

• **Différence entre la navigation réactive et la navigation globale :**
Nous pouvons synthétiser cette différence dans le tableau ci-dessous :

Navigation globale	Navigation réactive
Utilise des cartes préétablies de l'environnement pour planifier la trajectoire du robot.	Ne nécessite pas de cartes préexistantes de l'environnement.
Dépend de la précision des cartes pour une planification efficace.	Se base sur des capteurs pour percevoir en temps réel l'environnement et réagir aux obstacles.
Convient aux environnements statiques où les informations sont disponibles à l'avance.	Adaptée aux environnements dynamiques où les conditions changent rapidement.
Requiert souvent une phase de cartographie initiale pour créer des cartes précises de l'environnement.	Les décisions de déplacement sont prises en temps réel en fonction des informations sensorielles immédiates.

TAB. 1.1 – Différences entre la navigation réactive et la navigation globale

1.8 Conclusion

Ce premier chapitre a fourni un aperçu complet de la robotique mobile, en couvrant ses définitions, types, applications, et défis, ainsi qu'une focalisation spécifique sur les robots mobiles à quatre roues et les principes généraux de la navigation autonome qui représentent la thématique de notre travail.

Nous avons insister dans ce chapitre à examiner les robots mobiles à quatre roues, en détaillant leurs avantages, comme la stabilité et l'efficacité énergétique, ainsi que les

défis qu'ils rencontrent, notamment la mobilité sur terrains accidentés et les problèmes d'adhérence. Par la suite, nous avons exploré en profondeur la navigation autonome des robots mobiles et ses différentes approches, expliquant les techniques permettant au robot de se localiser, de cartographier son environnement et de planifier des chemins afin de compléter la navigation autonome. Nous avons également détaillé l'algorithme SLAM ainsi que le principe de fonctionnement du Grid-based SLAM et la planification du trajectoire en expliquant leurs méthodes telles que la DWA et Dijkstra.

Dans le chapitre suivant, nous expliquerons l'architecture matérielle et logicielle de notre projet de fin d'étude. Nous y détaillerons également les étapes de conception et de réalisation.

CHAPITRE

2

CONCEPTION ET RÉALISATION D'UN ROBOT MOBILE À 4 ROUES

2.1 Introduction

Ce chapitre décrit la conception et la réalisation d'un robot mobile à quatre roues, en se concentrant sur quatre aspects principaux : la modélisation cinématique, la structure mécanique, l'architecture électronique et l'architecture logicielle. La modélisation cinématique permet d'établir les équations de mouvement, cruciales pour un contrôle précis du robot. Ensuite, la section de conception et de réalisation mécanique détaille le processus de construction du robot à 4 roues.

L'architecture électronique proposée pour notre robot est divisée en deux segments haut-niveau et bas-niveau, couvrant respectivement les composants avancés et les éléments de base tels que les moteurs, calculateurs et capteurs. Enfin, l'architecture logicielle est détaillée, incluant les briques logicielles pour les différents calculateurs, assurant le contrôle et la communication au sein du robot. Cette approche intégrée montre la synergie entre mécanique, électronique et logicielle pour créer un robot fonctionnel et performant.

2.2 Modélisation cinématique du robot

Pour la modélisation cinématique, on considère les hypothèses suivantes :

1. Seul le mouvement sur un plan est pris en compte.
2. Les vitesses angulaires des roues motrices sont relativement faibles.
3. Le glissement des roues par rapport au sol n'est pas pris en considération.

La figure 2.1 illustre le mouvement de la plateforme robotique à quatre roues dans le plan. Pour décrire le déplacement du robot, deux repères sont utilisés : un repère fixe $R_0(x_0,\ y_0)$, représentant le système de coordonnées global, et un repère mobile $R_m(x_m,$

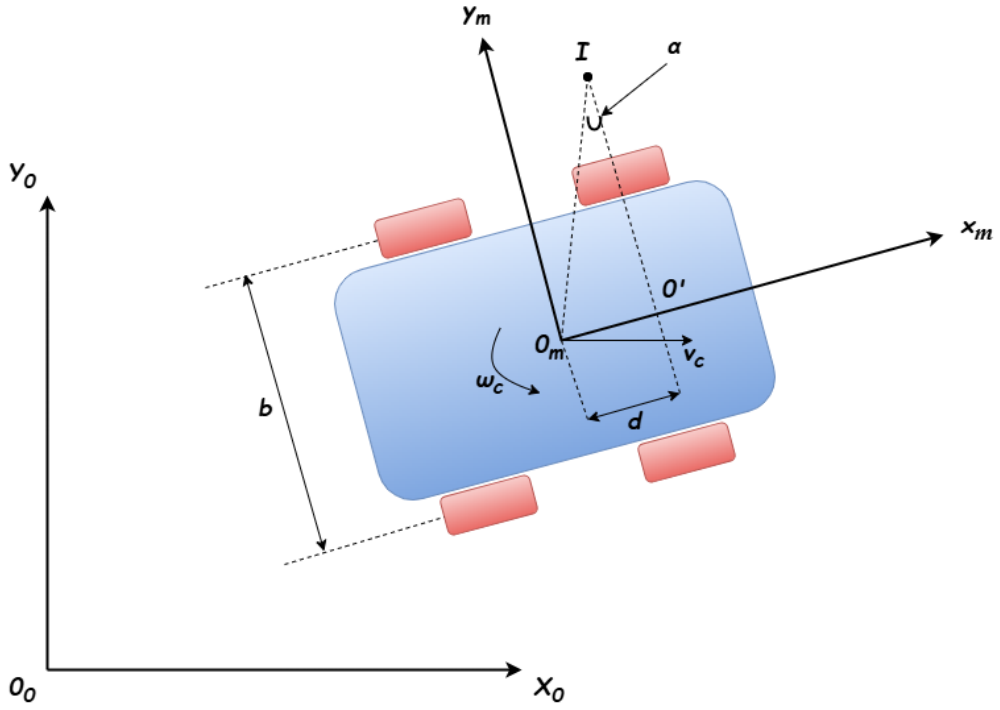


FIG. 2.1 – Le robot à 4 roues en mouvement dans le plan

y_m), attaché au corps de la plateforme. L'origine de ce repère mobile est située au centre de masse 0_m du robot. La matrice de rotation qui relie le repère R_0 au repère R_m est définie par :

$$R_0^m = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \tag{2.1}$$

Où θ représente l'angle d'orientation du robot (son cap). Le mouvement du robot mobile est composé de deux vitesses :

- La vitesse de translation : $[v_x, v_y]^T$
- La vitesse de rotation : $\omega_c = \dot{\theta}$

Avec v_x et v_y représentant les projections de v_c sur les axes du repère R_m, v_y est désignée comme la vitesse de glissement latéral. Le point I correspond au centre instantané de rotation. En raison du glissement, ce point se décale fréquemment vers l'avant d'une distance d. Un angle α se forme entre la ligne reliant I à O_m et la perpendiculaire de I à l'axe des x du repère R_m. Dans ce repère, le modèle cinématique s'exprime de la manière suivante :

$$\begin{cases} \dot{x}_m = v_x = \dfrac{r}{2}(\omega_D + \omega_G) \\ \dot{y}_m = v_y = \dfrac{r}{b}(\omega_D - \omega_G)d \\ \dot{\theta}_m = \omega_c = \dfrac{r}{b}(\omega_D - \omega_G) \end{cases} \tag{2.2}$$

Où b représente la distance entre les deux roues du robot, r est le rayon des roues du robot, et ω_G et ω_D désignent respectivement les vitesses angulaires des roues gauche et droite. En appliquant la transformation du repère R_m vers le repère R_0, le modèle cinématique devient :

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta & d\sin\theta \\ \sin\theta & -d\cos\theta \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{2.3}$$

En réécrivant simplement l'équation 2.8 sous la forme $\dot{q} = S(q)\eta$, où $q = (x, y, \theta)^T$ représente la pose du robot dans le repère cartésien inertiel R_0 ; (x, y) est la position du centre de masse du robot ; v est la vitesse de translation et ω est la vitesse angulaire du robot. L'entrée auxiliaire η est définie comme suit :

$$\eta = \begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} v_x \\ \omega_c \end{bmatrix} = \begin{bmatrix} \frac{r}{2}(\omega_D + \omega_G) \\ \frac{r}{b}(\omega_D - \omega_G) \end{bmatrix} \tag{2.4}$$

La variable $u = (\omega_G, \omega_D)^T$ est considérée comme l'entrée de commande réelle, pouvant être utilisée pour contrôler η selon la relation $\eta = Tu$, où la matrice de transformation T est définie comme suit :

$$T = r \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ -\frac{1}{b} & \frac{1}{b} \end{bmatrix} \tag{2.5}$$

En utilisant l'équation 1.5 et en notant que T est inversible, on peut déduire la relation $u = T^{-1}\eta$ comme suit :

$$\begin{bmatrix} \omega_G \\ \omega_D \end{bmatrix} = T^{-1} \begin{bmatrix} v \\ \omega \end{bmatrix} = \frac{1}{r} \begin{bmatrix} 1 & -\frac{b}{2} \\ 1 & \frac{b}{2} \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{2.6}$$

Il convient de noter que la vitesse latérale $v_y = -d\omega$, qui définit la vitesse de glissement, n'est pas intégrable. Par conséquent, la contrainte non-holonome peut être obtenue comme suit :

$$\begin{bmatrix} \sin\theta & -\cos\theta & d \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = A\dot{q} = 0 \tag{2.7}$$

D'après certains chercheurs, le paramètre de glissement d dépend principalement de deux types de facteurs :

1. L'état du sol, tel que sable sec, sol argileux, argile sèche, etc.
2. La vitesse du robot, qui influe sur les forces centrifuges.

Dans notre cas, nous avons assumé que le glissement de la roue par rapport au sol est nul $d = 0$ et donc la vitesse latérale $v_y = -d\omega = 0$. Le modèle cinématique obtenu pour le robot à quatre roues réalisé est un modèle d'un robot différentiel du type unicycle :

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{2.8}$$

2.3 Conception et réalisation de la structure mécanique du robot

Le châssis du robot est constitué de deux étages fabriqués à partir de feuilles de matériau Forex noir, assemblées à l'aide de colonnes en plastique, de boulons et d'écrous.

Une couverture protège l'ensemble des composants du robot. Pour la fixation des moteurs, les roues, le Raspberry, l'écran tactile et le Lidar sur le chassis, nous avons procédé à la conception et la réalisation des supports pour chacun de ces composants. Les étages, les couvertures, les supports ont été conçus avec SolidWorks. Les étages ont été découpés à l'aide d'une machine CNC, tandis que les couvertures et les supports ont été fabriquées à l'aide d'une imprimante 3D.

2.3.1 Conception des pièces par SolidWorks

Pour la conception des pièces, nous avons utilisé le logiciel SolidWorks. Ce dernier est un logiciel de conception assistée par ordinateur (CAO) largement utilisé dans les domaines de l'ingénierie mécanique et de la conception industrielle. Développé par la société Dassault Systèmes, SolidWorks offre une plate-forme intuitive et puissante pour la modélisation 3D, la simulation, la visualisation et la documentation de produits. Il permet aux ingénieurs et aux concepteurs de créer rapidement et efficacement des modèles virtuels de composants et d'assemblages, facilitant ainsi le processus de conception et d'ingénierie.

Nous avons utilisé SolidWorks à plusieurs reprises pour concevoir diverses pièces de notre robot, notamment :

— **Le premier étage :** Il constitue la base du robot et permet de fixer la majorité des éléments électroniques (Figure 2.2).

— **Le deuxième étage :** Il protège les composants du premier étage et offre des emplacements pour le LiDAR et l'écran avec la Raspberry Pi (Figure 2.3) .

— **La couverture du châssis :** Elle protège les éléments électroniques tout en apportant un aspect esthétique au robot (Figure 2.4).

— **Le support du LiDAR :** Il fixe le LiDAR au deuxième étage, garantissant la stabilité et le bon fonctionnement du capteur (Figure 2.5).

— **Les supports des moteurs :** Ils fixent les moteurs sous le premier étage (Figure 2.6).

— **Les supports des roues :** Ils assurent la liaison entre les roues et les bras des moteurs. Ils assurent qu'il n'y a aucun glissement dans la rotation des roues par rapport aux arbres des moteurs (Figure 2.7).

— **Le support de la Raspberry Pi** : (La Figure 2.8) Protection du Raspberry Pi et permet de le fixer avec l'écran.

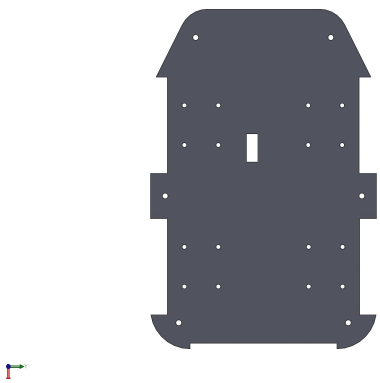


FIG. 2.2 – Premier étage du châssis

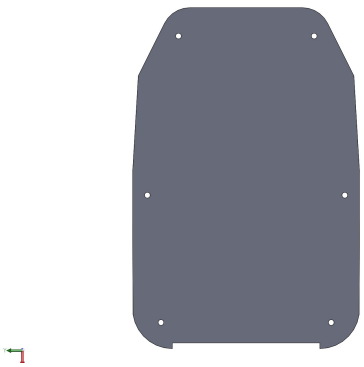


FIG. 2.3 – Deuxième étage du châssis

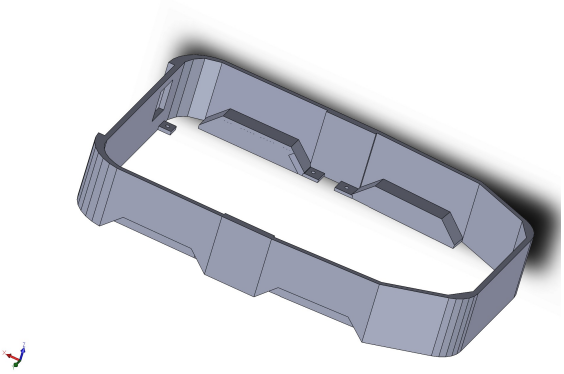


FIG. 2.4 – Couverture du châssis

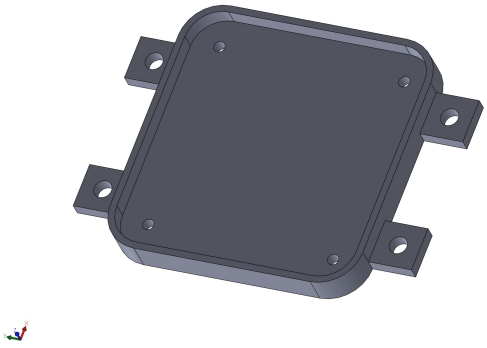


FIG. 2.5 – Support LiDAR

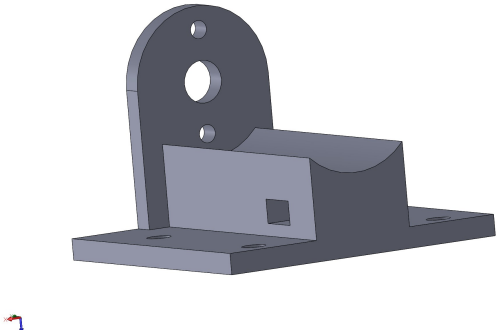


FIG. 2.6 – Support moteur

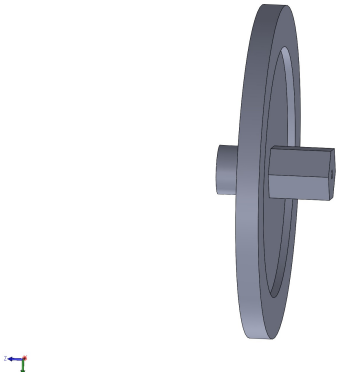


FIG. 2.7 – Support roue

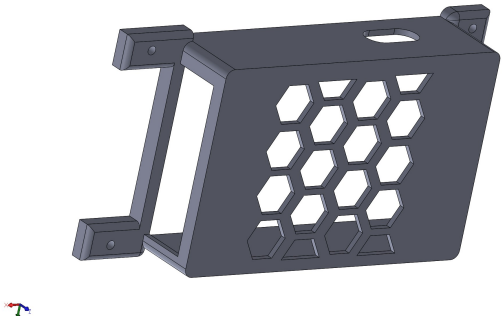


FIG. 2.8 – Support Raspberry Pi

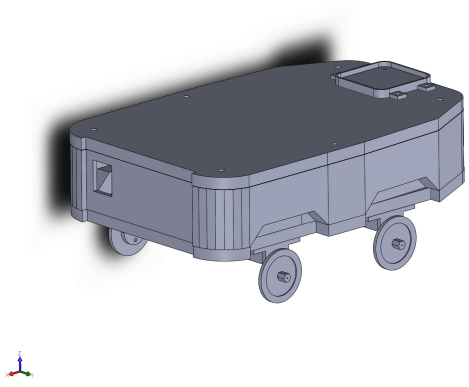


FIG. 2.9 – Assemblage des pièces

Ces éléments ont été soigneusement conçus pour garantir une intégration harmonieuse et des performances optimales du robot, comme illustré dans la figure 2.9.

2.3.2 Réalisation des pièces par CNC

La machine de découpe CNC par laser (la figure 2.10) est un équipement automatisé utilisé pour découper des matériaux tels que le bois, le plastique, le métal et bien d'autres, en utilisant un faisceau laser contrôlé par ordinateur. Pour utiliser une machine de découpe CNC par laser, il faut d'abord créer ou importer un fichier de conception sur un logiciel de conception assistée par ordinateur (CAO) tel que SolidWorks. Ensuite, ce modèle est converti en un format de fichier compatible avec la machine CNC (fichier DXF). Une fois cela fait, ce fichier est transféré à la machine qui contrôle le mouvement du faisceau laser pour découper précisément le matériau selon les spécifications du modèle.

Nous avons utilisé la machine de découpe CNC par laser dans la réalisation du châssis de notre robot mobile, notamment pour découper les étages du châssis en forex noir. Cette machine nous a permis d'obtenir des résultats précis et de haute qualité, garantissant une découpe nette et conforme aux spécifications du design. Son utilisation a grandement facilité le processus de fabrication du châssis, nous permettant de créer des pièces sur mesure avec une grande précision.

Fig. 2.10 – Machine de découpe CNC par laser

2.3.3 Réalisation des pièces par l'imprimante 3D

Une imprimante 3D (Figure 2.11) est un dispositif utilisé pour créer des objets tridimensionnels en ajoutant successivement des couches de matériau, tel que le plastique, le métal ou la résine, selon un modèle numérique spécifié. Pour exploiter l'imprimante 3D, on doit d'abord concevoir ou télécharger un modèle 3D sur un logiciel de modélisation assistée par ordinateur (CAO) SolidWorks par exemple. Ensuite, ce modèle est converti en un format de fichier compatible avec l'imprimante 3D, généralement un fichier STL. Après avoir chargé le matériau dans l'imprimante, celle-ci chauffe le matériau et le dépose couche par couche selon les spécifications du modèle, créant ainsi l'objet souhaité.

Nous avons utilisé l'imprimante 3D représentée sur la figure 2.11 dans la conception et la réalisation de divers composants du châssis de notre robot mobile, incluant le couvercle et tous les supports nécessaires pour la fixation des composants du robot sur

le châssis. Cette technologie permet une fabrication personnalisée et précise des pièces, répondant ainsi aux exigences spécifiques du projet.

FIG. 2.11 – Imprimante 3D

Après la réalisation de toutes ces pièces, nous avons procédé à l'assemblage final de la structure du robot. Le robot réalisé est décrit sur la figure 2.12 et ces caractéristiques sont présentés dans le tableau 2.1.

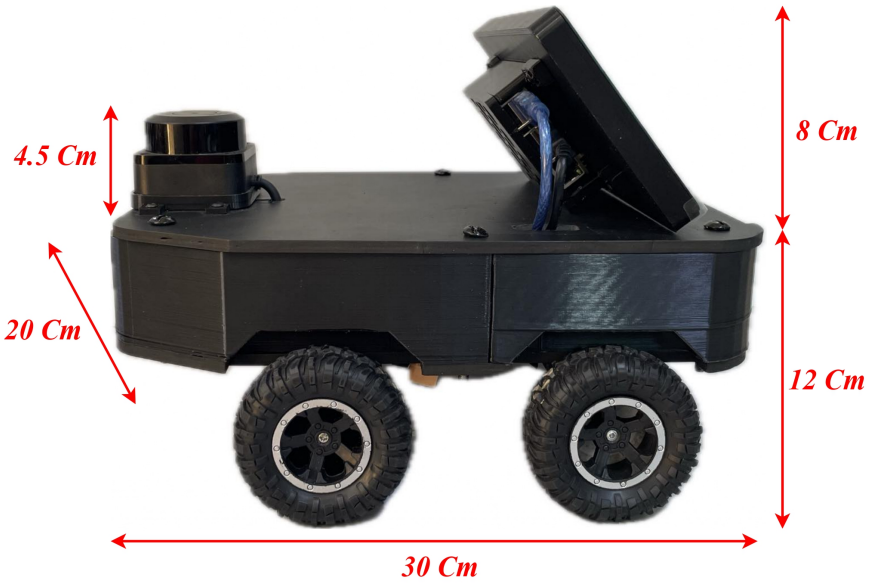


FIG. 2.12 – Dimensions du robot

Dimensions	$30cm \times 20cm \times 20cm$
Poids	environ 5 Kg
Châssis	Plaques en Forex Noir et Couvert en PLA
Roues	50 mm en caoutchouc
Charge maximale	Environ 7 Kg

TAB. 2.1 – Caractéristiques mécaniques du robot

2.4 Architecture électronique proposée

L'architecture matérielle proposée a pour rôle d'assurer la mise en marche du robot afin qu'il accomplisse une tache désirée qui est la navigation autonome dans notre projet. Elle est constituée de composants et de circuits électroniques reliés assurant ainsi la réalisation de cette architecture matérielle. Elle est composée principalement de deux (2) segments comme illustré sur la figure 3.14 : la commande haut-niveau et la commande bas-niveau. Le segment haut-niveau de l'architecture gère les algorithmes de navigation et de guidage de la plateforme robotique, tandis que le segment bas-niveau se charge d'exécuter les commandes envoyées par le segment haut-niveau, en supervisant les quatre moteurs de la plateforme robotique.

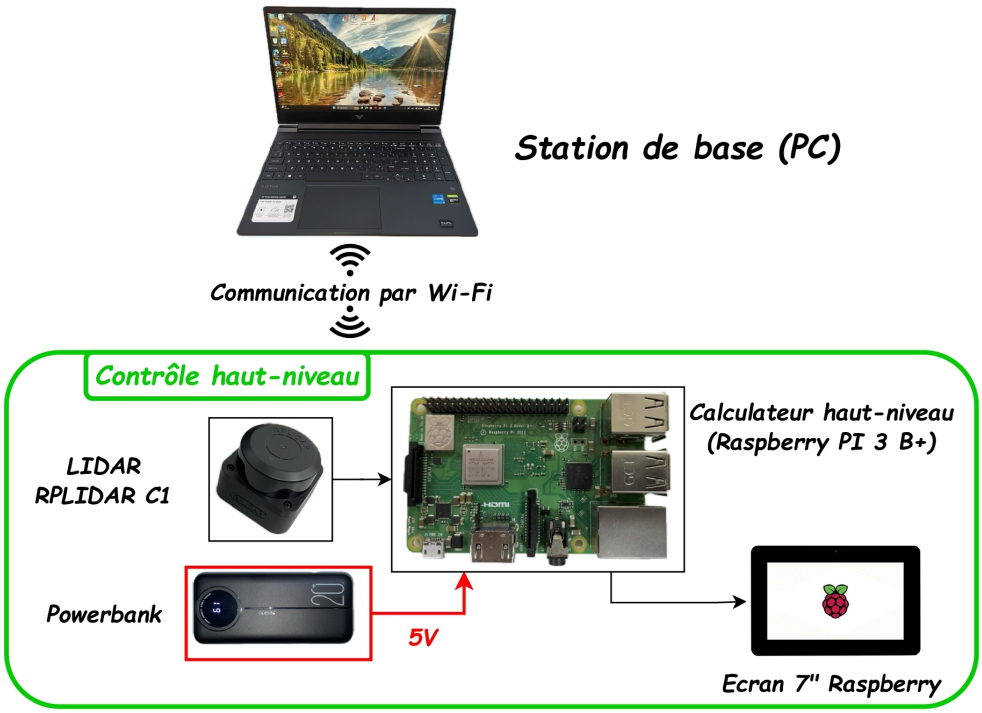


FIG. 2.13 – Segment haut-niveau

2.4.1 Segment Haut-niveau

Le segment haut-niveau (Figure 2.13) est constitué d'un calculateur Raspberry Pi3 modèle B+ qui communique avec la station sol (PC) via une connexion sans fil (WIFI), et est associé à un écran tactile 7 pouces compatible avec le Raspberry et un capteur extéroceptif RPLiDAR C1. Le Raspberry est alimenté par un power bank qui est une source de tension continue de 5v. Cette dernière assure une alimentation sécurisée pour le Raspberry.

Calculateur Raspebrry 3B+

Le Raspberry Pi est un ordinateur compact équipé du système d'exploitation Linux, conçu pour des applications d'informatique embarquée. Il peut être connecté directement à une interface utilisateur classique, telle qu'une souris, un clavier et un écran HDMI ou vidéo composite. En tant qu'ordinateur Linux, le Raspberry Pi offre également des outils de développement intégrés et une interface utilisateur accessible via SSH, permettant un contrôle à distance depuis un autre ordinateur via Ethernet ou WiFi [12]. Le segment haut-niveau de notre robot assurant la navigation et la commande de la plateforme robotique est à base d'un calculateur de type Raspberry Pi 3 modèle B+ illustré dans la figure 2.15.

Le Raspberry Pi 3 modèle B+ est la troisième génération de la gamme Raspberry Pi. Son processeur Broadcom BCM2837B0 bénéficie d'une augmentation de 200 MHz par rapport à son prédécesseur, offrant ainsi une exécution des calculs plus rapide et plus fluide [8]. Le tableau 2.2 résume les différentes caractéristiques de ce composant [7].

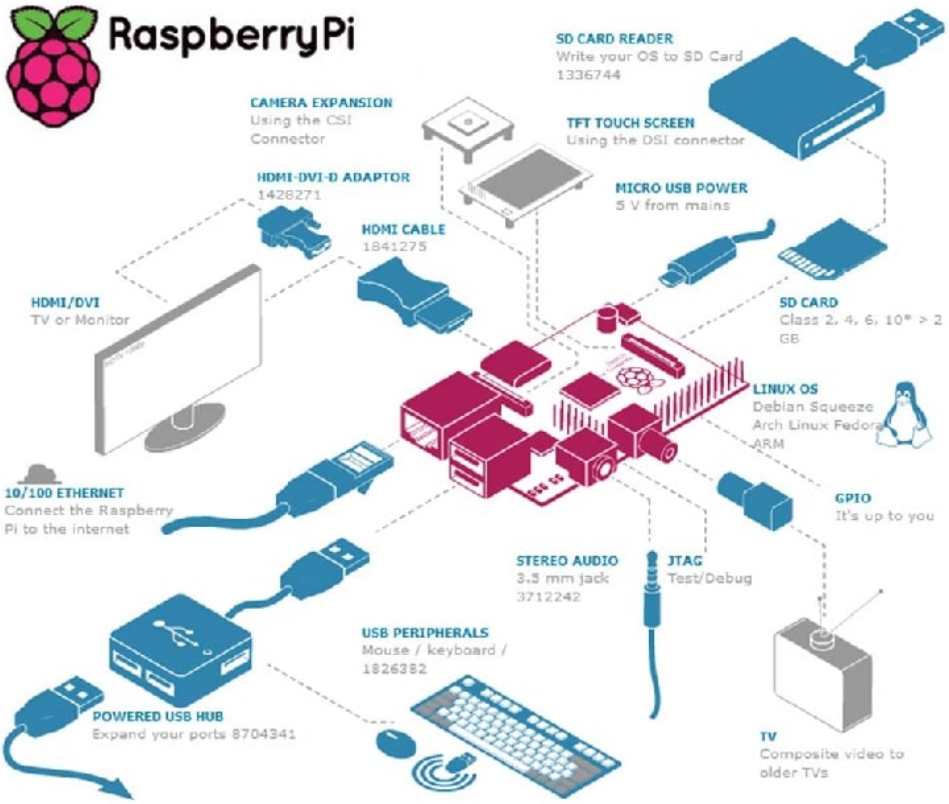


FIG. 2.14 – Architecture de connexions du Raspberry Pi

Fig. 2.15 – Raspberry Pi 3 modèle B+

Processeur	Broadcom BCM2837B0 Cortex-A53 64-bit SoC 1,4 GHz
Mémoire (SDRAM)	1GB LPDDR2
Nombre de ports USB	4 USB 2.0 ports
Port extension	GPIO 40 pins
Sorties vidéos	Port CSI pour la caméra Raspberry Pi Port DSI pour l'écran tactile Raspberry Pi HDMI
Sorties audio	Jack 3,5mm Femelle Stéréo
Sauvegarde des données	Carte MicroSD
Connexion sans fil	Bluetooth 4.2 BLE Wi-Fi Dual Band b/g/n/ac
Connexion filaire	Gigabit Ethernet
Alimentation	5V/2.5A via micro-USB
*Dimensions	$85mm \times 56mm \times 17mm$
*Poids	45 g
Prix	Environ 19,000.00 DA

Tab. 2.2 – Caractéristiques du Calculateur Raspberry Pi 3 modèle B+

Module de détection : RPLiDAR C1

Dans le cadre de notre projet de fin d'études, nous avons choisi d'utiliser un LiDAR de type RPLiDAR C1 (Figure 2.17) plutôt qu'une caméra. Cette décision découle de plusieurs avantages spécifiques du LiDAR dans le contexte de la navigation autonome des robots mobiles. Contrairement aux caméras, qui sont fortement tributaires des conditions d'éclairage ambiant, le LiDAR fournit des données de distance précises et fiables, même dans des conditions de faible luminosité ou de visibilité réduite. De plus, le LiDAR excelle dans la détection des obstacles en trois dimensions, une capacité essentielle pour une navigation efficace dans des environnements complexes ou encombrés [37].

Ce choix technologique est donc en accord avec notre objectif de développer un système de navigation robuste et fiable, capable de fonctionner dans une variété de conditions environnementales.

— **Principe de fonctionnement du LiDAR**

Le LiDAR utilise la lumière pour évaluer les distances entre le capteur et les objets environnants comme illustré dans la figure 2.16. Voici un aperçu global de son fonctionnement et de ses utilisations :

- **Émission du laser :** Le LiDAR envoie des impulsions laser vers la cible ou la zone à analyser.
- **Réflexion du signal :** Les objets renvoient le signal lorsque les impulsions laser les touchent, et celui-ci est dirigé vers le capteur.
- **Détection du signal :** Le capteur du LiDAR détecte les signaux réfléchis.
- **Calcul des distances :** Les distances sont calculées en mesurant précisément le temps nécessaire à chaque impulsion pour revenir au capteur. Étant donné que la vitesse de la lumière est constante, la distance entre le LiDAR et l'objet est calculée en utilisant la formule :

$$D = (C \times t)/2 \tag{2.9}$$

— **D** : La distance entre le capteur LiDAR et l'objet.
— **C** : La vitesse de la lumière, qui est approximativement 3×10^8 mètres par seconde (m/s).
— **t** : Le temps que met l'impulsion lumineuse pour effectuer l'aller-retour entre le LiDAR et l'objet.

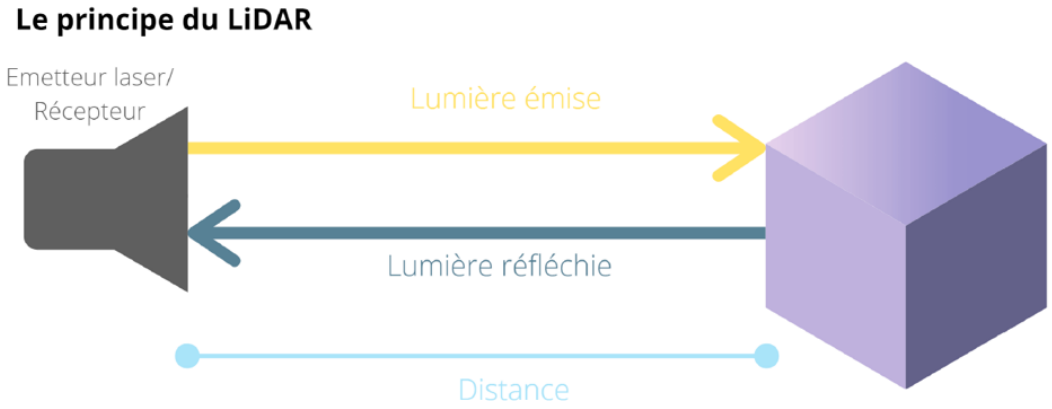


Fig. 2.16 – Principe de fonctionnement du LiDAR [25]

— **Applications de LiDAR** Ce système trouve son utilité dans divers scénarios d'application, notamment :

- La localisation et la cartographie simultanées générales (SLAM)
- Le balayage de l'environnement et la modélisation 3D
- L'utilisation par des robots de service ou industriels, même sur de longues périodes
- La navigation et la localisation de robots utilisés pour le service ou le nettoyage domestique
- La navigation et la localisation générales de robots
- La localisation et l'évitement d'obstacles pour les jouets intelligents

— **Caractéristiques du RPLiDAR C1**

Le RPLiDAR C1, produit par SLAMTEC illustré sur la figure 2.17, est un scanner laser 2D à 360 degrés de nouvelle génération, caractérisé par son faible coût et ses performances avancées. Grâce à sa capacité à réaliser jusqu'à 5000 échantillons de télémétrie laser par seconde et à sa vitesse de rotation élevée, il offre une acquisition rapide et précise des données.

Le RPLiDAR C1 offre une portée de mesure allant jusqu'à 12 mètres de rayon et présente une faible plage de cécité de seulement 0,05 mètre. Sa capacité à scanner et mesurer des objets à différentes distances facilite l'évitement des obstacles. Le tableau 2.3 résume les caractéristiques de RPLiDAR C1.

Fig. 2.17 – LiDAR "RPLiDAR C1"

Distance de réflexion	**-** ***Objet blanc :*** 0,05 à 12 mètres (sous une réflexion de 70 %) **-** ***Objet noir :*** 0,05 à 6 mètres (sous une réflexion de 10 %)
Taux d'échantillonnage	5KHz
Fréquence de balayage	8 à 12Hz
Résolution angulaire	0.72°
Planéité du champ de balayage	0°∼ 1.5°
précision	±30 mm
Résolution	15 mm
Dimensions	$55.6mm \times 55.6mm \times 41.3mm$
Protection	IP54
Prix	Environ 20,000.00 DA

Tab. 2.3 – Caractéristiques du RPLiDAR C1

Affichage : Raspberry Pi LCD - 7" TOUCHSCREEN

L'écran tactile Raspberry Pi LCD - 7" TOUCHSCREEN (la figure 2.18) est un périphérique d'affichage conçu spécifiquement pour les ordinateurs Raspberry Pi. Cet écran de 7 pouces offre une résolution de 800 x 480 pixels et supporte la fonctionnalité multi-touch à 10 points, ce qui permet une interaction intuitive avec le système.

Dans notre projet, nous avons utilisé l'écran tactile Raspberry Pi LCD - 7" TOUCHSCREEN pour l'affichage des cartes sur Rviz, un outil de visualisation dans ROS. Cet écran permet de visualiser en temps réel les données de navigation et les cartes générées par le robot. Grâce à sa fonctionnalité tactile, nous pouvons interagir directement avec le système, ce qui facilite la programmation et le contrôle du robot. De plus, il offre une interface pratique pour programmer le robot directement depuis la Raspberry Pi, simplifiant la modification des paramètres, l'exécution de scripts et la gestion des processus de contrôle sans nécessiter de périphériques d'entrée supplémentaires.

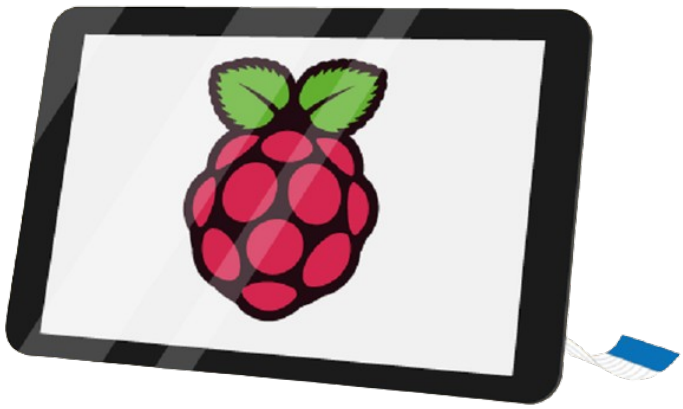


FIG. 2.18 – Raspberry Pi LCD - 7" TOUCHSCREEN

— **Caractéristiques Raspberry Pi LCD - 7"** Le tableau 2.2 résume les caractéristiques de notre écran.

Format de l'écran	800 (RGB) × 480 pixels
Taille de l'écran (diagonale)	7 pouces
Dimensions	$194mm \times 110mm \times 20mm$
Poids	200.00g
Prix	Environ 21,000.00 DA

TAB. 2.4 – Caractéristiques Raspberry Pi LCD - 7" [9]

Power bank

Pour notre projet de réalisation d'un robot mobile à 4 roues, nous avons choisi d'alimenter notre calculateur haut niveau (le Raspberry Pi 3), en utilisant une power bank (comme celui de la figure 2.19) plutôt qu'une batterie LiPo de 11.1V. Une power bank est un dispositif portable de stockage d'énergie utilisé pour recharger ou alimenter des appareils électroniques en l'absence de source d'alimentation directe. Elle contient une

batterie rechargeable interne et dispose de ports USB ou micro-USB pour connecter divers appareils. La power bank que nous avons sélectionnée possède une capacité de 20 000 mAh et fournit une tension de sortie de 5V, parfaitement adaptée aux besoins énergétiques du Raspberry Pi 3.

Cette power bank est également équipée d'un affichage numérique de la batterie, ce qui nous permet de surveiller en temps réel le niveau de charge restant. Cela garantit une alimentation stable et continue pour le Raspberry Pi, indispensable pour les opérations de traitement et de contrôle de notre robot. En optant pour cette solution, nous avons pu éviter les complications liées à la conversion de tension nécessaire avec une batterie de 11.1V, simplifiant ainsi l'intégration et augmentant la fiabilité de notre système.

Fig. 2.19 – Power bank 20.000 mAh

2.4.2 Segment bas-niveau

Quant au segment bas-niveau (Figure 2.20), il est composé d'un microcontrôleur Arduino Méga qui contrôle les quatre moteurs de la plateforme mécanique à l'aide d'un étage de puissance de type L298N. Pour mettre en place un module d'odométrie et garantir l'asservissement en vitesse des moteurs, nous avons utilisé quatre moteurs de types JGA25-370 équipés d'encodeurs.

Le système embarqué est alimenté par deux batteries fournissant respectivement des tensions de 5V et 11,1V. Une batterie 5V, sous forme d'un power-bank rechargeable, alimente le calculateur Raspberry Pi. La batterie 11,1V (polymères de lithium d'une capacité de 5200 mAh) alimente les moteurs. Cette configuration de double alimentation assure une protection pour le système embarqué en isolant les parties à basse puissance et à haute ou moyenne puissance, tout en garantissant une meilleure autonomie globale du système. Le rôle principale dédié à ce segment est d'assurer le contrôle bas niveau du robot c'est à dire l'asservissement en vitesses des quatre moteurs du robot. Dans ce qui suit, nous allons détailler chaque composant du segment bas-niveau.

Calculateur bas-niveau : Arduino Méga

Arduino est une plateforme open-source de développement électronique composée de cartes matérielles et d'un environnement de développement logiciel. Elles sont équipées de microcontrôleurs programmables qui peuvent être utilisés pour contrôler divers composants électroniques et créer une grande variété de projets, allant des simples capteurs aux systèmes embarqués complexes.

Les cartes Arduino sont disponibles dans une variété de types et de versions, chacune ayant des fonctionnalités et des spécifications uniques adaptées à différents besoins.

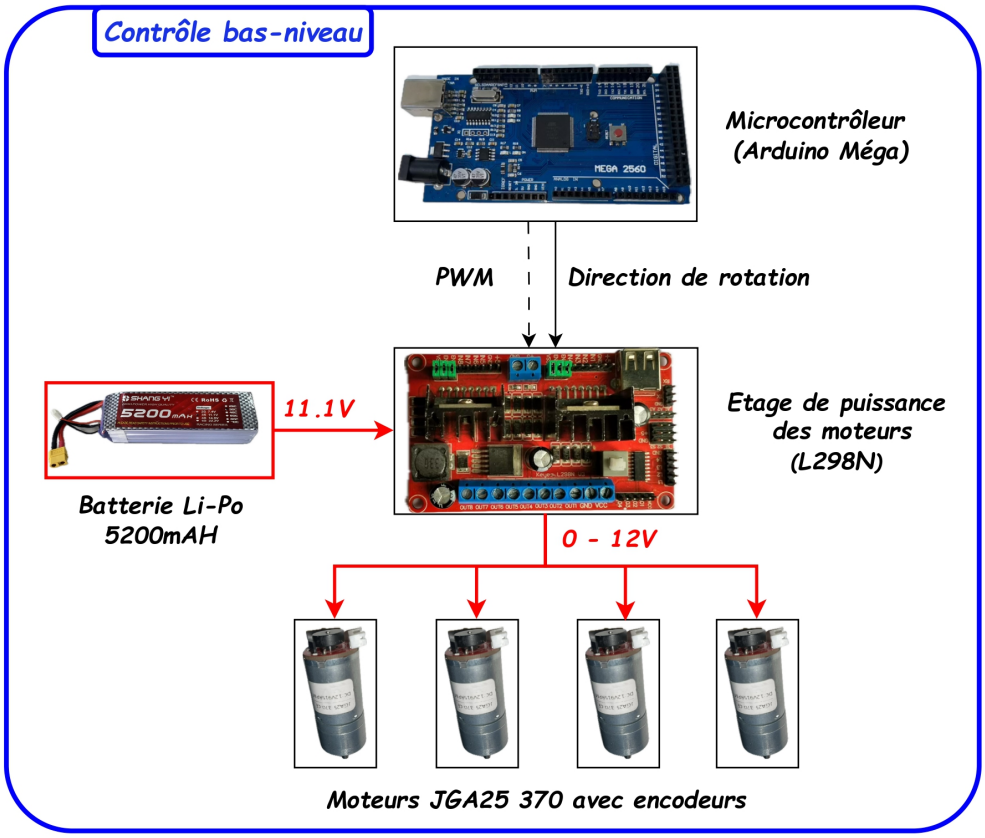


FIG. 2.20 – Segment bas-niveau

L'Arduino Uno est souvent utilisé pour les projets d'apprentissage et les applications de base, tandis que l'Arduino Mega offre une plus grande puissance de traitement et des entrées/sorties supplémentaires pour des projets plus complexes.

Pour réaliser le segment bas-niveau, nous avons sélectionné un microcontrôleur Arduino Mega comme celui présenté sur la figure 2.21 en raison de ses capacités en entrées/sorties requises pour notre architecture. De plus, ce choix présente plusieurs avantages, notamment une programmation aisée via un environnement de développement intégré gratuit, ainsi qu'une grande popularité au sein d'une vaste communauté de développeurs. Cette popularité garantit l'accès à une multitude de bibliothèques et de ressources en ligne, facilitant ainsi l'intégration de nouveaux composants. Le tableau 2.5 résume les caractéristiques de l'Arduino Méga.

FIG. 2.21 – Arduino Mega

Microcontrôleur	ATmega2560
Tension de fonctionnement	5 V
Tension d'alimentaion (recommandée)	7 - 12 V
Tension d'alimentaion (limites)	6 - 20 V
Broches d'E/S numériques	54 (15 sorties PWM)
Broches d'entrée analogique	16
Mémoire flash	256 Ko (8 Ko utilisés par le chargeur de démarrage)
Mémoire SRAM	8 kB
Mémoire EEPROM	4 kB
Vitesse horloge	16 MHz
Dimensions	$102mm \times 54mm \times 12mm$
Poids	36.8 g
Prix	Environ 4,000.00 DA

TAB. 2.5 – Caractéristiques microcontrôleur Arduino Mega [1]

L'étage de puissance des moteurs : module L298N

Le L298N (la figure 2.23) est un circuit intégré largement utilisé pour le contrôle d'un ou plusieurs moteurs. Il peut piloter des moteurs DC ayant une tension comprise entre 5V et 35V avec un courant de pic de 2 ampères (2A). Il s'agit d'un contrôleur de la direction de rotation des moteurs en utilisant la structure électronique appelée pont en H (H-bridge) (Figure 2.22). Cette structure permet de contrôler les moteurs de manière bidirectionnelle, en avant et en arrière, en inversant le courant.

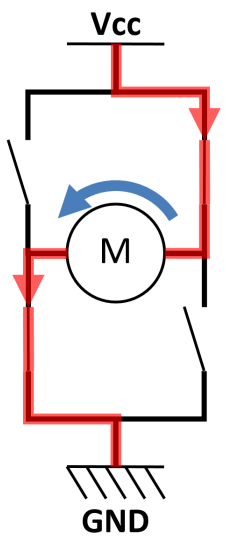


FIG. 2.22 – H bridge

Le module permet également de contrôler la vitesse des moteurs en utilisant une technique appelée modulation de largeur d'impulsion (PWM). Cette technique nous permet de réguler la valeur moyenne de la tension alimentant le Mosfet en alternant rapidement l'alimentation activée et désactivée. La valeur moyenne de la tension dépend du rapport cyclique, c'est-à-dire de la proportion de temps pendant lequel le signal est activé (ou désactivé) par rapport à la durée totale d'une période.

FIG. 2.23 – L298N Motor Driver

— **Caractéristiques du module L298N**

Circuit intégré	L298N (Pont double H)
Tension d'alimentation nominale du moteur	5 - 30 V
Courant maximum d'alimentation du moteur	2 A
Courant de commande	0 - 36 mA
Puissance maximale	20W (T = 75 °C)
Dimensions	$82mm \times 45mm \times 33mm$
Poids	100 g
Prix	Environ 2,700.00 DA

TAB. 2.6 – Caractéristiques du module L298N [5]

Le branchement détaillé des pins entre le calculateur Arduino Méga et l'étage de puissance L298N est représenté sur la figure 2.24. Tandis que l'identification du branchement de tous les pins est détaillée dans le tableau 2.7.

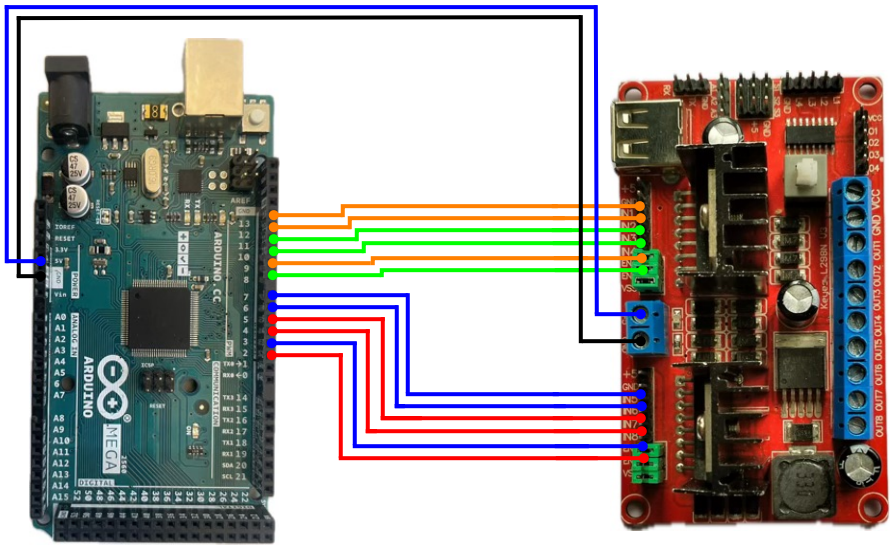


FIG. 2.24 – Schéma de branchement L298-Arduino

Pin	Description
12 V	Entrée 12V de la source d'alimentation DC (Batterie)
5 V	***Fournit l'énergie pour le circuit logique de commutation du L298N***
GND	Ground pin
IN1 et IN2	Broches de sens de rotation du moteur 1
IN3 et IN4	Broches de sens de rotation du moteur 2
IN5 et IN6	Broches de sens de rotation du moteur 3
IN7 et IN8	Broches de sens de rotation du moteur 4
ENA et ENB	Active le signal PWM pour moteur 1/3 et 2/4 respectivement (contrôle de vitesse)
OUT1 et OUT2	Broches de sortie du moteur 1
OUT3 et OUT4	Broches de sortie du moteur 2
OUT5 et OUT6	Broches de sortie du moteur 3
OUT7 et OUT8	Broches de sortie du moteur 4

TAB. 2.7 – Pins du module L298N.

Moteur JGA25 370 avec encodeur

Le moteur JGA25-370 DC12V 915RPM avec encodeur (Figure 2.25) est un moteur électrique à courant continu (DC) qui fonctionne sous une tension de 12 volts. Sa vitesse maximale de rotation est de 915 tours par minute. Il est équipé d'un encodeur, un dispositif qui convertit le mouvement mécanique en signaux électriques pour permettre la mesure précise de la vitesse, de la direction et de la position du moteur. Ce type de moteur est largement utilisé dans les applications nécessitant un contrôle précis du mouvement, telles que les robots mobiles, les drones, les imprimantes 3D, les véhicules autonomes, etc. Le tableau 2.8 résume les caractéristiques de ce moteur.

Les encodeurs, intégrés aux moteurs de notre robot mobile, mesurent la position, la vitesse et la direction de rotation. Ils convertissent les mouvements en signaux électriques,

permettant un contrôle précis en boucle fermée. Les encodeurs incrémentaux suivent les variations de position, tandis que les absolus fournissent la position angulaire exacte. Ils assurent des déplacements précis et fiables du robot. Ils fournissent également des informations essentielles pour corriger les erreurs de mouvement en temps réel, garantissant ainsi une navigation efficace et précise.

FIG. 2.25 – Moteur JGA25 370 avec encodeur

— **Caractéristiques du moteur JGA25 370**

Tension nominal du moteur	DC 12V
Tension de travail	Entre 6 V et 18 V
Vitesse moteur à vide	915 tr/min
Courant moteur à vide	50 mA
Courant décrochage	1200 mA
Torque décrochage	1 kg.cm
Prix	Environ 3,000.00 DA

TAB. 2.8 – Caractéristiques moteur JGA25-370 [6]

— **Schéma de branchement Moteurs-L298N**

La figure 2.26 détaille le branchement des quatre moteurs du robot avec l'étage de puissance L298N.

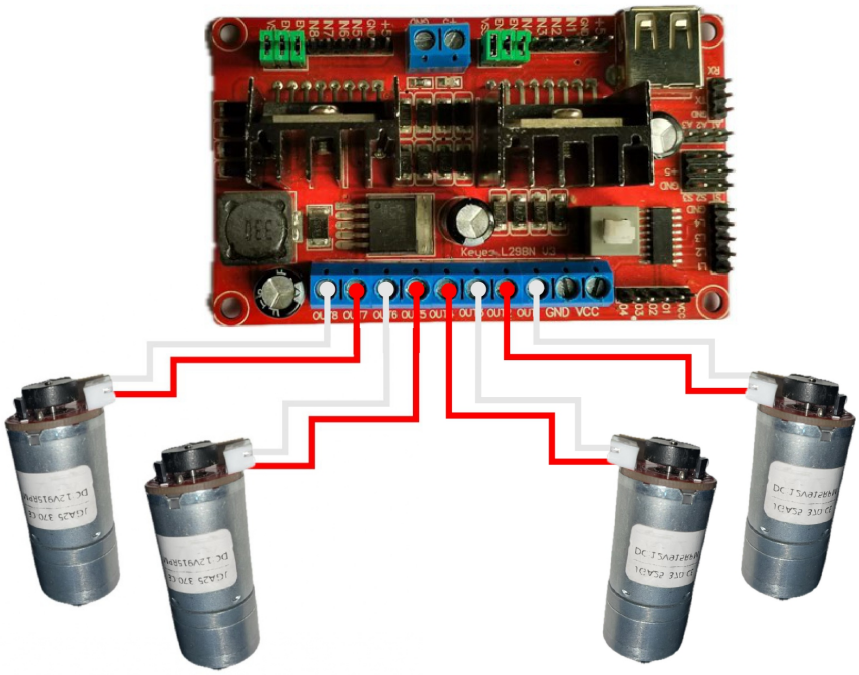


FIG. 2.26 – Schéma de branchement Moteurs-L298

— **Schéma de branchement Encodeurs-Arduino**

Les signaux issues des encodeurs sont branchés avec l'Arduino afin de les utiliser dans le calcul du déplacement et la vitesse actuelle du robot. La figure 2.27 détaille le branchement des encodeurs du robot avec l'Arduino.

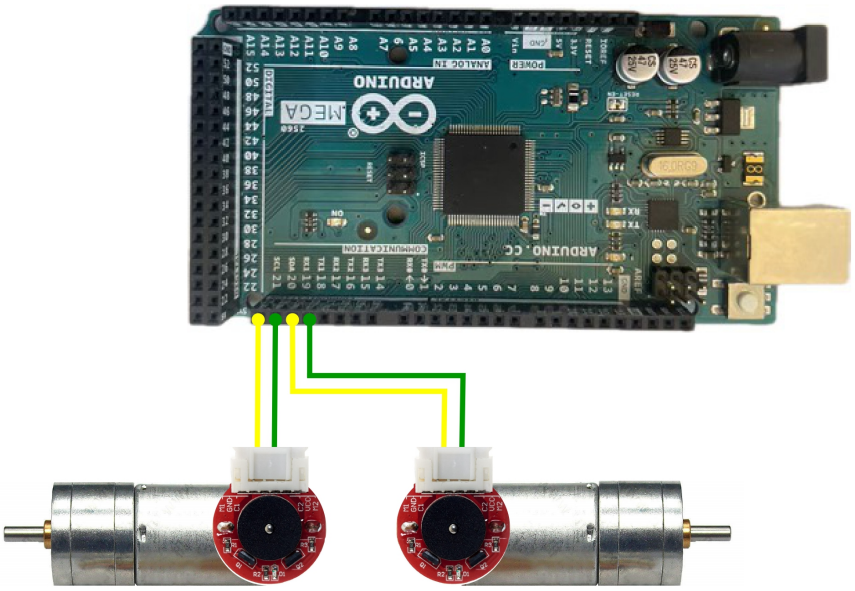


FIG. 2.27 – Schéma de branchement Encodeurs-Arduino

Batterie Lipo 11.1 V

Dans notre projet de réalisation d'un robot mobile à 4 roues, nous avons utilisé une batterie LiPo illustrée dans la figure 2.28 pour alimenter notre étage de puissance, le module L298N. Les batteries LiPo sont largement reconnues pour leur densité énergétique élevée, leur légèreté et leur capacité à fournir des courants élevés, ce qui les rend idéales pour des applications nécessitant une puissance importante et constante.

Notre batterie LiPo est caractérisée par une tension de 11.1V et une capacité de 5200 mAh. Ces caractéristiques offrent une alimentation stable et durable pour le module L298N, qui est responsable de la gestion et du contrôle des moteurs du robot. En utilisant cette batterie, nous avons assuré une performance optimale de l'étage de puissance, permettant au robot de fonctionner efficacement et de répondre aux exigences de puissance pendant les opérations de mobilité.

— **Caractéristiques du Batterie**

Tension de batterie	11.1 V
Capacité	5200 mAh
Décharge	35 C
Configuration	3S
Connecteur	Fiche XT60
Matériaux	Polymères de Lithium
Dimensions	$138mm \times 43.15mm \times 27.50mm$
Poids	368 g
Prix	Environ 13,000.00 DA

Tab. 2.9 – Caractéristiques du batterie [2]

Fig. 2.28 – Batterie

2.4.3 Réalisation globale du robot

Aprés la réalisation de la structure mécanique du robot et le branchement de l'architecture électronique proposée, nous procédons à l'assemblage des deux parties mécanique

et électronique pour réaliser le robot complet. Nous allons décrire cet assemblage en détaillant les composants électroniques placés dans chaque étage de la structure mécanique :

Sous le premier étage de notre robot (Figure 2.29), nous avons installé quatre moteurs de type JGA25 370 équipés d'encodeurs. Ces moteurs sont solidement fixés au châssis à l'aide de supports moteur spécialement conçus. Les quatre roues en caoutchouc sont attachées aux moteurs via des supports de roues. Toutes ces pièces, à savoir les supports de roues et les supports moteurs, ont été conçues à l'aide du logiciel SolidWorks et fabriquées à l'aide d'une imprimante 3D, garantissant ainsi une adaptation parfaite et une intégration optimale dans notre système robotique.

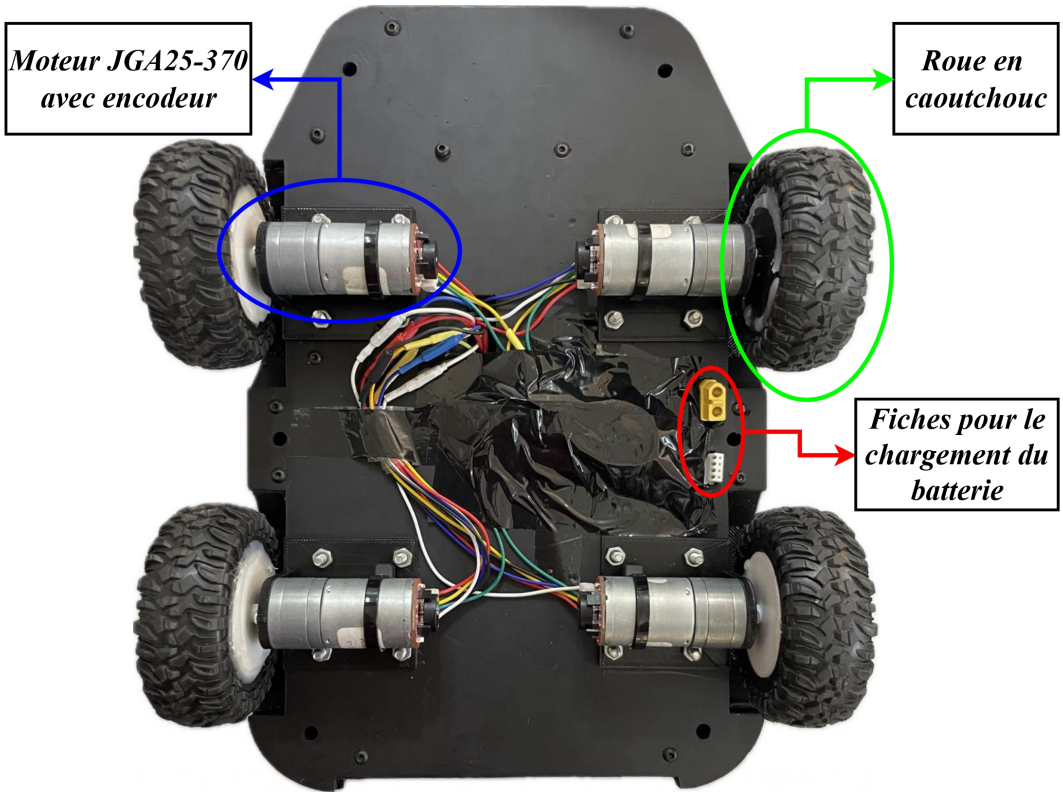


FIG. 2.29 – Vue de dessous du robot

Au premier étage (Figure 2.30), se trouvent les composants électroniques et de commande du robot, comprenant une carte Arduino, un driver L298N, une batterie et une power bank. Ces éléments sont interconnectés par des fils et des câbles électriques. Pour assurer leur stabilité et leur sécurité, ces composants sont fixés au châssis à l'aide de boulons, d'écrous et de colliers de serrage en plastique.

Au deuxième étage (Figure 2.31), un écran tactile Raspberry de 7 pouces est positionné sur un support incliné à un angle de 45 degrés. Sous cet écran se trouve notre Raspberry Pi 3, fixé avec l'écran grâce à un boîtier. À l'avant de l'étage, notre capteur LiDAR est installé sur un petit support. Les supports de l'écran et du LiDAR, ainsi que le boîtier, ont été conçus avec SolidWorks et fabriqués à l'aide d'une imprimante 3D.

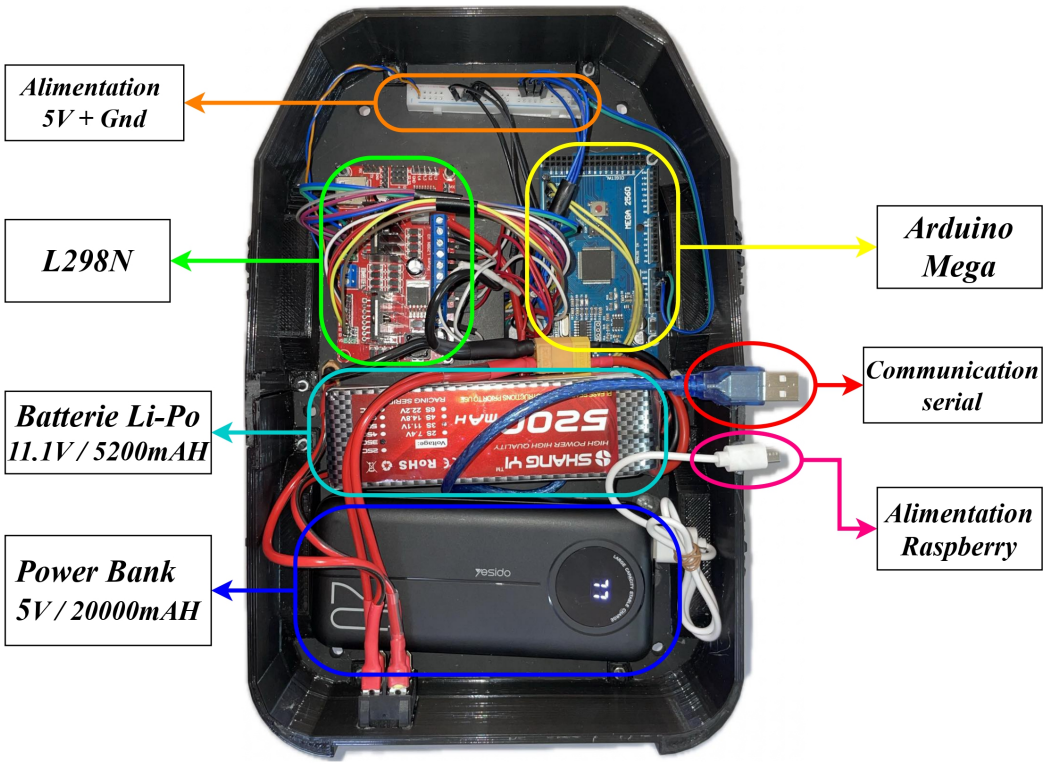


FIG. 2.30 – Premier étage

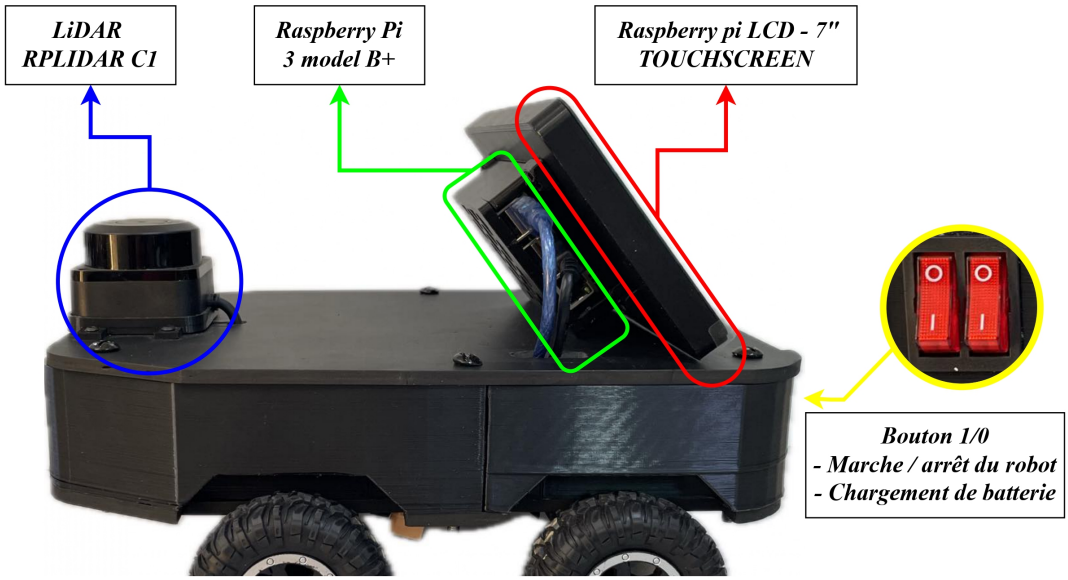


FIG. 2.31 – deuxième étage

2.5 Architecture logicielle proposée

Afin d'assurer la mise en marche de l'architecture matérielle proposée, nous avons développé l'architecture logicielle illustrée dans la figure 2.32 pour les trois calculateurs de notre système, notamment la station sol, le Raspberry et l'Arduino. Pour chaque calculateur, nous avons implémenté des logiciels permettant la mise en marche du système et la communication entre ces différentes parties. Dans ce qui suit, nous allons décrire les briques logiciels pour chaque calculateur.

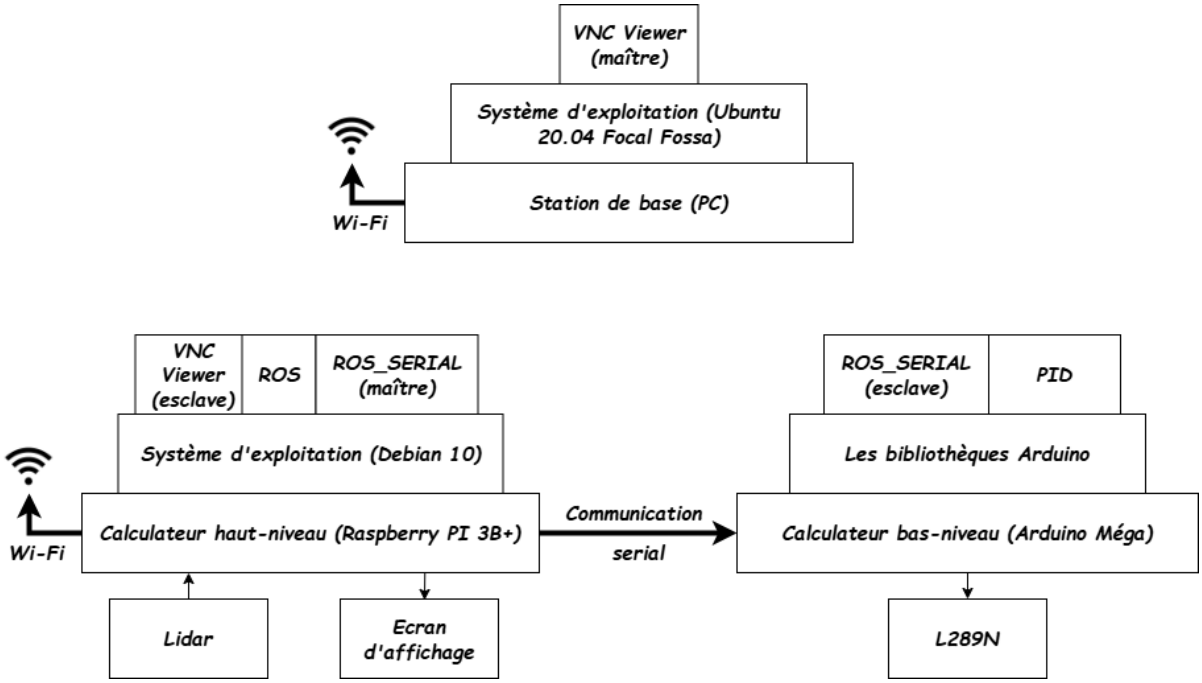


Fig. 2.32 – Les briques logiciels utilisées

2.5.1 Briques logiciels au niveau de la station sol (PC)

1. **Système d'exploitation Ubuntu 20.04 LTS (Focal Fossa)**
   Ubuntu est un système d'exploitation open source basé sur Linux, largement utilisé dans le monde entier pour ses performances, sa stabilité et sa sécurité. La version 20.04, également connue sous le nom de "Focal Fossa"(la figure 2.33), a été publiée en avril 2020 en tant que version LTS (Long Term Support), ce qui signifie qu'elle bénéficie d'un support étendu et de mises à jour de sécurité pendant cinq ans.

   Caractéristiques principales du système :

   — **Basé sur Linux :** Ubuntu est construit sur le noyau Linux, héritant ainsi de ses qualités intrinsèques telles que la fiabilité, la sécurité, et la compatibilité étendue avec divers matériels et logiciels.

   — **Caractère Open Source :** Distribué sous une licence open source, Ubuntu permet à tous d'accéder librement à son code source, favorisant ainsi la transparence, l'innovation, et la collaboration au sein de la communauté du développement logiciel.

   — **Accessibilité :** L'interface conviviale d'Ubuntu et son processus d'installation simplifié en font une option accessible même pour les utilisateurs débutants.

En outre, il est livré avec une gamme d'applications pré-installées, prêtes à être utilisées dès l'installation.

— **Support à Long Terme :** Les versions LTS d'Ubuntu, comme la 20.04, bénéficient d'un soutien prolongé, avec des mises à jour de sécurité et des corrections de bogues garanties pendant cinq ans à compter de leur publication. Cette stabilité en fait un choix populaire pour les déploiements à long terme et les environnements professionnels.

— **Écosystème Logiciel Étendu :** Ubuntu propose un large éventail de logiciels grâce à son vaste écosystème, avec des milliers d'applications disponibles dans ses dépôts officiels. Cela comprend des outils de productivité, des environnements de développement, des suites bureautiques, des serveurs, et plus encore, offrant ainsi une plateforme polyvalente adaptée à diverses utilisations.

Ubuntu 20.04 LTS, choisi pour notre projet de robotique, se démarque comme un système d'exploitation solide et flexible. Son architecture open source offre la possibilité d'intégrer le ROS sur notre station sol, ce qui n'est pas toujours le cas avec d'autres systèmes d'exploitation. La version Ubuntu 20.04 a été préférée pour sa stabilité et sa récence, assurant ainsi un environnement fiable et à jour pour le développement de notre projet.

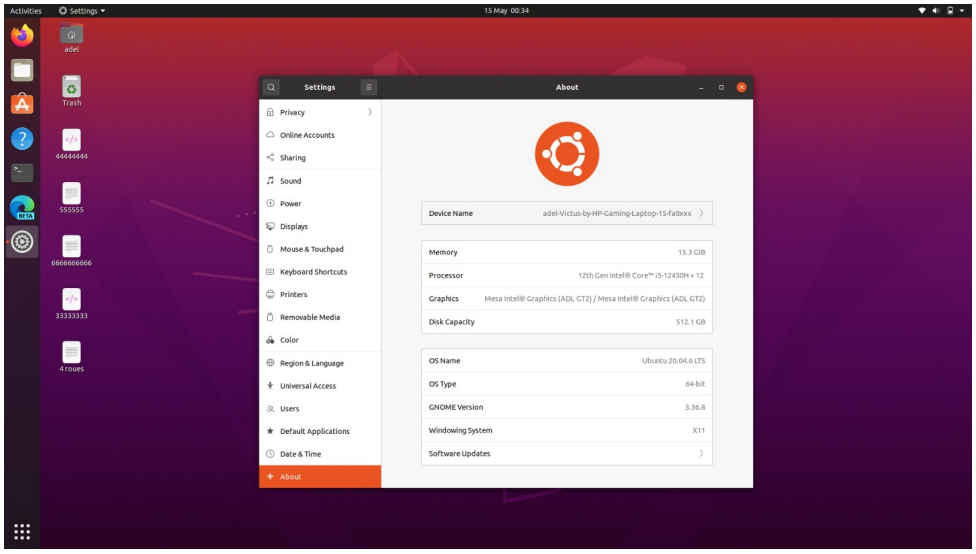


FIG. 2.33 – Interface du système Ubuntu 20.04

2. **Matlab**

Dans le cadre de notre projet de réalisation d'un robot mobile à 4 roues et afin de visualiser les différents résultats, nous avons utilisé Matlab, un logiciel de calcul numérique puissant et polyvalent. Matlab est couramment utilisé dans divers domaines pour l'analyse de données, le développement d'algorithmes et la création de modèles. Ses capacités de visualisation de données et de génération de graphiques en font un outil idéal pour notre projet.

Nous avons spécifiquement employé Matlab pour la visualisation des graphes, notamment les graphiques de vitesse du robot. Grâce à ses fonctionnalités avancées, nous avons pu représenter graphiquement les variations de vitesse du robot, ce qui nous a permis d'analyser et d'optimiser les performances en temps réel. L'utilisation

de Matlab a facilité l'interprétation des données et a contribué à une meilleure compréhension des dynamiques du système de contrôle du robot.

3. **VNC Viewer**
   Le logiciel VNC Viewer (Figure 2.34) est un outil utilisé pour la visualisation et le contrôle à distance de l'environnement de bureau d'un ordinateur, tel que le Raspberry Pi, à partir d'une station sol distante. Conçu pour permettre un accès distant facile et sécurisé, VNC Viewer utilise le protocole VNC pour établir une connexion entre l'ordinateur hôte et le dispositif client. Cette connexion permet à l'utilisateur de voir et d'interagir avec l'écran du Raspberry Pi, ainsi que d'exécuter des commandes et des applications à distance, comme s'il était physiquement présent devant l'appareil. Grâce à VNC Viewer, les utilisateurs peuvent gérer et contrôler efficacement leur Raspberry Pi depuis n'importe où, ce qui en fait un outil précieux pour la surveillance, la maintenance et le dépannage à distance des systèmes informatiques.

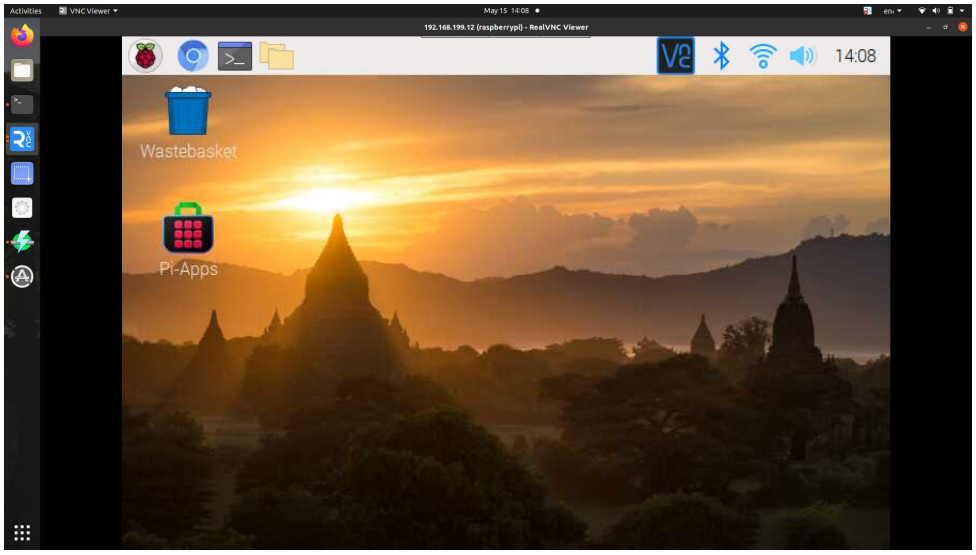


FIG. 2.34 – Interface du logiciel VNC

2.5.2 Briques logiciels du Raspberry PI 3

1. **Debian 10**
   Debian 10, aussi connu sous le nom de code "Buster", est une distribution de système d'exploitation libre de la famille GNU/Linux. Publiée officiellement le 6 juillet 2019, cette version de Debian est réputée pour sa stabilité, sa sécurité et sa robustesse. Debian 10 inclut une large gamme de logiciels, allant des outils de développement aux applications bureautiques, et bénéficie de mises à jour régulières assurant la sécurité et la fiabilité du système. En outre, Debian est maintenu et développé par une communauté mondiale de volontaires et bénéficie d'un support à long terme.

   Dans notre premier essai, nous avons installé Ubuntu 20.04 sur notre calculateur de haut niveau Raspberry Pi en raison de sa simplicité d'utilisation. Cependant, nous avons rencontré plusieurs difficultés et un blocage majeur lors de l'exécution. En conséquence, nous nous sommes tournés vers Debian 10. Bien que Debian 10 soit légèrement plus complexe à utiliser, il offre une exécution plus rapide et plus stable pour notre application.

Caractéristiques principales du système :

- **Stabilité :** Debian est réputée pour sa stabilité. Chaque version est soigneusement testée avant sa sortie, ce qui en fait un choix fiable pour les environnements de production.
- **Sécurité :** Debian inclut des mises à jour de sécurité régulières pour garantir la sécurité du système.
- **Vaste dépôt de logiciels :** Debian propose plus de 57 000 paquets logiciels, couvrant une large gamme d'applications.
- **Support matériel :** Debian 10 supporte une variété de matériels, y compris les architectures ARM, ce qui est idéal pour des systèmes comme Raspberry Pi.
- **Environnement de bureau :** Debian 10 offre plusieurs environnements de bureau, tels que GNOME, KDE Plasma, XFCE, et plus encore.

2. **Robot Operating System (ROS)**

Dans notre projet de réalisation d'un robot mobile, nous avons utilisé le Robot Operating System (ROS), qui est un cadre logiciel essentiel pour le développement de logiciels robotiques. Cependant, nous détaillerons ce logiciel de manière approfondie dans le prochain chapitre, en expliquant ses fonctionnalités, ses avantages et la manière dont nous l'avons intégré dans notre projet.

2.5.3 Briques logiciels d'Arduino Méga

1. **Application de contrôle embarquée bas-niveau**

L'application de contrôle de bas niveau est illustrée dans la figure 2.35. Elle garantit l'exécution des consignes envoyées par le calculateur de haut niveau. Cette couche de bas niveau reçoit les commandes de vitesse de translation et de rotation (V_{trans} et V_{rot}) en s'inscrivant dans le topic *cmd_vel*.

L'application de contrôle de bas niveau comprend deux boucles d'asservissement :

- La boucle droite, qui assure l'asservissement des moteurs droits.
- La boucle gauche, qui assure l'asservissement des moteurs gauches. Chaque boucle d'asservissement utilise un contrôleur PID.

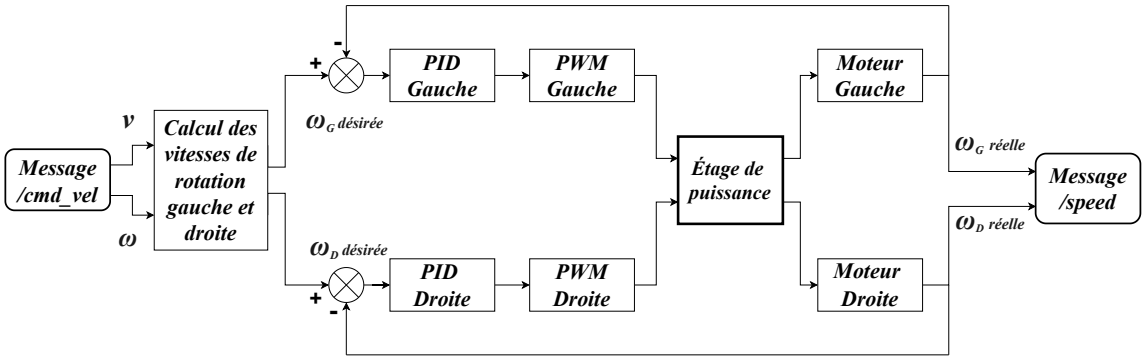


FIG. 2.35 – Application de contrôle embarquée bas-niveau.

La couche de bas niveau publie un message contenant les vitesses de rotation des roues (*omega*gauche et *omega*droite) dans le topic speed.

2. **Les bibliothèques Arduino**

   Pour implémenter l'application de contrôle embarquée de bas niveau, nous avons utilisé les bibliothèques Arduino suivantes :

   — **Bibliothèque *PID_b1_vc.h* :** Cette bibliothèque implémente un contrôleur PID permettant de calculer les commandes des moteurs droit et gauche de la plateforme robotique.

   — **Bibliothèque *Rosserial Arduino Library* :** Cette bibliothèque est utilisée pour établir la communication entre le calculateur de haut niveau (Raspberry Pi) et le calculateur de bas niveau (Arduino). Elle permet de s'abonner aux topics ROS et de partager les messages ROS.

2.6 Conclusion

Ce chapitre a détaillé les différentes étapes de la conception et de la réalisation de notre robot mobile à 4 roues. Nous avons commencé par la modélisation cinématique du robot, posant ainsi les bases théoriques nécessaires à sa conception. Cette modélisation nous a permis de comprendre et de définir les mouvements du robot, garantissant une navigation précise et contrôlée. Ensuite, nous avons décrit la conception et la réalisation de la structure mécanique du robot. Grâce à SolidWorks, nous avons conçu chaque pièce du robot avec précision. Ces pièces ont ensuite été réalisées à l'aide de techniques de fabrication avancées telles que le découpage CNC et l'impression 3D.

L'architecture électronique proposée a été divisée en deux segments principaux : le segment haut-niveau et le segment bas-niveau. Le segment haut-niveau est responsable de la gestion des algorithmes de navigation et de guidage et le segment bas-niveau exécute les commandes envoyées et supervise les quatre moteurs de la plateforme robotique.

Enfin, nous avons présenté l'architecture logicielle proposée, en détaillant les briques logicielles utilisées à différents niveaux. Au niveau de la station sol (PC), des outils tels qu'Ubuntu 20.04 et VNC Viewer ont été utilisés. Sur la Raspberry Pi, Debian 10 et ROS ont été les choix principaux pour gérer les tâches de navigation et de contrôle. Enfin, l'Arduino Mega a été programmé à l'aide de bibliothèques spécifiques pour assurer une communication et une exécution efficaces des commandes.

Dans le chapitre suivant, nous allons approfondir plusieurs aspects cruciaux de notre projet. Nous présenterons en détail le système ROS et son utilité dans notre projet. Nous aborderons également la commande bas-niveau et la commande manuelle du robot. L'implémentation sous ROS des différentes taches de la navigation autonome du robot sera aussi détaillée.

CHAPITRE

3

DÉVELOPPEMENT D'UNE ARCHITECTURE LOGICIELLE BASÉE SUR ROS

3.1 Introduction

En parcourant le premier chapitre, il est évident que la robotique mobile englobe divers domaines tels que l'industrie, l'agriculture, l'assistance médicale etc. En effet, les experts développeurs avaient besoin d'un outil pour simplifier leurs tâches. ROS est actuellement l'outil le plus largement utilisé dans le domaine de la robotique à l'échelle mondiale [29], non seulement pour la recherche scientifique sur des systèmes tels que les robots mobiles, mais aussi pour la gestion de produits commerciaux, personnels et même militaires.

Le développement d'une architecture logicielle basée sur ROS représente une approche innovante et efficace pour la conception de systèmes robotiques avancés. Dans ce chapitre, nous explorerons les fondements du ROS, son impact sur le développement logiciel en robotique, ses outils de visualisation, ainsi que les principaux avantages et les domaines d'application de cette architecture pour la création de robots autonomes. Nous allons passer par la suite à développer l'environnement du robot mobile en commençons par établir la communication entre la station sol, le Rasberry Pi et l'Arduino. Ensuite, nous allons détailler la commande bas-niveau du robot réalisé en commençant d'abord par l'asservissement en vitesse d'un seul moteur par un controleur PID puis l'asservissement en vitesse du robot complet. Nous allons par la suite présenter l'implémentation d'une commande manuelle du robot à partir de la station sol. Nous terminons ce chapitre par l'implémentation basée ROS de la commande haut-niveau en commençant par la cartographie de l'environnement par l'algorithme SLAM puis la localisation du robot par l'algorithme AMCL et la planification des chemins jusqu'à la navigation autonome du robot.

3.2 Robot Operating System

ROS est un ensemble d'outils open-source et d'infrastructures logicielles conçus pour simplifier le développement de logiciels pour robots. Il fournit une architecture modulaire qui facilite la création, le test et le déploiement de différents composants logiciels sur des robots. ROS permet également la communication entre ces composants, la gestion des capteurs et des actionneurs, ainsi que la simulation et la visualisation.

Avant 2007, le projet ROS a pris ses premiers pas à l'université de Stanford aux États-Unis, initié par Eric Berger et Keenan Wyrobek, deux doctorants impliqués dans un projet de robotique. Quelques mois avant l'annonce officielle de ROS, la start-up a été incubée par Willow Garage [26].

3.2.1 Pourquoi utiliser ROS ?

Une question souvent très posée, dont la réponse est très simple, comme l'ont dit leurs fondateurs : **ROS est l'outil le plus rapide pour construire un robot** [35]. Nous pouvons répondre aussi à cette question en donnant les cinqs principaux caractéristiques du ROS qui sont [47] :

1. **Réutilisabilité :** Les utilisateurs peuvent développer des fonctionnalités personnalisées et les partager sous forme de "packages" pour que d'autres puissent les réutiliser. Par exemple, les développeurs du Robonaut de la NASA ont utilisé ROS pour ce type de partage.
2. **Communication basée sur les noeuds :** ROS repose sur un modèle de communication entre noeuds, où chaque programme est décomposé en unités fonctionnelles plus petites, permettant une modularisation efficace. Cette approche favorise la réutilisation des logiciels et facilite le débogage des applications.
3. **Outils de développement :** ROS fournit des outils de débogage, de visualisation 2D (comme rqt) et 3D (comme RViz) qui simplifient le développement et la simulation de robots. Par exemple, les utilisateurs peuvent visualiser des modèles de robots et effectuer des simulations avec des simulateurs comme Gazebo.
4. **Communauté active :** ROS bénéficie d'une communauté dynamique d'utilisateurs open-source qui partagent des milliers de packages et collaborent sur des forums et des wikis. Cette collaboration favorise le développement et l'innovation dans le domaine de la robotique.
5. **Environnement en croissance :** ROS contribue à créer un écosystème vibrant dans le domaine de la robotique, favorisant l'innovation et la standardisation des logiciels [47].

3.2.2 Les concepts clés du ROS

ROS est un ensemble de concepts fondamentaux qui constituent sa philosophie et son architecture. On définit ci-dessous les principaux concepts de ROS [39] :

1. **Noeuds (Nodes) :** Les noeuds sont des entités de traitement dans ROS qui exécutent des tâches spécifiques. Chaque noeud accomplit une fonction particulière, comme la gestion des capteurs, la planification des mouvements ou le contrôle des actionneurs. Les noeuds interagissent en communiquant via des messages.

2. **Messages :** Les messages définissent la structure des données échangées entre les noeuds. Ils représentent des informations telles que des données de capteurs, des commandes de mouvement ou d'autres types de données pertinentes pour le système robotique.
3. **Sujets (Topics) :** Les sujets sont des canaux de communication à travers lesquels les noeuds échangent des messages. Un noeud peut publier des messages sur un sujet, tandis que d'autres noeuds s'abonnent à ce sujet pour recevoir ces messages.
4. **ROS Maitre (Master) :** Au coeur de ROS se trouve le ROS Master, un composant essentiel qui facilite la communication entre les noeuds. Il agit comme un registre centralisé où les noeuds peuvent se retrouver. Lorsqu'un noeud démarre, il s'enregistre auprès du maître ROS, fournissant des informations sur son nom, son type et ses fonctionnalités.
5. **Services :** Les services permettent à un noeud d'envoyer une requête à un autre noeud et d'attendre une réponse synchrone. Cela permet d'exécuter des actions spécifiques sur demande, comme l'obtention de données ou l'exécution de tâches particulières. La Figure 3.1 illustre le fonctionnement de services.

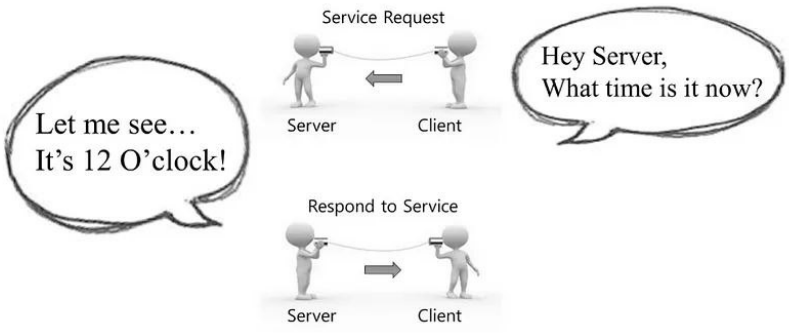


FIG. 3.1 – Services [47]

6. **Actions :** Les actions sont des interactions qui permettent de gérer des tâches complexes sur une période prolongée. Elles sont utiles pour des opérations telles que la navigation longue durée ou l'exécution de processus qui nécessitent un suivi d'état. La Figure 3.2 illustre le fonctionnement d'actions.
7. **Paramètres :** Les paramètres sont essentiels pour la communication de messages, se regroupant en sujets, services et actions. Ils agissent comme des variables globales utilisées localement et à grande échelle, facilitant la communication de messages.
8. **Packages :** Les packages sont des unités d'organisation du logiciel dans ROS. Chaque package regroupe des fonctionnalités connexes telles que des noeuds, des messages, des services et des bibliothèques nécessaires pour accomplir une tâche spécifique.
9. **Fichiers de Lancement (Launch Files) :** Les fichiers de lancement (launch files) sont des fichiers XML utilisés pour démarrer et configurer plusieurs noeuds ROS simultanément. Ils permettent de simplifier le processus de configuration et de démarrage d'un système ROS.
10. **Espace de Travail Catkin (Catkin Workspace) :** Catkin est le système de construction utilisé pour compiler et gérer les packages ROS. Un espace de travail

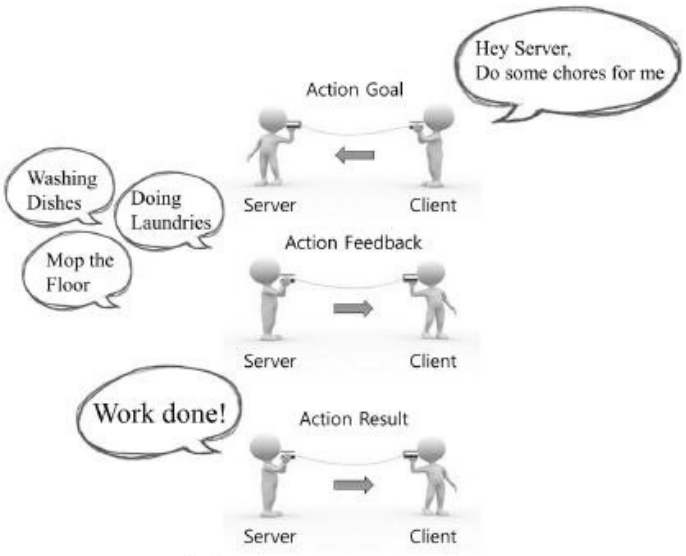


FIG. 3.2 – actions [47]

Catkin est un répertoire contenant tous les packages nécessaires à un projet ROS spécifique [39].
Ces différents concepts et les relations entre eux sont illustrés dans la figure 3.3.

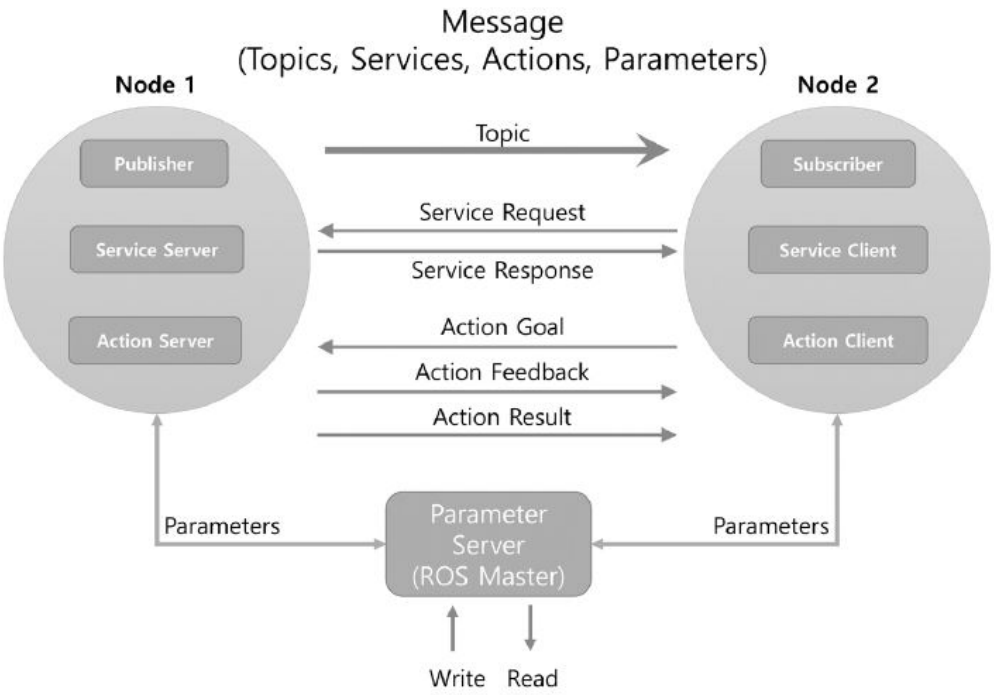


FIG. 3.3 – Les différents modes de communication [47]

3.2.3 Les applications du ROS

Le ROS est une plateforme open-source largement utilisée dans des différents applications et plus encore pour le développement et l'exploitation de robots. Dans ce paragraphe nous allons voir quelques-unes des différents applications utilisés dans la robotique [39] :

1. **Robotique industrielle :** La robotique industrielle adopte ROS en raison de sa modularité et de sa capacité à intégrer différents capteurs et actionneurs. Il permet de contrôler et de superviser des robots industriels dans des environnements variés. Cependent, l'utilisation de capteurs de proximité ou de vision assure la sécurité des opérateurs et des équipements.
2. **Robotique médicale :** ROS est essentiel dans le développement de robots chirurgicaux pour des procédures peu invasives. Il facilite l'intégration de systèmes d'imagerie, la précision du contrôle des bras robotiques, ainsi que la fourniture de retour tactile aux chirurgiens. Le robot chirurgical "Raven II" 3.4 développé par l'Université de Washington a utilisé ROS pour la recherche en robotique chirurgicale téléopérée [34].

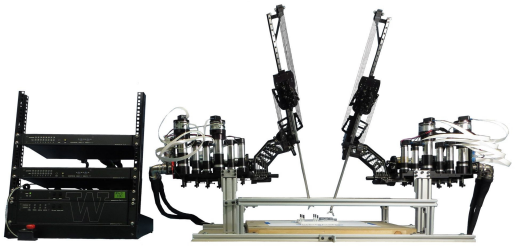


FIG. 3.4 – Raven II surgical robot [34]

3. **Robotique agricole :** ROS contribue au développement de robots autonomes pour des tâches telles que la plantation, la récolte et la surveillance des cultures. Il aide à intégrer des capteurs pour l'analyse du sol, le GPS pour la navigation et les algorithmes d'apprentissage automatique pour l'évaluation des cultures. Le BoniRob 3.5, un robot agricole polyvalent, a utilisé ROS pour son infrastructure logicielle [39].

FIG. 3.5 – Bonirob the robot farmer [28]

4. **Drones et véhicules autonomes :** ROS est adapté à la programmation de drones et de véhicules autonomes. Il offre des outils pour la planification de trajectoires, la perception de l'environnement et le contrôle des mouvements. Par exemple, la navigation GPS, le suivi d'objets, l'évitement d'obstacles, les radars et des caméras. La Figure 3.6 présente ces exemples.

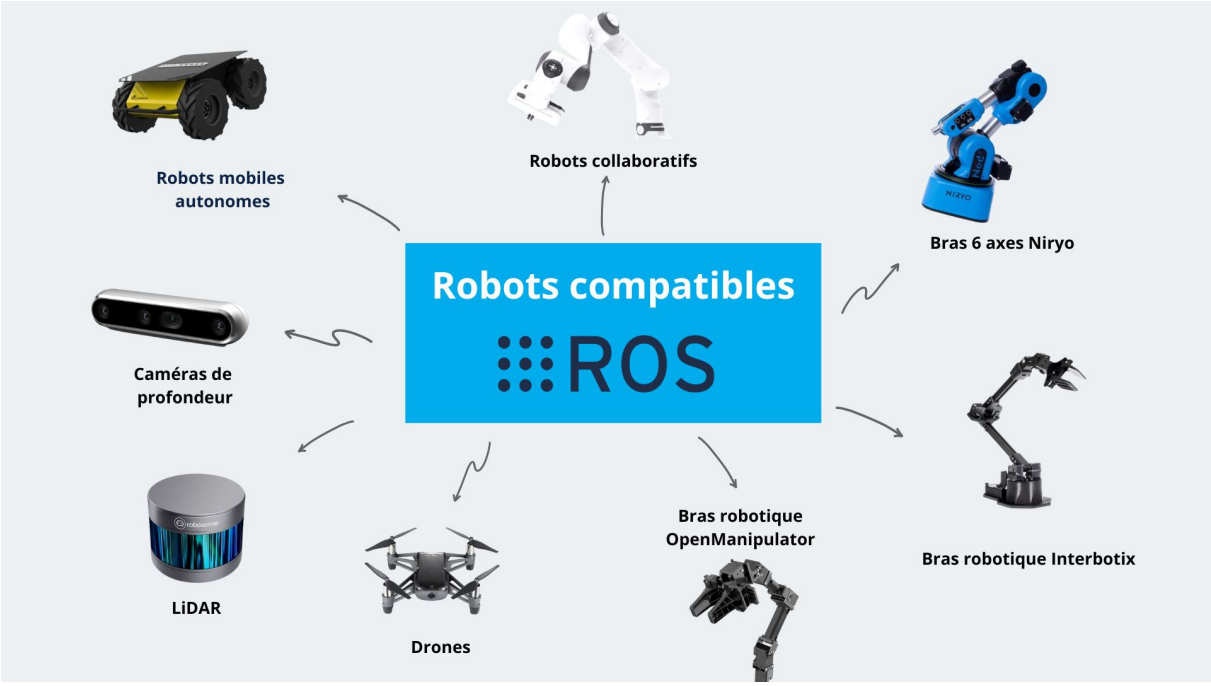


FIG. 3.6 – Différents robots compatibles avec ROS [33]

3.2.4 Les outils de visualisation de ROS

ROS se distingue par sa capacité à offrir une représentation en temps réel des états système et des résultats des algorithmes de traitement, ce qui constitue l'une de ses fonctionnalités les plus remarquables. Pour simplifier l'analyse du comportement des robots et des divers composants du système, ROS met à disposition une variété d'outils de visualisation. Parmi ces derniers, on peut mentionner RViz, Gazebo et rqt.

1. **RViz**

   RViz constitue l'outil de visualisation 3D de ROS. Son objectif principal réside dans la représentation des messages ROS en trois dimensions, offrant ainsi une vérification visuelle des données. Par exemple, il permet d'afficher la distance entre un capteur (LDS) et un obstacle, les données de nuage de points (PCD) issues de capteurs de distance 3D comme RealSense, Kinect ou Xtion, ainsi que les images capturées par une caméra, et bien d'autres éléments, sans nécessiter le développement de logiciels distincts. La Figure 3.7 présente le logo de RViz.

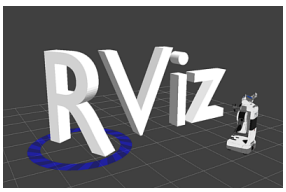


FIG. 3.7 – RViz logo

   RViz propose également une variété de visualisations à l'aide de polygones définis par l'utilisateur, tandis que les marqueurs interactifs permettent aux utilisateurs d'interagir avec les commandes et les données provenant du noeud utilisateur. De plus, ROS décrit les robots au moyen du format URDF (Unified Robot Description

Format), exprimé sous forme de modèle 3D modifiable. Le modèle de robot mobile peut être affiché, et les données de distance reçues du capteur laser (LDS) peuvent être exploitées pour la navigation, voir la figure 3.8.

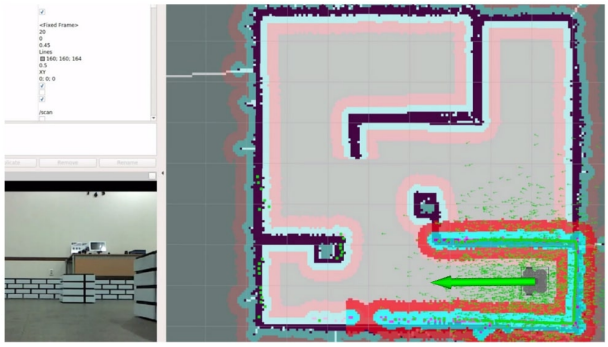


FIG. 3.8 – Navigation deTurtleBot3 et LDS [26]

RViz est également capable d'afficher les images de la caméra embarquée sur le robot. En outre, il peut recevoir des données de divers capteurs tels que Kinect, LDS, RealSense, et les représenter en 3D, voir les figures 3.9 3.10 3.11.

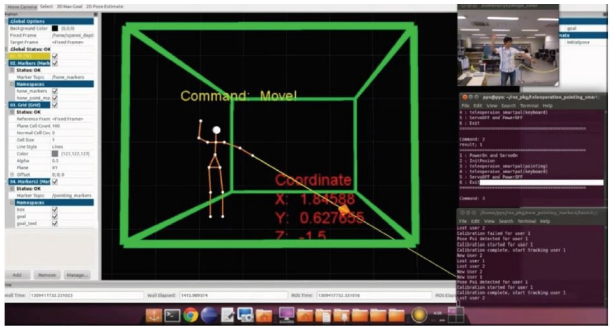


FIG. 3.9 – Obtenir le squelette d'une personne en utilisant Kinect[26]

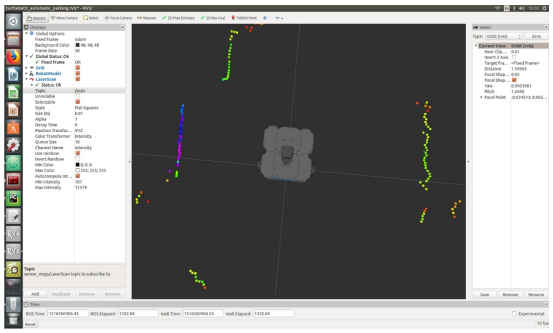


FIG. 3.10 – Mesure de distance en utilisant LDS[26]

FIG. 3.11 – Distance, infrarouge, valeur de l'image couleur obtenue à partir de Intel RealSense [26]

2. **Gazebo**

Gazebo permet une simulation précise et efficace de populations de robots dans des environnements intérieurs et extérieurs complexes. Il est doté d'un moteur physique robuste, de graphismes de qualité supérieure, ainsi que d'interfaces

graphiques et programmatiques pratiques. De plus, Gazebo est gratuit et bénéficie d'une communauté active. La Figure 3.12 présente le logo de Gazebo [4]. Ses principales caractéristiques incluent :

Fig. 3.12 – Gazebo Logo

• Simulation de la dynamique : Gazebo offre l'accès à plusieurs moteurs de physique de haute performance, tels que ODE, Bullet et Simbody.

• Graphiques 3D avancés : Gazebo assure un rendu réaliste des environnements, avec un éclairage, des ombres et des textures de haute qualité.

• Capteurs et bruit : Il permet la génération de données de capteurs à partir de divers équipements, y compris des télémètres laser, des caméras 2D/3D, des capteurs de type Kinect, des capteurs de contact, des couples de force...

• Simulation de nuages : Gazebo prend en charge CloudSim pour fonctionner sur Amazon AWS, ainsi que GzWeb pour interagir avec la simulation via un navigateur web.

• Transport TCP/IP : Il est possible d'exécuter la simulation sur des serveurs distants et d'interagir avec Gazebo via un système de transmission de messages basé sur des sockets utilisant Google Protobufs.

3. **rqt**

rqt est un cadre de développement multiplateforme largement utilisé pour la création d'interfaces graphiques. Cette base permet aux utilisateurs de développer et d'intégrer facilement des plugins, tels que rqt-image-view, rqt-graph, rqt-plot et rqt-bag.

• Le plugin Image View de rqt est un outil pratique pour visualiser des données d'images provenant de diverses sources, comme les caméras. Il fournit une interface permettant d'afficher et d'examiner en temps réel des images, ce qui facilite le débogage, la vérification des données des capteurs et la surveillance des environnements robotiques. Grâce à Image View, les utilisateurs peuvent aisément analyser les images capturées par les robots, ce qui en fait un élément indispensable de l'arsenal d'outils de développement avec ROS.

• Le plugin Plot de rqt est un outil essentiel pour représenter graphiquement des données en deux dimensions(2D). Il permet aux utilisateurs de créer des graphiques clairs et informatifs pour analyser diverses données, telles que des séries temporelles ou des résultats d'expériences. Grâce à cette fonctionnalité, les utilisateurs peuvent facilement visualiser et interpréter les données.

• Le plugin Graph de rqt est un outil précieux pour visualiser les relations entre les noeuds ou les flux de messages au sein d'un système ROS (Figure 3.13). Il offre une représentation visuelle claire de la structure et de l'interconnexion des composants, ce qui facilite la compréhension des systèmes robotiques complexes.

• Le plugin Bag de rqt est un outil pratique qui permet aux utilisateurs de capturer les messages publiés sur différents sujets et de les stocker dans des fichiers bag. Avec Bag, les utilisateurs peuvent facilement enregistrer des données pendant l'exécution d'expériences ou de tests, puis les examiner et les analyser à leur convenance.

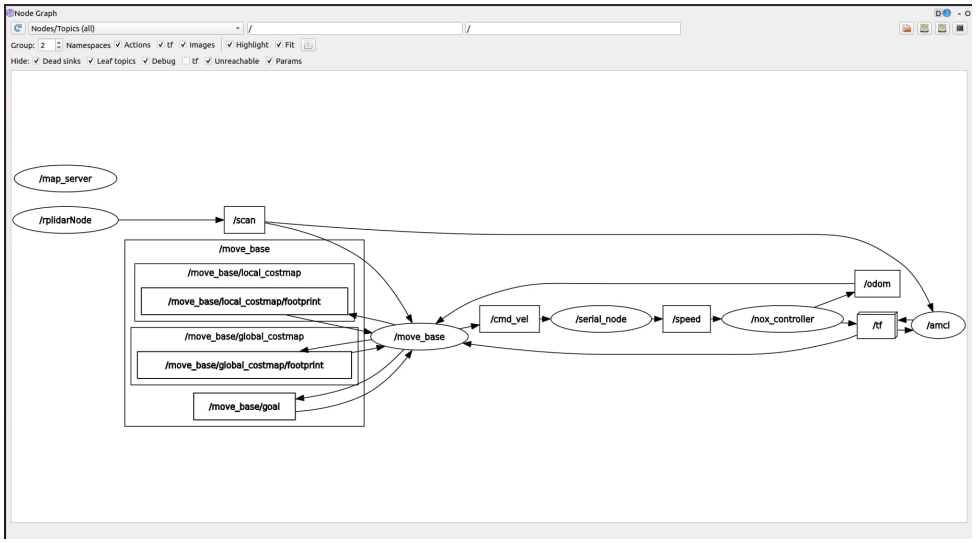


Fig. 3.13 – Un exemple du plugin Graph de rqt

3.3 Développement du Framework basé sur ROS pour le robot mobile

Cette section met en oeuvre le Framework reliant les différentes parties qui composent notre système décrit sur la figure 3.14. Il comprend une station sol (PC) permettant la visualisation des données et le contrôle à distance su robot. Lors de la commande manuelle du robot, la station sol communique avec le Raspberry en lui transférant les vitesses désirées du robot et elle reçoit des données pour les afficher sur Rviz lors de la navigation autonome du robot. Cette communication est établie à travers ROS comme nous allons l'expliquer dans ce qui suit. La commande haut-niveau communique à travers la communication série les vitesses désirées à la partie commande bas-niveau qui à son tour assure l'asservissement du robot. Cette communication série permet à l'Arduino d'être considéré comme un noeud dans le framework ROS. Ainsi, nous pouvons publier des données sur des topic directement à partir de l'Arduino comme nous pouvons aussi recevoir des données en souscrivant à leurs topics.

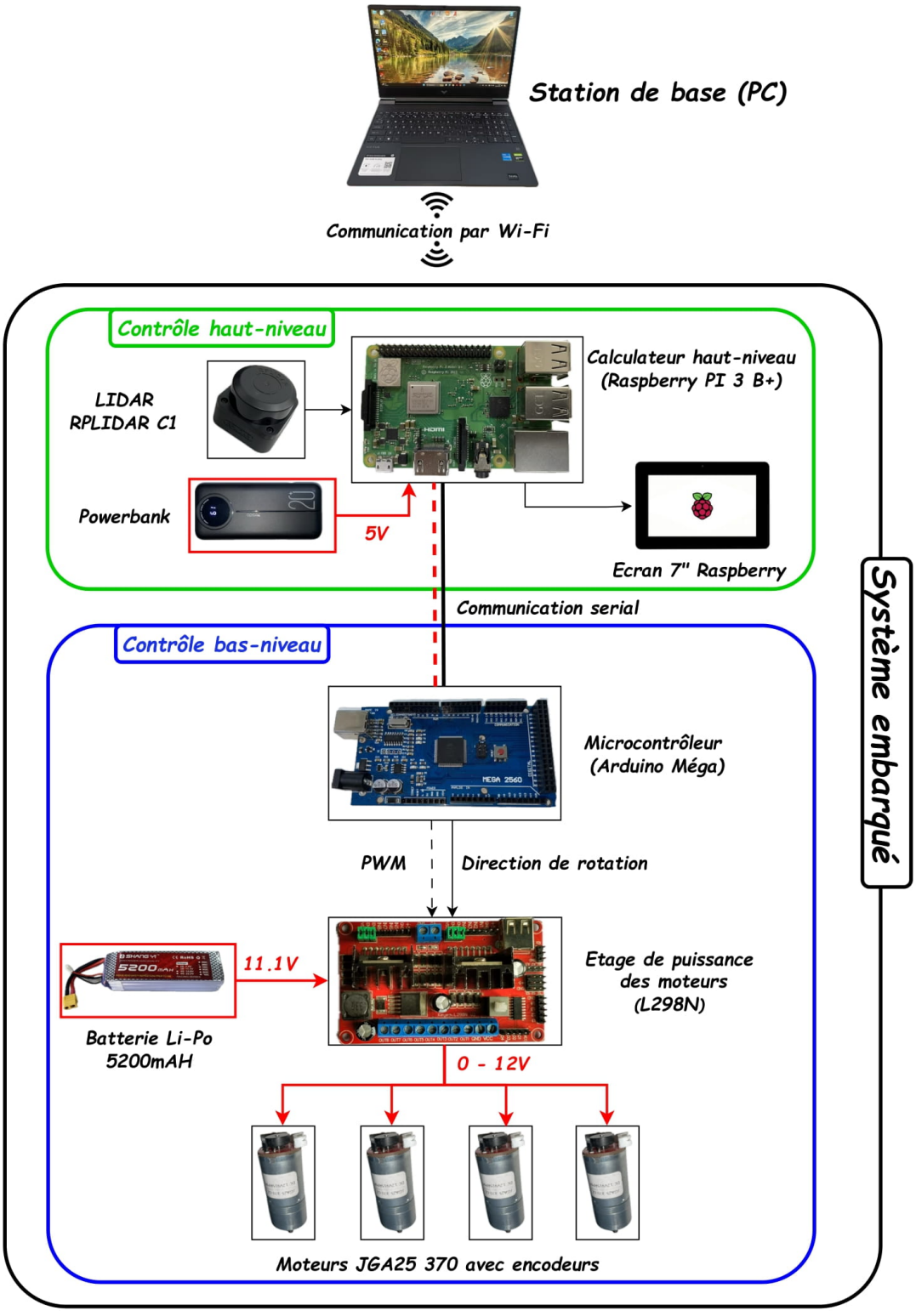


FIG. 3.14 – Architecture matérielle

3.3.1 Communication entre PC et Raspberry Pi

La communication entre le PC et le Raspberry Pi se fait sur un seul réseau, comme illustré sur la figure 3.15. Pour réaliser cette communication, il est nécessaire d'établir, en premier lieu, un réseau local qui assure une connexion Wifi entre le PC et le Raspberry Pi. Ensuite, nous identifierons le maître (Master) et l'esclave (Slave). Pour valider la communication, nous illustrerons avec un exemple en spécifiant un émetteur et un récepteur, afin de vérifier si le message parvient à destination.

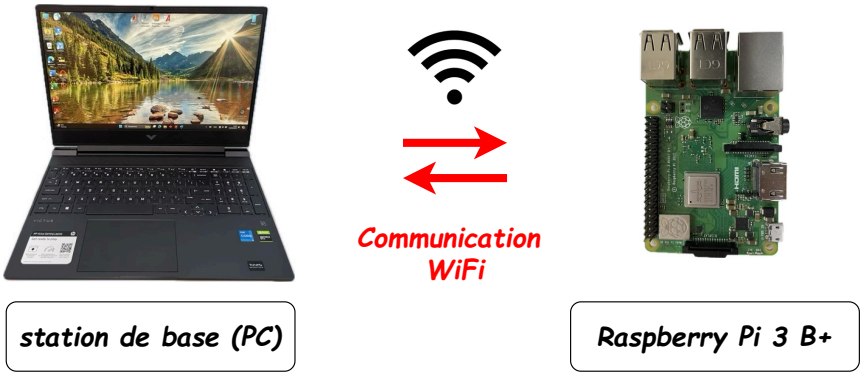


Fig. 3.15 – Schéma de la communication entre le PC et le Raspberry Pi

1. **Connexion entre PC et Raspberry Pi :** Pour établir cette connexion, nous allons utiliser le ROS-MASTER-URI et le ROS-IP dans les deux programmes en spécifiant que le Raspberry Pi est l'esclave (Raspberry=Slave) et le PC est le maître (PC=Master). Sur la station sol, pour informer le PC qu'il est le master et pour lui indiquer son adresse IP, nous exécutons respectivement les commandes :
"export ROS-MASTER-URI=http : //adresse-IP-PC :11311" et
"export ROS-IP=adresse-IP-PC".

```
hichem@hichem-ThinkPad-T570:~$ source /opt/ros/noetic/setup.bash
hichem@hichem-ThinkPad-T570:~$ export ROS_MASTER_URI=http://192.168.1.34:11311
hichem@hichem-ThinkPad-T570:~$ export ROS_IP=192.168.1.36
```

Fig. 3.16 – Programme du PC

Dans le programme du Raspberry Pi, on suit la même procédure mais on change l'adresse IP du Raspberry Pi par celle du PC pour lui indiquer qu'il n'est pas un Master.

```
hichem@ubuntu:~$ export ROS_MASTER_URI=http://192.168.1.34:11311/
hichem@ubuntu:~$ export ROS_IP=192.168.1.34
hichem@ubuntu:~$ source /opt/ros/noetic/setup.bash
```

Fig. 3.17 – Programme Raspberry Pi

2. **Lancement du ROSCORE :** On lance le ROSCORE pour démarrer le ROS Master (PC). Le ROSCORE est essentiel car il s'agit du noeud central de communication pour tous les noeuds.

```
hichem@ubuntu:~$ export ROS_MASTER_URI=http://192.168.1.34:11311/
hichem@ubuntu:~$ export ROS_IP=192.168.1.34
hichem@ubuntu:~$ source /opt/ros/noetic/setup.bash
hichem@ubuntu:~$ roscore
... logging to /home/hichem/.ros/log/5f09cf6e-d63d-11ee-8f15-03a65075b
e02/roslaunch-ubuntu-2080.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://192.168.1.34:39413/
ros_comm version 1.16.0


SUMMARY
========

PARAMETERS
 * /rosdistro: noetic
 * /rosversion: 1.16.0
```

FIG. 3.18 – Lancement du *ROSCORE*

3. **Exemple de communication :** Après avoir lancé le ROSCORE, nous lançons maintenant les noeuds en spécifiant l'émetteur (Talker) et le récepteur (Listener). Il s'agit donc de lancer un **ROSRUN** en définissant le PC comme Talker et le Raspberry comme Listener, en utilisant 'hello world' comme exemple. Les figures 3.33 et 3.34 illustrent la réussite de la communication entre les deux calculateurs.

```
hichem@hichem-ThinkPad-T570:~$ export ROS_MASTER_URI=http://192.168.1.34:11311
hichem@hichem-ThinkPad-T570:~$ export ROS_IP=192.168.1.36
hichem@hichem-ThinkPad-T570:~$ source /opt/ros/noetic/setup.bash
hichem@hichem-ThinkPad-T570:~$ rosrun rospy_tutorials talker.py
[INFO] [1709130670.052788]: hello world 1709130670.0524466
[INFO] [1709130670.153516]: hello world 1709130670.1531677
[INFO] [1709130670.253083]: hello world 1709130670.2527535
[INFO] [1709130670.353595]: hello world 1709130670.3532135
[INFO] [1709130670.453429]: hello world 1709130670.4531238
[INFO] [1709130670.553477]: hello world 1709130670.55312
[INFO] [1709130670.653137]: hello world 1709130670.6527297
[INFO] [1709130670.753490]: hello world 1709130670.75315
[INFO] [1709130670.853544]: hello world 1709130670.8531435
[INFO] [1709130670.952846]: hello world 1709130670.9526625
[INFO] [1709130671.052997]: hello world 1709130671.0526853
[INFO] [1709130671.152913]: hello world 1709130671.1526093
[INFO] [1709130671.253086]: hello world 1709130671.2527566
[INFO] [1709130671.353234]: hello world 1709130671.3528137
[INFO] [1709130671.453114]: hello world 1709130671.452789
[INFO] [1709130671.553157]: hello world 1709130671.5527444
[INFO] [1709130671.653439]: hello world 1709130671.6530159
[INFO] [1709130671.753087]: hello world 1709130671.7526965
[INFO] [1709130671.853154]: hello world 1709130671.8528016
```

FIG. 3.19 – Lancement du ROSRUN en tant qu'un émetteur

```
hichem@ubuntu:~$ source /opt/ros/noetic/setup.bash
hichem@ubuntu:~$ rosrun rospy_tutorials listener.py
[INFO] [1709126990.849866]: /listener_2147_1709126985494I heard hello
world 1709126991.9502885
[INFO] [1709126990.949667]: /listener_2147_1709126985494I heard hello
world 1709126992.0506053
[INFO] [1709126991.060712]: /listener_2147_1709126985494I heard hello
world 1709126992.150558
[INFO] [1709126991.149864]: /listener_2147_1709126985494I heard hello
world 1709126992.250602
[INFO] [1709126991.263944]: /listener_2147_1709126985494I heard hello
world 1709126992.3506591
[INFO] [1709126991.351055]: /listener_2147_1709126985494I heard hello
world 1709126992.4506109
[INFO] [1709126991.451854]: /listener_2147_1709126985494I heard hello
world 1709126992.5504441
[INFO] [1709126991.550607]: /listener_2147_1709126985494I heard hello
world 1709126992.6505914
[INFO] [1709126991.655692]: /listener_2147_1709126985494I heard hello
world 1709126992.7506187
```

FIG. 3.20 – Lancement du ROSRUN en tant qu'un récepteur

3.3.2 Communication entre Raspberry Pi et Arduino

La communication entre un Raspberry Pi et un Arduino est souvent utilisée dans le cadre de projets électroniques pour plusieurs raisons. En premier lieu, le Raspberry Pi est un ordinateur doté d'un processeur puissant, tandis que l'Arduino présente des capacités de traitement et de stockage moins importantes en comparaison. Cependant, la communication série permet au Raspberry Pi de communiquer avec un Arduino. Cette connexion permet un échange bidirectionnel de données entre les deux appareils, permettant au Raspberry Pi d'envoyer des commandes à l'Arduino et de recevoir des données en retour. Voici le schéma de la communication illustré dans la figure 3.21 :

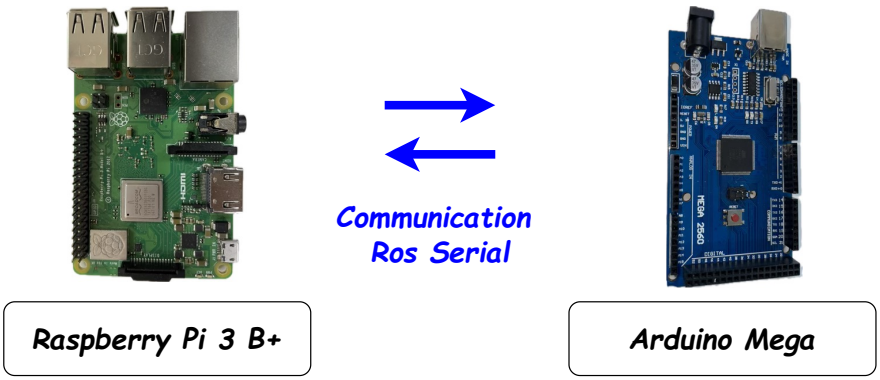


FIG. 3.21 – Schéma de la communication entre le Raspberry Pi et l'Arduino

Pour établir une communication série entre le Raspberry Pi et l'Arduino à travers ROS, il est d'abord nécessaire de configurer l'Arduino pour qu'il soit compatible avec ROS, puis de configurer le Raspberry Pi pour qu'il puisse communiquer avec l'Arduino. Pour ceci, nous avons utilisé la **Bibliothèque Rosserial_python**.. Cette bibliothèque propose une implémentation en Python de la connexion rosserial, simplifiant ainsi la configuration, la publication et l'abonnement des périphériques compatibles avec rosserial. Pour son fonctionnement, l'installation de pyserial est requise. Ce paquet doit être installé dans l'environnement ROS (*catkin_ws*).

1. **Installation de la bibliothèque Rosserial Arduino Library :** elle permet de rendre l'Arduino compatible avec ROS.
2. **Installation des bibliothèques : Rosserial (3.22), Rosserial-Arduino et Rosserial-Python** permet au Raspberry Pi de communiquer avec l'Arduino.

```
hichem@raspberrypi:~ $ sudo apt-get install ros-noetic-rosserial
Reading package lists... Done
Building dependency tree
Reading state information... Done
ros-noetic-rosserial is already the newest version (0.9.2-1buster.20230216.00182
8).
```

FIG. 3.22 – Installation de la bibliothèque Rosserial

3. **Définition des autorisations du port :** Pour définir les autorisations appropriées sur notre port de communication, nous avons exécuté la commande suivante : "sudo chmod 666 dev/ttyUSB1" tel que "dev/ttyUSB1" est le nom du port série utilisé pour la communication avec l'Arduino depuis notre Raspberry Pi.
4. **Confirmation de la communication :** Afin de confirmer la communication établie entre le Raspbeery Pi et l'Arduino, nous allons voir si le noeud "*serial_node*" est actif dans la liste des noeuds en cours d'exécution sur le Raspberry Pi. Pour le faire, en utilisant la commande *rosnode_list*. La Figure 3.23 illustre cette confirmation de la communication.

```
hichem@raspberrypi:~ $ rosnode list
/rosout
/serial_node
```

FIG. 3.23 – Liste des noeuds en cours d'exécution sur le Raspberry Pi

De plus, la Figure 3.24 présente les informations détaillées de ce noeud. Nous observons que le noeud "*serial_node*" publie le topic "*speed*" et que le topic "*cmd_vel*" y est abonné. Alors, nous pouvons conclure que la communication entre le Raspberry Pi et l'Arduino est établie correctement.

```
hichem@raspberrypi:~ $ rosnode info /serial_node
--------------------------------------------------------------------------------
Node [/serial_node]
Publications:
 * /diagnostics [diagnostic_msgs/DiagnosticArray]
 * /rosout [rosgraph_msgs/Log]
 * /speed [geometry_msgs/Vector3Stamped]

Subscriptions:
 * /cmd_vel [unknown type]

Services:
 * /serial_node/get_loggers
 * /serial_node/set_logger_level


contacting node http://192.168.117.158:37995/ ...
Pid: 1432
Connections:
 * topic: /rosout
    * to: /rosout
    * direction: outbound (36795 - 192.168.117.7:60332) [9]
    * transport: TCPROS
```

FIG. 3.24 – Informations sur le noeud "*serial_node*"

3.4 Commande du niveau bas

Afin que le robot roule avec les vitesses de translation et de rotation désirées, nous devrons l'asservir en vitesse. Ceci revient donc à asservir en vitesse les quatre moteurs du robot. Nous allons dans ce qui suit décrire d'abord comment nous avons pu asservir en vitesse un seul moteur en utilisant un contrôleur PID et justifier les choix de ses paramètres, et après nous passons à réaliser l'asservissement des quatre moteurs, c'est-à-dire du robot complet. La commande du niveau bas est synthétisé dans la Figure 3.25.

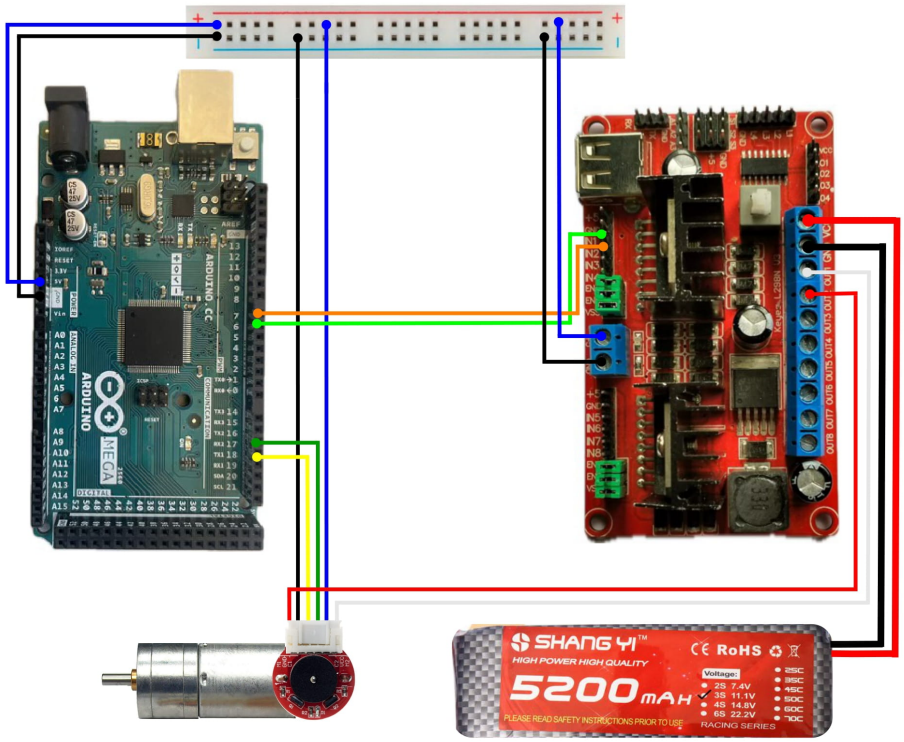


FIG. 3.25 – L'architecture de commande du niveau bas

3.4.1 Asservissement en vitesse d'un moteur à courant continu

Les moteurs à courant continu doivent être contrôlés avec une grande précision afin de maintenir une vitesse de rotation constante malgré les perturbations extérieures. Ce contrôle est essentiel pour réduire au minimum les erreurs internes qui pourraient compromettre la précision du système de localisation associé aux moteurs .

1. **La commande en boucle ouverte :**
   La commande d'un moteur en boucle ouverte s'effectue par le calcul d'une relation entre la vitesse de rotation désirée et le rapport cyclique correspondant comme illustré sur la Figure 3.26. Le rapport cyclique se réfère à la proportion du temps pendant lequel une charge est activée (ON) par rapport à la durée totale du cycle de fonctionnement, lorsque cette charge bascule plusieurs fois entre les états ON et OFF chaque seconde elle représente un rapport cyclique.

   Pour calculer la vitesse du moteur nous avons utiliser les encodeurs du moteur. La Figure 3.27 illustre une visualisation des signaux de l'encodeur à l'aide de l'oscilloscope. La Figure 3.28 montre la vitesse mesurée pour un rapport cyclique=30% sur le sérial moniteur.

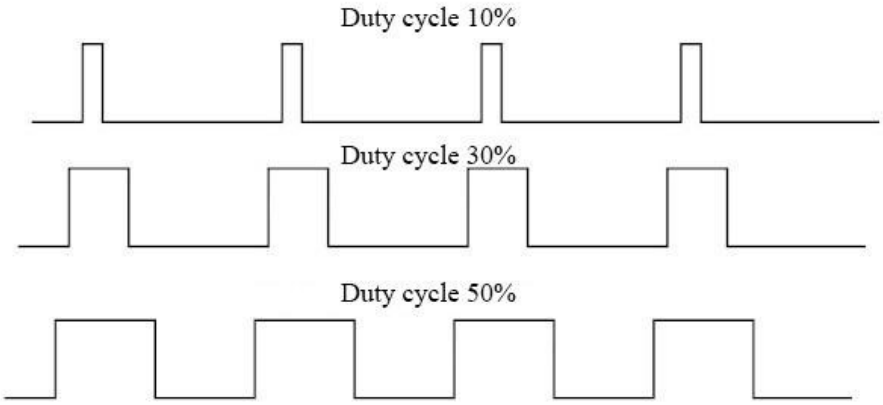


FIG. 3.26 – Le rapport cyclique

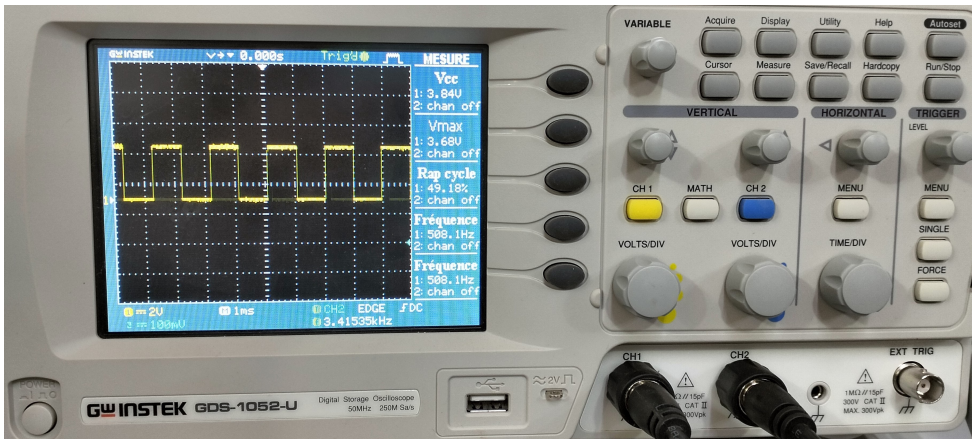


FIG. 3.27 – Mesure d'encodeur

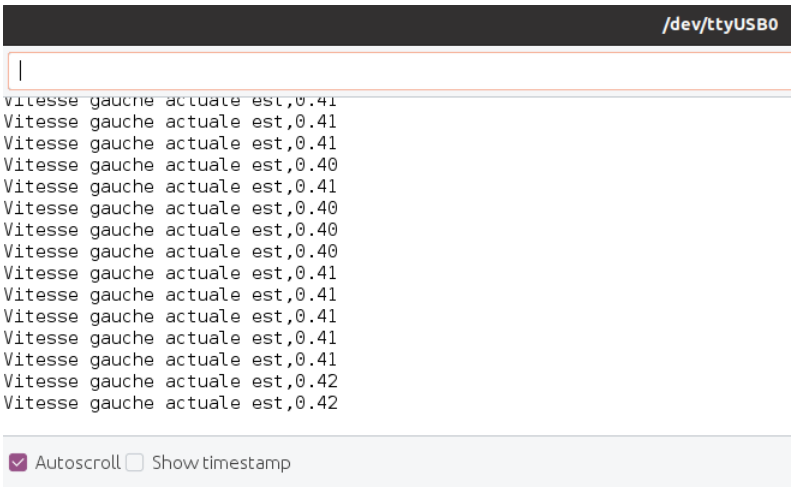


FIG. 3.28 – Visualisation de la vitesse mesurée pour un rapport cyclique de 30% sur le serial monitor

Dans ce qui suit, nous allons faire varier les différentes valeurs du rapport cyclique et visualiser la variation de la vitesse de notre moteur en fonction du rapport cyclique.

RC(%)	30	45	60	90	100
v(m/s)	0.41	1.18	1.80	2.40	2.55

TAB. 3.1 – Valeurs de vitesses mesurées pour différentes valeurs du rapport cyclique

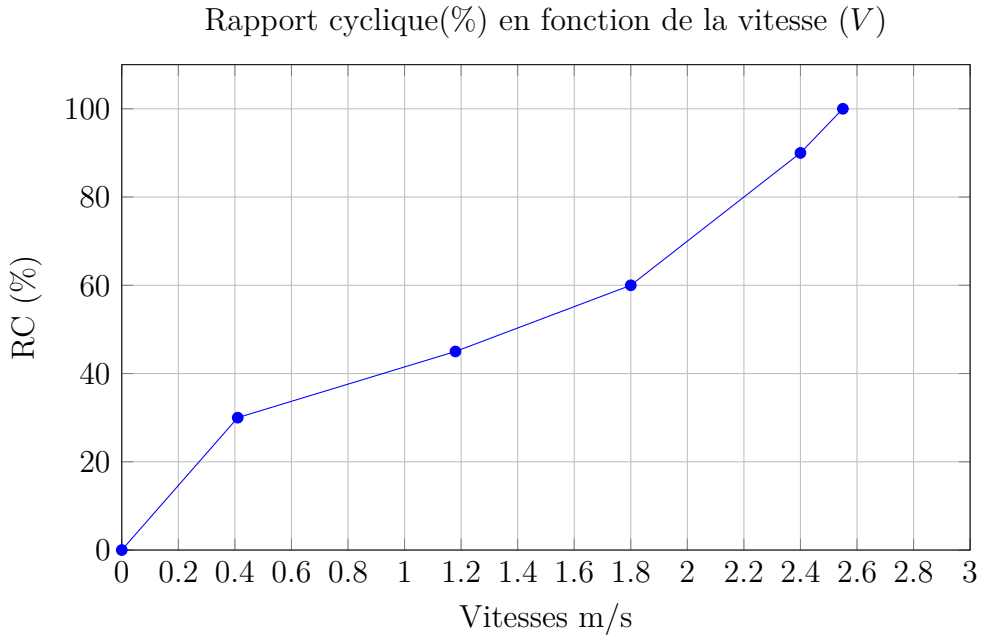


FIG. 3.29 – Représentation du rapport cyclique(%) en fonction de la vitesse (V)

Cette représentation met en évidence l'importance du réglage du rapport cyclique pour contrôler la vitesse du moteur, car elle permet d'observer clairement une augmentation progressive de la vitesse avec l'augmentation du rapport cyclique. Pour valider les vitesses calculées par rapport aux vitesses réelles, nous avons utilisé un tachymètre (Figure 3.30). Ce dernier est un dispositif qui mesure la vitesse en utilisant la vitesse angulaire ou les fréquences de vibration comme paramètres de mesure.

FIG. 3.30 – Un tachymètre

Pour la confirmation de l'asservissement en boucle ouverte, on trace une courbe, représenté dans la Figure 3.31, des valeurs des vitesses réelles v(m/s) en fonction des vitesses mesurées par le tachymètre V_t(m/s) . Le tableau ci-dessous regroupe ces valeurs ansi que les valeurs de tachymètre en (tr/min).

v(m/s)	0	0.41	1.18	1.80	2.40	2.55
V_t(tr/min)	0	96	290	445	590	627
V_t(m/s)	0	0.38	1.17	1.80	2.39	2.54

TAB. 3.2 – Vitesses réelles en fonction des vitesses mesurées par le tachymètre

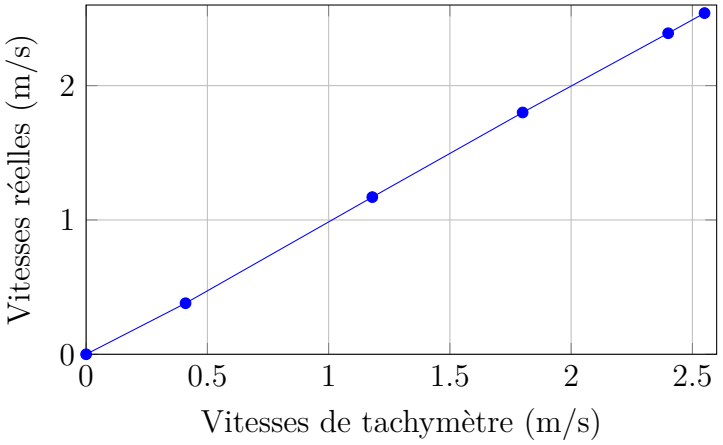


FIG. 3.31 – Vitesses réelles en fonction des vitesses mesurées par le tachymètre

Le graphique montre comment les mesures de vitesse prises par un tachymètre correspondent aux vitesses réelles du moteur en action. Cela signifie que lorsque le tachymètre indique une certaine vitesse, c'est effectivement la vitesse à laquelle le moteur tourne. Cela assure que nos mesures sont précises et fiables, ce qui est essentiel pour que le moteur fonctionne comme prévu, sans erreurs indésirables de vitesse.

2. **La commande en boucle fermée :**
   Le processus de correction implique la comparaison entre une consigne de vitesse et la vitesse mesurée par les encodeurs. L'écart (consigne - mesure) est ensuite corrigé pour appliquer une commande. Dans ce contexte, nous optons pour un régulateur PID (Proportionnel-Intégral-Dérivé) pour l'asservissement en vitesse d'un moteur DC (figure 3.32).

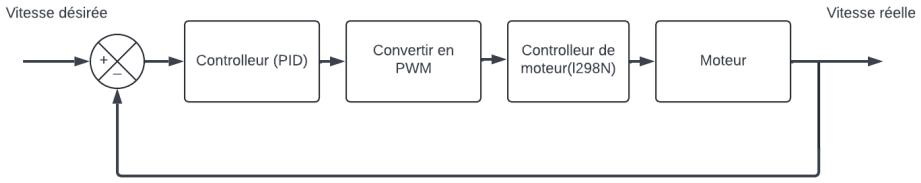


FIG. 3.32 – Commande en boucle fermée d'un moteur

- **Principe du contrôleur PID :**

Le contrôleur PID est conçu pour améliorer trois aspects essentiels d'un système : sa rapidité, sa précision et sa stabilité. Dans le contexte d'un moteur, cela se traduit par une accélération rapide (un temps de monté petit), un maintien de la vitesse réelle très proche de la consigne, et un fonctionnement sans oscillations. Le rôle du PID est de minimiser l'écart entre la vitesse réelle du moteur et la consigne qui lui est assignée dans notre projet.

$$\epsilon = \Omega_{\text{consigne}} - \Omega_{\text{mesure}} \tag{3.1}$$

Nous pouvons observer ci-dessous que le correcteur PID peut fonctionner selon trois modalités :

∗ Action proportionnelle : l'erreur est multipliée par un gain K_p.

$$w(t) = K_p \epsilon(t) \tag{3.2}$$

∗ Action intégrale : Dans le contrôle proportionnel, nous avons la possibilité d'ajouter l'intégration de l'erreur pour obtenir une régulation PI (Proportionnel et Intégral). Dans ce cas, l'erreur entre la consigne et la mesure est accumulée sur une période de temps et multipliée par une constante qui doit également être ajustée en fonction du système.

$$w(t) = K_p \epsilon(t) + K_i \int_0^t \epsilon(\tau)\, d\tau \tag{3.3}$$

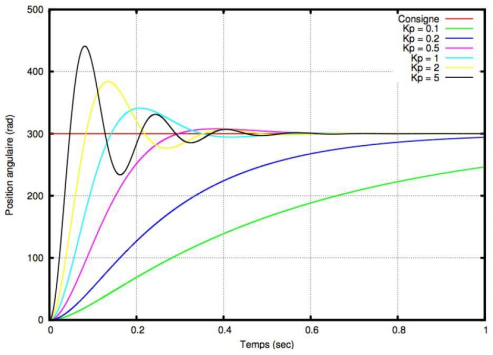


FIG. 3.33 – L'effet de l'action proportionnel sur la réponse du système [22]

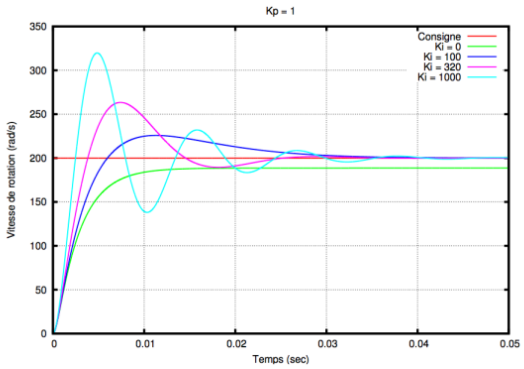


FIG. 3.34 – L'effet de l'action intégrale sur la réponse du système [22]

∗ Action dérivée : Pour mettre en place un contrôle PID complet, nous ajoutons un terme supplémentaire qui consiste à dériver l'erreur entre la consigne et la mesure par rapport au temps, puis à la multiplier par un gain K_d.

$$w(t) = K_p \epsilon(t) + K_i \int_0^t \epsilon(\tau)\, d\tau + K_d \frac{d}{dt}\epsilon(t) \tag{3.4}$$

Le contrôleur PID remplit trois principales fonctions :

— Il génère un signal de commande $u(t)$ en analysant l'évolution de la sortie $y(t)$ par rapport à la consigne $w(t)$.

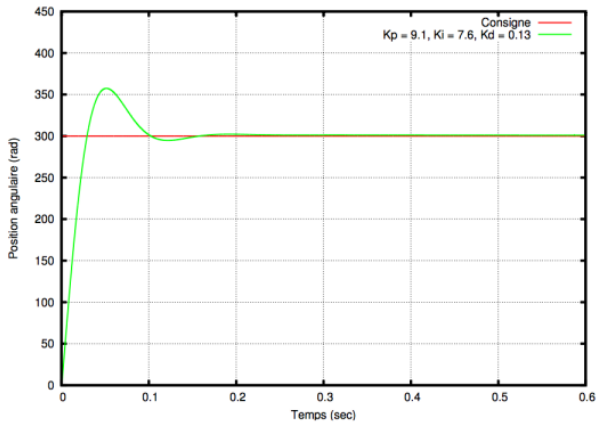


FIG. 3.35 – L'effet de l'action dérivé sur la réponse du système [22]

— Il élimine l'erreur statique en utilisant le terme intégral.
— Il anticipe les changements de sortie en utilisant le terme dérivé.

Nous pouvons regrouper les caractéristiques du PID dans le tableau ci-dessous :

Coefficient	Temps de montée	Temps de stabilisation	Dépassement	Erreur statique
K_p	Diminue	Augmente	Augmente	Diminue
K_i	Diminue	Augmente	Augmente	Annule
K_d	–	Diminue	Diminue	–

TAB. 3.3 – Caractéristiques du PID

• **Calcul des coefficients du PID :** La méthode que nous avons utilisé pour calculer les coefficients du PID repose sur une approche expérimentale itérative.
∗ Pour K_p=5, K_i=4 et K_d=2 tel que v désirée est égale à 1.5 m/s, la réponse du moteur est affichée sur la Figure 3.36.

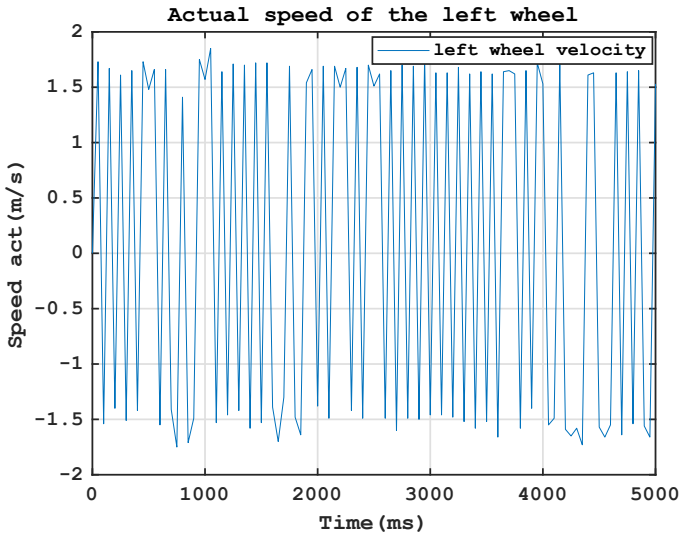


FIG. 3.36 – Vitesse mesurée pour K_p=5, K_i=4 et K_d=2

∗ Pour K_p=4, K_i=1 et K_d=0, nous obtenons la réponse affichée sur la figure 3.37 :

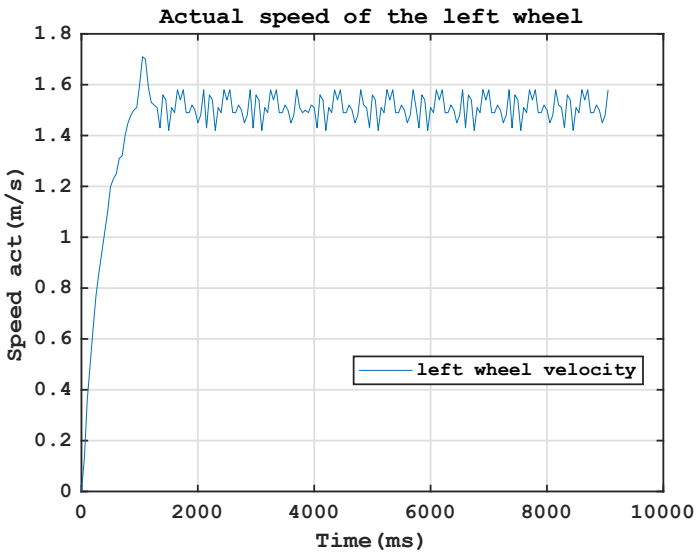


FIG. 3.37 – Vitesse mesurée pour K_p=4, K_i=1 et K_d=0

∗ Pour K_p=0.5, K_i=2 et K_d=0, la figure 3.38 illustre la réponse du moteur :

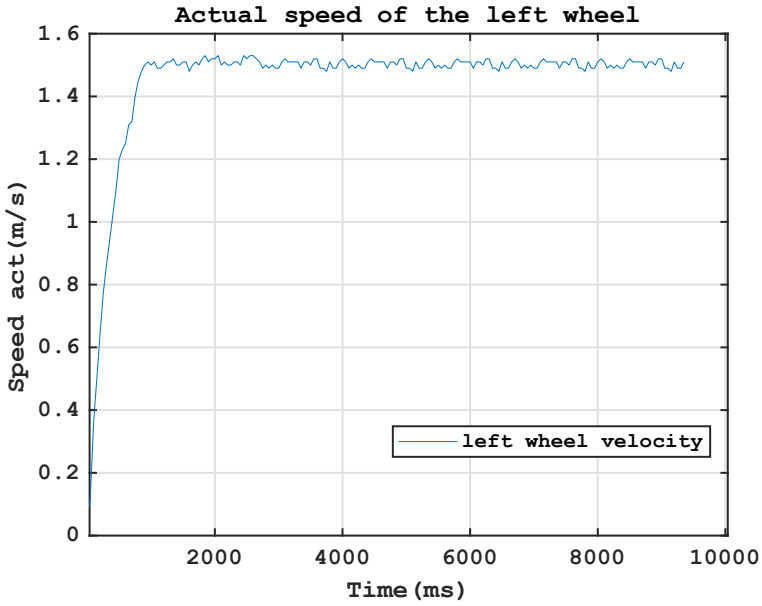


FIG. 3.38 – Vitesse mesurée pour K_p=0.5, K_i=2 et K_d=0

∗ Après plusieurs essais de réglage des coefficients du PID par tâtonnement, nous constatons que pour K_p=0.5, K_i=2 et K_d=0 la vitesse réelle est presque égale à la vitesse désirée. Pour une confirmation supplémentaire, les coefficients du PID sont toujours K_p=0.5, K_i=2 et K_d=0 et pour une vitesse désirée égale à 2 m/s, la réponse du moteur est affichée sur la figure 3.39 :

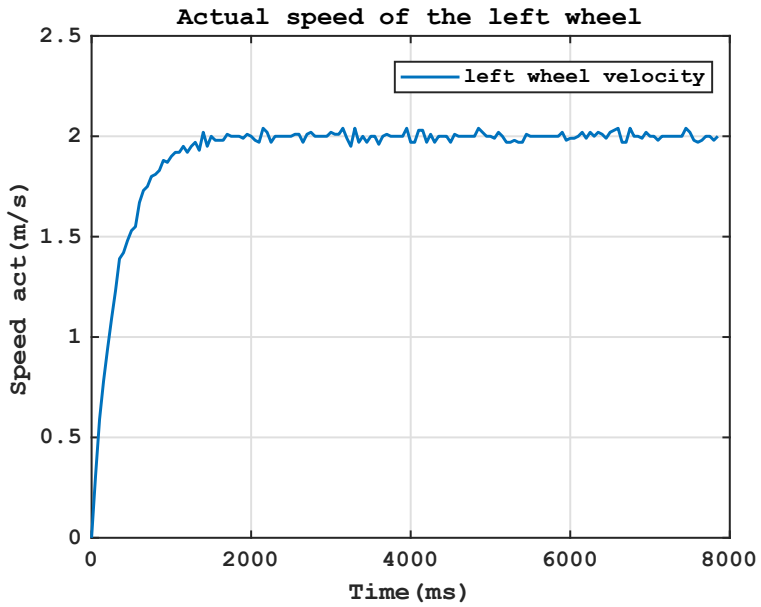


FIG. 3.39 – Vitesse mesurée pour K_p=0.5, K_i=2 et K_d=0

3.4.2 Asservissement en vitesse du robot complet :

Après avoir asservi un seul moteur, nous allons généraliser l'asservissement à quatre moteurs, ce qui permet d'asservir le robot complet tout en respectant les mouvements d'un robot différentiel de type unicycle. Pour cela, nous devons utiliser les relations cinématiques d'un robot de type unicyle afin de calculer les vitesses des roues gauche (v_g) et droite (v_d) à partir de la vitesse linéaire (v) et de la vitesse de rotation ω du robot :

Remarque : Les vitesses des roues arrière et avant du côté gauche seront égales, tout comme celles du côté droit.

$$v_g = v - \frac{L \cdot \omega}{2} \tag{3.5}$$

$$v_d = v + \frac{L \cdot \omega}{2} \tag{3.6}$$

Où : v est la vitesse linéaire du robot, ω est la vitesse de rotation du robot et L est la distance entre les roues gauche et droite.
Ces équations permettent de calculer les vitesses désirées des moteurs pour chaque coté du robot en se basant sur les vitesses linéaires et de rotation désirées.

3.5 Commande manuelle du robot mobile

La commande manuelle des robots mobiles est essentielle pour nous permettre de contrôler précisément les mouvements et les interactions du robot dans divers environnements. Dans cette partie de notre projet de fin d'études, nous expérimenterons le contrôle directionnel de notre robot en ajustant sa vitesse (rotation et translation) à l'aide des

touches du clavier sur notre ordinateur personnel, qui communique à travers une liaison WiFi avec la carte Raspberry Pi comme présentée dans la section 3.3.1. Cette dernière transmettra les commandes de vitesse de rotation et de translation à la carte Arduino bas niveau par le biais d'une communication série détaillée dans la section 3.3.2. Cette commande manuelle se base essentiellement sur un paquet ROS nommé "Teleop".

3.5.1 Description du package "Teleop" :

C'est un paquet ROS conçu pour la commande manuelle d'un robot mobile via le clavier. Il publie des messages du type *geometry_msgs/Twist* vers le topic *cmd_vel*. Ces messages définissent les valeurs de vitesse linéaire et angulaire désirées. Voici l'interface du package illustré ci-dessous :

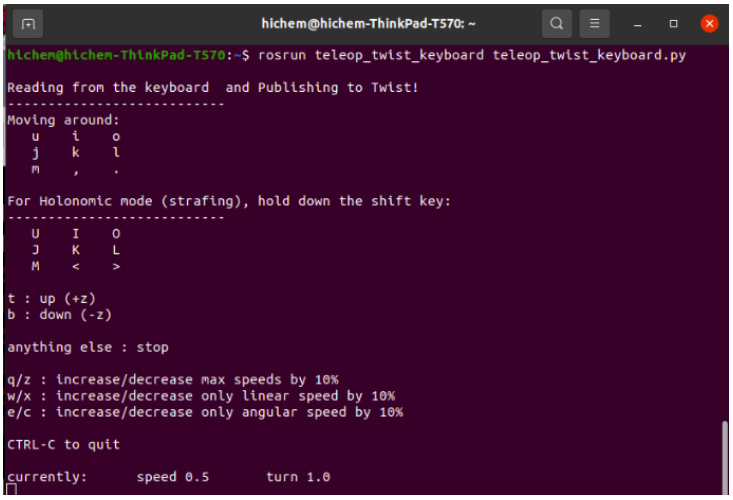


FIG. 3.40 – Interface *teleop_twist_keyboard*

- **i :** Marche avant.
- **, :** Marche arrière.
- **j :** Tourner à gauche.
- **l :** Tourner à droite.
- **u :** Avancer en tournant à gauche.
- **o :** Avancer en tournant à droite.
- **m :** Reculer en tournant à gauche.
- **. :** Reculer en tournant à droite.
- **k :** Arrêter.
- **w :** Augmenter la vitesse linéaire.
- **x :** Diminuer la vitesse linéaire.
- **e :** Augmenter la vitesse angulaire.
- **c :** Diminuer la vitesse angulaire.

Nous effectuons l'installation de ce package avec la commande suivante : *sudo apt install ros − noetic − teleop − twist − keyboard* .

3.5.2 Implémentation de la commande manuelle :

Après le lancement du Ros Master : *roscore* et de la communication établit entre le maître (Master) et l'esclave (Slave) comme décrite dans la section 1), nous lançons la communication rosserial pour établir la communication entre l'Arduino et le Raspberry Pi : *rosrun rosserial_pythonserial_node.py*.

Puisque le paquet "Teleop" publie ces vitesses de translation et de rotation désirées pour le robot sur le topic "*cmd_vel*", nous avons implémenté un souscripteur sur l'Arduino afin de collecter ces vitesses désirées et les transformer en vitesses désirées pour les moteur gauches et droits selon les équations 3.6 et 3.5. La commande manuelle du robot est prête à être exécutée en lançant le package teleop pour contrôler le robot à travers les touches du clavier : *rosrun teleop_twist_keyboard teleop_twist_keyboard.py*. Nous pouvons contrôler le robot par le clavier comme illustré sur la figure 3.40.

3.5.3 Test de la commande manuelle

Après le lancement de la commande, nous allons effectuer un test de celle-ci. Ce test consistera à varier les différentes vitesses du robot pour différentes vitesses désirées de translation v lors d'un mouvement rectiligne, et de rotation (ω) pour des rotations sur place.

1. **Mouvement rectiligne :** Visualisation des vitesses mesurées des côtés droit et gauche du robot illustrée sur la figure 3.41, où la vitesse linéaire v est d'abord fixée à 2 m/s, puis à -2 m/s illustrée sur la figure 3.42 en sens inverse :

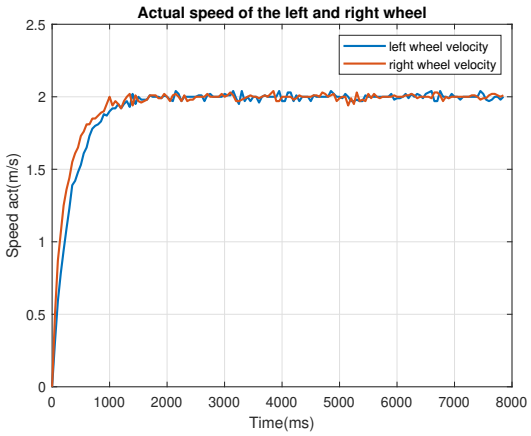


FIG. 3.41 – Commande de mouvement rectiligne du robot pour v =2 m/s

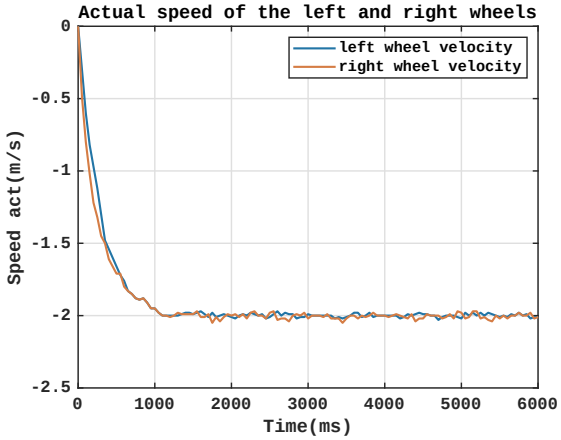


FIG. 3.42 – Commande de mouvement rectiligne du robot pour v = -2 m/s

2. **Mouvement de rotation sur place :** La figure 3.43 présente la visualisation des vitesses mesurées du robot lors d'une rotation sur place vers la gauche, avec une vitesse angulaire de $\omega = 4$ rd/s et une vitesse linéaire nulle. Ensuite, sur la figure 3.44, la rotation est vers la droite, avec une vitesse angulaire de $\omega = 8$ rd/s. Sur la figure 3.45, le robot tourne à gauche avec une vitesse angulaire de $\omega = 4$ rd/s et une vitesse linéaire nulle $v = 0m/s$. Enfin, sur la figure 3.46, le robot tourne à gauche avec une vitesse angulaire de $\omega = 8$ rd/s. Pendant ces rotations, les roues gauches tournent dans le sens directe tandis que les roues droites tournent dans le sens inverse pour que le robot tourne à droite, et vice versa pour tourner à gauche.

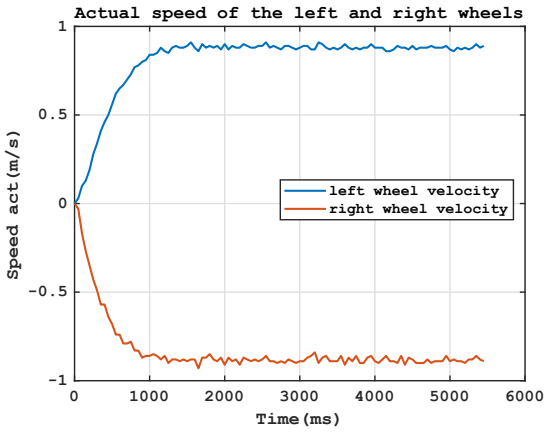


FIG. 3.43 – Robot tourne sur place (sens droite) avec $\omega = 4rd/s$ et $v = 0$

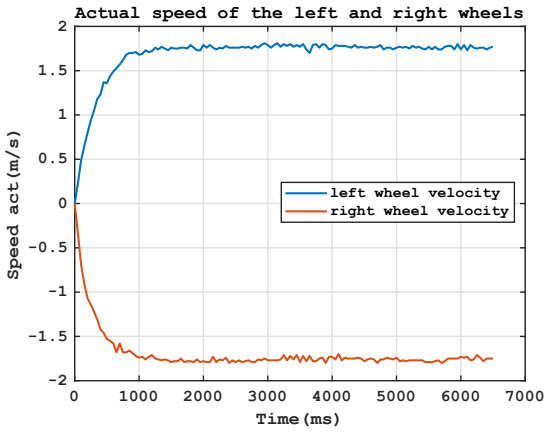


FIG. 3.44 – Robot tourne sur place (sens droite) avec $\omega = 8rd/s$ et $v = 0$

FIG. 3.45 – Robot tourne sur place (sens gauche) avec $\omega = 4rd/s$ et $v = 0$

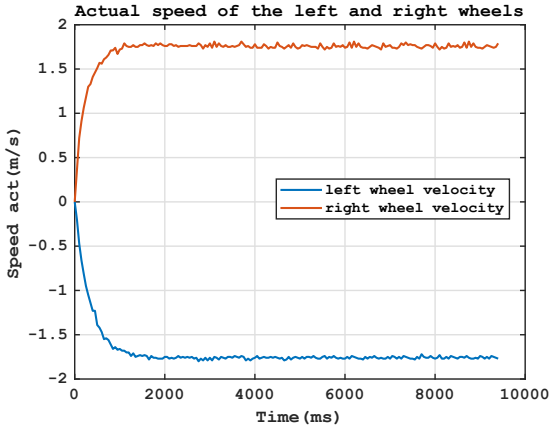


FIG. 3.46 – Robot tourne sur place (sens gauche) avec $\omega = 8rd/s$ et $v = 0$

3.6 Implémentation du SLAM

Après avoir la commande bas-niveau du robot et vérifier sa fiabilité et sa précision, nous passons maintenant à l'implémentation de la commande haut niveau qui a pour but d'assurer la navigation autonome du robot qu'on peut décomposer en plusieurs taches. Dans cette section nous allons commencé par la tache de la cartographie de l'environnement. Cette dernière est effectuée à l'aide de l'algorithme du grid-based SLAM déja décrit dans la section 1.7.3, qui utilise un filtre particulaire pour fusionner les données d'odométrie avec les données exterioceptifs du LiDAR. Cet algorithme est implémenté sous ROS sous forme d'un package appelé le Gmapping qui utilise des informations sur la distance parcourue (l'odométrie) et les données extéroceptives comme celles d'un LiDAR ou d'une caméra, afin d'évaluer la position du robot tout en élaborant une carte de son environnement. Nous allons maintenant expliquer ce package, comment l'installer, développer les noeuds nécessaires pour son fonctionnement, configurer ses paramètres et l'utiliser pour créer la carte de l'environnement.

3.6.1 Installation du package GMapping

L'installation de ce package se fait sur le Raspberry Pi on exécutant la commande suivante dans un terminal : *sudo apt − get install ros − noetic − gmapping*.
Afin d'établir la cartographie de l'environnement nous développons le framework nécessaire pour le paquet Gmapping. La figure 3.47 montre les différents noeuds et topics qui doivent être connectés avec le Gmapping :

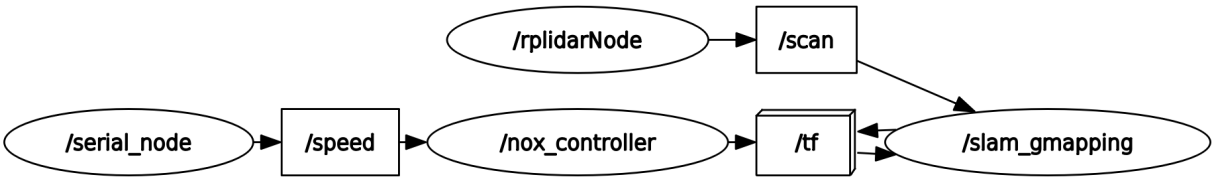


FIG. 3.47 – Schéma des noeuds utilisés pour le GMapping

Les informations venant du capteur LiDAR, les transformations entre les repères du robot et les données d'odométrie du robot doivent être mises à disposition, car le Gmapping s'abonne à ces topics pour générer la carte de l'environnement. Nous transmettons les données d'observation du LiDAR via le topic "*scan*", les données d'odométrie pour la prédiction via le topic "*odom*", et les transformations entre les repères du système via le topic "tf".

3.6.2 Développement des noeuds

1. **Exploitation du LiDAR C1 pour la cartographie :**
   Afin de visualiser les données produites par le LiDAR C1 dans l'outil de visualisation sous ROS qui est RViz, nous devons implémenter d'abords le noeud nommé *rpLiDARNode*. Ce noeud est responsable de la publication des données du LiDAR sur le topic ROS nommé "*scan*", ce qui permet à RViz de les récupérer et de les visualiser. Nous commençons par l'installation du package *rpLiDAR_ros* depuis le github de Slamtec, le fournisseur de ce Lidar :
   $ *git clone https* : *//github.com/Slamtec/rpLiDAR_ros.git*
   Pour lancer ce noeud, nous avons développé un fichier *launch* qui contient le noeud *rpLiDAR* en identifiant sur quel port le LiDAR C1 est connecté au Raspberry Pi :

```
<launch>
<!-- Paramètre du port USB -->
<param name="serial_port" value="/dev/ttyUSB0" />
<!-- Lancement du noeudrpLiDAR_node -->
<node pkg="rpLiDAR_ros" type="rpLiDARNode" name="rpLiDARNode" output="screen">
</node>
</launch>
```

   Nous pouvons désormais lancer le $launch\_file$ développé avec la commande :
   *roslaunch rpLiDAR_ros rpLiDAR_c*1.*launch*

   Après ces étapes, le noeud *rpLiDARNode* envoie des commandes de configuration

au capteur pour spécifier la fréquence de balayage souhaité. Une fois que le LiDAR reçoit la commande, il commence à faire le balayage laser de son environnement et il envoie ensuite ces données. Le noeud prend en charge le traitement des données provenant du capteur, puis les diffuse sur un topic ROS spécifique. Dans notre configuration, nous diffusons ces données sur le topic "scan". Enfin, nous avons visualisé les données du LiDAR sur RViz comme illustré dans la figure 3.48.

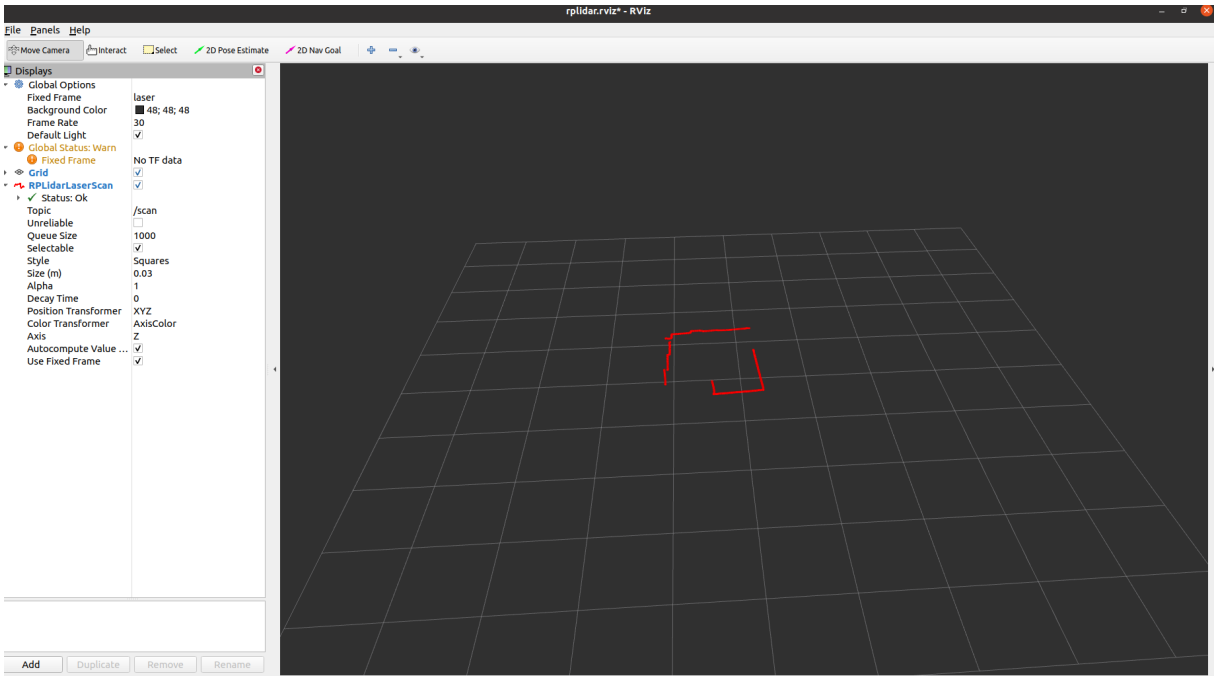


FIG. 3.48 – Visualisation des données du LiDAR sous RViz

2. **Développement du noeud de publication des transformations entre les repères du robot :**
   Un Publisher qui publie ces données sur le topic $'tf'$ a été programmé au sein d'un noeud développé. En premier lieu, nous créons un repère nommé "*map*" qui représente l'origine de la carte globale de l'environnement. Ce repère est fixé par rapport à un point de référence absolu dans l'environnement, souvent défini à la position initiale du robot sur la carte. Par la suite, nous avons le repère "*odom*" qui est fixé au robot, indiquant ainsi sa position par rapport au repère "*map*". Ce repère "*odom*" est utilisé pour établir une référence stable à la position du robot dans son environnement local. Passons maintenant au repère "*baselink*" qui représente la pose du robot (sa position et son orientation). Finalement, nous disposons du repère "*laser*" qui est associé au capteur LiDAR. Ce repère "*laser*" décrit la position et l'orientation du LiDAR par rapport au repère "*baselink*". Les données de balayage du LiDAR sont exprimées dans ce repère "*laser*", et grâce aux transformations fournies par le système "tf", ces données peuvent être transposées dans les repères "*baselink*", "*odom*" et "*map*". La figure 3.49 montre ces transformations :

   En utilisant les distances entre les repères sur le robot réel, nous avons réussi à mettre en oeuvre ces transformations entre eux en termes de translation et de rotation, puis à les publier sur le topic "tf". Ce système "tf" de ROS est essentiel pour la visualisation des données et des transformations dans RViz. Grâce à "tf", RViz

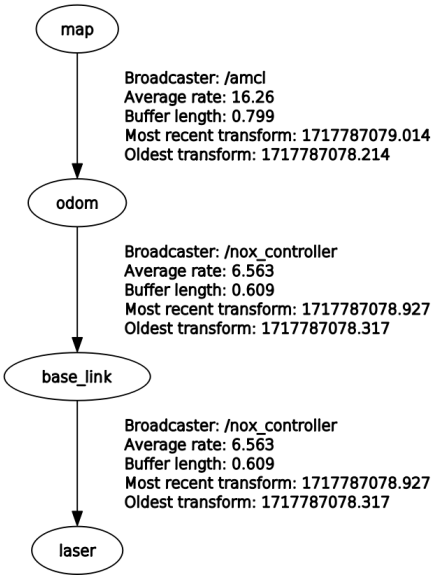


FIG. 3.49 – Les transformations entre les repères du robot

peut représenter visuellement les transformations entre les divers repères du robot, en affichant des flèches et des cadres pour illustrer leurs relations spatiales, comme le montre la Figure 3.50.

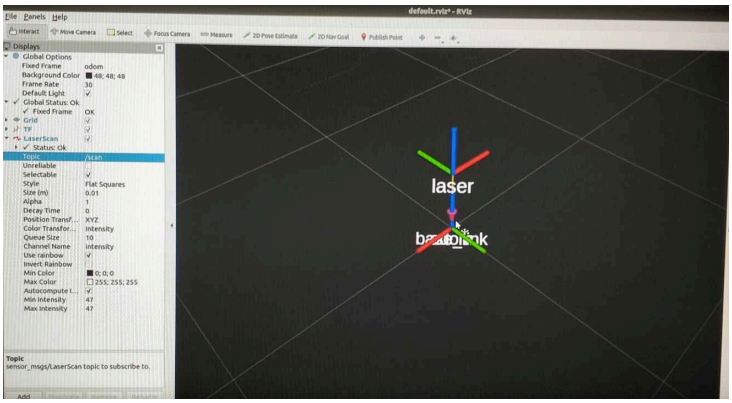


FIG. 3.50 – Visualisation des transformations entre les repères du robot sous RViz

3. **Publication des données odométriques dans le topic** "*odom*" **:**
Les données odométriques représentent l'emplacement du robot par rapport au point de départ. Pour obtenir ces calculs et les diffuser sur le topic "*odom*'", nous avons développé un noeud nommé "*odom*" sur le Raspberry Pi. Ce noeud calcule la distance parcourue par le robot en utilisant une méthode d'intégration d'Euler décrite dans les équations 3.7 et 3.8 à partir des vitesses gauche et droite du robot déterminées par les encodeurs comme expliqué précédemment dans la section 2.4.2. Ces vitesses sont publiées dans un topic nommé "*speed*", auquel notre noeud"*odom*" doit s'abonner pour calculer les nouvelles positions du robot. La position linéaire est calculée par :

$$\begin{aligned} x_{t+1} &= x_t + v_t \cdot \cos(\theta_t) \cdot \Delta t \\ y_{t+1} &= y_t + v_t \cdot \sin(\theta_t) \cdot \Delta t \end{aligned} \tag{3.7}$$

Tandis que l'orientation du robot est déterminée par :

$$\theta_{t+1} = \theta_t + \omega_t \cdot \Delta t \tag{3.8}$$

où x_t et y_t représentent les coordonnées actuelles du robot, v_t est la vitesse du robot, θ_t est l'angle de rotation du robot et Δt est l'intervalle de temps entre les mises à jour des positions (égale à 50 ms).

Pour démarrer tous ces noeuds simultanément, nous avons développé deux fichiers de lancement distincts (Launch files). Le premier est conçu pour être exécuté sur le Raspberry Pi et comprend le noeud de connexion avec l'Arduino, appelé "*ROSSerial*", qui publie les vitesses des deux moteurs sur le topic "*speed*". Ce fichier contient également le noeud "*nox_controller*", responsable de la publication de la distance parcourue par le robot sur le topic "*odom*" et de la gestion des transformations sur le topic "*tf*". De plus, il initialise le noeud de connexion avec le LiDAR, nommé "*rpLiDARNode*", en spécifiant le port USB auquel il est connecté et le noeud GMapping qui permet à faire la cartographie de l'environnement. La figure 3.51 illustre notre premier launch file. Le deuxième fichier de lancement intègre le noeud RViz, permettant la visualisation des données en temps réel. La figure 3.52 illustre notre deuxième launch file.

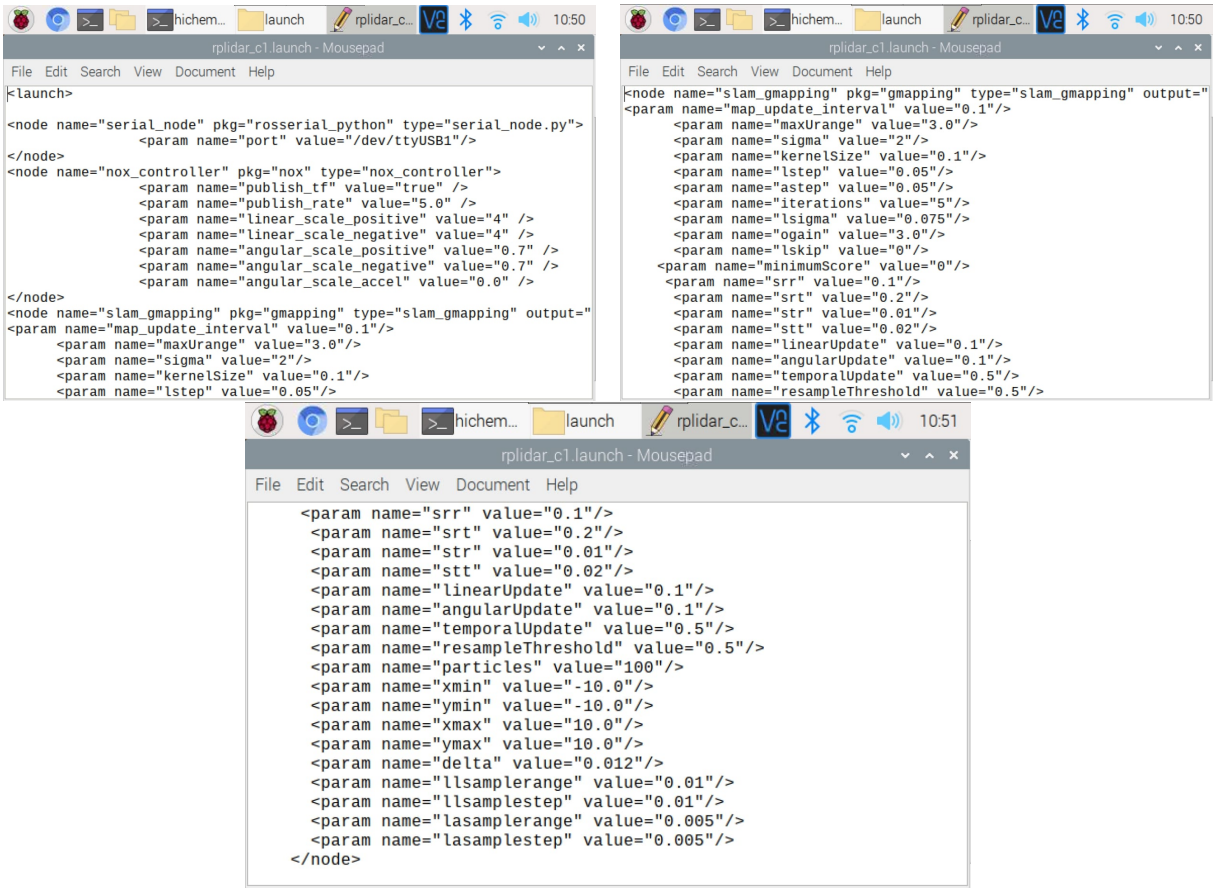


FIG. 3.51 – *rpLiDAR_c1* Launch file

```
<!--
  Used for visualising rplidar in action.

  It requires rplidar.launch.
 -->
<launch>

  <node name="rviz" pkg="rviz" type="rviz" args="-d $(find nox)/cfg/rviz_amcl_navigation.rviz" />
</launch>
```

FIG. 3.52 – Launch file pour lancer la visualisation sous RViz

3.6.3 Configuration des paramètres de Gmapping

Pour améliorer l'efficacité de GMapping et garantir une cartographie précise représentant fidèlement l'environnement, il est crucial d'ajuster certains paramètres afin d'optimiser le scan matching et d'augmenter le nombre effectif de particules n_{eff}. Ce dernier constitue une mesure clé dans les algorithmes de filtre à particules, évaluant la diversité des particules et indiquant la nécessité de ré-échantillonner, comme défini par l'équation 3.9.

$$n_{eff} = \frac{1}{\sum_{i=1}^{N} \omega_i^2} \tag{3.9}$$

où N représente le nombre total de particules. Lorsque les poids des particules deviennent très disparates, signalant une faible diversité, certaines particules présentent des poids considérablement plus élevés que d'autres. Dans ce contexte, n_{eff} diminue, indiquant la nécessité de ré-échantillonner les particules pour éviter leur dégradation. En résumé, le scan matching aligne les observations avec les prédictions pour ajuster les poids des particules, tandis que n_{eff} reflète la qualité de cette distribution de particules.
Les paramètres du package Gmapping nécessaires pour régler le coefficient n_{eff} sont :

1. **Sigma σ :** C'est la mesure de l'écart type de la distribution gaussienne employée afin de représenter la variabilité des observations des capteurs.
2. **KernelSize :** Ce paramètre influence la dimension du noyau tel qu'illustré dans la figure 3.53, pour lisser les observations et atténuer les effets des oscillations mineures et du bruit dans les mesures. Un noyau plus large peut rendre la carte plus résistante au bruit, mais peut aussi entraîner une perte possible de détails fins.

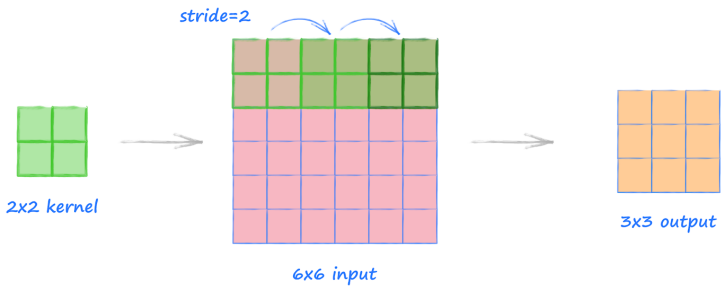


FIG. 3.53 – Connexion entre les noeuds de l'opération de cartographie [31]

3. **LinearUpdate :** Ce réglage détermine la distance linéaire (en mètres) que le robot doit parcourir avant que le processus de cartographie avec GMapping ne mette à jour la carte.

4. **AngularUpdate :** Ce paramètre détermine l'angle de rotation (en radians) que le robot doit effectuer avant que GMapping ne mette à jour la carte. Tout comme pour LinearUpdate, une valeur plus basse entraîne des mises à jour plus fréquentes, ce qui peut être bénéfique pour capturer des détails.

3.7 La localisation sur la carte construite

Après avoir accomplit la cartographie de l'environnement, une autre tache essentielle dans la navigation est la localisation du robot au sein de cette carte construite. Pour situer notre robot sur la carte que nous avons construit, nous ferons appel à l'algorithme de l'AMCL qui a le but d'estimer la pose du robot dans une carte préalablement établie car cet algorithme repose sur l'hypothèse que la carte est statique et connue à l'avance. Nous allons décrire dans ce qui suit, l'implémentation de l'AMCL sur ROS afin de permettre à notre robot de se localiser avec précision.

3.7.1 Implémentation de l'AMCL sur ROS

1. **Préparation de l'environnement :** Nous avons lancé le noeud "*mapserver*" pour charger la carte construite à partir des fichiers image pgm et du fichier YAML correspondant et qui sont déjà enregistré à l'aide de la commande *rosrun map_server map_saver −f my_map* . Le noeud "*mapserver*" diffuse la carte chargée sur le topic "*map*" pour permettre au noeud de l'AMCL de l'accéder.
2. **Configuration des paramètres :** Pour obtenir des performances précises de l'AMCL, il est essentiel de bien régler ses paramètres. Cette tâche peut sembler facile, mais en réalité, elle est assez difficile car elle nécessite un ajustement délicat en fonction des caractéristiques spécifiques de notre robot. La figure 3.54 illustre la configuration des paramètres de notre système AMCL, ainsi que leurs valeurs correspondantes.

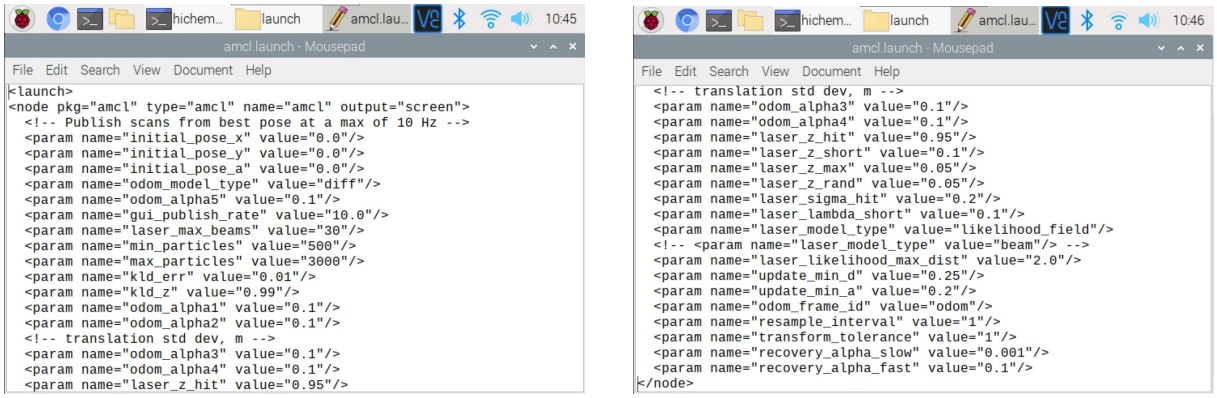


Fig. 3.54 – Launch file de l'AMCL

.

3. **Exécution d'AMCL :** Pour l'exécution de l'AMCL nous avons développé un fichier *launch* pour lancer tout ces noeuds ensemble. La figure 3.55 montre l'architecture de l'AMCL pour lancer ces noeuds.

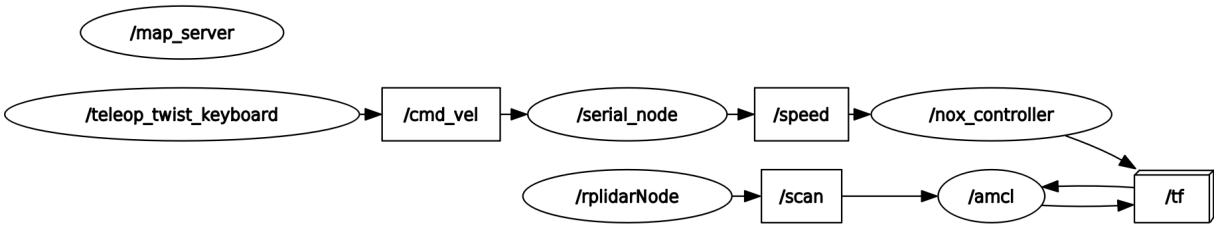


FIG. 3.55 – Rqt d'AMCL

3.8 Planification de trajectoire

Une fois la cartographie et la localisation sont effectuées, le robot doit pouvoir se déplacer de manière efficace et sécurisée vers des objectifs spécifiques. La planification de trajectoire vise à définir le chemin optimal que le robot doit emprunter pour atteindre sa destination. Pour cela, nous avons utilisé deux planificateurs : un planificateur global basé sur l'algorithme Dijkstra et un planificateur local utilisant l'algorithme DWA. Ces méthodes sont détaillées dans la section 1.7.4.

3.8.1 Implémentation des planificateurs global et local sur ROS

Afin d'implémenté la planification des chemins sur ROS, nous avons développé un fichier de lancement exécutant le noeud "*move_base*", intégrant les planificateurs global et local, ainsi qu'un fichier de lancement pour le noeud de l'algorithme de localisation AMCL sur le Raspberry Pi. Le noeud "*move_base*" de ROS combine ces deux planificateurs pour assurer une navigation autonome efficace et sécurisée. Le planificateur global (basé sur l'algorithme Dijkstra) génère une trajectoire optimale depuis la position actuelle du robot jusqu'à la destination désirée. En parallèle, le planificateur local (utilisant l'algorithme DWA) suit cette trajectoire en temps réel tout en réagissant aux obstacles imprévus rencontrés sur le chemin. Lorsqu'un obstacle est détecté, le planificateur local ajuste dynamiquement les commandes de vitesse du robot pour éviter la collision tout en restant aussi proche que possible de la trajectoire globale. La Figure 3.57 affiche le fichier *launch* qui exécute le noeud "*move_base*", et la Figure 3.56 montre l'architecture de cette implémentation pour lancer touts ces noeuds.

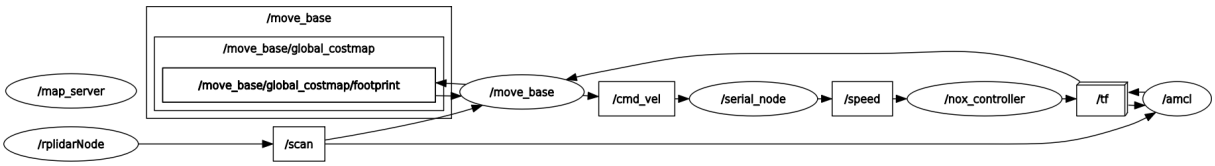


FIG. 3.56 – Architecture des planificateurs du chemin

FIG. 3.57 – Fichier launch "*move_base*"

3.9 Conclusion

Dans ce chapitre, nous avons décrit la mise en oeuvre d'un Framework basé sur ROS permettant l'implémentation des stratégies de navigation autonome. Nous entamons par une définition détaillée du ROS, ses avantages, ses applications et ses outils de visualisation. Nous passons par la suite à développé un framework ROS afin d'établir la communication entre les différentes parties de notre système. Ensuite, nous passons à assurer la commande bas-niveau du robot en mettant en place un asservissement en vitesse des moteurs en utilisant des contrôleurs PID, dont la validation s'opère grâce à des mesures précises obtenues via un tachymètre. Une commande manuelle du robot est également développée pour tester l'exactitude de l'asservissement en vitesse à travers des différents mouvements et le bon fonctionnement des liaisons de communication.

En parallèle, la commande haut-niveau développée sous ROS est présentée. Nous intégrons la technique de grid-based SLAM, via le package GMapping, permettant au robot de générer une carte de son environnement tout en se localisant avec précision. L'ajustement de ces paramètres contribue à améliorer à la fois la précision et l'efficacité de cette cartographie. Par la suite, nous recourons à l'algorithme AMCL pour localiser le robot sur la carte élaborée, augmentant ainsi la robustesse et la précision de la navigation. Enfin, nous avons implémenté des planificateurs de chemin local et global pour permettre au robot d'assurer une navigation autonome. Dans le prochain chapitre, nous discuterons des résultats obtenus et exposerons les résultats de l'implémentation de ces méthodes de navigation autonome du robot présentées dans ce chapitre.

CHAPITRE

4

INTERPRÉTATION ET DISCUSSION DES RÉSULTATS

4.1 Introduction

Ce chapitre se concentre sur la présentation et l'analyse des résultats obtenus tout au long des différentes étapes de notre projet. Nous débutons par une simulation de notre robot sur Gazebo afin de reproduire son comportement dans des environnements virtuels, cela nous permet d'analyser et de comprendre comment il réagit dans différentes situations avant de le déployer dans le monde réel. Par la suite, nous passons à évaluer le système odomètrique développé afin de tester sa précision et sa fiabilité dans des conditions variées et pour garantir son bon fonctionnement lors de la navigation du robot. Nous allons ensuite exposer les résultats obtenus lors de la cartographie de notre environnement, en évaluant l'efficacité de notre implémentation du SLAM sous ROS, présentée dans le chapitre précédent. Par la suite, nous analyserons la performance de la localisation du robot sur la carte crée, afin de confirmer la fiabilité de notre approche de localisation hybride basée sur l'AMCL. Nous terminons le chapitre par la présentation des résultats relatifs à la navigation autonome du robot qui constitue l'objectif ultime de notre travail.

4.2 Simulation du robot sous Gazebo

Avant de déployer notre robot dans le monde réel, il est essentiel de le tester en simulation sur Gazebo. Cela nous permet de vérifier son comportement dans des environnements virtuels, ce qui nous aide à l'améliorer avant de le déployer dans un environnement réel. Avant de procéder à la simulation du robot sous Gazebo, il est nécessaire d'installer des packages spécifiques qui nous permettront d'effectuer cette simulation. Ces packages sont illustrés dans la Figure 4.1.

```
adel@adel-Victus-by-HP-Gaming-Laptop-15-fa0xxx:~$ sudo apt-get install ros-noetic-gazebo-ros-pkgs
[sudo] password for adel:
Reading package lists... Done
Building dependency tree
Reading state information... Done
ros-noetic-gazebo-ros-pkgs is already the newest version (2.9.2-1focal.20240111.185854).
0 upgraded, 0 newly installed, 0 to remove and 52 not upgraded.
adel@adel-Victus-by-HP-Gaming-Laptop-15-fa0xxx:~$ sudo apt-get install ros-noetic-gazebo-msgs
Reading package lists... Done
Building dependency tree
Reading state information... Done
ros-noetic-gazebo-msgs is already the newest version (2.9.2-1focal.20230620.191337).
0 upgraded, 0 newly installed, 0 to remove and 52 not upgraded.
adel@adel-Victus-by-HP-Gaming-Laptop-15-fa0xxx:~$ sudo apt-get install ros-noetic-gazebo-plugins
Reading package lists... Done
Building dependency tree
Reading state information... Done
ros-noetic-gazebo-plugins is already the newest version (2.9.2-1focal.20240111.182925).
0 upgraded, 0 newly installed, 0 to remove and 52 not upgraded.
adel@adel-Victus-by-HP-Gaming-Laptop-15-fa0xxx:~$ sudo apt-get install ros-noetic-gazebo-ros-control
Reading package lists... Done
Building dependency tree
Reading state information... Done
ros-noetic-gazebo-ros-control is already the newest version (2.9.2-1focal.20231030.154912).
0 upgraded, 0 newly installed, 0 to remove and 52 not upgraded.
adel@adel-Victus-by-HP-Gaming-Laptop-15-fa0xxx:~$ sudo apt-get install ros-noetic-teleop-twist-keyboard
Reading package lists... Done
Building dependency tree
Reading state information... Done
ros-noetic-teleop-twist-keyboard is already the newest version (1.0.0-1focal.20230620.184914).
0 upgraded, 0 newly installed, 0 to remove and 52 not upgraded.
adel@adel-Victus-by-HP-Gaming-Laptop-15-fa0xxx:~$
```

FIG. 4.1 – Packages installés

4.2.1 Modélisation du robot

Avant de simuler le robot, il est nécessaire de créer son modèle sur le simulateur. Le robot peut être modélisé en trois fichiers, le premier est nommé "*robot.xacro*", le deuxième est nommé "*robot.gazebo*" et le troisième est nommé "*robot_xacro.launch*". Le premier fichier définit la géométrie du robot, notamment les dimensions du châssis et des roues, sa masse, la densité des matériaux et les moments d'inertie des roues. La Figure 4.2 illustre le contenu de ce fichier.

Open ~/robot_four/src/robot_model_pkg/urdf robot.xacro Save

4 roues robot.xacro

```
1 <?xml version="1.0"?>
2 <!-- ######################################  -->
3 <!-- DESCRIPTION OF THE 4-WHEELED ROBOT      -->
4 <!-- Made by ALLAK and BESSGHIER              -->
5 <!-- FEBRUARY 2024                           -->
6 <!-- ######################################  -->
7 <robot name="differential_drive_robot" xmlns:xacro="http://www.ros.org/wiki/xacro">
8
9 <!-- Body dimensions -->
10 <xacro:property name="body_link_x_dim" value="1"/>
11 <xacro:property name="body_link_y_dim" value="0.6"/>
12 <xacro:property name="body_link_z_dim" value="0.3"/>
13
14 <!-- Wheel dimensions -->
15 <xacro:property name="wheel_link_radius" value="0.15"/>
16 <xacro:property name="wheel_link_length" value="0.1"/>
17 <xacro:property name="wheel_link_z_location" value="-0.1"/>
18
19 <!-- Material density -->
20 <xacro:property name="body_density" value="2710.0"/>
21 <xacro:property name="wheel_density" value="2710.0"/>
```

FIG. 4.2 – Fichier robot.xacro

Le deuxième fichier contient les coefficient de frottement, les couleurs du châssis et des roues et les paramètres de notre contrôleur. La Figure 4.3 illustre le contenu de ce fichier.

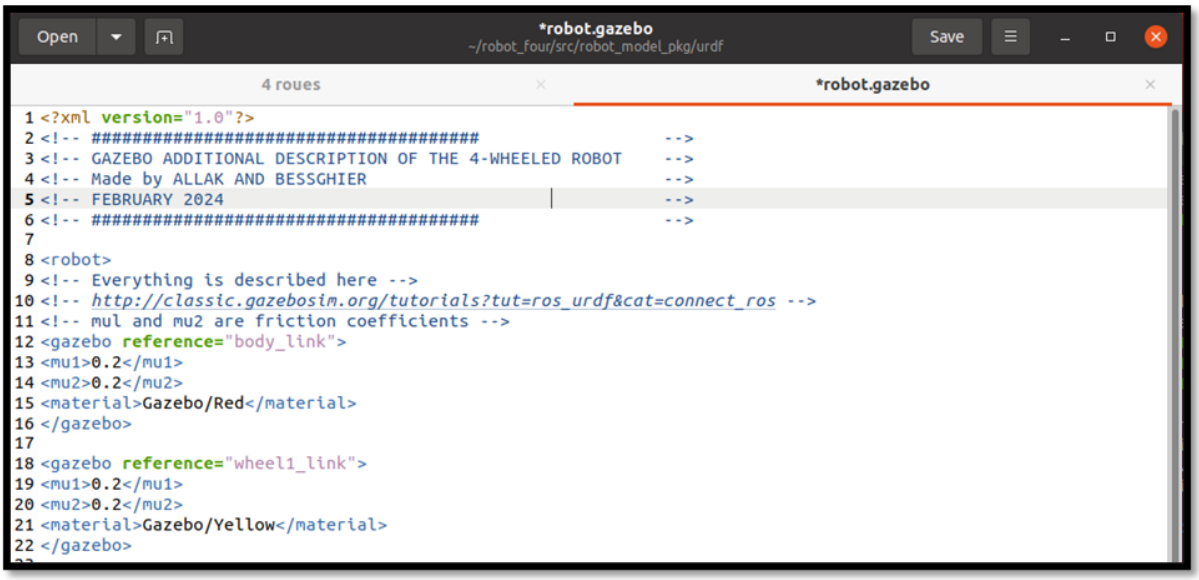


FIG. 4.3 – Fichier robot.gazebo

Le troisième fichier est un fichier de lancement (launch file) qui inclut le monde sur Gazebo dans lequel nous souhaitons réellement intégrer notre robot, c'est-à-dire un environnement en 3D. La Figure 4.4 illustre le contenu de ce fichier.

FIG. 4.4 – Fichier de lancement robot_xacro.launch

4.2.2 Contrôle du robot

Aprés avoir modélisé le robot, nous allons le déployer sous Gazebo afin de simuler ces mouvements dans un environnement que nous avons définit précédemment. Ceci est établit en exécutant la commande *roslaunch robot_model_pkg robot_xacro.launch*. La Figure 4.5 montre la simulation sous Gazebo.

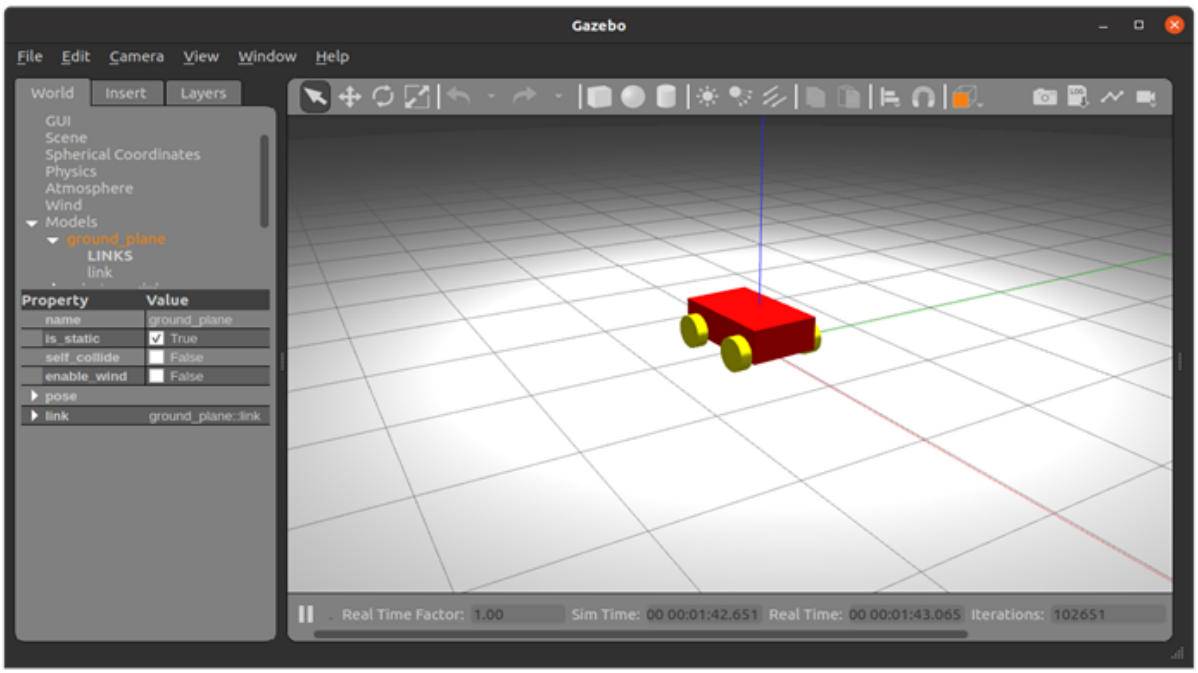


FIG. 4.5 – Simulation sous Gazebo

Le contrôle du robot est géré par le package "*teleop*", défini précédemment dans la section 3.5.1, en utilisant la commande *rosrun teleop_twist_keyboard teleop_twist_keyboard.py*. L'exécution à la fois du fichier de lancement *robot_xacro.launch* et le package *teleop* nous permet de contrôler notre robot et changer sa vitesse angulaire et linéaire par les touches du clavier comme la Figure 4.6 l'illustre.

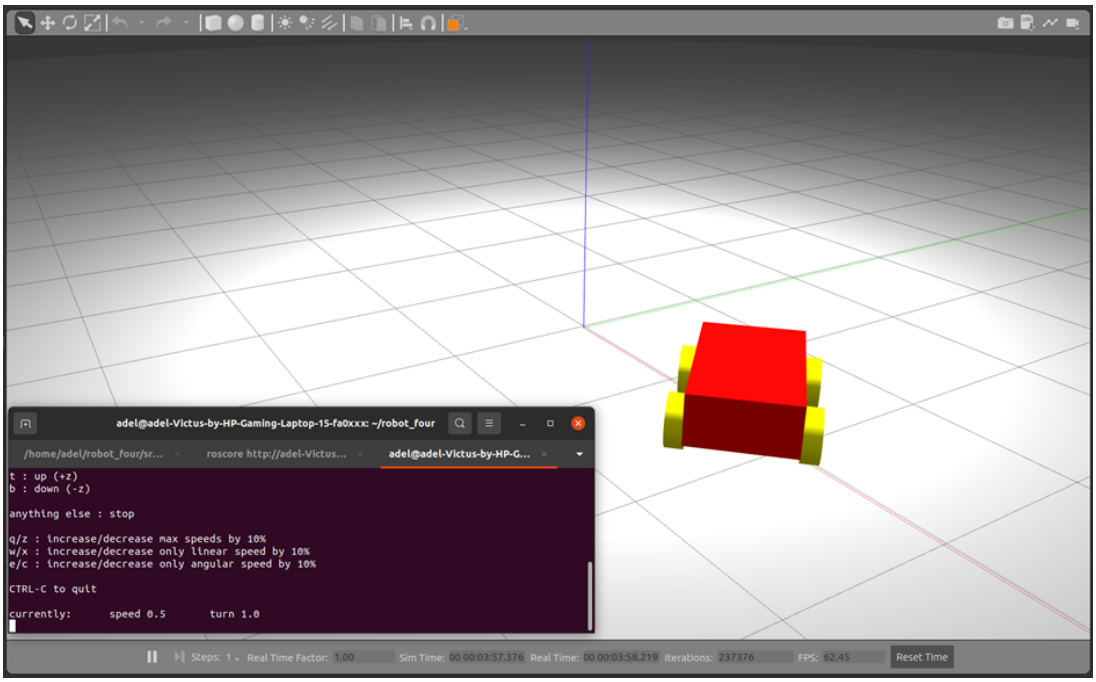


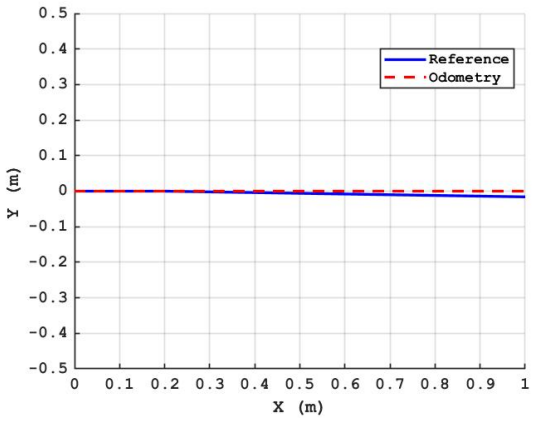
FIG. 4.6 – Contrôle du robot sous Gazebo

4.3 Vérification du système odométrique

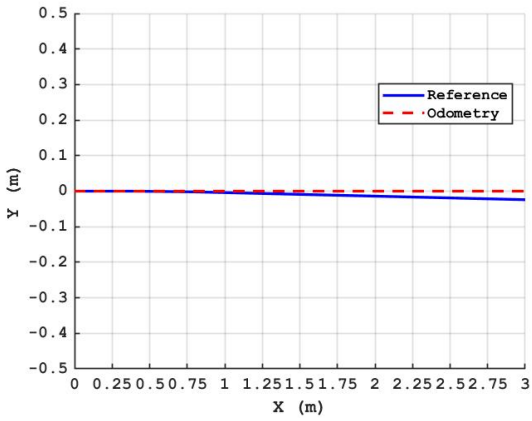
Dans cette section, nous allons évaluer la précision d'odomètrie de notre robot réalisé. Pour ce faire, nous allons le soumettre à deux types de tests de mouvement : d'abord, un test de déplacement en ligne droite, puis un test de déplacement circulaire. En examinant comment le robot se comporte dans ces deux situations différentes, nous pourrons mieux comprendre ses capacités d'odométrie, c'est-à-dire sa capacité à suivre et à enregistrer ses propres déplacements avec précision.

4.3.1 Mouvement rectiligne :

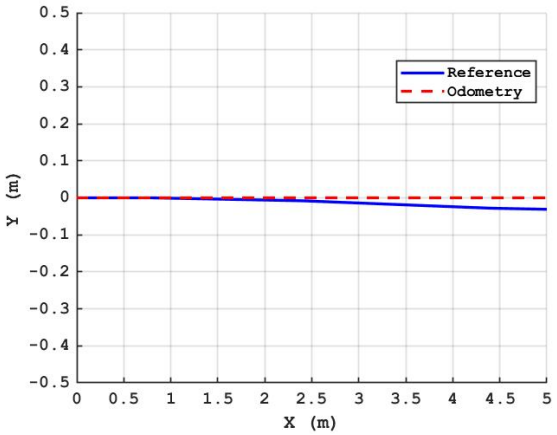
Nous allons démarrer notre robot à différentes vitesses de translation (v= 0.1 m/s, v = 0.3 m/s et v = 0.5 m/s)) pendant 10 secondes, ce qui signifie que le robot parcourra respectivement 1 mètres, 3 mètres et 5 mètre à ces vitesses. Les résultats de l'odomètre pour le mouvement rectiligne sont présentés dans la figure 4.7. La pose du robot est calculée en utilisant l'intégration d'Euler comme il est indiqué par les équations 3.7 et 3.8.



(a) Résultat de l'odomètre pour 1 mètre



(b) Résultat de l'odomètre pour 3 mètre



(c) Résultat de l'odomètre pour 5 mètre

FIG. 4.7 – Résultat de l'odomètre pour le mouvement rectiligne du robot

Il est clairement visible dans la figure 4.7 que la déviation de l'odomètre par rapport à la référence augmente avec la distance parcourue, ce qui est l'un des inconvénients du

système de localisation par odomètre. Cependant, malgré cela, nous pouvons constater que la précision de ce module reste satisfaisante pour ces différentes vitesses, mais il sera nécessaire d'apporter des ajustements futurs pour améliorer la précision et réduire les erreurs accumulées sur de longues distances.

4.3.2 Mouvement circulaire :

Pour évaluer la performance de notre robot lorsqu'il est en mouvement circulaire, nous avons réalisé des tests à une vitesse de déplacement linéaire de 0.50 m/s, en variant les rayons de courbure (0.5 m, 1 m, 1.5 m). Les vitesses de rotation ont été calculées à l'aide de la formule $\omega = \frac{v}{r}$.

À l'aide de l'équation 4.1, nous avons calculer les erreurs associées pour chaque combinaison de vitesse de translation et de rayon de courbure. Les vitesses de rotation et les erreurs pour les différentes vitesses de translation et rayons de courbure sont détaillées dans le tableau 4.1.

$$\varepsilon = \frac{\sqrt{(x_{\text{réel}} - x_{\text{désiré}})^2 + (y_{\text{réel}} - y_{\text{desire}})^2}}{\sqrt{x_{\text{désiré}}^2 + y_{\text{désiré}}^2}} \tag{4.1}$$

Tab. 4.1 – Résultats de mouvement circulaire

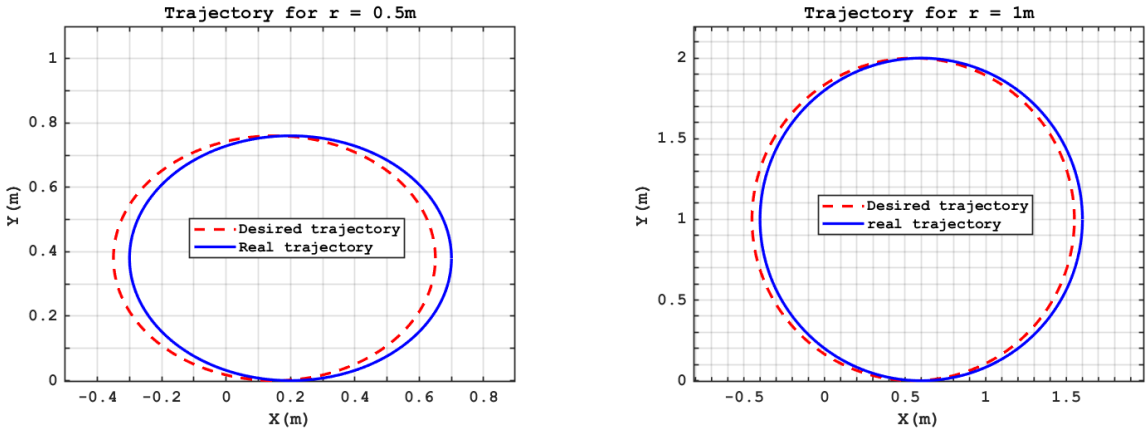
Vitesse de translation	r = 0.5m	Erreur (%)	r = 1m	Erreur (%)	r = 1.5m	Erreur (%)
0.50 [m/s]	ω =1 rd/s	9.6	ω =0.5 rd/s	2.2	ω =0.33 rd/s	5.7

Nous pouvons observé que l'erreur est minimale avec un rayon de 1 m (2.2%) cela indique une trajectoire optimale pour notre robot, comparativement aux rayons de 0.5 m et 1.5 m. L'erreur augmente avec un rayon de 0.5 m en raison de la courbe serrée et du glissement, tandis qu'elle augmente avec un rayon de 1.5 m en raison de la grande distance parcourue. La Figure 4.8 illustre les trajectoires réelles et les trajectoires désirées pour différents rayons de courbure.

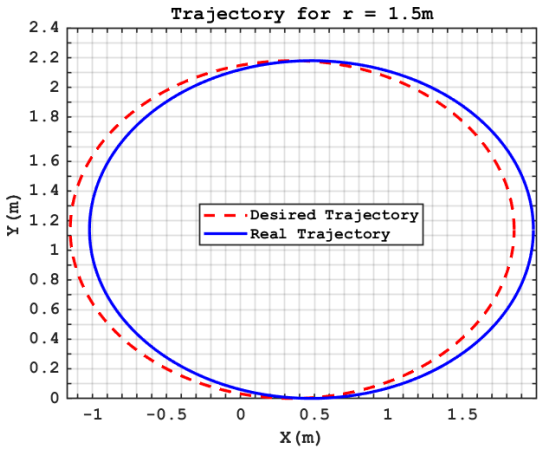
Les résultats démontrent que notre robot présente une performance optimale lorsqu'il effectue des mouvements circulaires avec un rayon de 1m, où l'erreur est minimale. Cette erreur tend à augmenter pour des rayons plus petits (0.5m) et plus grands (1.5m), en raison des limitations mécaniques et des glissements survenant dans les trajectoires resserré ou sur de longues distances.

4.4 Cartographie de l'environnement

Aprés la vérification de l'asservissement en vitesse du robot et le système odométrique implémenté, nous passons maintenant à la commande haut-niveau qui permet d'assurer la navigation autonome du robot en débutant par la cartographie de l'environnement. Dans notre projet, nous avons adopté l'algorithme Grid-based SLAM, une méthode permettant de construire une représentation détaillée de notre environnement. Cette méthode est déjà expliquée en détail dans la section 1.7.3. Cette implémentation a été réalisée dans l'environnement ROS en utilisant le package 'Gmapping' déjà détaillé dans la section 3.6.



(a) Rayon égal à 0.5m (b) Rayon égal à 1m



(c) Rayon égal à 1.5m

FIG. 4.8 – Trajectoires réelles et trajectoires désirées pour différents rayons de courbure

Avant de procéder à la cartographie de notre environnement réel avec le robot, nous avons réalisé une étape préliminaire en simulant le processus avec un robot mobile à 4 roues virtuel nommé "Husky". Ce dernier, comme notre robot réalisé, est considéré aussi comme un robot différentiel de type unicycle. Cette phase nous a permis de simuler la création de la carte de l'environnement dans Gazebo et d'observer les résultats via RViz. Les détails de cette simulation seront présentés dans la section qui suit.

4.4.1 Cartographie d'un environnement virtuel sur Gazebo

Afin de simuler le robot virtuel "Husky", il est nécessaire d'installer des packages spécifiques du robot "Husky" présentés sur la Figure 4.9. Ces packages comprennent des modèles de simulation du robot "Husky" lui-même, ainsi que des modèles pour ses capteurs, tels que les LiDARs et les caméras. De plus, des packages pour l'intégration du "Husky" dans Gazebo ainsi que pour la visualisation dans RViz sont également inclus.

```
adel@adel-Victus-by-HP-Gaming-Laptop-15-fa0xxx:~$ sudo apt-get install ros-noetic-gazebo-ros-pkgs
[sudo] password for adel:
Reading package lists... Done
Building dependency tree
Reading state information... Done
ros-noetic-gazebo-ros-pkgs is already the newest version (2.9.2-1focal.20240111.185854).
0 upgraded, 0 newly installed, 0 to remove and 52 not upgraded.
adel@adel-Victus-by-HP-Gaming-Laptop-15-fa0xxx:~$ sudo apt-get install ros-noetic-gazebo-msgs
Reading package lists... Done
Building dependency tree
Reading state information... Done
ros-noetic-gazebo-msgs is already the newest version (2.9.2-1focal.20230620.191337).
0 upgraded, 0 newly installed, 0 to remove and 52 not upgraded.
adel@adel-Victus-by-HP-Gaming-Laptop-15-fa0xxx:~$ sudo apt-get install ros-noetic-gazebo-plugins
Reading package lists... Done
Building dependency tree
Reading state information... Done
ros-noetic-gazebo-plugins is already the newest version (2.9.2-1focal.20240111.182925).
0 upgraded, 0 newly installed, 0 to remove and 52 not upgraded.
adel@adel-Victus-by-HP-Gaming-Laptop-15-fa0xxx:~$ sudo apt-get install ros-noetic-gazebo-ros-control
Reading package lists... Done
Building dependency tree
Reading state information... Done
ros-noetic-gazebo-ros-control is already the newest version (2.9.2-1focal.20231030.154912).
0 upgraded, 0 newly installed, 0 to remove and 52 not upgraded.
adel@adel-Victus-by-HP-Gaming-Laptop-15-fa0xxx:~$ sudo apt-get install ros-noetic-teleop-twist-keyboard
Reading package lists... Done
Building dependency tree
Reading state information... Done
ros-noetic-teleop-twist-keyboard is already the newest version (1.0.0-1focal.20230620.184914).
0 upgraded, 0 newly installed, 0 to remove and 52 not upgraded.
adel@adel-Victus-by-HP-Gaming-Laptop-15-fa0xxx:~$
```

FIG. 4.9 – Les packages nécessaires pour la cartographie

Une fois que ces packages sont installés et configurés correctement, nous pouvons lancer la simulation du robot "Husky" dans Gazebo et visualiser son comportement dans RViz. Le lancement de la simulation s'effectue en exécutant la commande suivante dans un terminal : `roslaunch husky_gazebo husky_playpen.launch`. La Figure 4.10 affiche la simulation du robot sous Gazebo. Par la suite, nous lançons la visualisation de l'environnement (Figure 4.11) par la commande : `roslaunch husky_viz view_robot.launch`.

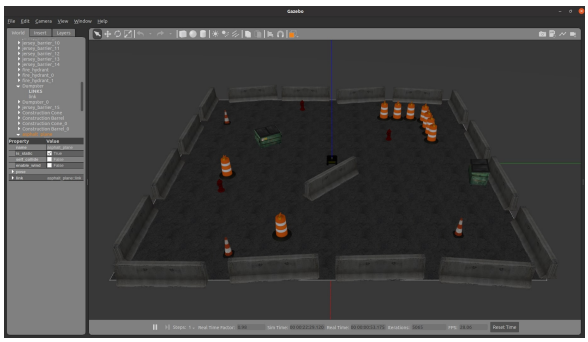


FIG. 4.10 – Simulation sous Gazebo

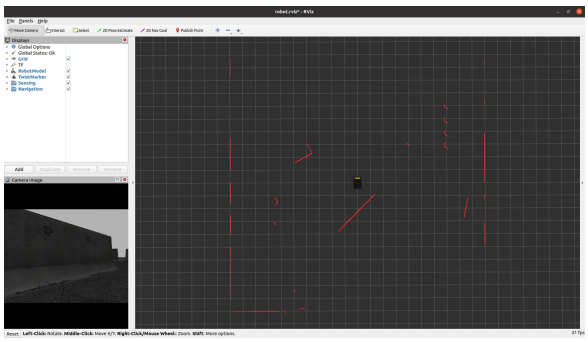


FIG. 4.11 – Visualisation sous RViz

Afin d'établir la cartographie de l'environnement, nous avons implémenté le package gmapping. Le lancement de la cartographie est éffectuée par la commande : `roslaunch husky_navigation gmapping.launch`. La Figure 4.12 montre la cartographie avant le déplacement du robot. Afin de cartographier l'environnement, le robot doit être déplacé dans les différents localisations sur l'environnement. Ceci est effectué en implémentant le package "*teleop*" détaillé dans la section 3.5.1. Lors du déplacement du robot, le LiDAR reçoit de nouvelles données de perception et la carte est établit en parallèle. La Figure 4.13 montre la cartographie complète du notre environnement après avoir déplacé le robot.

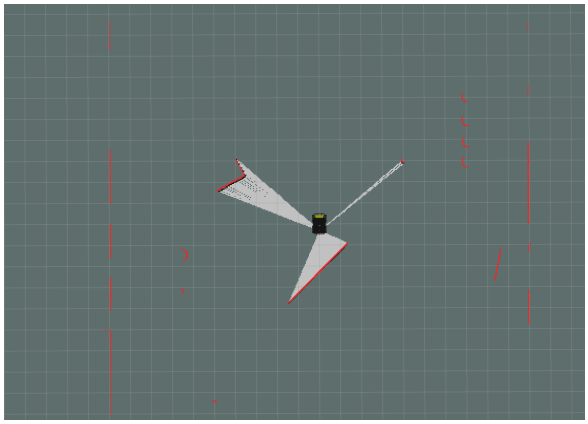


FIG. 4.12 – Début de la cartographie

FIG. 4.13 – La cartographie complète

Une fois la cartographie de l'environnement terminée, nous sauvegardons la carte pour l'utiliser ultérieurement dans la navigation. Pour ce faire, nous implémentons le package $'map_server'$ et nous lançons la commande suivante : `rosrun map_server map_saver -f <NOM_DU_FICHIER>`.

4.4.2 Implémentation réelle du SLAM sur le robot mobile

Après avoir cartographié le monde virtuel 4.4.1 avec notre robot, nous entamons maintenant la cartographie de l'environnement réel en déplaçant le robot réalisé équipé d'un capteur LiDAR à partir du clavier de la station sol. Les détails de la cartographie de l'environnement concernant les noeuds et les fichiers 'launch' développés, les algorithmes SLAM implémentés avec le réglage des paramètres sont tous exposés dans la section 3.6. Dans cette section, nous allons présenter les résultats concernant l'analyse de la cartographie effectuée.

Pour ce faire, nous choisissons un environnement réel à cartographier où nous mettons quelques obstacles statiques comme représenté sur la Figure 4.14. Notre objectif est d'établir une carte de cet environnement en déplaçant le robot le long de cet environnement tout en enregistrant les données captées par le LiDAR sur la carte.

FIG. 4.14 – Notre carte réelle

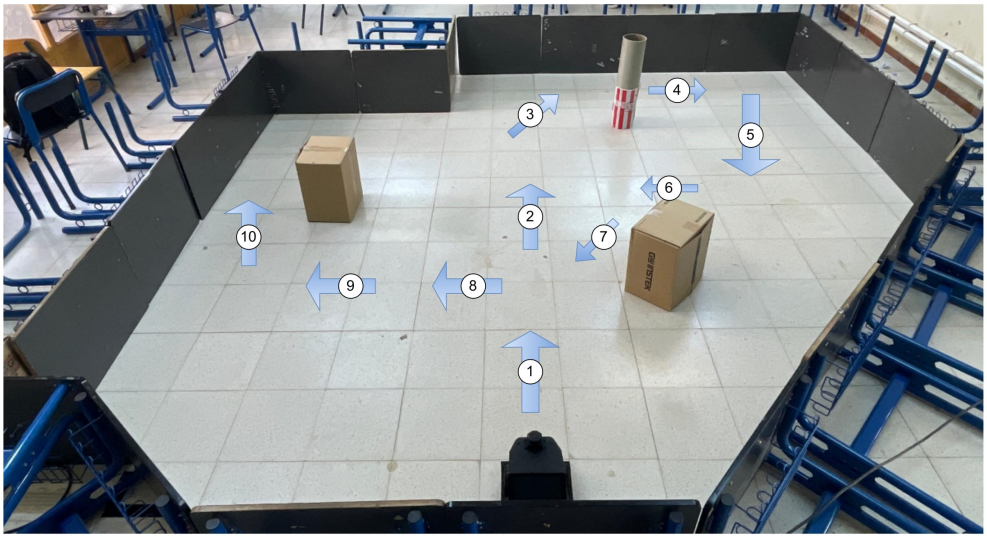


FIG. 4.15 – La trajectoire suivie par notre robot pour cartographier l'environnement

Après avoir lancé les noeuds de Gmapping (section 3.6) et le package "teleop", nous sommes prêts à prendre le contrôle de notre robot pour cartographier notre environnement. Le robot a été déplacé dans notre environnement à partir de la station sol suivant les figures 4.15,4.16 afin de créer une carte détaillée de son environnement.

La cartographie de notre environnement est établit avec le package Gmapping. Ce dernier se base sur l'algorithme Grid-based SLAM déja détaillé dans la section 1.7.3. Le package Gmapping nécessite des données relatifs au déplacement du robot et les transformations entre ses différents repères. Pour ceci, nous avons développé des noeuds et des fichiers de lancement comme déja présenté dans la section 3.6. Les différents paramètres de ce package sont aussi présenté dans la section 3.6. Dans ce qui suit, nous allons exposés les résultats de la cartographie de l'environnement ainsi que le choix optimal de ces paramètres pour une cartographie fiable et fidèle à l'environnement réel.

Après avoir lancé le GMapping, nous avons rencontré plusieurs problèmes, notamment en ce qui concerne la précision de la cartographie. Ce problème est lié aux paramètres mentionnés dans la section 3.6.3. La Figure 4.17 illustre diverses cartographies de notre environnement, mal construites, pour différentes valeurs des paramètres Sigma et de KernelSize.

La Figure 4.18 présente une représentation visuelle de la carte environnementale, considérée comme étant de qualité supérieure. Cette qualité est à cause d'ajustemet des paramètres clés Sigma (σ) et KernelSize. Lorsque ces paramètres sont réglés à 2 pour Sigma (σ) et à 0.1 pour KernelSize, la précision de la cartographie s'améliore de manière significative et la Figure 4.18 démontre une carte environnementale bien définie et fidèle à la réalité. Cette carte obtenue et validée est enregistrée avec le package 'map-server' comme présentée précédemment lors de la cartographie d'un environnement virtuel sur Gazebo.

Après avoir établit une cartographie fiable de l'environnement, nous passons à la validation de la localisation hybride du robot au sein de cette carte. Cette stratégie combine les données odométriques et les données du LiDAR afin de déterminer avec précision la localisation du robot. Dans la section qui suit, nous allons présenter les résultats concernant l'implémentation sous ROS de la localisation hybride du robot.

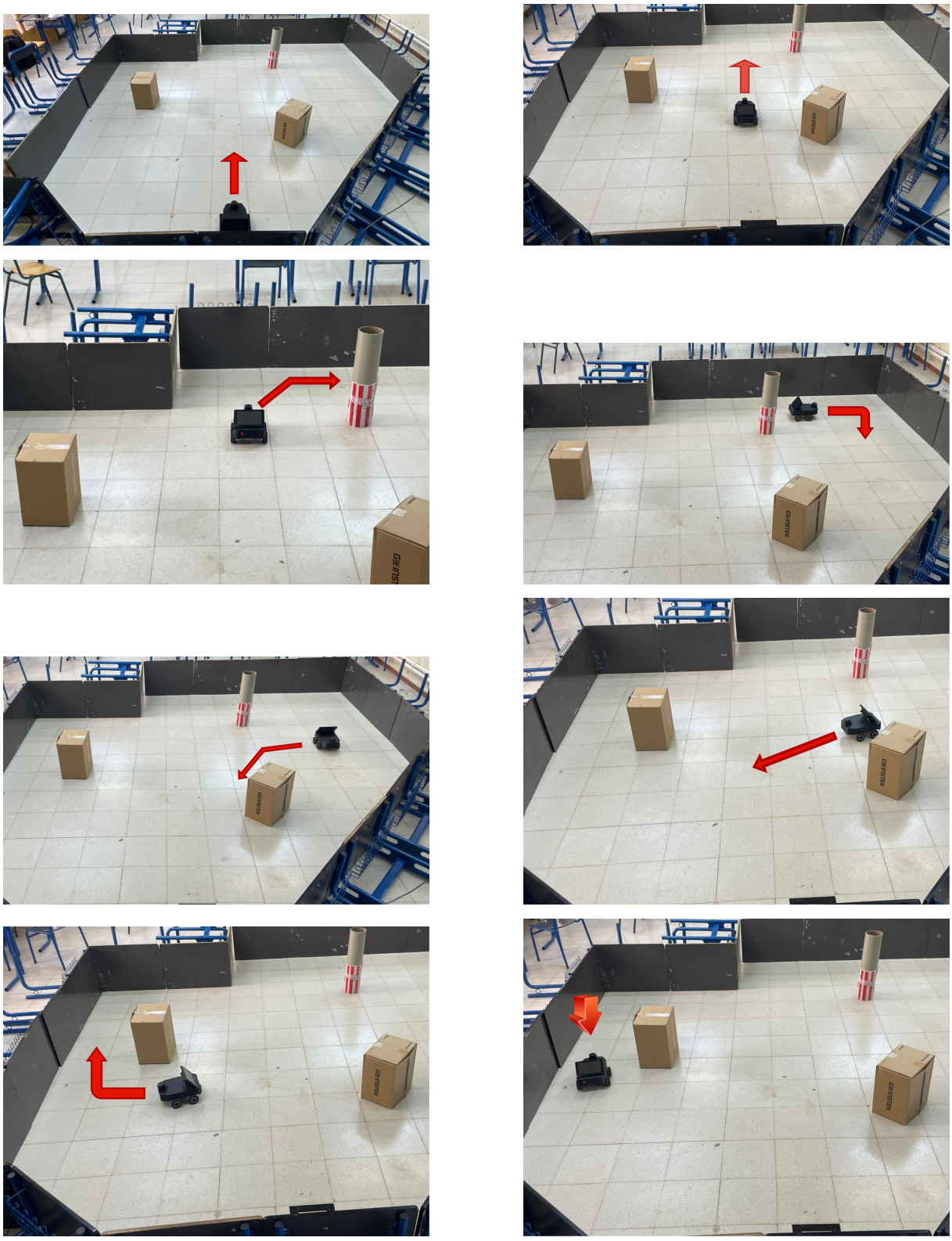
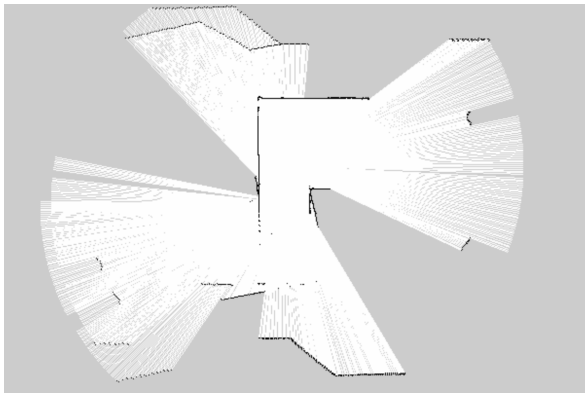


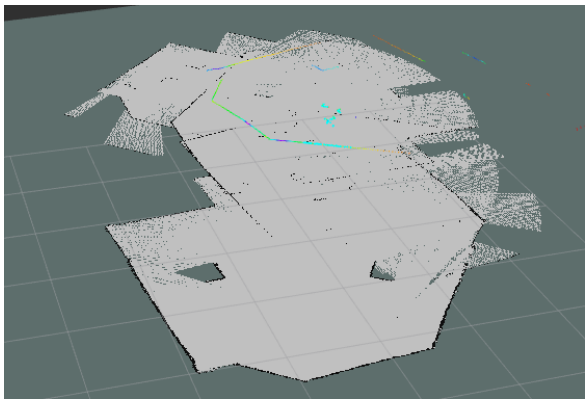
FIG. 4.16 – Photos successives du robot en train de cartographier

4.4.3 Validation de la localisation du robot

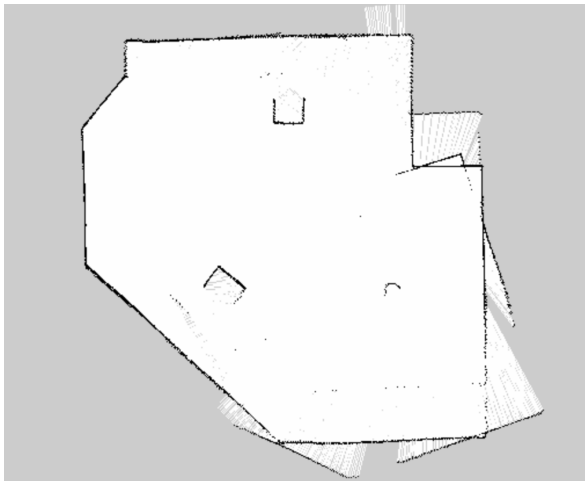
Avant d'entamer la phase de navigation, nous effectuerons d'abord une validation de la localisation hybride du robot. Cette approche repose sur l'utilisation des données de perception afin de corriger les déviations du système odométrique. Cette méthode de



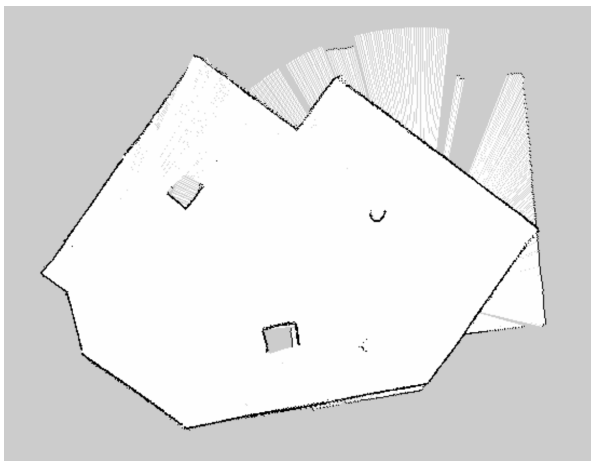
(a) Sigma $\sigma = 0.1$, KernelSize = 1



(b) Sigma $\sigma = 1$, KernelSize = 1



(c) Sigma $\sigma = 1$, KernelSize = 0.5



(d) Sigma $\sigma = 1.5$, KernelSize = 0.5

FIG. 4.17 – Différentes cartes pour des différentes valeurs de paramètres

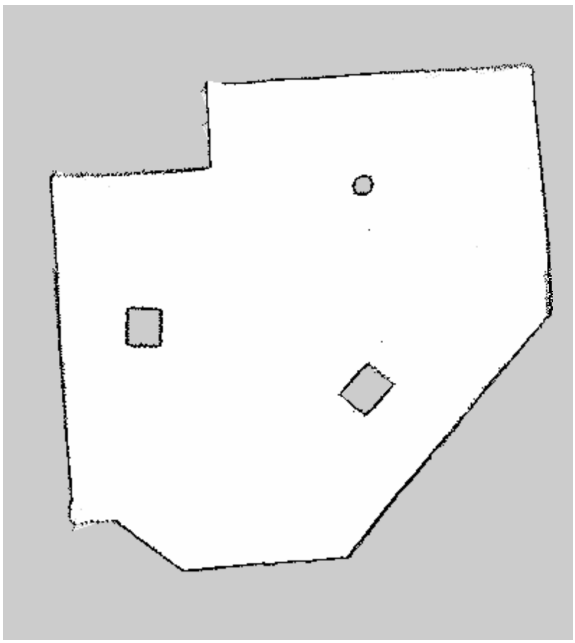


FIG. 4.18 – La cartographie de notre environnement pour Sigma (σ) = 2 et KernelSize = 0.1

localisation hybride est effectuée par l'algorithme de l'AMCL qui se base sur un filtre particulaire. L'implémentation de cet algorithme sous ROS est expliquée dans la section 3.7.

Afin de vérifier la précision du système de localisation hybride, nous procéderons à des commandes manuelles du robot depuis le clavier de la station sol. Nous vérifierons ensuite si la position affichée sur RViz correspond effectivement à la position réelle du robot. Pour garantir une correspondance précise, les données de RViz doivent être projetées ou alignées avec les obstacles de la carte. Tout au long de l'opération, les scans du LiDAR sont restés alignés avec les obstacles de la carte, ce qui confirme la précision de la localisation. La capture d'écran présentée ci-dessous dans la Figure 4.19 illustre les résultats obtenus, où les flèches rouges indiquent les hypothèses de pose du robot.

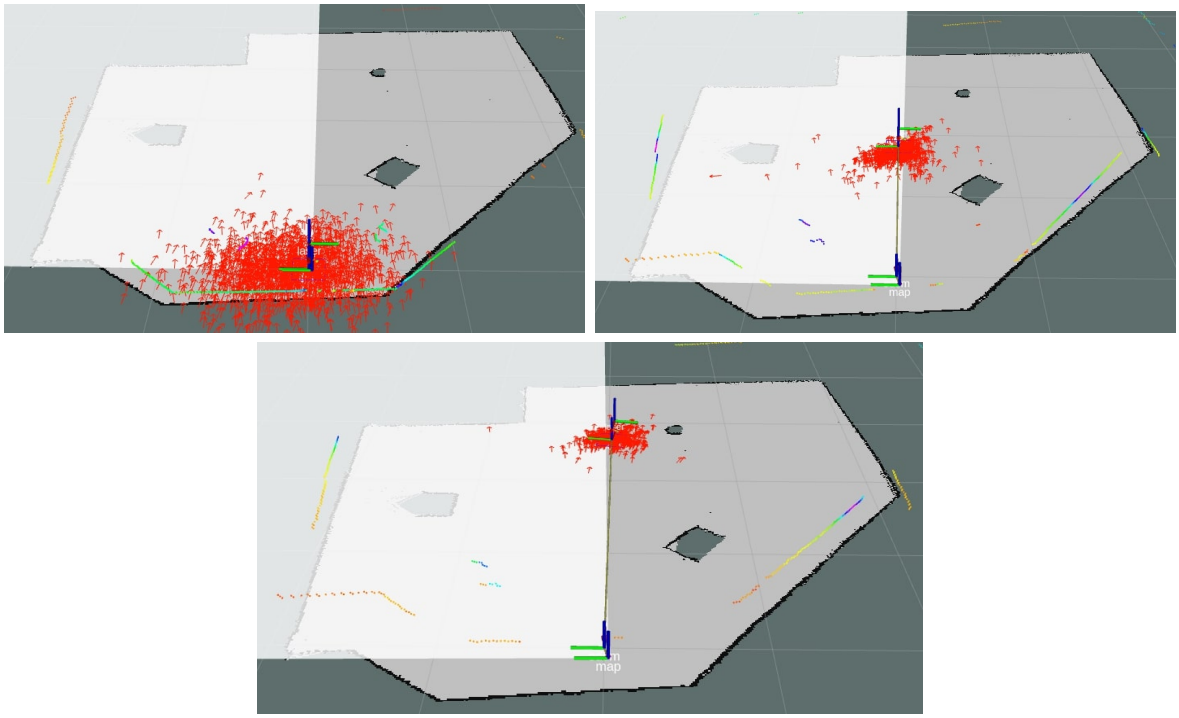


Fig. 4.19 – Validation de la localisation

4.5 Navigation autonome du robot mobile

Après avoir développé le framework sous ROS, établit les communications, commandé les moteurs au niveau bas, cartographié l'environnement, réalisé la localisation hybride du robot sur la carte construite et implémenté les planificateurs local et global, notre robot est maintenant prêt pour la navigation autonome. Dans cette section, nous examinons notre robot afin de vérifier sa capacité à naviguer et atteindre une localisation spécifique dans son environnement.

Avant de tester la navigation autonome sur notre robot réel, nous allons d'abord la tester sur le robot virtuel pour lequel nous avons déjà réalisé la cartographie, comme indiqué dans la section 4.4.1.

4.5.1 Navigation autonome du robot sur Gazebo

Après avoir enregistré la carte avec le "*map_server*", nous pouvons lancer la navigation du robot avec la commande : *roslaunch husky_navigation map_based_navigation.launch*. Nous pouvons la visualisée dans la Figure 4.20.

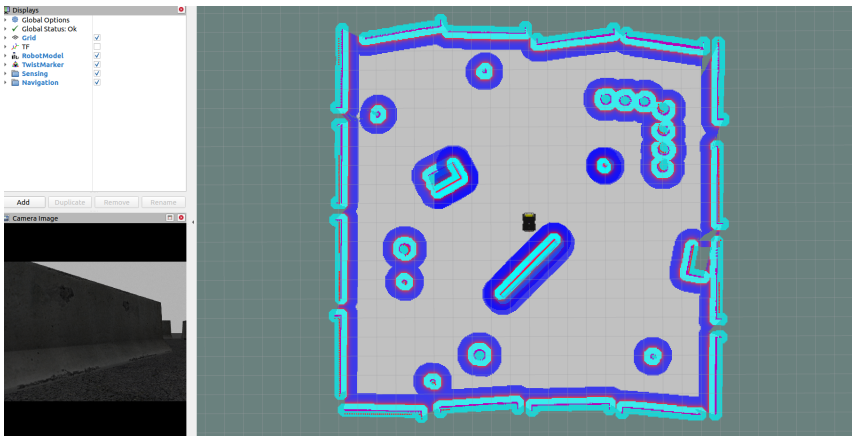


FIG. 4.20 – Visualisation de la carte de navigation

Une fois que la carte est affichée dans RViz, nous plaçons un point de destination en utilisant le bouton '2D Nav Goal' sur la fenetre RViz, comme montré à la Figure 4.21. Cela permet de publier la pose (x, y, Θ) de la position but dans le topic *move_base*.

FIG. 4.21 – Publication de la pose désirée sur RViz

Après avoir défini la destination souhaitée, la Figure 4.22 montre le robot en train de se diriger vers le but sélectionné. En parallèle, la Figure 4.23 montre la pose du robot dans l'environnement virtuel sous Gazebo.

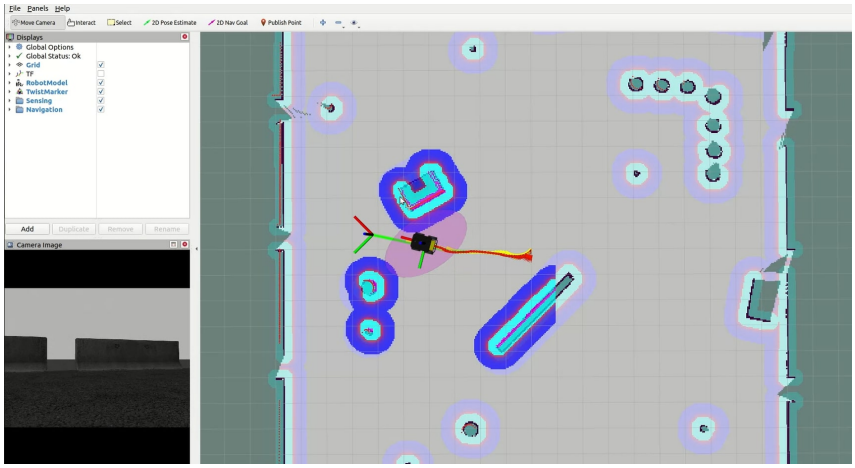


FIG. 4.22 – Le robot se dirige vers la destination souhaitée

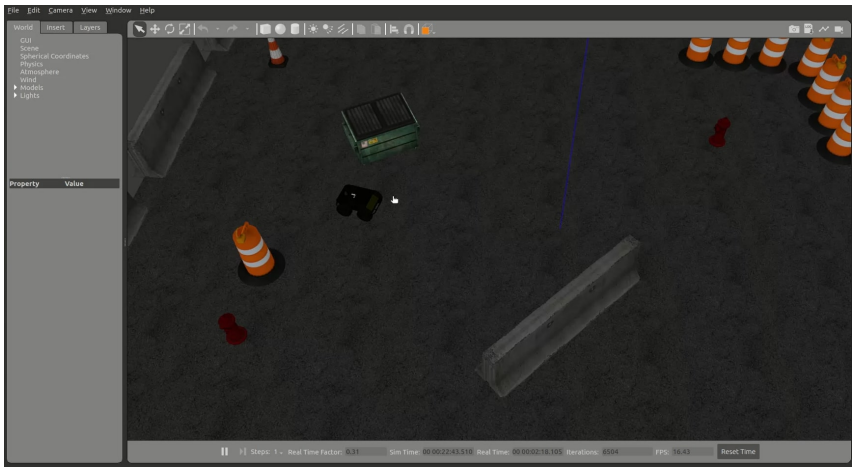


FIG. 4.23 – La pose du robot sous Gazebo

4.5.2 Navigation autonome du robot réel

Afin de reproduire cette tache sur le robot réel et de le déplacer vers une localisation désirée dans l'environnement, nous avons développé un fichier *.launch* (voir la figure 3.57) sur le Raspberry Pi qui lance tous les noeuds qui concernent le contrôle bas niveau du robot, la communication entre les différentes parties du framework développé, la planification et la localisation. De plus, sur la station sol, un autre fichier *.launch* est développé afin de lancer RViz pour visualiser le processus de navigation, comme déjà discuté dans la section 3.8.1. La carte utilisée pour la navigation est illustrée dans la Figure 4.24.

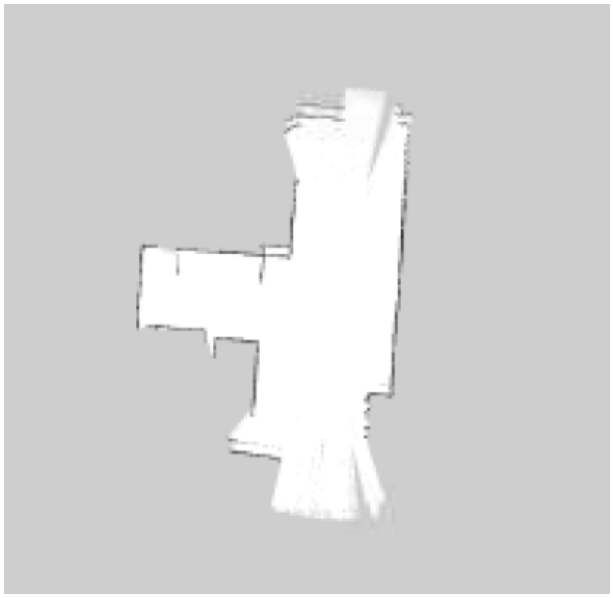


FIG. 4.24 – La carte utilisée pour la navigation

Pour lancer la navigation aller au but, nous plaçons le point de destination en utilisant le même bouton sur la fenêtre RViz, comme montré sur la Figure 4.25. La Figure 4.26 indique que le robot a élaboré un nouveau chemin pour atteindre sa destination. Ce dernier est établit par les planificateurs local et global implémentés.

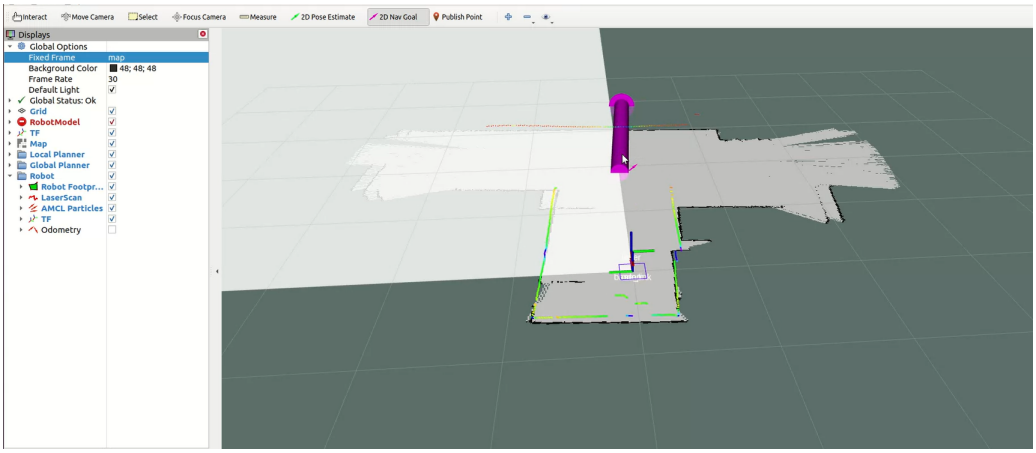


FIG. 4.25 – La destination souhaitée

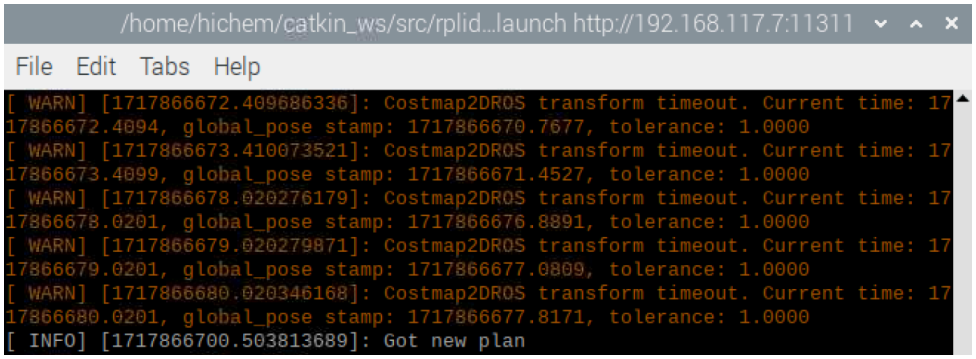


FIG. 4.26 – Le plan de la destination souhaitée

Après que le robot ait suivi le chemin pour atteindre sa destination, les résultats peuvent être observés dans la Figure 4.27, où l'on constate que le robot a effectivement atteint sa destination souhaitée. La Figure 4.28 confirme que l'objectif a été atteint. Tandis que la Figure 4.29 illustre cette navigation autonome du robot dans le monde réel.

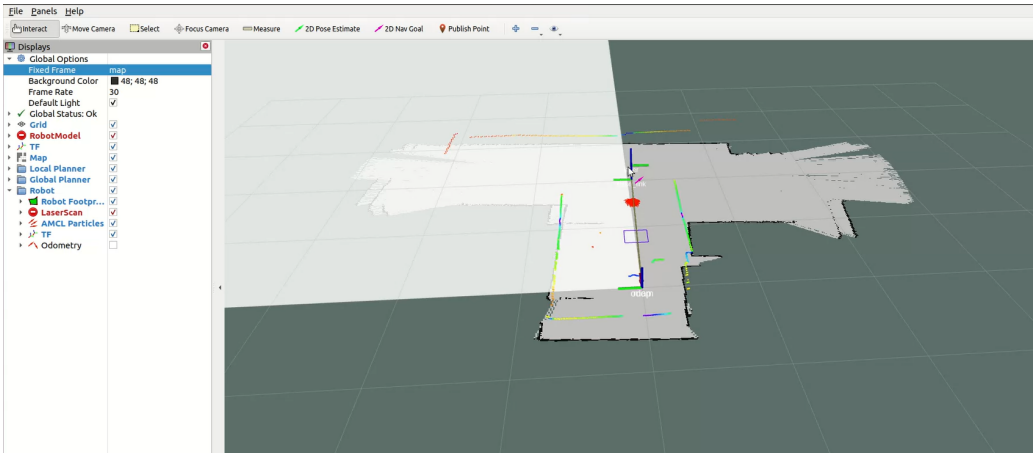


FIG. 4.27 – La navigation vers la destination est réussie

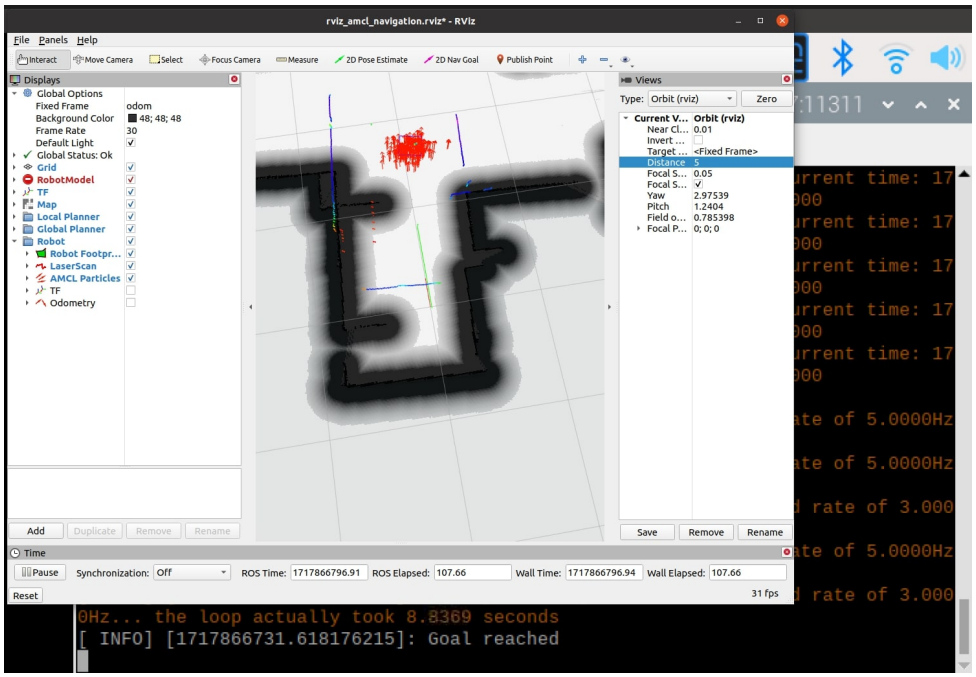


FIG. 4.28 – Atteindre le but

FIG. 4.29 – Photos successives de la navigation autonome du robot

4.6 Conclusion

Ce chapitre présente et analyse les résultats obtenus au cours des différentes phases de notre projet. Nous avons commencé d'abord par la simulation du robot dans un environnement virtuel sous Gazebo. Par la suite, les tests réalisés sur le système odométrique

ont confirmé la précision de la localisation relative du robot. Bien que des erreurs soient présentes, elles demeurent dans des limites acceptables pour les déplacements rectilignes et circulaires. L'implémentation du SLAM a été évaluée et ajustée en fonction de paramètres critiques tels que σ et KernelSize. Les cartes générées montrent la capacité efficace de notre système à cartographier l'environnement, malgré les contraintes liées à la portée du LiDAR. La validation de la localisation basée sur l'AMCL a démontré la précision du robot à se situer avec précision dans l'environnement cartographié.

Enfin, le scénario de navigation autonome a mis en lumière l'efficacité de notre système pour accomplir des tâches telles que la navigation vers un but. Ces résultats soulignent l'importance d'un ajustement précis des paramètres et d'une intégration cohérente des différentes composantes du système de navigation autonome. Les performances globales obtenues sont prometteuses et ouvrent la voie à des améliorations futures, notamment en ce qui concerne le réglage des paramètres des planificateurs local et global pour résoudre les défis d'évitement d'obstacles, qu'ils soient statiques ou dynamiques.

CHAPITRE

5

CONCLUSION GÉNÉRALE

La navigation autonome des robots mobiles est un domaine en constante évolution dans la robotique moderne. Ce projet s'attache à concevoir et mettre en oeuvre un système de navigation pour un robot mobile à quatres roues en utilisant ROS et la technologie du LiDAR. Notre objectif principal était de développer une architecture solide, tant sur le plan matériel que logiciel, permettant au robot d'effectuer des tâches autonomes comme la cartographie, la localisation, la planification de trajectoires et l'évitement d'obstacles.

Nous avons commencé par un état de l'art comprenant les bases théoriques nécessaires pour la réalisation de notre projet, explorant les généralités de la robotique mobile, y compris les applications, les types de robots mobiles, et plus spécifiquement, les robots à quatre roues. Ensuite, nous nous sommes approfondis dans les aspects de la navigation autonome, incluant la cartographie notamment l'algorithme SLAM, la localisation hybride par l'algorithme de l'AMCL, la planification de trajectoires locale et globale par les algorithmes DWA et Dijkstra, ainsi que la navigation autonome elle-même.

Pour la réalisation de notre robot mobile, nous avons débuté par la conception et la réalisation de la structure mécanique du robot ensuite nous nous sommes passés à la réalisation de l'architecture électronique que nous avons décomposé en deux segments bas-niveau et haut-niveau. Le premier se base sur un calculateur Arduino qui a pour rôle d'assurer la commande bas-niveau du robot notamment l'asservissement en vitesses des moteurs. Par contre, le segment haut-niveau se basant sur le Raspberry et le capteur Lidar assure les taches telles que la cartographie, la planification des chemins et la navigation autonome du robot. Afin d'assurer l'échange de données entre les deux segments, nous avons détaillé le développement du framework basé sur ROS, en établissant une communication efficace entre les plates-formes de communication notamment le Raspberry Pi, l'Arduino et la station de base.

Nous avons ensuite procédé à l'implémentation pratique des stratégies de navigation autonome. Cela inclut la mise en oeuvre de la commande bas-niveau pour l'asservissement en vitesse des moteurs et la lecture des encodeurs de déplacement via un contrôleur PID sur Arduino. La commande haut-niveau, gérée par le Raspberry Pi sous ROS, a super-

visé l'implémentation des algorithmes de navigation autonome, y compris l'intégration de SLAM pour la cartographie, la localisation via AMCL, et la planification de trajectoires. Pour évaluer l'efficacité de notre système de navigation autonome, nous avons présenté et analysé les résultats obtenus en simulant d'abord un robot virtuel sur Gazebo, en effectuant sa navigation et en visualisant les résultats sous RViz. Nous avons ensuite évalué l'odométrie du robot ainsi que les algorithmes de cartographie SLAM, de localisation hybride et de planification de chemins mis en oeuvre en terminant par la réalisation d'une tache de navigation autonome du robot notamment l'allez au but.

En synthèse, ce projet nous a permis de développer un système de navigation autonome performant pour un robot mobile à quatre roues. Les résultats obtenus démontrent que le robot est capable de naviguer efficacement vers des objectifs spécifiques et de construire des cartes précises de son environnement. Cependant, certaines limitations persistent, notamment en termes d'optimisation des algorithmes pour des environnements complexes et dynamiques, ainsi que pour l'évitement d'obstacles statiques et dynamiques. Pour surmonter ces défis, les perspectives d'amélioration incluent l'intégration de techniques de machine learning pour améliorer la perception et la prise de décision, ainsi que l'optimisation des algorithmes de planification de trajectoires pour des performances améliorées en temps réel.

En conclusion, ce projet représente une contribution significative à la recherche en navigation autonome pour les robots mobiles, ouvrant la voie à des applications pratiques dans divers domaines tels que l'industrie, la défense, les missions de sauvetage, le transport d'objets, la détection de mines et l'exploration de terrains inconnus.

BIBLIOGRAPHIE

[1] Arduino mega 2560 r3 compatible | dzduino. https://www.dzduino.com/arduino.

[2] Batterie lipo 11.1v 3s | dzduino. https://www.dzduino.com/batterie-lipo.

[3] Faulhaber - véhicules robotisés. https://www.faulhaber.com.

[4] Gazebo Simulation Software. https://gazebosim.org/home.

[5] L298n | dzduino. https://www.dzduino.com/l298n.

[6] Moteur jga25-370 915rpm avec encodeur | dzduino. https://www.dzduino.com/arduino.

[7] Raspberry pi 3 model b+ | raspberry pi. https://www.raspberrypi.com/products/raspberry-pi-3-model-b-plus/. Consulté le 25 avril 2024.

[8] Raspberry pi 3 modèle b+. https://www.ldlc.com.

[9] Raspberry pi touch display. https://www.raspberrypi.com/products/raspberry-pi-touch-display/. 2024.

[10] Robots mobiles autonomes (amr) | kuka ag. https://www.kuka.com/robots-mobiles.

[11] La robotique mobile prend son indépendance dans les usines. https://www.usinenouvelle.com/article, 2021.

[12] BELKACEM ANES. Commande à distance d'un robot mobile via un smartphone android.

[13] A. Bazoula. Cours de navigation des véhicules autonomes emp, 2020.

[14] Janusz Będkowski. Introductory chapter : Autonomous mobile mapping robots–current state and future real-world challenges. *Autonomous Mobile Mapping Robots*, 2023.

[15] Bernard Benet. Perception de l'environnement naturel pour des applications d'agriculture de précision. In *Journées nationales sur ROS 2018*, pages 89–p, 2018.

[16] Abdeslam BENMAKHLOUF. *Navigation Intelligente Autonome Pour Robot Mobile*. PhD thesis, Université Kasdi Merbah Ouargla.

[17] Rodney Brooks. A robust layered control system for a mobile robot. *IEEE journal on robotics and automation*, 2(1) :14–23, 1986.

[18] Russell Buchanan, Tirthankar Bandyopadhyay, Marko Bjelonic, Lorenz Wellhausen, Marco Hutter, and Navinda Kottege. Walking posture adaptation for legged robot navigation in confined spaces. *IEEE Robotics and Automation Letters*, 4(2) :2148–2155, 2019.

[19] Guy Caverot*. Vers de nouveaux usages des robots mobiles. *Réalités industrielles*, (1) :42–48, 2012.

[20] Ford Media Center. Robot humanoÏde "digit", 2019.

[21] CHLOE'S. Chloe's robot électronique spiderbot. https://www.amazon.fr/CHLOES, 2017.

[22] Abdelhakim Debib. modelisation d4un robot mobile a quatre roues, 2016.

[23] Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1) :269–271, 1959.

[24] David Filliat. Tutoriel sur la robotique mobile, insérez l'année.

[25] Gabriel Fouché. Le scanner mobile ZEB-HORIZON : nouvel outil pour la réalisation des PCRS. Analyse du couplage de méthodes de levés pour identifier et déterminer les différentes erreurs du LIDAR. Master's thesis, Cabinet Altuis, 42 rue Marcelin Berthelot, 93701 Drancy, July 2021.

[26] IMED EDDINE GUEBLI. Navigation et commande d'un robot mobile à 4 roues sous ros. 2020.

[27] Boujelkha Ichrak. Conception et réalisation d'un robot mobile fennec camouflé basé sur l'iot pour des applications militaires.

[28] ImpactLab. New robot may have eliminated need for herbicides, 2015. 2024.

[29] Anis Koubâa et al. *Robot Operating System (ROS).*, volume 1. Springer, 2017.

[30] Tae-jae Lee, Chul-hong Kim, and Dong-il Dan Cho. A monocular vision sensor-based efficient slam method for indoor service robots. *IEEE Transactions on Industrial Electronics*, 66(1) :318–328, 2018.

[31] MAAROUF Mahmoud and LAIB Amar Salem. Navigation autonome basée sur ros pour un robot mobile à chenilles. Mémoire de Fin d'Études, Mai 2024.

[32] Pierre-Yves Oudeyer. Robotique : les grands défis à venir. *Futuribles*, 339:5–22, 2014.

[33] Generation Robots. Ros (robot operating system), 2019). 2024.

[34] RobotsGuide. Raven surgical robot, Année d'invention 2012.

[35] ROS (Robot Operating System). Why ROS ? https ://www.ros.org/blog/why-ros/, 2024.

[36] Francisco Rubio, Francisco Valero, and Carlos Llopis-Albert. A review of mobile robots : Concepts, methods, theoretical framework, and applications. *International Journal of Advanced Robotic Systems*, 16(2) :1729881419839596, 2019.

[37] SLAMTEC. Slamtec rplidar datasheet c1. https://www.slamtec.ai/product/slamtec-rplidar-c1/, 2023.

[38] Noureddine Slimane. *Système de localisation pour robots mobiles.* PhD thesis, Université de Batna 2, 2008.

[39] Spiceworks. What is robot operating system ?, 2018. 2024.

[40] Ryosuke Tajima and Keisuke Suga. One-legged jumping robot. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2006.

[41] Zakaria TELDJOUNE and Ismail GUENNICHE. *Navigation autonome des robots mobiles*. PhD thesis, Directeur : Mme SEBBAGH Hafidha/Co-Directeur : Mr ABDELLAOUI Ghouti, 2022.

[42] Anne-Laure Thessard. Robotisation du travail et compétition symbolique entre les espèces. *Giornale di Filosofia*, 2(2), 2021.

[43] Sebastian Thrun, Wolfram Burgard, Dieter Fox, et al. Probabilistic robotics, vol. 1, 2005.

[44] Sabrina TOUAMI. Vehicules de livraison autonomes.

[45] Olivier Trullier, Sidney I Wiener, Alain Berthoz, and Jean-Arcady Meyer. Biologically based artificial navigation systems : Review and prospects. *Progress in neurobiology*, 51(5) :483–544, 1997.

[46] Christopher Von Alt. Autonomous underwater vehicles. In *Autonomous Underwater Lagrangian Platforms and Sensors Workshop*, volume 3, page 2, 2003.

[47] PYO Yoonseok, Cho HanCheol, Jung RyuWoon, and Lim TaeHoon. Ros robot programming. *Seoul, Republic of Korea : ROBOTIS Co., Ltd*, 2017.