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 de Master

Filière : Automatique
Spécialité : Automatique

Présenté par : ALLAK Mohamed Adel
BESSEGHIEUR Mohammed Hichem

Thème

Contribution à la navigation autonome d'un robot mobile de type voiture basé sur ROS2.

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

Année universitaire :2023 /2024

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

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

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 tenons également à remercier chaleureusement **M. BENNEKROUF Mohammed**, et **Mme. OUHOUD Amina**, pour leur soutien précieux et leur engagement déterminant tout au long de ce projet.

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

توضح هذه المذكرة مفهوم الروبوتات الحديثة، وهو مجال متعدد التخصصات يستخدم الميكانيك والإلكترونيات وعلوم الكمبيوتر والذكاء الاصطناعي لتطوير أنظمة مستقلة. يدرس التطور التاريخي للروبوتات ويركز على الروبوتات المتنقلة، حيث تستخدم الروبوتات أجهزة الاستشعار للإدراك واتخاذ القرار والتنقل بكفاءة. تمت دراسة نظام تشغيل الروبوت عن طريق الروس ، وهو منصة مفتوحة المصدر، لدوره في تبسيط تطوير الأنظمة الروبوتية المتقدمة، من خلال مقارنة إصدارات الروس 1 و الروس 2 . توضح دراسة حالة حول التنقل المستقل للروبوت هاسكي تحسين أداء الروس 2 للأنظمة الموزعة والأمن.

Abstract

This dissertation explores modern robotics, a multidisciplinary field that uses mechanics, electronics, computer science and artificial intelligence to develop autonomous systems. It examines the historical evolution of robotics and focuses on mobile robotics, where robots use sensors to perceive, decide, and navigate efficiently. The Robot Operating System (ROS), an open source platform, is studied for its role in simplifying the development of advanced robotic systems, by comparing ROS 1 and ROS 2 versions. A case study on the autonomous navigation of the "Husky" robot demonstrates the improved performance of ROS 2 for distributed systems and security.

Résumé

Ce mémoire explore la robotique moderne, un domaine multidisciplinaire qui utilise la mécanique, l'électronique, l'informatique et l'intelligence artificielle pour développer des systèmes autonomes. Il examine l'évolution historique de la robotique et se concentre sur la robotique mobile, où les robots utilisent des capteurs pour percevoir, décider et naviguer efficacement. Le Robot Operating System (ROS), plateforme open source, est étudié pour son rôle dans la simplification du développement de systèmes robotiques avancés, en comparant les versions ROS 1 et ROS 2. Une étude de cas sur la navigation autonome du robot "Husky" démontre les performances améliorées de ROS 2 pour les systèmes distribués et la sécurité.

TABLE DES MATIÈRES

Liste des abréviations vi

Table des figures vii

Liste des tableaux ix

Introduction générale 1

1 Généralités sur la robotique 2
- 1.1 Introduction . . . 2
- 1.2 Généralités sur la robotique . . . 2
  - 1.2.1 Historique de la robotique mobile . . . 3
  - 1.2.2 Définition d'un robot . . . 4
  - 1.2.3 Avantages des robots . . . 4
  - 1.2.4 Inconvénients des robots . . . 5
- 1.3 Généralité sur la robotique mobile . . . 5
  - 1.3.1 Définition . . . 5
  - 1.3.2 Classisfication des robots mobiles . . . 6
- 1.4 Généralités sur la navigation autonome . . . 12
  - 1.4.1 Perception . . . 13
  - 1.4.2 Décision . . . 13
  - 1.4.3 Action . . . 14
- 1.5 Conclusion . . . 14

2 ROS (Robot Operating System) 16
- 2.1 Introduction . . . 16
- 2.2 Historique de ROS . . . 16
- 2.3 Définition de ROS . . . 17
- 2.4 Avantages du ROS . . . 17
- 2.5 Les Principaux Concepts de ROS . . . 18
- 2.6 Outils de visualisation . . . 20

2.6.1 Gazebo 20
2.6.2 RViz 21
2.6.3 rqt 21
2.7 ROS 1 22
2.8 ROS 2 23
2.9 Définition des fondamentaux de la navigation autonome sur ROS 24
2.10 Conclusion 27

3 Simulation du robot Husky sous ROS 28
3.1 Introduction 28
3.2 Robot Husky 28
3.2.1 Définition 28
3.2.2 Équations de robot Husky 30
3.2.3 Caractéristiques du robot terrestre Husky 30
3.2.4 Exemples d'applications du robot terrestre Husky 30
3.3 Navigation autonome du robot Husky 31
3.3.1 Simulation du robot Husky sous ROS 1 31
3.3.2 Simulation du robot Husky sous ROS2 36
3.3.3 Comparaison entre ROS 1 et ROS 2 40
3.4 Conclusion 41

4 Conclusion générale 42

LISTE DES ABRÉVIATIONS

AMCL Adaptive Monte Carlo Localization.
DDS Data Distribution Service.
DWA Dynamic Window Approach.
GPS Global Positioning System.
Hz Point Cloud Library.
IMU Inertial Measurment Unit.
LiDAR Light Detection And Ranging.
OSRF Open Source Robotics Foundation.
ROS Robot Operating System.
RViz Ros Visualization.
SLAM Simultaneous Localization And Mapping.
STAIR Stanford AI Robot.
TLS Transport Layer Security.

TABLE DES FIGURES

1.1 Robot **"Tortue"** de Grey Walter **1950** . . . 3
1.2 Robot "**Beast**" de l'université John Hopkins **1960** . . . 4
1.3 Robot **"Shakey"** de Stanford **1969** . . . 4
1.4 Robot de déminage . . . 7
1.5 Robot de combat . . . 7
1.6 Véhicule autonome [13] . . . 7
1.7 Robot de livraison . . . 7
1.8 Drone de livraison . . . 8
1.9 Robot humanoïde sous-marine . . . 8
1.10 Robot de Cartographie et exploration [8] . . . 8
1.11 Robot d'assistance médicale . . . 9
1.12 Robot chirurgical [18] . . . 9
1.13 Récolte automatisée . . . 9
1.14 Surveillance des cultures [10] . . . 9
1.15 Robot aspirateur . . . 9
1.16 Désinfection des surfaces . . . 9
1.17 robot laveur de vitre . . . 9
1.18 Robot de type unicycle . . . 10
1.19 Robot de type tricycle . . . 10
1.20 Robot Ackerman . . . 11
1.21 Robot de type omnidirectionnel . . . 11
1.22 Illustration des contrôleurs hybrides . . . 13

2.1 Communication noeuds-topics à travers les messages [14] . . . 18
2.2 Exécution des services [14] . . . 19
2.3 Communication entre le noeud ROS Master et les noeuds slaves . . . 19
2.4 Gazebo Logo [2] . . . 20
2.5 RViz Logo . . . 21
2.6 Logo de ROS 1 (Noetic Ninjemys) . . . 22
2.7 Logo du ROS2 (Humble Hawksbill) . . . 23

2.8 interface *teleop_twist_keyboard* 25

3.1 Robot Husky 29
3.2 Robot Husky équipé de caméras, LiDAR, GPS 29
3.3 Les packages nécessaires pour la cartographie 32
3.4 Simulation sous Gazebo 32
3.5 Visualisation sous RViz 33
3.6 Début de la cartographie 33
3.7 La cartographie complète 33
3.8 Visualisation de la carte de navigation 34
3.9 Publication de la pose désirée sur RViz 34
3.10 Le robot se dirige vers la destination souhaitée 35
3.11 La pose du robot sous Gazebo 35
3.12 Installation des packages 36
3.13 Simulation sous Gazebo 36
3.14 Visualisation sous RViz 37
3.15 Début de la cartographie 37
3.16 La cartographie complète 37
3.17 Visualisation de la carte de navigation 38
3.18 Publication de la pose désirée sur RViz 38
3.19 Visualisation de la destination souhaitée 39
3.20 Visualisation de la destination atteinte 39

LISTE DES TABLEAUX

1.1 Les avantages et les inconvénients des différents types de robots à roues [16] 12

3.1 Tableau comparatif entre ROS 1 et ROS 2 [1] 40

INTRODUCTION GÉNÉRALE

La robotique représente aujourd'hui un domaine essentiellement multidisciplinaire, au carrefour de la mécanique, de l'électronique, de l'informatique et de l'intelligence artificielle, visant à concevoir et à mettre en oeuvre des systèmes capables d'effectuer des tâches de manière autonome ou semi-autonome. Ce mémoire explore en profondeur divers aspects de la robotique moderne, en se concentrant sur la robotique mobile.

La robotique mobile désigne le domaine spécialisé de la robotique où les robots sont conçus pour se déplacer de manière autonome dans leur environnement. Ces robots peuvent être équipés de capteurs pour percevoir leur environnement, prendre des décisions en fonction de ces perceptions et exécuter des actions pour atteindre leurs objectifs. Ce sous-domaine de la robotique joue un rôle essentiel dans divers secteurs tels que la logistique, l'exploration spatiale, l'agriculture de précision et les applications domestiques, en permettant aux robots de naviguer efficacement et en toute sécurité dans des environnements variés et souvent changeants.

Dans le premier chapitre, nous plongerons dans les fondements théoriques de la robotique, en retraçant son évolution historique et en définissant ce qu'est un robot. Nous examinerons les avantages significatifs que ces technologies apportent, notamment en termes de productivité accrue, de sécurité renforcée dans des environnements dangereux, ainsi que les inconvénients et les défis éthiques et économiques associés à leur utilisation. Nous nous concentrerons par la suite sur la robotique mobile, un domaine en expansion qui permet aux robots de se déplacer de manière autonome. Nous classerons les robots selon leur degré d'autonomie, leurs domaines d'application et leur type de locomotion. Un accent particulier sera mis sur les principes de la navigation autonome, crucial pour permettre aux robots de percevoir leur environnement, de prendre des décisions et d'agir en conséquence.

Dans le deuxième chapitre, nous étudierons le Robot Operating System (ROS), une plateforme logicielle open source de premier plan utilisée pour simplifier le développement de systèmes robotiques avancés. Nous analyserons ses caractéristiques, ses outils de simulation et de visualisation, ainsi que les différences entre ses versions ROS 1 et ROS 2, en mettant en lumière leurs avantages et leurs applications spécifiques.

Enfin, dans une étude de cas pratique dans le troisième chapitre, nous mettrons en oeuvre des algorithmes de navigation autonome sur un robot mobile nomé "Husky", d'abord sur ROS 1 puis sur ROS 2, pour évaluer et comparer leurs performances et leur fiabilité dans des conditions variées.

CHAPITRE

1

GÉNÉRALITÉS SUR LA ROBOTIQUE

1.1 Introduction

La robotique, un domaine scientifique et technologique en plein essor, englobe l'étude, la conception et l'utilisation de machines capables d'exécuter des tâches de manière autonome ou semi-autonome. Ce premier chapitre, intitulé "Généralités sur la robotique", offre une vue d'ensemble des concepts fondamentaux de ce domaine dynamique.

Nous commencerons par une introduction générale à la robotique, incluant un aperçu historique de la robotique mobile et une définition des robots, en précisant leurs caractéristiques essentielles. Nous discuterons des avantages des robots, tels que l'augmentation de la productivité et la capacité à opérer dans des environnements dangereux, ainsi que des inconvénients, notamment les préoccupations économiques et éthiques.

Ensuite, nous nous pencherons sur les généralités de la robotique mobile, définissant ce sous-domaine et classifiant les robots mobiles selon le degré d'autonomie, les domaines d'application, et le type de locomotion. Enfin, nous explorerons les principes de la navigation autonome, en nous concentrant sur la perception et la prise de décision des robots pour se déplacer de manière autonome et l'action.

1.2 Généralités sur la robotique

La robotique est un domaine en pleine expansion qui révolutionne de nombreux aspects de la vie moderne, des industries manufacturières aux services de santé. Dans ce qui suit, nous commencerons par un aperçu historique de la robotique et définirons ce qu'est un robot, avant d'examiner ses avantages et ses inconvénients.

1.2.1 Historique de la robotique mobile

Le terme "robot" apparaît pour la première fois en **1920** dans une pièce de Karel Capek intitulée "Rossum's Universal Robots". Issu du tchèque "robota" (servitude), il présente les robots comme des serviteurs dociles.

Dans les années **1950**, Grey Walter construit l'un des premiers robots mobiles autonomes, appelé la Tortue (la figure 1.1). Ce robot pouvait se diriger vers une lumière, éviter des obstacles et recharger ses batteries. Bien que fonctionnant dans un environnement préparé, ces fonctions de base restent des sujets de recherche pour les rendre plus robustes.

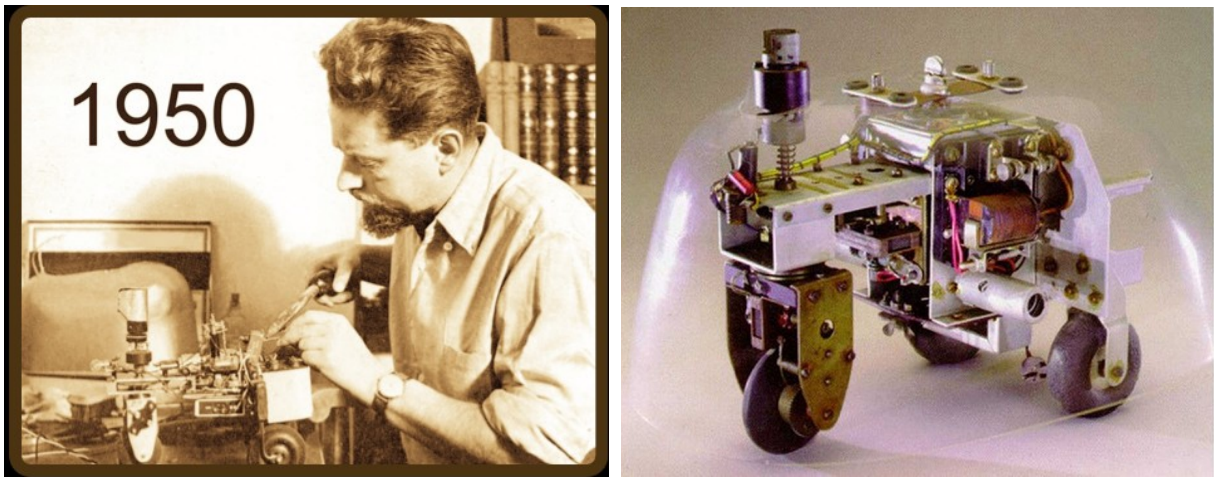

FIG. 1.1 – Robot **"Tortue"** de Grey Walter **1950**

Les années **1960** voient l'arrivée du transistor, permettant des robots plus complexes, tels que "Beast" (la figure 1.2) de l'université John Hopkins, capable de se déplacer en utilisant des capteurs ultrason et des photo-diodes pour trouver des prises électriques et se recharger.

Les premiers liens entre l'intelligence artificielle et la robotique se forment en **1969** avec Shakey (la figure 1.3) à Stanford. Ce robot, utilisant des télémètres à ultrason et une caméra, sert de plateforme de recherche en intelligence artificielle axée sur la planification symbolique. La perception de l'environnement se révèle complexe, influençant fortement les contraintes environnementales. Les développements se poursuivent avec le Stanford Cart à la fin des années **1970**, utilisant la stéréo-vision pour la détection d'obstacles.

Depuis les années **1990**, des plateformes intégrées comme le Pioneer de Mobile Robots ont permis à de nombreux laboratoires de diversifier leurs recherches en robotique mobile. Les défis de déplacement et de modélisation de l'environnement demeurent, mais de nouveaux thèmes de recherche émergent, comme les approches multi-robots, l'apprentissage et les interactions homme-robot [7].

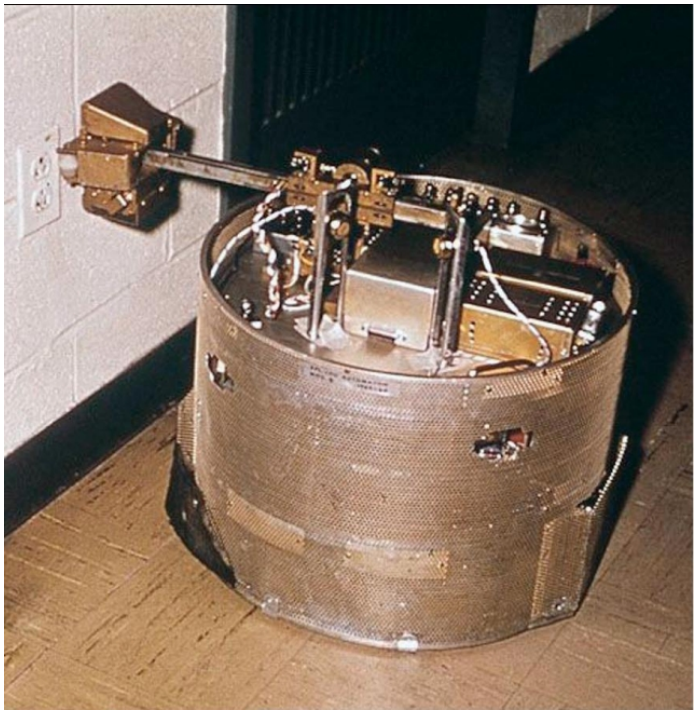

FIG. 1.2 – Robot "**Beast**" de l'université John Hopkins **1960**

FIG. 1.3 – Robot **"Shakey"** de Stanford **1969**

1.2.2 Définition d'un robot

Un robot est un dispositif mécatronique, intégrant mécanique, électronique et informatique, capable de réaliser automatiquement des tâches généralement dangereuses, répétitives ou impossibles pour les humains, ou des tâches simples mais de manière plus efficace qu'un humain. Ces systèmes sont dotés d'un certain degré d'intelligence ainsi que de moyens de perception et de commande permettant de réagir de manière autonome aux changements de circonstances, y compris les situations imprévues, dans leur environnement de travail [12].

Le Petit Larousse définit un robot comme un appareil automatique capable de manipuler des objets ou d'exécuter des opérations selon un programme fixe ou modifiable. En réalité, la perception commune d'un robot est souvent vague et le terme est fréquemment utilisé pour désigner un manipulateur automatique à cycles programmables.

Pour être qualifié de robot, un système doit posséder une certaine flexibilité, caractérisée par les propriétés suivantes :

— **Versatilité :** Un robot doit être capable d'exécuter une variété de tâches ou la même tâche de différentes manières.

— **Auto-adaptativité :** Un robot doit pouvoir s'adapter à un environnement changeant au cours de l'exécution de ses tâches.

1.2.3 Avantages des robots

Un système robotique comprend non seulement des robots, mais aussi divers dispositifs et systèmes complémentaires utilisés pour accomplir les tâches requises. Les avantages des robots mobiles incluent [16] :

— **Productivité et Qualité :** La robotique et l'automatisation peuvent augmenter la productivité, la sécurité, l'efficacité, la qualité et la cohérence des produits dans de nombreuses situations.

— **Sécurité :** Les robots peuvent opérer dans des environnements dangereux sans nécessiter de mesures de survie ou de sécurité pour les humains.

— **Environnement de travail :** Les robots n'ont pas besoin d'éclairage, de climatisation, de ventilation ou de protection contre le bruit, ce qui simplifie les conditions de travail.

— **Endurance :** Les robots peuvent travailler en continu sans ressentir de fatigue ou d'ennui, et ils n'ont pas besoin d'assurance médicale ni de vacances.

— **Précision :** Les robots offrent une précision répétable en tout temps, sauf en cas de dysfonctionnement ou d'usure.

— **Exactitude :** Les robots peuvent atteindre une précision bien supérieure à celle des humains.

1.2.4 Inconvénients des robots

Les robots présentent certaines limitations, notamment une capacité réduite à réagir en cas d'urgence, sauf si ces situations et leurs réponses ont été préalablement programmées. Les mesures de sécurité sont cruciales pour éviter de blesser les opérateurs ou d'endommager les machines collaborant avec les robots. Les principaux inconvénients des robots mobiles incluent [16] :

— **Réponse inadéquate :** Les robots peuvent réagir de manière inappropriée ou incorrecte face à des situations imprévues.

— **Absence de prise de décision :** Les robots manquent de pouvoirs décisionnels autonomes.

— **Consommation d'énergie :** Les robots nécessitent une quantité significative d'énergie pour fonctionner.

— **Risques de dommages :** Les robots peuvent potentiellement endommager d'autres équipements ou causer des blessures humaines.

— **Coûts élevés :** Les robots sont coûteux en raison des frais initiaux de l'équipement, des coûts d'installation, des besoins en périphériques, de la formation nécessaire et des exigences de programmation.

1.3 Généralité sur la robotique mobile

1.3.1 Définition

Un robot mobile est une machine automatique capable de se déplacer dans un environnement donné, contrairement aux robots fixes, tels que les robots manipulateurs utilisés dans l'industrie. Les robots mobiles peuvent être classés en fonction de leur type de locomotion, c'est-à-dire le milieu dans lequel ils évoluent et leur mode de propulsion [9]. Ces robots peuvent évoluer sur terre, dans les airs, ou sur et sous l'eau. Par exemple, les robots terrestres peuvent se déplacer à l'aide de roues, de chenilles, ou de pattes.

Pour notre étude, nous nous concentrerons sur les robots mobiles à roues, en particulier les robots mobiles à quatre roues. Ces robots sont bien adaptés à des environnements plans, ont une construction mécanique relativement simple, et offrent une bonne contrôlabilité.

1.3.2 Classisfication des robots mobiles

La classification des robots mobiles peut être basée sur divers critères tels que le degré d'autonomie, le système de locomotion, l'énergie utilisée ou encore les domaines d'application. Parmi ces critères, la classification selon le degré d'autonomie est la plus pertinente et la plus couramment utilisée.

Classisfication selon le degré d'autonomie

La classification selon le degré d'autonomie est particulièrement intéressante. Un robot autonome est un système capable de prendre des décisions et de traiter des données pour accomplir diverses tâches avec peu ou pas de supervision humaine, même dans des environnements inconnus. Voici les principales catégories de robots mobiles selon ce critère [11] :

1. **Robot télécommandé**
   Un robot télécommandé est un robot qui est contrôlé directement par un opérateur humain en temps réel. Les commandes sont envoyées via un dispositif de contrôle, comme une télécommande, un ordinateur, ou une autre interface de commande à distance. Ces robots sont souvent utilisés dans des environnements dangereux où il n'est pas sûre pour les humains d'entrer, tels que le déminage, l'exploration sous-marine, ou les opérations de secours. La précision et la rapidité des actions du robot dépendent fortement des compétences et de la réactivité de l'opérateur humain.
2. **Robot semi-autonome**
   Un robot semi-autonome combine des éléments de télécommande et d'autonomie. Ces robots peuvent effectuer certaines tâches de manière autonome en suivant des algorithmes programmés, mais nécessitent une supervision humaine pour des tâches plus complexes ou des situations imprévues. Par exemple, un robot semi-autonome peut naviguer de manière autonome dans un environnement connu en évitant les obstacles, mais pourrait nécessiter une intervention humaine pour des décisions complexes ou des ajustements en temps réel. Ce type de robot est souvent utilisé dans des applications où l'environnement est partiellement structuré où l'autonomie complète n'est pas encore fiable.
3. **Robot autonome**
   Un robot autonome est capable de réaliser des tâches sans intervention humaine directe, grâce à des systèmes avancés de capteurs, de traitement de données et de prise de décision. L'autonomie implique la capacité de capter, percevoir, analyser, communiquer, planifier, prendre des décisions et agir pour atteindre les objectifs définis par un opérateur humain. Ces robots utilisent des algorithmes de planification, d'apprentissage automatique et de traitement en temps réel pour percevoir leur environnement, prendre des décisions et s'adapter aux changements. Les robots autonomes sont utilisés dans des domaines tels que les véhicules autonomes, la logistique et la manutention, l'agriculture de précision, et l'exploration spatiale.

Leur capacité à fonctionner de manière indépendante et à gérer des situations imprévues les rend particulièrement précieux dans des environnements dynamiques et complexes.

Classisfication selon les domaines d'application

1. **Militaire**
   - Intervention dans des situations à haut risque pour désamorcer des bombes (la figure 1.4)
   - Robots de combat (la figure 1.5)
   - Surveillance des frontières

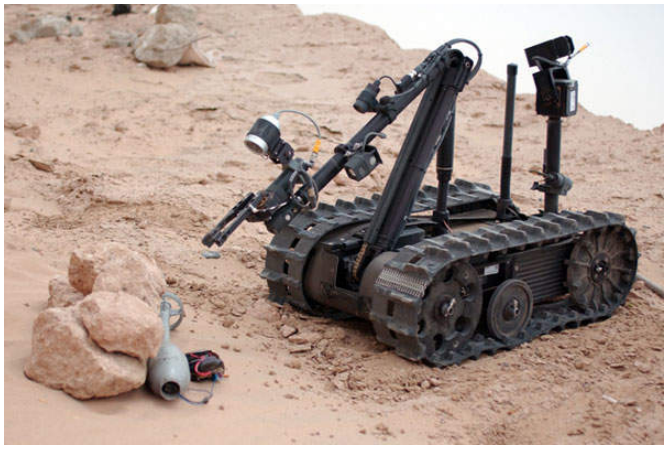

FIG. 1.4 – Robot de déminage

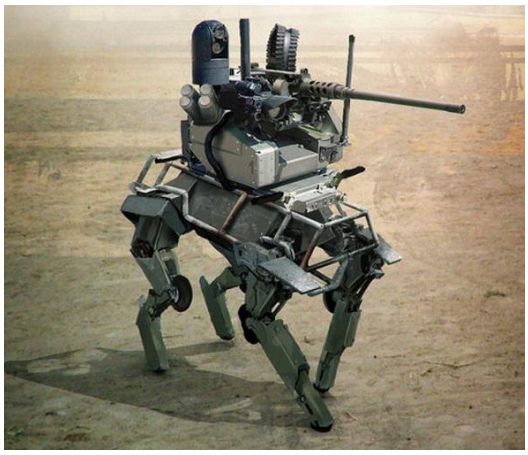

FIG. 1.5 – Robot de combat

2. **Transport et logistique**
   - Véhicule autonome (la figure 1.6)
   - Drones et robots de livraison (les figure 1.7 et 1.8)

FIG. 1.6 – Véhicule autonome [13]

FIG. 1.7 – Robot de livraison

FIG. 1.8 – Drone de livraison

3. **Sous-marine**
   - Pose de câbles, déminage sous-marin, assistance dans les opérations de sauvetage en mer (par un robot humanoïde comme illustré dans la figure 1.9)
   - Cartographie et exploration (la figure 1.10)

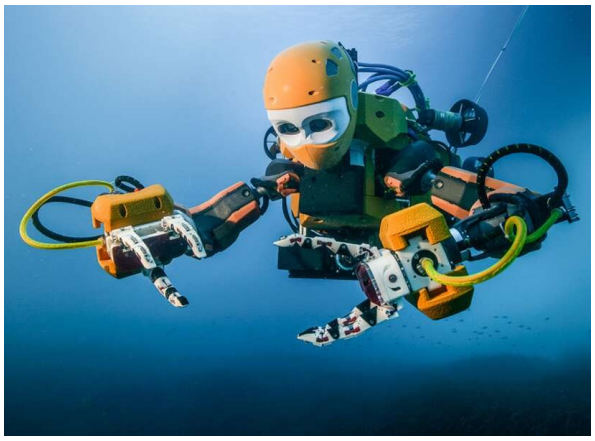

FIG. 1.9 – Robot humanoïde sous-marine

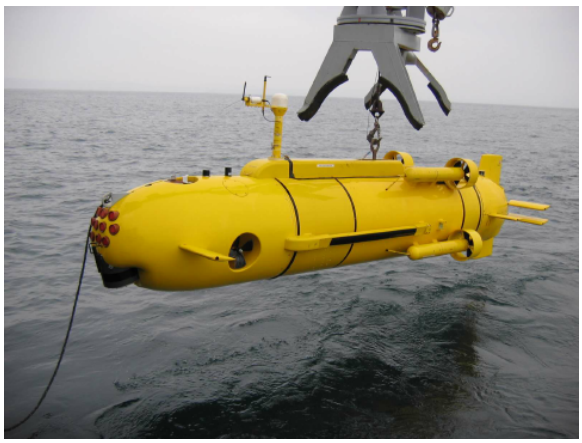

FIG. 1.10 – Robot de Cartographie et exploration [8]

4. **Santé**
   - Assistance médicale et réhabilitation (la figure 1.11)
   - Chirurgie assistée par robot (la figure 1.12)
5. **Agriculture**
   - Récolte automatisée (la figure1.13)
   - Irrigation intelligente et pulvérisation des pesticides
   - Surveillance des cultures (santé des cultures, détecter les maladies, les parasites, les besoins en irrigation et estimation du rendement) comme illustré dans la figure 1.14

FIG. 1.11 – Robot d'assistance médicale

FIG. 1.12 – Robot chirurgical [18]

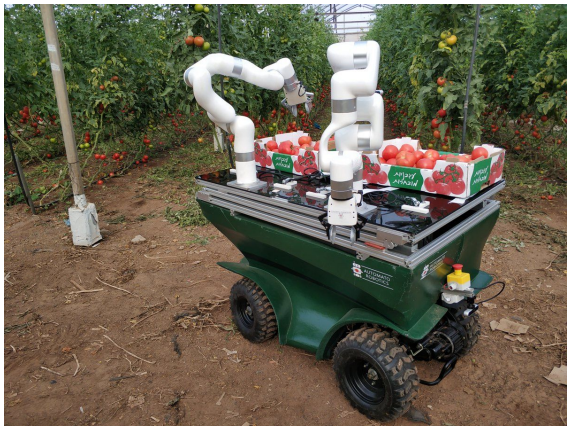

FIG. 1.13 – Récolte automatisée

FIG. 1.14 – Surveillance des cultures [10]

6. **Nettoyage**
   - Aspirateurs robots (la figure 1.15)
   - Désinfection des surfaces (la figure 1.16)
   - Robots laveurs de vitres (la figure 1.17)

FIG. 1.15 – Robot aspirateur

FIG. 1.16 – Désinfection des surfaces

FIG. 1.17 – robot laveur de vitre

Classisfication selon le type de locomotion

Les robots mobiles à roues se classifient principalement selon la position et le nombre de roues utilisées. Les quatre classes principales de robots à roues sont les suivantes [9] :

1. **Robot unicycle**

   Ce type de robot est équipé de deux roues indépendantes et parfois de roues supplémentaires pour la stabilité. Son centre de rotation se trouve sur l'axe reliant les deux roues motrices.

   Il s'agit d'un robot "Non-holonome" [1] , incapable de se déplacer perpendiculairement aux roues de locomotion. La commande de ce robot est relativement simple, permettant des déplacements par une série de rotations et de lignes droites.

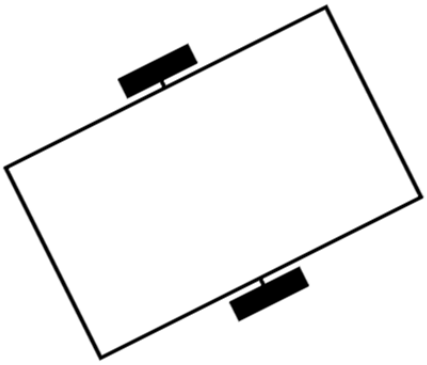

FIG. 1.18 – Robot de type unicycle

2. **Robot tricycle**

   Composé de deux roues fixes sur un même axe et d'une roue orientable centrée sur l'axe longitudinal. Le mouvement est contrôlé par la vitesse des roues fixes et l'orientation de la roue orientable. Le centre de rotation se situe à l'intersection des axes des roues fixes et orientable.

   Également non-holonome, ce type de robot ne peut pas se déplacer perpendiculairement aux roues fixes. Sa commande est plus complexe en raison du rayon de braquage limité de la roue orientable, rendant les rotations simples difficiles.

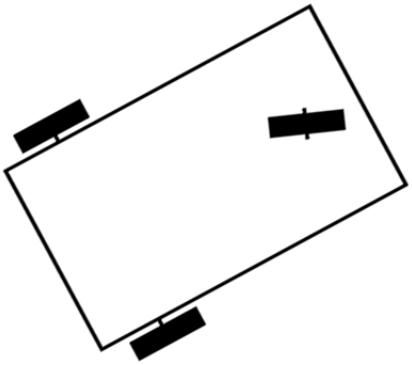

FIG. 1.19 – Robot de type tricycle

[1] Non-holonome signifie que le système comporte une contrainte non intégrale (i.e. le système ne peut pas effectuer certains mouvements).

3. **Robot Ackerman**

   Semblable au tricycle, il comporte deux roues fixes sur un axe et deux roues orientables sur un autre axe, offrant une stabilité accrue grâce à un point d'appui supplémentaire.

   Les propriétés du robot Ackerman sont identiques à celles du tricycle. En remplaçant les deux roues avant par une seule roue centrale, le robot Ackerman peut être simplifié en un robot tricycle sans changer le centre de rotation.

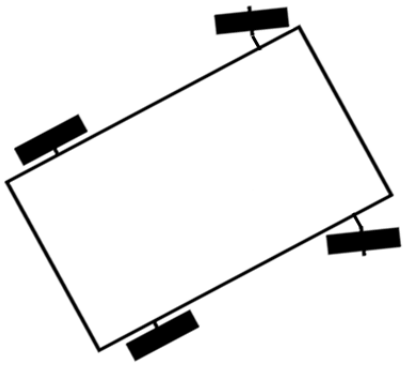

Fig. 1.20 – Robot Ackerman

4. **Robot omnidirectionnel**

   Capable de se déplacer librement dans toutes les directions, ce robot utilise généralement trois roues orientables placées en triangle équilatéral.

   L'avantage majeur de ce robot est sa nature holonome, permettant des déplacements dans toutes les directions. Cependant, cela implique une complexité mécanique plus élevée.

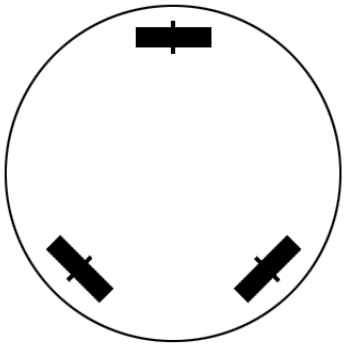

Fig. 1.21 – Robot de type omnidirectionnel

- **Comparaison des différents types**

Le tableau ci-dessous présente un récapitulatif des avantages et des inconvénients des différents types de robots à roues.

Type du robot	Avantages	Inconvénients
Unicycle	- Stable - Rotation sur soi-même - Complexité mécanique faible	- Non-holonome
Tricycle	- Complexité mécanique modérée	- Non-holonome - Peu stable - Pas de rotation sur soi-même
Ackerman	- stable - Complexité mécanique modérée	- Non-holonome - Pas de rotation sur soi-même
Omnidirectionnel	- Holonome - Stable - Rotation sur soi-même	- Complexité mécanique imprtante

TAB. 1.1 – Les avantages et les inconvénients des différents types de robots à roues [16]

1.4 Généralités sur la navigation autonome

La navigation autonome représente une avancée significative dans le domaine de la robotique mobile, permettant aux robots de se déplacer de manière indépendante dans des environnements variés sans intervention humaine directe. À la base de la navigation autonome se trouve la capacité du robot à percevoir son environnement, à prendre des décisions basées sur ces perceptions et à exécuter des actions pour atteindre des objectifs spécifiques. Les robots autonomes utilisent une combinaison de capteurs, d'algorithmes de traitement de données, de planification de trajectoire et de contrôle pour naviguer efficacement et en toute sécurité.

La mise en oeuvre de ces méthodes pour atteindre l'objectif de navigation requiert une architecture de contrôle. L'architecture de contrôle d'un robot désigne l'organisation et la disposition des logiciels qui régissent son comportement et ses actions. Cette structure est conçue pour 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.

Les contrôleurs hybrides, qui combinent les avantages des architectures réactives et planifiées, offrent une solution largement adoptée pour leur réactivité rapide et leur capacité de planification. La Figure 1.22 illustre cette méthode, que nous allons détailler dans ce qui suit.

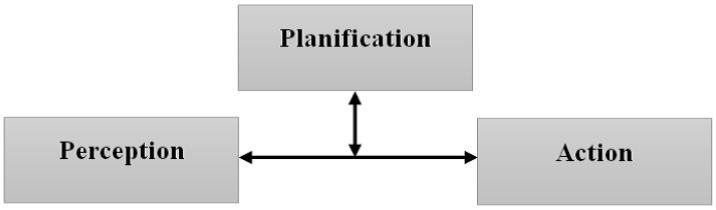

FIG. 1.22 – Illustration des contrôleurs hybrides

1.4.1 Perception

Pour qu'un robot puisse se déplacer librement, la perception de son environnement est essentielle. Les capteurs jouent un rôle crucial en lui fournissant les informations nécessaires pour repérer et réagir aux différentes situations. On distingue deux types de capteurs :

— **Capteurs proprioceptifs :** Mesurent les données internes du robot, comme la vitesse des roues et l'angle d'inclinaison. Ils incluent des accéléromètres, gyroscopes et encodeurs de roues, permettant de contrôler l'état et les mouvements du robot.

— **Capteurs extéroceptifs :** Fournissent des informations sur l'environnement externe du robot. Ils incluent des caméras, LiDARs et sonars, utilisés pour la détection d'obstacles, la localisation et la cartographie.

1.4.2 Décision

Dans cette étape, le robot utilise les données environnementales pour définir les actions à entreprendre, établissant ainsi des séquences d'actions appropriées. Lors de la navigation, cette étape repose principalement sur le SLAM.

SLAM, en français "Localisation et Cartographie Simultanées", est le processus par lequel un robot autonome construit une carte d'un environnement inconnu tout en se localisant simultanément à l'intérieur de cette carte. Ce processus est dynamique et itératif, impliquant deux tâches principales : la localisation et la cartographie.

La localisation permet au robot de déterminer sa position exacte dans l'environnement, en utilisant des capteurs tels que les LIDAR, les caméras, et les IMU (Unités de Mesure Inertielle). Ces capteurs fournissent des données brutes que le robot traite pour estimer sa position relative par rapport à des points de référence dans l'environnement. Cette étape est cruciale pour assurer une navigation précise et éviter les collisions.

La cartographie, quant à elle, consiste à construire et à mettre à jour en temps réel une carte de l'environnement. Le robot utilise les données collectées par ses capteurs pour identifier les caractéristiques clés de l'environnement, telles que les murs, les obstacles et les passages. Ces informations sont intégrées dans une carte qui devient de plus en plus précise à mesure que le robot explore davantage l'environnement.

Au fur et à mesure que le robot se déplace, il met à jour sa carte et sa compréhension de l'environnement, tout en améliorant sa propre localisation. Ce processus itératif permet au robot de naviguer de manière autonome, même dans des environnements inconnus ou changeants. En combinant les informations de localisation et de cartographie, le robot est

capable de planifier des trajectoires optimales, d'éviter les obstacles, et de s'adapter aux changements imprévus dans son environnement.

Cette capacité de décision basée sur le SLAM est essentielle pour plusieurs raisons. Elle permet au robot de prendre des décisions éclairées en temps réel, ce qui est crucial pour la sécurité et l'efficacité de la navigation. Par exemple, si un obstacle imprévu apparaît sur le chemin du robot, celui-ci peut rapidement recalculer sa trajectoire pour éviter une collision. De plus, le robot peut ajuster ses plans en fonction de nouvelles informations, comme un passage initialement fermé qui devient accessible.

1.4.3 Action

L'action effectuée par le robot consiste en faire la planification de chemin, lui permettant de déterminer le meilleur chemin entre son point de départ et sa destination. Cette action se divise en deux méthodes : la planification locale et la planification globale.

— **Planification locale :** La planification locale implique la création d'une trajectoire immédiate en se basant sur les données environnementales disponibles, permettant ainsi au robot d'éviter efficacement les obstacles proches. Les planificateurs de trajectoire locaux sont particulièrement adaptés aux environnements dynamiques car ils utilisent des informations en temps réel pour ajuster la trajectoire du robot. Des méthodes telles que la méthode DWA (Dynamic Window Approach), le champ de potentiel et l'histogramme de champ vectoriel sont flexibles et peuvent être adaptées pour fonctionner avec des cartes locales dynamiques ou sans carte du tout. Leur capacité à traiter les données en temps réel pour l'évitement d'obstacles et la navigation directe les rend applicables dans divers contextes de navigation [3].

— **Planification globale :** La planification globale se concentre sur la création d'une trajectoire complète du point de départ au point d'arrivée avant que le robot ne commence son déplacement. Cette approche nécessite généralement des cartes détaillées et préalablement établies de l'environnement complet. L'algorithme de Dijkstra est largement utilisé pour cette planification de trajectoire globale. Il maintient une liste mise à jour des distances les plus courtes depuis un noeud de départ vers tous les autres noeuds du graphe. Initialement, la distance jusqu'au noeud de départ est fixée à zéro, tandis qu'elle est considérée comme infinie pour tous les autres noeuds. À chaque étape, Dijkstra sélectionne le noeud avec la plus petite distance dans un ensemble appelé "frontière", met à jour les distances de ses voisins si nécessaire, et l'ajoute à l'ensemble des noeuds traités. Ce processus continue jusqu'à ce que la destination soit atteinte ou que tous les noeuds accessibles aient été traités [6].

1.5 Conclusion

Ce premier chapitre a fourni une vue d'ensemble essentielle de la robotique, en établissant les bases pour une compréhension approfondie des concepts et des technologies clés de ce domaine. Nous avons débuté par une présentation des généralités sur la robotique, avec un bref historique de la robotique mobile pour situer le contexte de son évolution. La définition d'un robot a permis de clarifier les caractéristiques fondamentales de ces machines, tandis que l'exploration des avantages et des inconvénients a mis en lumière les bénéfices et les défis associés à leur utilisation.

Nous avons ensuite approfondi les généralités sur la robotique mobile, en offrant une définition précise et en classifiant les robots mobiles selon différents critères. Cette section a couvert le degré d'autonomie (télécommandé, autonome et semi-autonome), les domaines d'application (militaire, santé, transport, agriculture, etc.), et les types de locomotion (unicycle, tricycle, etc.), incluant une comparaison des robots en fonction de leurs modes de déplacement.

Enfin, nous avons abordé les principes de la navigation autonome, en examinant les processus de perception, de décision, et d'action. Ces trois éléments sont cruciaux pour permettre aux robots de se déplacer de manière autonome dans des environnements variés.

En résumé, ce chapitre a établi une base solide de connaissances sur la robotique, couvrant à la fois les aspects théoriques et pratiques. Cela prépare le terrain pour les discussions plus techniques et spécialisées des chapitres suivants. Dans le chapitre suivant, nous allons détailler le Robot Operating System (ROS), une plateforme logicielle essentielle pour le développement et le contrôle des robots modernes, en expliquant ses fonctionnalités, son architecture et ses applications pratiques.

CHAPITRE

2

ROS (ROBOT OPERATING SYSTEM)

2.1 Introduction

Le Robot Operating System (ROS) est un ensemble de bibliothèques et d'outils open source conçu pour simplifier le développement de logiciels pour la robotique. Concevoir une architecture logicielle basée sur ROS constitue une approche novatrice et efficace pour développer des systèmes robotiques avancés. Concevoir une architecture logicielle basée sur ROS constitue une approche inovante et efficace pour développer des systèmes robotiques avancés. Dans ce chapitre, nous examinerons l'évolution du ROS, en commençant par sa définition et les fondements qui ont conduit à sa création. Nous explorerons également les outils de visualisation qu'il offre, qui sont essentiels pour la simulation des systèmes robotiques. De plus, nous analyserons les principaux avantages de son utilisation, tels que la communauté active, la réutilisation du code, et la facilité de collaboration. Enfin, nous discuterons des différentes versions de ROS, en mettant l'accent sur ROS 1 et ROS 2.

2.2 Historique de ROS

En mai 2007, le projet ROS a été lancé en s'inspirant des premiers cadres logiciels robotiques open source, notamment Switchyard, développé par le Dr Morgan Quigley au laboratoire d'intelligence artificielle de Stanford pour soutenir le projet Stanford AI Robot (STAIR).

En novembre 2007, l'entreprise américaine Willow Garage a pris en charge le développement de ROS. Willow Garage est réputée dans le domaine des robots personnels et de service. Elle est également connue pour avoir développé et soutenu la bibliothèque de nuages de points (PCL), largement utilisée pour les dispositifs 3D tels que Kinect, ainsi que la bibliothèque open source de traitement d'images OpenCV.

Willow Garage a commencé à développer ROS en novembre 2007, et le 22 janvier 2010, ROS 1.0 a été officiellement lancé. La première version stable, ROS Box Turtle,

a été publiée le 2 mars 2010. Par la suite, plusieurs versions ont été diffusées par ordre alphabétique, comme C Turtle, Diamondback, etc., à l'instar des versions d'Ubuntu et d'Android.

Depuis lors, ROS a connu de nombreuses évolutions, chaque nouvelle version apportant des améliorations significatives et de nouvelles fonctionnalités. Willow Garage a cessé ses activités en 2014, mais le développement de ROS a été repris par la fondation Open Source Robotics Foundation (OSRF), qui supervise actuellement ROS 1 et ROS 2 [17].

2.3 Définition de ROS

ROS (Robot Operating System) est un cadre open-source conçu pour faciliter le développement de logiciels en robotique, offrant une gamme étendue de services similaires à ceux d'un système d'exploitation. Il fournit l'abstraction matérielle, le contrôle des dispositifs de bas niveau, l'implémentation de fonctionnalités courantes, ainsi que la communication inter-processus et la gestion des packages. En plus de ces services, ROS propose des outils et des bibliothèques pour obtenir, construire, écrire et exécuter du code sur plusieurs ordinateurs, ce qui permet aux développeurs de créer des applications robotiques complexes de manière plus efficace et collaborative. En réunissant ces fonctionnalités sous une seule plateforme, ROS joue un rôle crucial dans la standardisation et l'accélération du développement en robotique.

2.4 Avantages du ROS

ROS est l'un des meilleurs moyens pour construire un robot, non seulement en raison de sa gratuité et de sa nature open source, mais aussi grâce à de nombreuses autres caractéristiques que nous allons énumérer parmi elles :

1. **Réutilisation des Packages :** Les utilisateurs peuvent développer des fonctionnalités personnalisées et les partager sous forme de "packages" pour que d'autres puissent les réutiliser.
2. **Outils de Développement Avancés :** ROS offre des outils de visualisation telle que rqt et RViz. Ces outils simplifient le développement et la simulation de robots, permettant, par exemple, la visualisation de modèles de robots et la réalisation de simulations avec des simulateurs comme Gazebo.
3. **Large Communauté et Support :** ROS bénéficie d'une large communauté d'utilisateurs et de développeurs, ce qui facilite l'apprentissage, le partage de connaissances et le support technique.
4. **Compatibilité avec Divers Matériels :** ROS est compatible avec une variété de plates-formes matérielles et de systèmes d'exploitation, ce qui le rend flexible et adaptable à différents environnements robotiques.
5. **Gestion des matériaux :** ROS offre des bibliothèques et des outils pour la gestion des capteurs, des actionneurs et des données provenant de ceux-ci, cela facilite l'intégration matérielle.

2.5 Les Principaux Concepts de ROS

ROS est construit autour de concepts fondamentaux qui définissent sa philosophie et son architecture. Voici les principaux concepts de ROS :

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. La Figure 2.1 illustre cette communication entre les noeuds et les sujets à travers les messages :

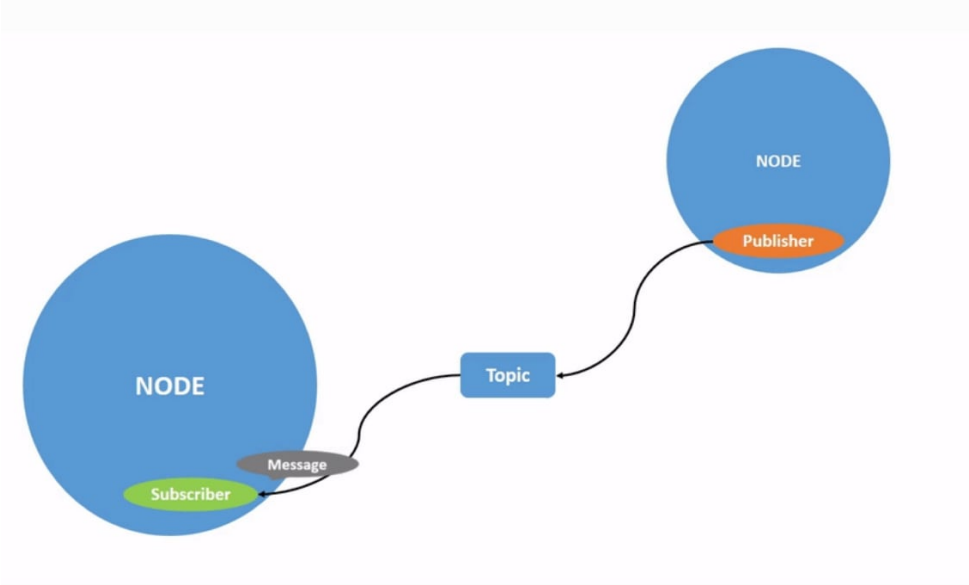

Fig. 2.1 – Communication noeuds-topics à travers les messages [14]

4. **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 2.2 illustre cette communication entre les noeuds pour l'exécution des services :
5. **Actions :** Les actions sont des interactions permettant de gérer des tâches complexes sur une période prolongée, comme la navigation à long terme. Elles facilitent le suivi d'état et la gestion de processus nécessitant une interaction continue.
6. **Packages :** Les packages regroupent des fonctionnalités connexes, tels que des noeuds, des messages et des services, facilitant l'organisation et la réutilisation du code dans ROS.

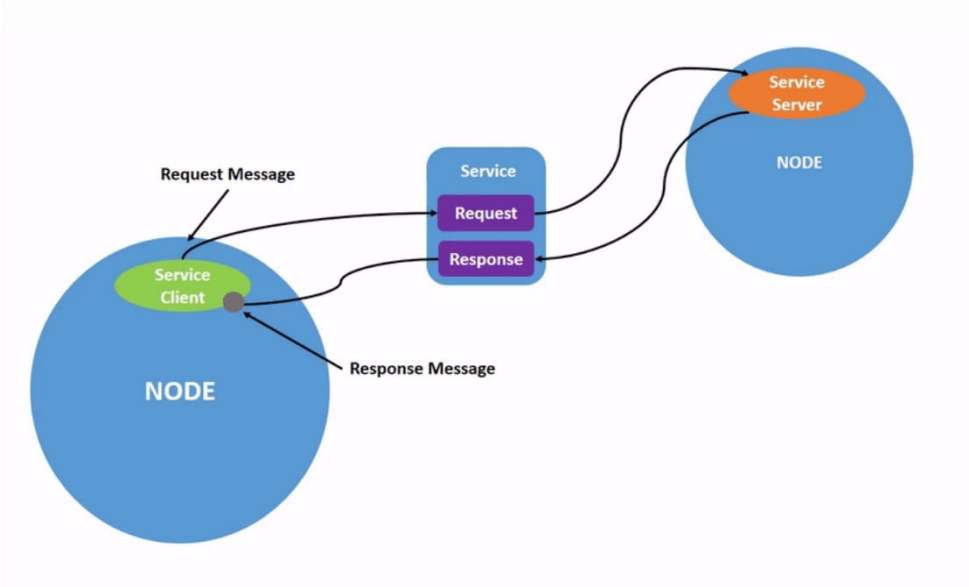

FIG. 2.2 – Exécution des services [14]

7. **ROS 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 et partager des informations. La Figure 2.3 illustre cette communication.

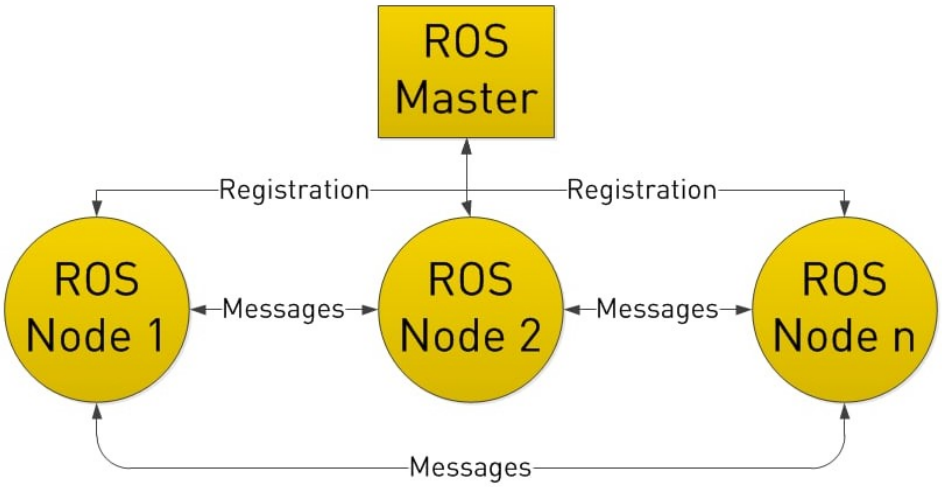

FIG. 2.3 – Communication entre le noeud ROS Master et les noeuds slaves

8. **Fichiers de Lancement (Launch Files) :** Les fichiers de lancement sont des fichiers XML utilisés pour démarrer et configurer plusieurs noeuds ROS simultanément, simplifiant ainsi la mise en place de systèmes ROS complexes.
9. **Espace de Travail Catkin (Catkin Workspace) :** Catkin est le système de construction utilisé pour compiler et gérer les packages ROS. Un espace de travail Catkin contient tous les packages nécessaires à un projet ROS spécifique [15].

2.6 Outils de visualisation

ROS se démarque par sa capacité à fournir une représentation en temps réel des états système et des résultats des algorithmes de traitement, ce qui représente l'une de ses fonctionnalités les plus notables. Pour faciliter l'analyse du comportement des robots et des différents composants du système, ROS propose une gamme d'outils de visualisation, notamment RViz, Gazebo et rqt.

2.6.1 Gazebo

Gazebo est un simulateur 3D largement utilisé dans l'écosystème ROS, il nous a offert la possibilité de modéliser fidèlement les différents aspects de l'environnement , y compris les obstacles, les structures et les conditions spécifiques qui peuvent influencer la navigation autonome d'un robot. Grâce à Gazebo, les développeurs peuvent tester et valider leurs algorithmes dans des environnements virtuels avant de les déployer sur des robots réels, assurant ainsi une préparation efficace et sécurisée des systèmes robotiques. La Figure 2.4 présente le logo de Gazebo.

Fig. 2.4 – Gazebo Logo [2]

- **Capteurs et simulation du bruit :** Gazebo prend en charge la simulation de divers capteurs tels que des télémètres laser, des caméras 2D/3D, des capteurs de type Kinect, des capteurs de contact et des couples de force, incluant la génération de données bruitées pour une simulation plus réaliste.
- **Graphismes 3D avancés :** Gazebo offre un rendu réaliste des environnements, incluant un éclairage sophistiqué, des ombres réalistes et des textures de haute qualité.
- **Simulation de nuages :** Gazebo supporte CloudSim pour fonctionner sur Amazon AWS, ainsi que GzWeb pour interagir avec la simulation via un navigateur web.
- **Simulation de la dynamique :** Gazebo permet l'utilisation de plusieurs moteurs de physique performants.
- **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.

2.6.2 RViz

RViz est l'outil de visualisation 3D de ROS qui se concentre principalement sur la représentation des messages ROS en trois dimensions. Son principal objectif est de permettre une vérification visuelle des données. Par exemple, il peut afficher la distance entre un capteur (LDS) et un obstacle, les données de nuage de points (PCD) provenant de capteurs de distance 3D comme RealSense, Kinect ou LiDAR, ainsi que les images capturées par une caméra, parmi d'autres éléments. La Figure 2.5 présente le logo de RViz.

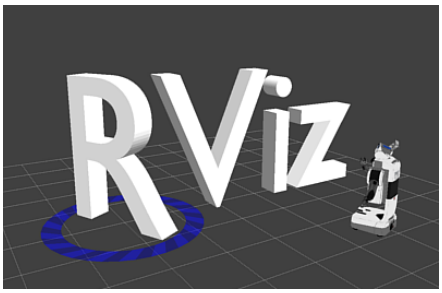

FIG. 2.5 – RViz Logo

RViz offre également une gamme variée de visualisations, y compris des polygones personnalisables et des marqueurs interactifs pour interagir avec les données et les commandes du noeud utilisateur. Il utilise le format URDF (Unified Robot Description Format) pour décrire les robots sous forme de modèles 3D modifiables. RViz peut afficher des modèles de robots mobiles et utiliser les données de distance provenant de capteurs LDS pour la navigation. De plus, RViz prend en charge l'affichage d'images provenant de caméras embarquées sur le robot ainsi que la représentation 3D de données provenant de capteurs tels que Kinect, LDS, et RealSense.

2.6.3 rqt

rqt est un cadre de développement multiplateforme largement utilisé pour la création d'interfaces graphiques dans l'écosystème ROS. Cette plateforme flexible permet aux utilisateurs de développer, personnaliser et intégrer facilement une variété de plugins. Parmi les plugins les plus couramment utilisés, on trouve :

- **rqt-image-view** : Permet d'afficher et de visualiser les images provenant des caméras embarquées sur le robot.
- **rqt-graph** : Offre une représentation graphique des noeuds et des sujets ROS, facilitant ainsi la compréhension et le débogage des systèmes de communication.
- **rqt-plot** : Utilisé pour tracer et surveiller en temps réel les données publiées sur les sujets ROS, ce qui est particulièrement utile pour l'analyse des performances et le diagnostic.
- **rqt-bag** : Permet de lire, écrire et visualiser les fichiers de données ROS enregistrés (rosbags), facilitant ainsi l'analyse des données historiques et le développement itératif.

Grâce à rqt, les développeurs peuvent rapidement créer des interfaces graphiques interactives qui améliorent l'expérience utilisateur et la gestion des systèmes robotiques.

Dans ce qui suit, nous allons explorer en détail les différentes versions de ROS, en les catégorisant selon leurs évolutions et leurs fonctionnalités spécifiques. Cette analyse approfondie nous permettra de mieux comprendre l'évolution et les améliorations apportées à cette plateforme dans le domaine de la robotique, avant que nous aborderons la simulation de la navigation autonome sous ROS 1 et ROS 2.

2.7 ROS 1

ROS 1, acronyme de Robot Operating System, est un ensemble d'outils logiciels open-source destiné à la robotique. Il fournit les services courants que l'on pourrait attendre d'un système d'exploitation, tels que l'abstraction matérielle, le contrôle des périphériques de bas niveau, l'implémentation de fonctionnalités couramment utilisées, la communication entre processus et la gestion des paquets. ROS 1 facilite le développement de logiciels pour la robotique en permettant aux développeurs de créer des applications robustes et modulaires. Parmi les versions notables de ROS 1, nous trouvons ROS 1.0 (Box Turtle) lancé en mars 2010, Indigo Igloo en juillet 2014, Kinetic Kame en mai 2016, Melodic Morenia en mai 2018, et la dernière version Noetic Ninjemys sortie en mai 2020.

Dans notre projet, nous avons utilisé ROS1 (Noetic Ninjemys), offrant des améliorations notables en termes de stabilité et de compatibilité, le logo de cette version est illustré dans la figure 2.6.

Fig. 2.6 – Logo de ROS 1 (Noetic Ninjemys)

ROS 1 Noetic Ninjemys est la treizième version de ROS 1, publiée en mai 2020. Il s'agit de la version finale de ROS 1, conçue pour être compatible avec Ubuntu 20.04 (Focal Fossa) et avec Python 3, reflétant les évolutions des systèmes d'exploitation modernes et des langages de programmation.

ROS1 Noetic Ninjemys se distingue par plusieurs caractéristiques et améliorations :

- **Compatibilité avec Python 3 :** Une des améliorations majeures de Noetic est la transition complète vers Python 3.
- **Stabilité et Longévité :** En tant que version LTS (Long Term Support), Noetic bénéficiera de mises à jour de sécurité et de maintenance jusqu'en 2025.

— **Support des Systèmes Modernes :** Noetic est conçu pour fonctionner sur Ubuntu 20.04 LTS, assurant ainsi une meilleure intégration avec les dernières versions des bibliothèques et des outils système.
— **Améliorations des Performances :** Noetic intègre plusieurs optimisations et corrections de bugs des versions précédentes.
— **Large Écosystème des packages :** Grâce à la vaste communauté de développeurs ROS, Noetic bénéficie d'un large éventail des packages et de bibliothèques compatibles, facilitant l'intégration de nouveaux capteurs, actionneurs et algorithmes.
— **Documentation et Support Communautaire :** La documentation de Noetic est complète et bien maintenue, et la communauté active de ROS continue de fournir un support précieux pour le développement et le déploiement des systèmes robotiques.

En conclusion, ROS1 Noetic Ninjemys constitue une plateforme robuste et flexible pour le développement de projets robotiques modernes, offrant à la fois une compatibilité avec les technologies actuelles et une stabilité à long terme.

2.8 ROS 2

ROS 2, une évolution majeure de ROS 1, est un ensemble d'outils logiciels open-source conçu pour répondre aux besoins croissants de la robotique moderne. Il offre des services similaires à ceux de ROS 1, tels que l'abstraction matérielle, le contrôle des périphériques de bas niveau, la communication entre processus, et la gestion des paquets, mais avec une architecture repensée pour améliorer les performances, la flexibilité et la sécurité. ROS 2 facilite le développement de logiciels robotiques en supportant des systèmes distribués, en temps réel et hautement fiables. Les versions notables de ROS 2 incluent Ardent Apalone (décembre 2017), Crystal Clemmys (décembre 2018), Dashing Diademata (mai 2019), Foxy Fitzroy (juin 2020), et Humble Hawksbill (mai 2022).

Dans notre projet, nous avons utilisé ROS 2 (Humble Hawksbill), offrant des améliorations significatives en termes de flexibilité, performance, et sécurité, répondant ainsi aux exigences des systèmes robotiques modernes. Voici le logo de cette version dans la figure 2.7.

Fig. 2.7 – Logo du ROS2 (Humble Hawksbill)

ROS 2 Humble Hawksbill est une version LTS publiée en mai 2022. Cette version vise à fournir une plateforme stable et performante pour le développement de systèmes

robotiques avancés, en tirant parti des dernières avancées en matière de communication, de sécurité et de gestion des systèmes distribués.

ROS 2 Humble Hawksbill se distingue par plusieurs caractéristiques et améliorations remarquables :

— **Architecture Repenser pour la Flexibilité et la Sécurité :** Conçu pour répondre aux besoins des systèmes robotiques modernes, ROS 2 utilise DDS (Data Distribution Service) pour une communication flexible et fiable.

— **Support des Systèmes en Temps Réel :** Humble Hawksbill améliore le support des systèmes en temps réel, permettant une exécution précise et déterministe des tâches robotiques.

— **Sécurité Renforcée :** Cette version met un accent particulier sur la sécurité, intégrant des fonctionnalités pour la communication sécurisée et la gestion des identités.

— **Performance Optimisée :** Grâce à des optimisations diverses, Humble Hawksbill offre des performances accrues par rapport aux versions précédentes, avec une latence réduite et une meilleure utilisation des ressources.

— **Large Écosystème et Support Communautaire :** Comme avec ROS 1, la communauté active autour de ROS 2 continue de croître, offrant des ressources abondantes, des paquets compatibles et un support communautaire solide.

En conclusion, ROS 2 Humble Hawksbill offre une plateforme avancée et fiable pour le développement de projets robotiques modernes, intégrant les dernières technologies pour répondre aux exigences croissantes de la robotique.

2.9 Définition des fondamentaux de la navigation autonome sur ROS

ROS joue un rôle essentiel dans le développement de systèmes de navigation autonome. En fournissant une infrastructure solide et modulaire, ROS facilite l'intégration et le déploiement de capteurs, d'algorithmes de navigation et de systèmes de contrôle, permettant aux développeurs de créer des robots autonomes capables de fonctionner dans divers contextes et de réaliser des tâches complexes.

Alors, pour réaliser la navigation autonome d'un robot mobile, il est essentiel de définir les fondamentaux implémentés dans ROS qui facilitent cette fonctionnalité. Parmi ces éléments clés, on trouve des packages tels que *teleop*, le SLAM (Simultaneous Localization and Mapping), et l'algorithme d'AMCL (Adaptive Monte Carlo Localization).

1. **Description du package** *teleop*

   Ce package est un composant ROS conçu pour permettre la commande manuelle d'un robot mobile à l'aide du clavier. Il publie des messages du type $geometry\_msgs/Twist$ vers le topic cmd_vel. Ces messages spécifient les valeurs de vitesse linéaire et angulaire souhaitées pour contrôler le déplacement du robot. Voici l'interface du package décrite ci-dessous :

FIG. 2.8 – 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.

Pour installer le package *teleop* dans un environnement ROS, nous pouvons utiliser la commande suivante : $sudo\ apt\ install\ ros- < ROS - DISTRO > -teleop - twist - keyboard$. Nous remplaçons $< ROS - DISTRO >$ par le nom de notre distribution ROS (Noetic/Melodic pour ROS 1, Humble/Foxy/Jazzy pour ROS 2). Après avoir installé le package, nous pouvons maintenant lancer le node pour commencer à commander votre robot à l'aide du clavier. Pour se faire, nous lançons la commande : $rosrun\ teleop\_twist\_keyboard\ teleop\_twist\_keyboard.py$.

2. **Description du SLAM**

La cartographie environnementale du robot est effectuée à l'aide du SLAM (Simultaneous Localization and Mapping) qui est implémenté sous ROS sous forme d'un package appelé le *Gmapping*. Le package GMapping représente une implémentation clé de l'algorithme SLAM dans le cadre de ROS. Il utilise les données provenant de l'odométrie du robot (distance parcourue) ainsi que des capteurs extéroceptifs tels que les données provenant d'un LiDAR ou d'une caméra. L'objectif principal de GMapping est d'estimer précisément la position du robot en temps réel tout en générant simultanément une carte détaillée de son environnement. À travers des techniques probabilistes avancées, telles que les filtres de Kalman étendus et les

méthodes de Monte Carlo, GMapping gère l'incertitude associée à la localisation et à la cartographie dans des environnements réels souvent sujets au bruit et aux imprécisions des capteurs.

L'installation de ce package se faire on exécutant la commande suivante : $sudo\ apt - get\ install\ ros- < ROS - DISTRO > -gmapping$.

3. **Description d'AMCL**

   L'AMCL (Adaptive Monte Carlo Localization) est un algorithme avancé de localisation pour les robots mobiles, intégré dans ROS. Il combine la localisation basée sur l'odométrie avec les données de perception du robot pour estimer précisément sa position dans son environnement. Ci-dessous, nous allons voir comment l'AMCL fonctionne en détail :

   • **Filtres de particules :** L'AMCL utilise des filtres de particules pour représenter de manière probabiliste la position du robot. Ces particules représentent différentes hypothèses sur la position actuelle du robot dans l'environnement.

   • **Données d'odométrie :** L'algorithme utilise les données d'odométrie pour estimer la position du robot en se basant sur les mouvements qu'il effectue, tels que la distance parcourue et l'orientation.

   • **Observations des capteurs externes :** Les observations provenant des capteurs externes, comme un LiDAR, une caméra, sont utilisées pour corriger et mettre à jour les estimations de position du robot. Ces observations comprennent des informations sur les obstacles et les caractéristiques de l'environnement capturées par le capteur.

   • **Mise à jour continue de la pose :** L'AMCL maintient une mise à jour continue de la pose du robot en intégrant en temps réel les données d'odométrie et les observations des capteurs. Cela permet de corriger les erreurs accumulées dans l'estimation de la position du robot au fil du temps.

   • **Référence à une carte préétablie :** Pour améliorer la précision de la localisation, l'AMCL utilise une carte préétablie de l'environnement dans lequel le robot évolue. Cette carte sert de référence pour comparer les observations des capteurs et ajuster les estimations de position en conséquence.

4. **Description du planificateur des trajectoires**

   La planification du trajectoire permet au robot mobile de déterminer le trajet optimal entre son point de départ et sa destination. Ce processus se subdivise en deux approches : la planification locale et la planification globale.

   • **Planification locale :** consiste à générer une trajectoire adaptée aux conditions environnementales immédiates, permettant ainsi au robot de contourner efficacement les obstacles proches. Les méthodes telles que la fenêtre dynamique (DWA), le champ de potentiel et l'histogramme de champ de vecteur sont particulièrement flexibles, car elles peuvent fonctionner en temps réel dans des environnements dynamiques, même sans carte préétablie. Ces approches sont capables de s'ajuster aux données en temps réel pour naviguer et éviter les obstacles, les rendant adaptables à une variété de contextes de navigation [3].

   • **Planification globale :** La planification globale se concentre sur la création d'une trajectoire complète, du point de départ au point d'arrivée, avant que le robot ne commence à se déplacer. Cette approche nécessite des cartes détaillées et souvent préétablies de tout l'environnement. L'algorithme le plus communément utilisé pour cette planification est l'algorithme de Dijkstra. L'algorithme de Dijkstra

maintient une liste mise à jour des distances les plus courtes depuis un noeud de départ vers tous les autres noeuds du graphe. Initialement, la distance jusqu'au noeud de départ est fixée à zéro, tandis qu'elle est considérée comme infinie pour tous les autres noeuds [6].

2.10 Conclusion

En conclusion, ROS (Robot Operating System) représente une plateforme puissante et polyvalente pour le développement de systèmes robotiques avancés. À travers ses outils de visualisation comme RViz, Gazebo et rqt. ROS facilite aussi la création, la simulation et l'analyse des comportements robotiques. De plus, les versions ROS 1 et ROS 2 offrent des fonctionnalités adaptées aux besoins diversifiés des projets robotiques modernes. Avec une communauté active et un support continu, ROS continue de jouer un rôle central dans l'innovation et l'évolution de la robotique.

Dans le chapitre suivant, nous allons simuler la navigation d'un robot mobile à quatre roues nommé "Husky" sur ROS 1 et ROS 2. Nous débuterons d'abords par une introduction détaillant ce qu'est un robot "Husky", sa définition, ses caractéristiques et quelques exemples d'applications courantes. Nous passons par la suite à faire notre simulation sur ROS1 et ROS 2. Pour conclure ce chapitre, nous allons examiner de manière approfondie les différences entre ROS 1 et ROS 2, en mettant en évidence les avantages spécifiques de chacun par rapport à l'autre.

CHAPITRE

3

SIMULATION DU ROBOT HUSKY SOUS ROS

3.1 Introduction

Ce chapitre intitulé "Simulation du robot Husky sous ROS", explore l'utilisation de la plateforme Robot Operating System (ROS) pour simuler le robot terrestre Husky.

Nous commencerons par une présentation du robot Husky, en fournissant une définition claire et en décrivant ses caractéristiques techniques. Ensuite, nous illustrerons ses applications pratiques à travers des exemples concrets, démontrant sa polyvalence et son efficacité.

Ensuite, nous aborderons la navigation autonome du robot Husky. Nous détaillerons la simulation du robot sous ROS 1, puis sous ROS 2, en mettant en évidence les améliorations apportées par cette version plus récente de ROS.

Enfin, nous effectuerons une comparaison entre les simulations sous ROS 1 et ROS 2, permettant de comprendre les avantages et les inconvénients de chaque version.

3.2 Robot Husky

3.2.1 Définition

Le Husky (illustré dans la figure 3.1) est un robot terrestre sans pilote, robuste et facile à utiliser, spécialement conçu pour les applications de prototypage rapide et de recherche. Sa base robotique mobile est performante même dans des conditions difficiles, grâce à son châssis à quatre roues motrices puissantes et sans entretien, ses pneus tout-terrain robustes [5].

FIG. 3.1 – Robot Husky

Le Husky est une plateforme de développement robotique de taille moyenne, offrant une grande capacité de charge utile et des systèmes d'alimentation capables de supporter une variété de charges personnalisées pour les besoins de recherche. Il peut être équipé de caméras stéréo, de LiDAR, de GPS, d'IMU, de manipulateurs, et bien plus encore comme illustré dans la figure 3.2. Sa construction robuste et sa transmission à couple élevé lui permettent de mener des recherches dans des environnements inaccessibles à d'autres robots [4].

Entièrement pris en charge par ROS, le Husky dispose de code et d'exemples Open Source, facilitant son intégration et son utilisation dans divers projets de recherche.

FIG. 3.2 – Robot Husky équipé de caméras, LiDAR, GPS

3.2.2 Équations de robot Husky

La relation suivante entre la vitesse des roues et la vitesse de la plate-forme [5] :

$$v = \frac{v_D + v_G}{2} \tag{3.1}$$

$$\omega = \frac{v_D - v_G}{L} \tag{3.2}$$

Sachant que :

— v : vitesse de translation instantanée de la plateforme.

— ω : vitesse angulaire instantanée de la plateforme.

— v_D et v_G : les vitesses des roues droite et gauche respectivement.

— L : Distance entre le coté droit et le coté gauche.

3.2.3 Caractéristiques du robot terrestre Husky

— **Facile à utiliser :** Le Husky, première plateforme robotique de terrain à prendre en charge ROS dès sa configuration d'usine, utilise un protocole série open source pour ROS, C++ et Python. Cela permet d'intégrer rapidement les recherches existantes et d'exploiter les connaissances de la communauté ROS pour des résultats rapides.

— **Robuste et tout-terrain :** Husky, conçu avec des matériaux durables possède une transmission haute performance sans entretien et des pneus à crampons robustes. Cette conception simple et efficace permet au robot de naviguer sur des terrains difficiles.

— **Une référence de confiance :** Husky est largement utilisé et reconnu par des centaines de chercheurs et ingénieurs à travers le monde. Il a été mentionné dans de nombreux articles de recherche comme une plateforme de test fiable.

— **Contrôle de précision :** Husky, doté d'encodeurs à haute résolution, fournit des estimations précises de l'état et du mouvement, même à des vitesses très lentes avec un excellent rejet des perturbations.

— **Personnalisable :** Husky permet la sélection et l'intégration de diverses charges utiles, adaptant le robot à des besoins spécifiques. Il est compatible avec une large gamme d'accessoires robotiques, offrant une flexibilité maximale pour diverses applications.

3.2.4 Exemples d'applications du robot terrestre Husky

— **Systèmes multi-robots :**
Le robot terrestre Husky possède une architecture évolutive, ce qui en fait une plateforme idéale pour développer, tester et mettre en œuvre des systèmes impliquant plusieurs robots simultanément.

— **Perception et navigation :** Le robot Husky permet d'améliorer l'estimation de son état grâce à des paramètres de contrôle adaptatifs, des mesures d'odométrie précises et un système de diagnostic performant.

— **Manipulation :** Le robot Husky peut facilement être intégré avec divers manipulateurs et préhenseurs industriels pour effectuer des tâches de téléopération ou de manipulation.

— **Téléopération :** Le robot Husky est prêt à l'emploi pour toutes les opérations de téléopération, incluant une interface de commande par joystick sans fil [1].

3.3 Navigation autonome du robot Husky

Afin de faire la navigation autonome d'un robot mobile de type voiture (Husky) en utilisant ROS 2, nous commencerons d'abords par une phase de simulation sur ROS 1, où nous mettrons en oeuvre des algorithmes de navigation autonome standard disponibles dans cette version. Nous analyserons les performances et les limitations rencontrées lors de cette phase. Ensuite, nous répliquerons la même simulation sur ROS 2, en adaptant les algorithmes au nouvel environnement. Nous évaluerons ensuite les améliorations potentielles en termes de performance, de latence et de fiabilité par rapport à ROS 1.

3.3.1 Simulation du robot Husky sous ROS 1

Cette étape est importante pour notre projet, car elle nous a permis de simuler de manière détaillée la création de la carte de l'environnement dans Gazebo (déjà définie dans la partie 2.6.1) sur ROS 1 afin de faire une différence à celle du ROS 2 3.3.2.

Une fois la simulation effectuée dans Gazebo, nous allons utilisé RViz (déjà définie dans la partie 2.6.2)qui nous a permis de représenter graphiquement la carte générée par Gazebo, ainsi que les données sensorielles collectées par le robot pendant sa navigation. Cela inclut les informations provenant des capteurs tels que les LiDARs, les caméras et les scanners laser, essentiels pour une navigation précise et sûre.

Les prochaines étapes détailleront le processus de simulation de la navigation autonome sur ROS 1. Nous explorerons en détail comment nous allons configurer l'environnement de simulation, intégrer les algorithmes de navigation et évaluer les performances du système. Chaque étape sera soigneusement conçue pour assurer une simulation fidèle et significative de la navigation autonome, visant à offrir des perspectives précieuses pour le développement et l'amélioration des systèmes autonomes basés sur ROS 1.

1. **Cartographie d'un environnement virtuel sur Gazebo**

   Pour lancer la simulation du robot virtuel "Husky", il est essentiel d'installer des packages spécifiques dédiés à cette simulation. La Figure 3.3 illustre ces packages, qui englobent les modèles de simulation du robot "Husky", ainsi que ceux de ses capteurs tels que les LiDARs et les caméras. De plus, ces packages comprennent des composants pour intégrer le robot "Husky" dans Gazebo et pour le visualiser efficacement dans RViz.

```
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. 3.3 – Les packages nécessaires pour la cartographie

Une fois que les packages requis sont installés et correctement configurés, nous pouvons débuter la simulation du robot "Husky" dans Gazebo et observer son comportement via RViz. Pour démarrer la simulation, il suffit d'exécuter la commande suivante dans un terminal : `roslaunch husky_gazebo husky_playpen.launch`.

La Figure 3.4 présente la simulation du robot dans Gazebo, fournissant une représentation visuelle de son interaction avec l'environnement simulé. Ensuite, pour visualiser cet environnement, nous utilisons la commande : `roslaunch husky_viz view_robot.launch`. Cette étape permet de lancer RViz et d'afficher l'environnement dans lequel évolue le robot "Husky", comme illustré dans la Figure 3.5.

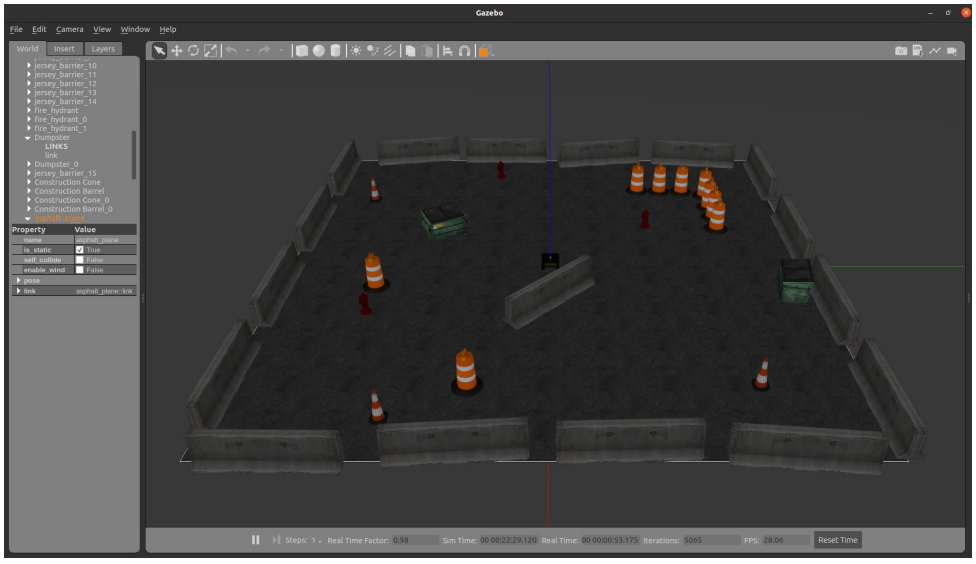

FIG. 3.4 – Simulation sous Gazebo

Nous allons maintenant procéder au lancement du noeud *gmapping* (détaillé dans la section 2) afin de réaliser la cartographie de notre environnement. Pour cela, nous allons exécuté la commande suivante dans un terminal : `roslaunch`

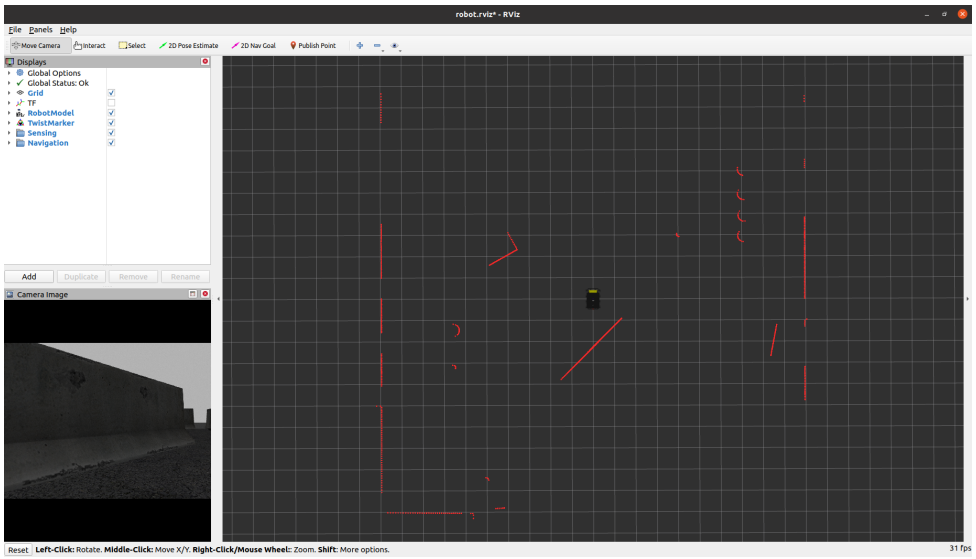

FIG. 3.5 – Visualisation sous RViz

`husky_navigation gmapping.launch`. La Figure 3.6 illustre la carte initiale avant que le robot ne commence à se déplacer.
Ensuite, pour obtenir la carte complète de notre environnement après avoir déplacé le robot, nous avons utilisé le package "*teleop*" détaillé dans la section 2.8. La Figure 3.7 présente cette cartographie complète. Le package "*teleop*" permet de piloter le robot à distance et ainsi de générer une cartographie détaillée de l'environnement exploré.

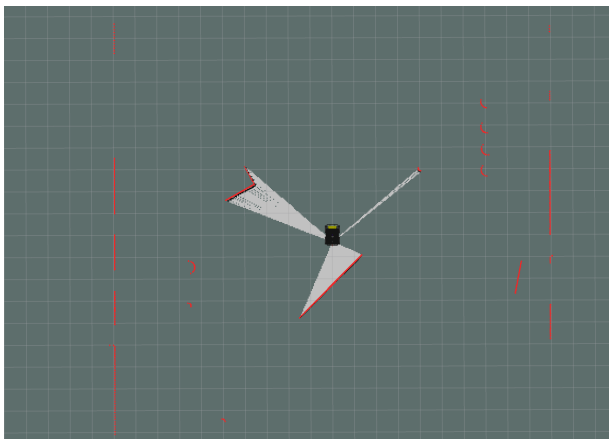

FIG. 3.6 – Début de la cartographie

FIG. 3.7 – La cartographie complète

Une fois que nous avons achevé la création de la carte de l'environnement à l'aide de la cartographie, nous procédons à sa sauvegarde afin de pouvoir l'utiliser plus tard lors de la navigation. Cette étape essentielle nous permet de capturer et de conserver une représentation précise de la carte générée. Pour sauvegarder la carte, nous utilisons la commande suivante dans notre environnement de développement : `rosrun map_server map_saver -f <NOM_DU_FICHIER>`.

2. **Navigation autonome du robot sur Gazebo**

   Une fois que nous avons enregistré la carte à l'aide du nœud *map_server*, nous procédons d'abord à la validation de la localisation en utilisant l'algorithme AMCL

(détaillé dans la section 3). Pour ce faire, nous lançons la commande suivante : `roslaunch husky_navigation amcl.launch`. Cela permet d'initialiser et de paramétrer l'algorithme AMCL pour que le robot puisse se localiser précisément dans l'environnement cartographié.

Ensuite, nous pouvons démarrer la navigation du robot en lançant la commande : `roslaunch husky_navigation map_based_navigation.launch` . Cette commande lance le processus de navigation basée sur la carte en utilisant la carte enregistrée précédemment. Cela permet au robot de planifier et d'exécuter des trajectoires autonomes dans l'environnement défini par la carte.

Pour visualiser ces processus, nous pouvons voir la Figure 3.8, qui illustre la simulation de la navigation basée sur la carte. Cette étape finale nous permet de voir le robot se déplacer de manière autonome dans son environnement en utilisant les informations de localisation et de cartographie préalablement établies.

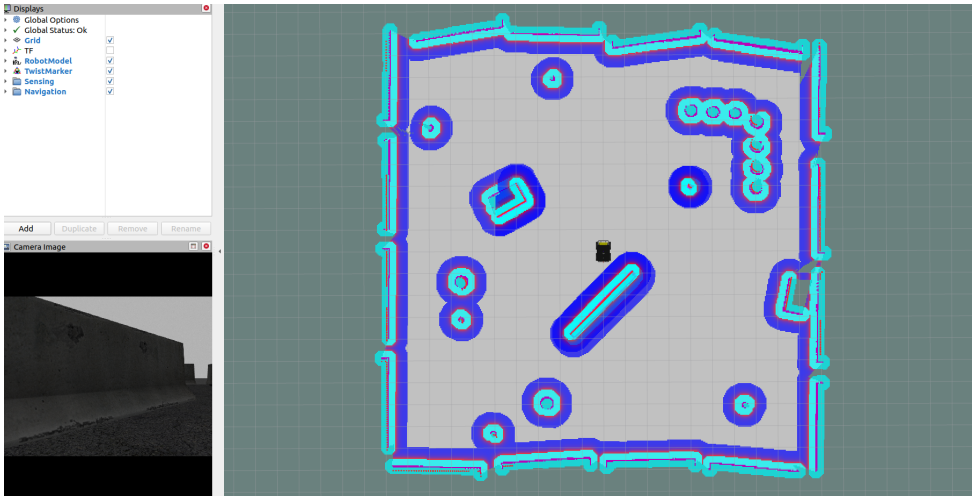

FIG. 3.8 – Visualisation de la carte de navigation

Après avoir affiché la carte dans RViz, nous déterminons un point de destination en utilisant la flèche de RViz, comme illustré dans la Figure 3.9. Cela autorise la publication de la position (x, y, Θ) de l'objectif dans le topic *move_base*.

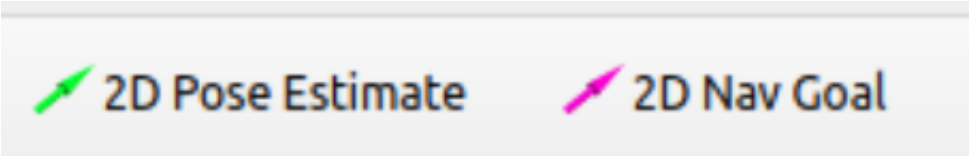

FIG. 3.9 – Publication de la pose désirée sur RViz

Une fois que la destination désirée a été définie, la Figure 3.10 montre le robot en train de se dirige vers elle. En même temps, la Figure 3.11 illustre la pose du robot à Gazebo.

FIG. 3.10 – Le robot se dirige vers la destination souhaitée

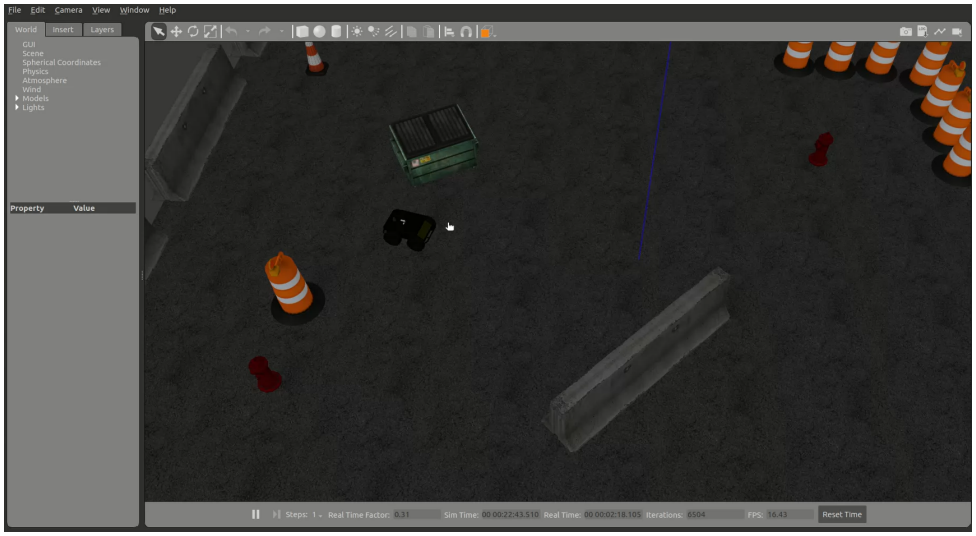

FIG. 3.11 – La pose du robot sous Gazebo

3. **Interprétation des résultats**

Après avoir réalisé la simulation de la navigation autonome du robot "Husky" sur ROS 1, nous avons observé que ROS 1 peut présenter des limitations en termes de performance, notamment lorsqu'il s'agit de simulations complexes. Cela peut se traduire par des ralentissements ou des temps de réponse plus longs au sein de la simulation. En comparaison avec des solutions plus récentes, ROS 1 pourrait manquer de support matériel avancé et d'une intégration optimale, notamment pour des dispositifs comme les LiDAR à haute résolution.

L'intégration matérielle limitée avec ROS 1 peut se manifester lorsque des applications de simulation nécessitent une gestion avancée des capteurs ou des actionneurs spécifiques. Dans de telles situations, une synchronisation précise et une gestion efficace des données en temps réel deviennent importantes pour garantir la performance et la précision des simulations.

3.3.2 Simulation du robot Husky sous ROS2

1. **Cartographie d'un environnement virtuel sur Gazebo**

   Pour lancer la simulation du robot virtuel "Husky" sous ROS2, il est essentiel d'installer des packages spécifiques contenant les fichiers de simulation, y compris le modèle du robot Husky et ses capteurs comme le LiDAR et la caméra, ainsi que l'architecture de l'environnement ou la carte de simulation. comme illustré dans la Figure 3.12.

```
hichem@hichem-ThinkPad-T570:~$ sudo apt-get install ros-humble-hardware-interface ros-humble-controller-manager ros-humble-gazebo-ros2-contro
ebo-plugins ros-humble-gazebo-msgs ros-humble-gazebo-ros ros-humble-gazebo-ros2-control-demos ros-humble-slam-toolbox ros-humble-navigation2
[sudo] password for hichem:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
ros-humble-controller-manager is already the newest version (2.41.0-1jammy.20240524.011651).
ros-humble-gazebo-msgs is already the newest version (3.7.0-1jammy.20240523.212917).
ros-humble-gazebo-plugins is already the newest version (3.7.0-1jammy.20240530.093311).
ros-humble-gazebo-ros is already the newest version (3.7.0-1jammy.20240523.231945).
ros-humble-gazebo-ros2-control is already the newest version (0.4.8-1jammy.20240524.013025).
ros-humble-gazebo-ros2-control-demos is already the newest version (0.4.8-1jammy.20240530.094031).
ros-humble-hardware-interface is already the newest version (2.41.0-1jammy.20240523.233911).
ros-humble-nav2-bringup is already the newest version (1.1.15-1jammy.20240530.115600).
ros-humble-navigation2 is already the newest version (1.1.15-1jammy.20240530.113754).
ros-humble-slam-toolbox is already the newest version (2.6.8-1jammy.20240530.113302).
ros-humble-xacro is already the newest version (2.0.8-1jammy.20240516.163442).
The following packages were automatically installed and are no longer required:
  libwpe-1.0-1 libwpebackend-fdo-1.0-1 python3-rospkg
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 4 not upgraded.
```

FIG. 3.12 – Installation des packages

Une fois les packages installés, la simulation du robot "Husky" peut être démarrée dans Gazebo en exécutant la commande `ros2 launch husky_simulation husky_gazebo.launch.py`. Cela ouvrira la fenêtre Gazebo, visualisant notre environnement de simulation, comme illustré dans la figure 3.13.

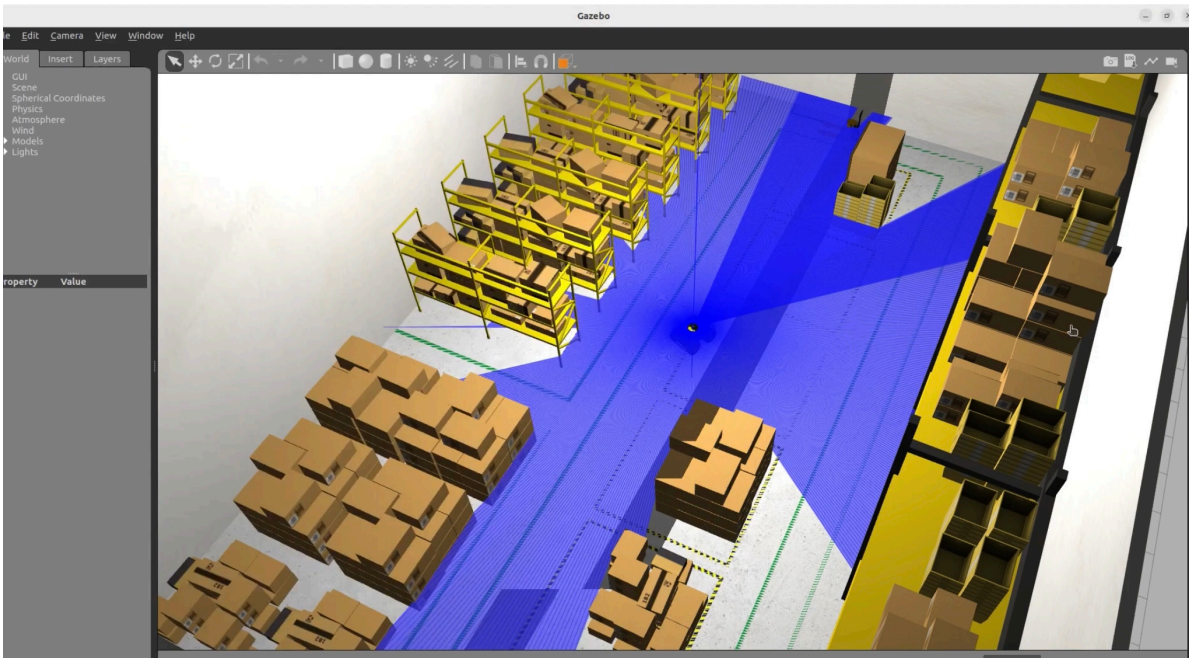

FIG. 3.13 – Simulation sous Gazebo

À cette étape, nous pouvons observer le comportement du robot Husky via la fenêtre RViz, comme illustré dans la figure 3.14. Pour ce faire, il suffit d'exécuter la commande `ros2 launch husky_navigation online_async_launch.py`. Cette

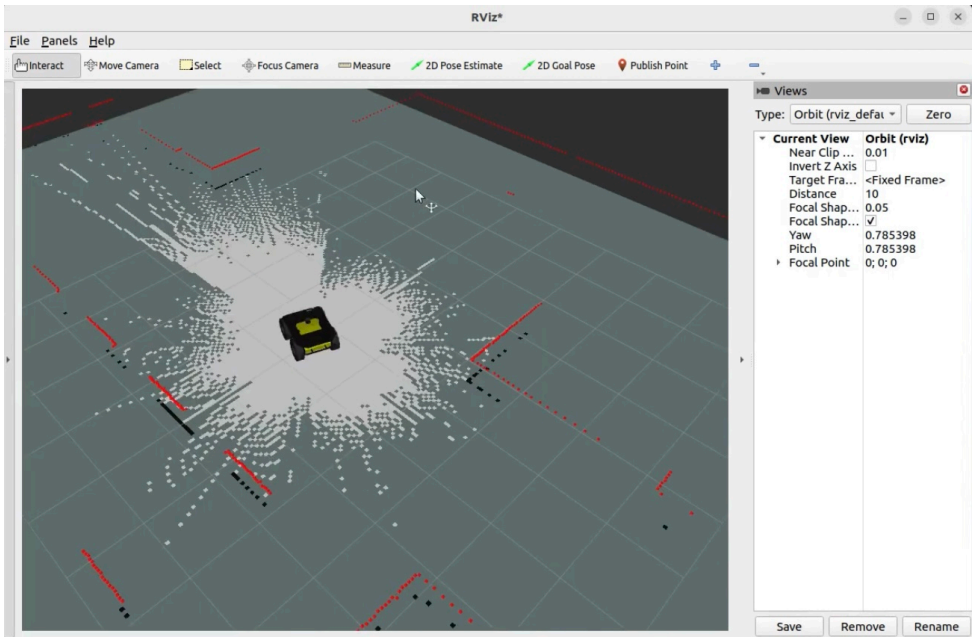

FIG. 3.14 – Visualisation sous RViz

commande permet de lancer RViz et d'afficher l'environnement dans lequel évolue le robot Husky, après avoir ajusté certains paramètres d'affichage.

Pour réaliser la cartographie de notre environnement, il est nécessaire de déplacer le robot Husky. Cela peut être fait en utilisant la commande `ros2 run teleop_twist_ keyboard teleop_twist_keyboard`, qui permet de contrôler le robot à l'aide des touches du clavier, Cette méthode de téléopération a été illustrée dans la partie précédente (dans la partie 2.8).

Pour débuter l'opération de cartographie, il est nécessaire de déplacer le robot Husky dans l'ensemble de l'environnement afin que le LiDAR puisse scanner et capturer toutes les caractéristiques de la zone. La figure 3.15 illustre le début de cette opération, tandis que la figure 3.16 montre la cartographie complète obtenue.

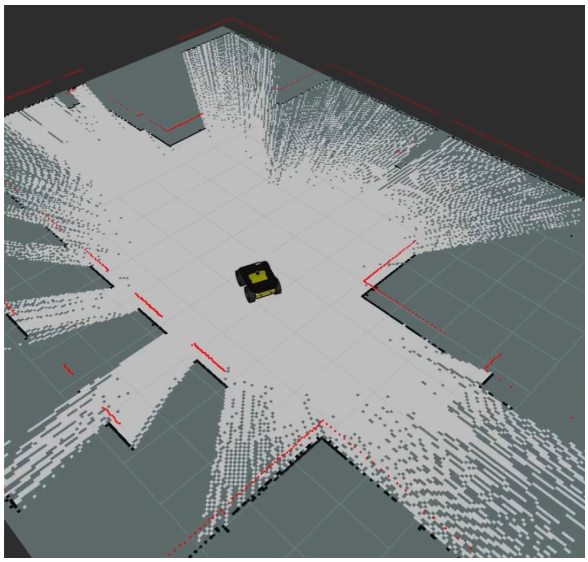

FIG. 3.15 – Début de la cartographie

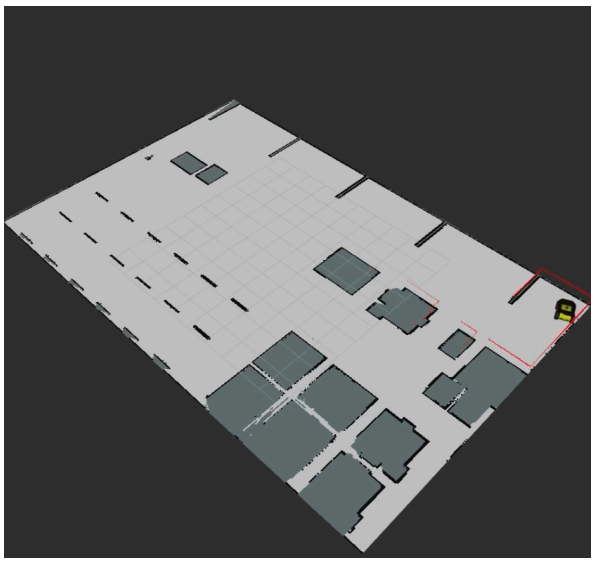

FIG. 3.16 – La cartographie complète

Une fois la carte de l'environnement créée, il est essentiel de la sauvegarder pour une utilisation ultérieure, notamment pour les tâches de navigation. Pour ce faire, la commande suivante est utilisée : `ros2 run nav2_map_server map_saver_ cli -f <NOM_DE_LA_CARTE>`.

2. **Navigation autonome du robot sur Gazebo**

   Après l'enregistrement de notre carte, nous pouvons démarrer la navigation autonome du robot en lançant la commande suivante :

   `ros2 launch husky_navigation husky_navigation.launch.py`

   Cette commande permet de lancer RViz, qui visualise le robot sur la carte de navigation que nous avons enregistrée, comme illustré à la figure 3.17. Nous pouvons alors initier le processus de navigation autonome.

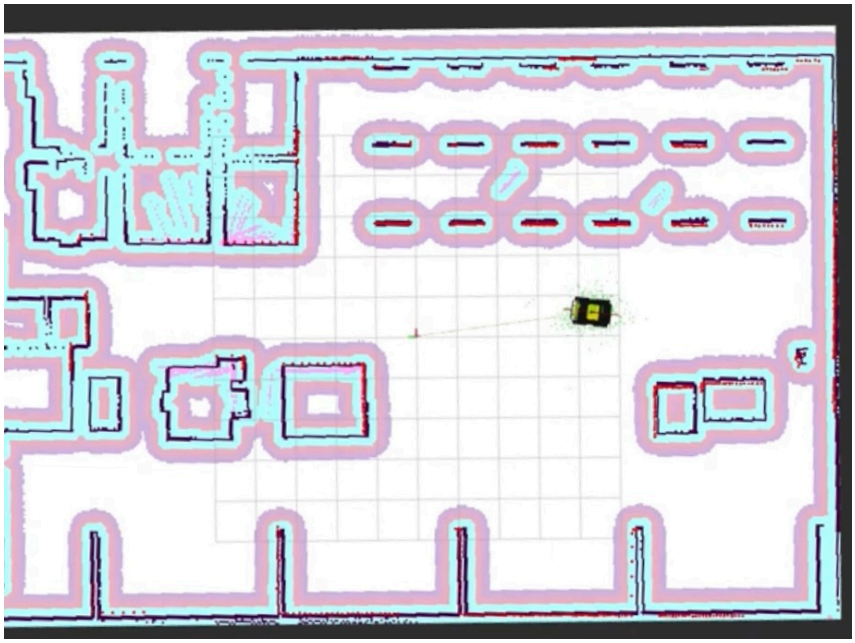

FIG. 3.17 – Visualisation de la carte de navigation

La commande permet au robot de planifier et d'exécuter des trajectoires autonomes dans l'environnement défini par la carte enregistrée. Pour démarrer la navigation, il suffit de sélectionner l'option "Nav2 Goal" affichée au-dessus de la barre d'état de RViz, comme montré à la figure 3.18.

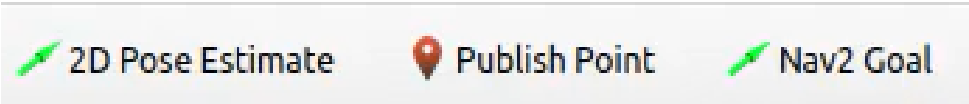

FIG. 3.18 – Publication de la pose désirée sur RViz

Après avoir sélectionné cette option, représentée par une flèche verte, il faut placer cette flèche sur le point de destination souhaité sur la carte, comme illustré à la figure 3.19. Enfin, nous observons que le robot se dirige vers le point souhaité, comme montré à la figure 3.20.

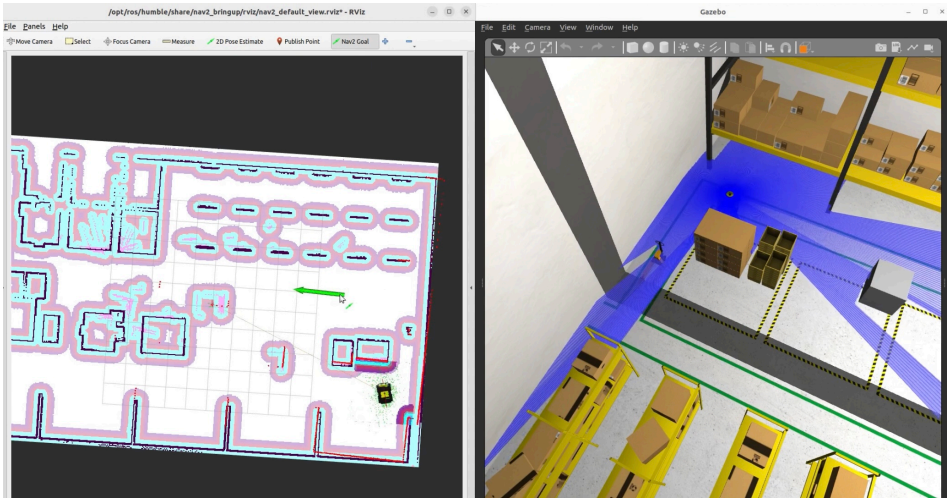

FIG. 3.19 – Visualisation de la destination souhaitée

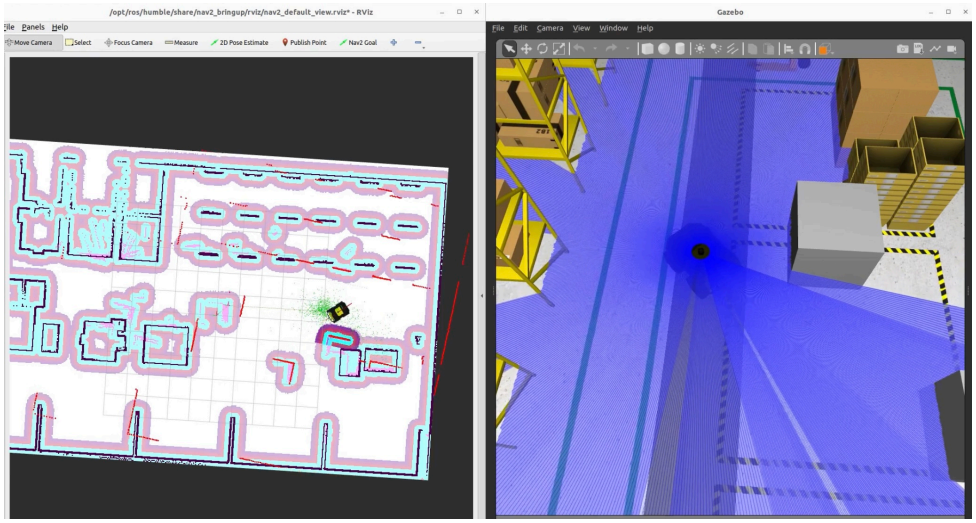

FIG. 3.20 – Visualisation de la destination atteinte

3. **Interprétation des résultats**

La simulation de la navigation autonome du robot "Husky" sous ROS 2 a montré des améliorations significatives par rapport à ROS 1 dans plusieurs aspects clés. La rapidité et la flexibilité de ROS 2 sont nettement supérieures, permettant des temps de réponse plus courts et une gestion plus efficace des tâches complexes. Cette amélioration se traduit par des simulations plus fluides, même lorsque des configurations avancées de capteurs, comme les LiDAR à haute résolution, sont utilisées.

En outre, ROS 2 offre un meilleur support matériel et une intégration optimisée pour divers dispositifs, ce qui améliore considérablement la performance globale. La capacité de ROS 2 à gérer les données en temps réel de manière synchronisée et efficace permet une simulation plus précise et fiable. Ces avantages rendent ROS 2 particulièrement adapté pour des applications nécessitant une gestion avancée des capteurs et des actionneurs, garantissant ainsi une performance supérieure et une précision accrue dans les simulations.

3.3.3 Comparaison entre ROS 1 et ROS 2

Les divergences sont synthétisées dans le tableau comparatif suivant 3.1 :

Caractéristique	**ROS 1**	**ROS 2**
Communication	Basée sur les sockets (TCP/UDP)	Introduit un middleware (DDS) pour la communication
Langages de programmation	Principalement C++ et Python	Support étendu pour divers langages (C++, Python, Java, etc.)
Systèmes d'exploitation	Linux principalement, avec des efforts pour Windows	Meilleur support multiplateforme (Linux, Windows, macOS)
Performances	Limitées par le système de communication	Améliorations de performances avec DDS
Fiabilité	Moins robuste dans les environnements distribués	Meilleure gestion des erreurs et de la fiabilité
Sécurité	Faible en termes de sécurité des communications	Améliorations de la sécurité avec DDS et TLS
Support temps réel	Limité	Meilleur support pour les systèmes temps réel
Modularité	Moins modulaire, souvent monolithique	Architecture plus modulaire
Outils et Écosystème	Vaste écosystème, nombreux packages disponibles	Transition progressive vers ROS 2
Compatibilité ascendante	Supporte les versions antérieures, mais des limitations	Améliorations de la compatibilité avec ROS 1
Utilisation industrielle	Utilisé largement dans la recherche et l'industrie	Adoption croissante dans divers secteurs industriels
Développement communautaire	Communauté active avec de nombreux contributeurs	Croissance de la communauté et des contributeurs

Tab. 3.1 – Tableau comparatif entre ROS 1 et ROS 2 [1]

Après avoir simulé le robot à la fois sur ROS 1 3.3.1 et ROS 2 3.3.2, nous avons identifié des différences significatives entre ces deux plateformes. ROS 2 représente une évolution notable par rapport à ROS 1, offrant des améliorations significatives en termes de performance, de support multi-plateforme et de gestion des communications. Ces évolutions sont spécifiquement conçues pour répondre aux exigences croissantes de la robotique moderne, offrant une fiabilité accrue et une flexibilité améliorée pour le développement et le déploiement des systèmes robotiques.

L'installation des packages sur ROS 2 a posé plusieurs défis comparativement à ROS 1, où l'accès à une base de données bien établie facilite davantage la recherche et le développement. Cependant, ROS 2 montre clairement son potentiel à surmonter ces défis

grâce à des avancées technologiques continues.

ROS 2 repose en grande partie sur les mêmes fondations que ROS 1 tout en intégrant de nouvelles fonctionnalités, notamment l'utilisation du protocole DDS pour la distribution de données en temps réel. DDS agit comme un middleware améliorant la communication entre les nœuds, offrant des avantages significatifs en termes de temps de réponse, de scalabilité, de performances et de sécurité.

Une des améliorations notables de ROS 2 par rapport à ROS 1 est l'annonce de la fin de vie de ROS 1 Noetic prévue pour mai 2025. Bien que ROS 1 restera utilisable jusqu'à cette date, l'OSRF [1] recommande fortement la transition vers ROS 2. Pour faciliter cette transition, des 'bridges' sont disponibles pour permettre l'interopérabilité entre les outils ROS 1 et ROS 2.

3.4 Conclusion

Ce chapitre a exploré en détail la simulation du robot terrestre Husky en utilisant la plateforme ROS. Nous avons commencé par définir le robot Husky et examiner ses caractéristiques techniques, mettant en lumière ses capacités et son adaptabilité à diverses applications pratiques.

Ensuite, nous avons abordé la navigation autonome du robot Husky, en présentant les étapes et les outils nécessaires pour simuler le robot sous ROS 1 et ROS 2. Les avantages et les améliorations de ROS 2 par rapport à ROS 1 ont été particulièrement soulignés.

Enfin, une comparaison des simulations sous ROS 1 et ROS 2 a été effectuée, permettant de mettre en évidence les forces et les faiblesses de chaque version. Cette comparaison offre des insights précieux pour les développeurs et les chercheurs dans le choix de l'environnement de simulation le plus approprié à leurs besoins.

[1] OSRF une organisation à but non lucratif qui soutient le développement de logiciels open source pour la robotique. Fondée en 2012, l'OSRF est connue pour son implication dans des projets comme ROS.

CHAPITRE

4

CONCLUSION GÉNÉRALE

Ce mémoire a exploré en profondeur les divers aspects de la robotique moderne, offrant une compréhension approfondie des concepts théoriques et des avancées technologiques qui façonnent ce domaine multidisciplinaire.

Dans le premier chapitre, nous avons dressé une vue d'ensemble essentielle de la robotique, en commençant par un historique de la robotique et son évolution. Nous avons ensuite approfondi notre exploration avec la robotique mobile, en définissant ce sous-domaine et en classifiant les robots selon divers critères tels que le degré d'autonomie, les domaines d'application et les types de locomotion. Cette section a été enrichie par une comparaison détaillée des différents modes de déplacement des robots, soulignant leur adaptation à des environnements variés et complexes. La démonstration des avantages et des inconvénients des robots mobiles a mis en lumière les bénéfices potentiels ainsi que les défis éthiques et économiques associés à leur utilisation.

Le deuxième chapitre sur le Robot Operating System (ROS) a mis en évidence l'importance de cette plateforme pour le développement de systèmes robotiques avancés. Avec des outils comme RViz, Gazebo et rqt, ROS simplifie la création, la simulation et l'analyse des comportements robotiques, répondant ainsi aux besoins diversifiés des projets modernes. L'explication des différentes plateforme du ROS 1 et ROS 2 a permis de comprendre leurs fonctionnalités spécifiques et leurs applications respectives, soulignant leur évolution et leur adaptation aux nouvelles exigences technologiques.

Dans le dernier chapitre, nous avons examiné en détail les différences entre ROS 1 et ROS 2. Cette comparaison approfondie a permis de mettre en évidence les forces et les faiblesses de chaque version. Nous avons analysé les performances des simulations de navigation autonome réalisées sur ROS 1 et ROS 2, en évaluant notamment la fiabilité, la latence et l'adaptabilité à des environnements variés. Cette étude nous a également permis de comprendre les améliorations apportées par ROS 2 par rapport à ROS 1, notamment en termes de support pour les systèmes distribués, de modularité et de sécurité.

En conclusion, ce mémoire a établi une base solide de connaissances sur la robotique, couvrant à la fois les aspects théoriques et pratiques. Il a exploré les avancées significatives

dans la navigation autonome, mettant en lumière les processus complexes de perception, de décision et d'action nécessaires pour permettre aux robots de fonctionner de manière autonome dans des environnements dynamiques. Ces contributions sont essentielles pour l'innovation continue et l'évolution future de la robotique, jouant un rôle central dans la transformation de divers secteurs industriels et technologiques à l'échelle mondiale.

BIBLIOGRAPHIE

[1] Clearpath robotics - drone terrestre husky a200 ugv. https://www.generationrobots.com.

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

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

[4] Clearpath Robotics. Husky unmanned ground vehicle robot. https://clearpathrobotics.com/husky-unmanned-ground-vehicle-robot/.

[5] Clearpath Robotics. *Husky User Manual*. Clearpath Robotics, Kitchener, ON, Canada.

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

[7] David Filliat. *Robotique mobile*. PhD thesis, EDX, 2011.

[8] Luc Jaulin, Michel Legris, and Frédéric Dabe. Gesmi, un logiciel por l'aide à localisation de mines sous-marines. 01 2006.

[9] Stéphane Lens. Locomotion d'un robot mobile. 2008.

[10] Luiz Fernando Oliveira, A. Moreira, and Manuel Silva. Advances in agriculture robotics : A state-of-the-art review and challenges ahead. *Robotics*, 10:52, 03 2021.

[11] Icheroufene Omar. *Conception et réalisation d'un Robot mobile télécommandé à base de la PCDUINO V3*. PhD thesis, Université Mouloud Mammeri, 2016.

[12] ABEDLKADER OUMAYA, ABEDELOUAHAB HADJAJ FAROUK, and Djallal ABDESSEMED. *Etude et Réalisation d'un Robot Mobile Multi Taches Destiné au Domaine Agricole*. PhD thesis, UNIVERSITÉ KASDI MERBAH OUARGLA.

[13] Manon Prédhumeau. *Modélisation et simulation de comportements piétons réalistes en espace partagé avec un véhicule autonome*. PhD thesis, 12 2021.

[14] ROS (Robot Operating System). Understanding ROS. https ://www.ros.org/Understanding-ROS/.

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

[16] Hocine TAKHI. Conception et réalisation d'un robot mobilea base d'arduino. *Université Amar Telidji-Instrumentation*, 2014.

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

[18] Nabila Zrira. Master thesis, 08 2016.