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 : TELDJOUNE Zakaria
GUENNICHE Ismail

Thème

Navigation autonome des robots mobiles

Soutenu publiquement, le 06 / 07 / 2022, devant le jury composé de :

M BRAHAMI Mustapha Anwar	MCA	ESSA. Tlemcen	Président
Mme SEBBAGH Hafidha	MCA	ESSA. Tlemcen	Directeur de mémoire
M ABDELLAOUI Ghouti	MCB	ESSA. Tlemcen	Co- Directeur de mémoire
M ABDI Sidi Mohammed	MCB	ESSA. Tlemcen	Examinateur 1
M M'HAMEDI Mohammed	MAA	ESSA. Tlemcen	Examinateur 2

Année universitaire : 2021 /2022


بسم الله الرحمن الرحيم

Dédicaces

Je dédie cet ouvrage

À mes chers parents, pour tous leurs sacrifices, leur amour, leur tendresse, leur soutien et leurs prières tout au long de mes études,

À mes chers frères ZOHEIR, ZINNE EDDINE et MOHAMED AMINE pour leurs encouragements permanents, et leur soutien moral,

À ma grand-mère et toute ma famille pour leur soutien et leur encouragement,

À mon ami Younes et tous mes amis qui m'ont toujours encouragé, et à qui je souhaite plus de succès,

Sans oublier mon binôme pour son soutien moral, sa patience et sa compréhension tout au long de ce projet.

Que ce travail soit l'accomplissement de vos vœux tant allégués, et le fuit de votre soutien infaillible,

Merci d'être toujours là pour moi.

ISMAIL.GUENNICHE

Dédicaces

Je dédie ce travail

A mes parents qui m'a soutenu et encouragé ces années d'études.

Qu'ils trouvent ici le témoignage de ma profonde reconnaissance.

A mes 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 supporté et encouragé tout au long de mon parcours.

A mon âme sœur, et ma famille qui m'a donné de l'amour et de la vivacité.

A tous mes amis qui m'ont toujours encouragé, et à qui je souhaite plus de succès.

Zakaria.TELDJOUNE

Remerciements

Nous rendons nos profondes gratitudes à Dieu tout puissant qui nous a aidés à réaliser ce modeste travail.

Nous exprimons nos profondes gratitudes à nos parents pour leurs encouragements, leurs soutiens et pour les sacrifices qu'ils ont enduré.

Nous remercions, notre encadreur et co-encadreur Mme SEBBAGH Hafidha et M ABDELLAOUI Ghouti pour les efforts qu'il a déployé, pour nous aider, conseiller, encourager et corriger.

Nous sommes sensibles à l'honneur que nous a fait monsieur Mustapha Anwar BRAHAMI, pour avoir accepté de présider notre mémoire et de nous honorer de sa présence au sein du jury.

Nous tenons egalement à adresser nos vifs remerciements à monsieur Sidi Mohamed ABDI, et à monsieur mohammed MHAMEDI, nous les remercions chaleureusement pour avoir accepté d'examiner le present memoire et pour ses observations et remarques pertinentes et constructives.

Nous remercions vivement M ADJIM RAMZ-EDDINE Abderrezak, l'ingénieur de FABLAB à l'Ecole supérieur en science appliqué de Tlemcen, pour avoir aidé à la réalisation de notre travail. Sans oublier tous nos amis Que tous ceux qui ont contribué de près ou de loin à la réalisation de ce travail, trouvent ici notre sincère reconnaissance.

Résumé

Naviguer dans des environnements non structurés est une capacité de base des créatures intelligentes, qui est un intérêt fondamental dans ce projet. La navigation est une tâche complexe qui repose sur le développement d'une représentation interne de l'espace, fondée sur des repères reconnaissables et un traitement visuel robuste, qui peut simultanément soutenir l'auto-localisation continue de sa place et une représentation de l'endroit où il va.

Le nombre de robots déployés dans l'industrie manufacturière a augmenté rapidement et cette tendance devrait se poursuivre à l'avenir, car les robots autonomes ont le potentiel d'automatiser un large éventail de tâches à forte intensité de main-d'œuvre dans l'environnement de l'usine et d'améliorer la production. De nombreux défis techniques doivent être résolus pour réaliser une navigation autonome.

Dans ce projet, nous visons à résoudre le problème principal de la navigation autonome des robots en effectuant une simulation dans la plate-forme ROSDS. Ce dernier est l'outil web de construction pour la programmation en ligne des robots ROS. Notre robot sera capable de reconnaître les balises dans l'environnement et de naviguer sur le chemin désigné en évitant les obstacles sur son chemin de sa position actuel à sa position cible de manière autonome. Nous avons utilisé une pile de navigation qui fournit toutes les fonctions nécessaires pour que les robots mobiles soient autonomes. L'application de notre modèle de navigation sur le robot mobile à donner des résultats très satisfaisantes.

Le robot atteint une position cible et évite tous les obstacles, contrôlés par notre modèle de navigation.

Abstract

Navigating in unstructured environments is a basic capacity of intelligent creatures, which is a fundamental interest in this project. Navigation is a complex task that relies on the development of an internal representation of space, based on recognizable landmarks and robust visual processing, which can simultaneously support continuous self-localization of one's place and a representation of the place where he is going.

The number of robots deployed in the manufacturing industry has increased rapidly and this trend should continue in the future, because autonomous robots have the potential to automate a wide range of tasks with high intensity of labor in the factory environment and improve production. Many technical challenges must be resolved to carry out autonomous navigation.

In this project, we aim to solve the main problem of autonomous robot navigation by performing a simulation in the ROSDS platform. The latter is the construction web tool for online programming of ROS robots. Our robot will be able to recognize beacons in the environment and navigate the designated path avoiding obstacles on its way from its current position to its target position autonomously. We used a navigation stack that provides all the functions needed for mobile robots to be autonomous. The application of our navigation model on the mobile robot to give very satisfactory results. The robot reaches a target position and avoids all obstacles, controlled by our navigation model.

ملخص

يعد التنقل في البيئات غير المهيكلة قدرة أساسية للمخلوقات الذكية ، لذا فهي ذات أهمية أساسية في هذا المشروع. التنقل هو مهمة معقدة تعتمد على تطوير تمثيل الفضاء الداخلي ، بناءً على معالم يمكن التعرف عليها ومعالجة بصرية قوية ، والتي يمكن أن تدعم في الوقت نفسه التحديد الذاتي المستمر لمكان الفرد وتمثيل المكان الذي يتجه إليه.

زاد عدد الروبوتات المنتشرة في الصناعة التحويلية بسرعة ومن المتوقع أن يستمر هذا الاتجاه في المستقبل ، حيث تتمتع الروبوتات المستقلة بالقدرة على أتمتة مجموعة واسعة من المهام كثيفة العمالة في التصنيع بيئة المصنع وتحسين الإنتاج. يجب حل العديد من التحديات التقنية لتحقيق التنقل المستقل.

في هذا المشروع ، نهدف إلى حل المشكلة الرئيسية المتمثلة في التنقل باستخدام الروبوت المستقل من خلال إجراء محاكاة في منصة ROSDS . هذا الأخير هو أداة الويب الإنشائية للبرمجة عبر الإنترنت لروبوتات ROS . سيتمكن الروبوت الخاص بنا من التعرف على الحواجز في البيئة والتنقل في المسار المحدد لتجنب العقبات في طريقه من موقعه الحالي إلى موقعه المستهدف بشكل مستقل. استخدمنا مكدس تنقل يوفر جميع الوظائف اللازمة لكي تكون الروبوتات المتنقلة مستقلة. تطبيق نموذج الملاحة الخاص بنا على الروبوت المتحرك لإعطاء نتائج مرضية للغاية. يصل الروبوت إلى موقع مستهدف ويتجنب جميع العوائق التي يتحكم فيها نموذج الملاحة الخاص بنا

Table des matières

Table des figures

Liste des tableaux

Table des acronymes

AGV	Automated Guided Vehicle
AMCL	Adaptive Monte Carlo Localization
AMR	Autonomous Mobile Robot
CNC	Commande Numérique par Calculateur
DWA	Dynamic Window Approach
GUI	Graphical User Interface
IMU	Inertial Measurement Unit
MCL	Monte Carlo Localization
MIT	Massachusetts Institute of Technology
OGM	Occupancy Grid Map
R.U.R	Rossum's Universal Robots
RGB	Red-Green-Blue
ROI	Return on investement
ROS	Robot Operating System
ROSDS	Robot Operating System Development Studio
Rviz	ROS visualization
SLAM	Simultaneous Localization And Mapping
SPLAM	Simultaneous Planning Localization and Mapping
TEB	Timed Elastic Band
TR	Trajectory Rollout

Introduction générale

Suit aux évolutions des modes de vie et l'apparition des robots qui remplacent l'homme dans divers domaines tel que : l'agriculture, la chirurgie, l'industrie, etc. Les entreprises sont de plus en plus à la recherche des machines qui ont des qualités importantes par rapport aux humains d'où l'apparition des robots industriels.

Un robot mobile peut être défini comme un système mécanique capable de se déplacer de manière autonome dans son environnement. Pour cela, le robot doit être équipé de :

— Capteurs qui vont l'aider à percevoir son environnement (qui plus ou moins il le connaît) et à se localiser.

— Actionneurs qui lui permet de se déplacer.

— Intelligence (algorithme, ou régulateur) qui lui permettra de calculer les commandes à envoyer à l'actionneur en fonction des données recueillies par les capteurs pour effectuer la tâche demandée.

Enfin, il faut joindre au robot un environnement qui correspond au monde dans lequel il évolue, et aux tâches qu'il doit accomplir. Les robots mobiles sont en développement constant depuis les années 2000 et ils sont principalement utilisés dans les domaines militaires (drones volants, robots sous-marins, etc.), médical ou agricole. Ils sont notamment amenés à effectuer des tâches jugées difficiles ou dangereuses pour l'homme. C'est

le cas des opérations de déminage, de recherche de boîtes noires d'avions endommagées sur le fond marin ou d'exploration de planètes. Il existe aussi d'autres exemples sur les robots mobiles comme les satellites, les lanceurs (Ariane V), les métros autonomes et les ascenseurs. Les avions de ligne, les trains et les voitures continuent d'évoluer vers des systèmes de plus en plus automatisés et deviendront probablement des robots mobiles dans les décennies à venir.

La robotique mobile est une discipline liée à la conception de robots mobiles. Elle fait appel à d'autres disciplines telles que l'automatique, le traitement du signal, la mécanique, l'informatique et l'électronique. Le but de ce mémoire est de fournir un aperçu des outils et des méthodes robotiques qui aident à la conception des robots mobiles. Le robot sera modélisé par une équation d'état (c'est-à-dire une équation différentielle du premier ordre). Ces outils sont essentielles à la fois pour la simulation du robot et à la conception du régulateur.[7]

Nous allons toutefois illustrer une généralité sur la robotique au chapitre 1 pour introduire des concepts importants de la robotique comme l'environnement dynamique et incertain, les robots à pattes, les types de navigation, la différence entre les robots holonome et les robots non-holonome, la différence entre un AGV et un AMR.

Le chapitre 2 s'appuie essentiellement sur les différents concepts de sécurité certifiés pour les robots mobiles en environnements industriels.

Le chapitre 3 est consacré à la simulation de notre robot sur la Platform open-source de ROS Navigation, dans le but concrétiser notre modèle de navigation (mapping, localisation, planification de trajectoire et l'évitement d'obstacles) et visualiser le comportement dans un environnement similaire à l'environnement réel.

A la fin de ce mémoire, une conclusion générale résumera notre étude sur les robots mobiles d'une manière générale et la navigation autonome de ces derniers en particulier.

Ainsi, elle présentra aussi le future de ce projet de master en citant quelques perspectives prévues pour améliorer ce travail.

Chapitre 1

Généralité sur la robotique

1.1 Introduction

Le but de la robotique est l'automatisation des systèmes mécaniques. En dotant le système de capacités de perception, d'action et de décision, l'objectif est de lui permettre d'interagir de manière rationnelle et autonome avec son environnement. La robotique est un domaine d'étude à l'intersection de l'intelligence artificielle, de l'automatisation, de l'informatique et de la perception par ordinateur, et cette interdisciplinarité découle d'une certaine complexité.

Ce chapitre vise à introduire le sujet de ce travail en introduisant brièvement quelques généralités sur les robots mobiles afin de mieux comprendre la navigation des robots mobiles.

1.2 Historique

Le mot "robot" a été introduit pour la première fois en 1921 dans une pièce de théâtre écrite par le tchèque Karel Capek, et intitulée R.U.R (Rossum's Universal Robots).

En 1942, Issac Asimove formule, dans sa nouvelle "Runaround" les trois lois de la robotique censées protéger l'être humain :

— Un robot ne peut pas blesser un être humain ou, par inaction, ne permet pas à un être humain de se blesser.

— Un robot doit obéir aux ordres qui lui sont donnés par les êtres humains, sauf lorsque de tels ordres entreraient en conflit avec la première loi.

— Un robot doit protéger sa propre existence tant qu'une telle protection n'est pas en conflit avec la première ou la deuxième loi.

En 1953, le premier AGV au monde a été introduit au Royaume-Uni pour le transport, qui a été modifié à partir d'un tracteur et peut être guidé par un câble aérien.

En 1961, le premier robot industriel commence à travailler sur une ligne d'assemblage dans l'usine du General Motors, pour soulever et tordre des pièces chaudes de métal.

En 1979, Le premier bras articulé fait son apparition sur une chaine de production.

En 1986, Honda a créé le premier robot capable de marcher sur deux pieds comme un humain.

En 1988, le premier robot de service fait son apparition à l'hôpital de Danbury (Etats-Unis).

En 1997, La NASA envoie son premier robot explorateur sur Mars.

Le 3 juin 2014, La Commission Européenne et 180 entreprises de recherche ont lancé le programme SPARC, programme civil de recherche et d'innovation en robotique. [1]

1.3 Définition d'un robot

De manière générale, la robotique joue un rôle très important dans l'industrie et prend de plus en plus d'importance dans des domaines de services tels que la maintenance, l'exploration, la médecine, etc.

Étymologiquement, le mot robot vient du mot tchèque robota, qui signifie travail et désigne un système automatique contrôlé par une unité de contrôle informatique. Un

robot est un groupe d'appareils mobiles associés dont les mouvements sont contrôlés et synchronisés numériquement. La structure, la forme et la fonction de ces robots doivent être adaptées à l'environnement avec lequel ils interagissent.

Chaque mouvement de base est un axe de commande numérique par calculateur (CNC). Cependant, un robot n'est qu'une machine programmable qui ne fait rien d'autre qu'exécuter ce que les humains lui enseignent. D'une manière générale, ils peuvent être divisés en deux catégories : les robots ou manipulateurs à base fixe et les robots mobiles.

1.4 Applications des robots mobiles

Les robots mobiles ont une place particulière en robotique. Leur intérêt réside dans leur mobilité qui ouvre des applications dans de nombreux domaines. Comme les robots manipulateurs, qui sont destinés à assister l'homme dans les tâches pénibles (transport de charges lourdes), monotones ou en ambiance hostile (nucléaire, marine, spatiale, lutte contre l'incendie, surveillanc,etc.)

Voici quelques applications de la robotique mobile [13] :

1.4.1 Industrie nucléaire

— Surveillance des sites.

— Manipulation des matériaux radio-actifs.

— Démantèlement des centrales nucléaires.

1.4.2 Sécurité civile

— Neutralisation d'activité terroriste.

— Déminage.

— Pose d'explosif.

— Surveillance de munitions.

1.4.3 Militaire

— Surveillance, patrouillage.

— Pose d'explosifs.

— Manipulation de munitions.

1.4.4 Chimique

— Surveillance des sites.

— Manipulation des matériaux toxiques.

1.4.5 Médecine

— Assistance d'urgence.

— Aide aux handicapés physiques, non voyant.

1.4.6 Espace

— Exploration des planètes.

— Maintenance des satellites.

1.4.7 Industriel

— Convoyage.

— Surveillance.

1.5 Non-holonomie et holonomie des robots

1.5.1 Non-holonomie

Les systèmes mobiles dits non-holonomes sont ceux que nous rencontrons le plus dans notre vie courante (voiture, bus, camion, etc.). Ces systèmes ont une structure mécanique relativement simple (roues motrices, des roues directrices et des roues libres). Une

roue peut avoir une, deux ou trois fonctions. Mais tous ces systèmes ont une caractéristique commune, qui est le sens de la vitesse d'entrainement (vitesse linéaire) donné par la direction des roues directrices.

1.5.2 Holonomie

Les systèmes holonomes sont encore plus rares dans notre quotidien. Ils ont une structure mécanique complexe qui leur permet de se déplacer dans toutes les directions sans intervention humaine.

Dans ce domaine, le terme robot mobile holonome s'applique à un concept abstrait appelé robot ou base qui ne prend pas en compte les corps rigides qui composent le robot et le mécanisme lui-même. Par conséquent, tout robot mobile avec trois degrés de liberté de mouvement dans le plan s'appelle un robot mobile holonome, exemples de robot omnidirectionnel à 3 roues. [4]

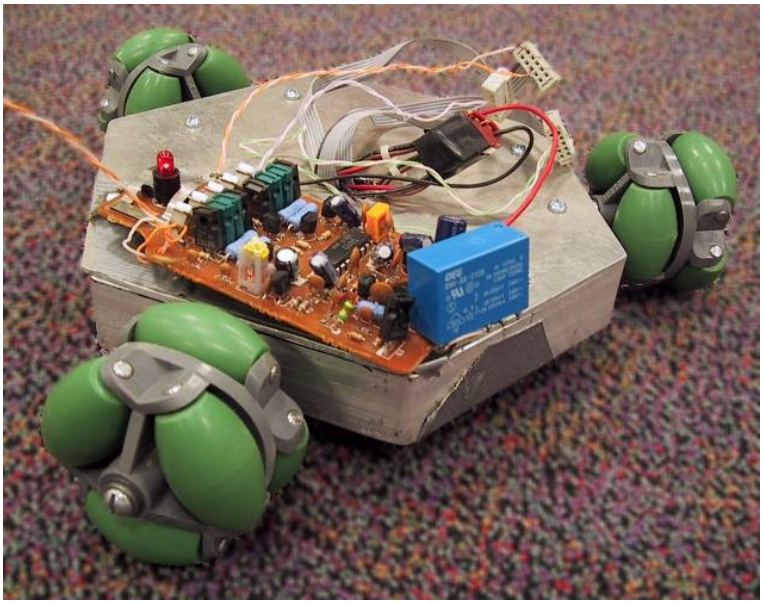

FIGURE 1.1: Robot mobile holonome.

1.6 Environnement dynamique et incertain

1.6.1 Notion d'environnement dynamique

Nous dirons qu'un environnement est dynamique lorsque l'ensemble des positions occupées par les obstacles est possible de changer au cours du temps.

C'est le cas des obstacles qui se déplacent, de ceux qui changent de forme (un piéton tenant un chien en laisse est perçu par le robot comme un seul objet à géométrie variable) ou de ceux qui apparaissent/disparaissent (une porte coulissante semble dans le mur quand elle s'ouvre).

1.6.2 Notion d'incertitude

Nous dirons généralement qu'une information est incertaine, si elle est bruitée (mauvaise conditions de mesures), incomplète (obstruction d'un capteur ou portée limitée, absence d'informations sur l'évolution d'un objet ou d'un phénomène) ou imprécise (les glissements des roues par rapport au sol sont observables mais rarement mesurables). [8]

1.7 Vision pour la navigation de robots mobiles

1.7.1 Introduction

La vision est la sensation qui transmet la plupart des informations au cerveau, qui à son tour nous fournit une grande quantité de données de l'environnement qui nous entoure et nous permet d'effectuer une interaction intelligente avec l'environnement dynamique (évitement d'obstacles mobiles, etc.). Par conséquent, il existe un grand nombre de recherches sur le développement de capteurs qui tentent d'imiter le système visuel humain . Les capteurs visuels que nous utilisons doivent contenir les robots intelligents qui ont les mêmes sensibilités et réponses à la lumière que notre système de vision.[14]

Dans le domaine de la robotique, ces vingt dernières années, des innovations tech-

nologiques dans la fabrication des caméras et dans l'évolution des ordinateurs ont permis d'intégrer des systèmes de vision complexes aux systèmes embarqués, que ce soit sur des robots mobiles de navigation autonome ou sur des véhicules d'aide à la conduite. La vision artificielle est particulièrement importante car elle est un élément essentiel du développement des nouvelles technologies, elle fournit à la machine les capacités nécessaires pour réagir avec son environnement et les représentations à partir des quelles le robot prend des décisions.

1.7.2 Navigation visuelle en milieu intérieur

Depuis les premiers projets sur les robots mobiles, des séquences d'images ont été proposées pour fournir des informations utiles à la navigation des robots. Ils sont usuellement traités par des méthodes de reconnaissance de formes pour détecter des objets dans des images successives dont le modèle (l'apparence de l'objet ou le modèle géométrique de l'objet) est connu. La plupart de ces méthodes utilisent des modèles structurés, le processus de navigation est associé à l'une des manières suivantes :

— Utilisation de cartes connues de l'environnement.

— Construction incrémentale d'une carte au fur et à mesure des d'emplacements.

— Navigation dépourvue de carte.

Au final, les méthodes géométriques se sont avérées très adaptées aux environnements intérieurs, et des modèles mathématiques formels sont souvent proposés dans la littérature. Ces modèles incluent implicitement les actions d'évitement d'obstacles, de détection de points de repère, de construction ou de mise à jour de cartes et d'estimation de la position du robot.[5]

1.7.3 Navigation visuelle en milieu extérieur

Pour les milieux naturels extérieurs, la construction de la carte est beaucoup plus complexe (structuration de bas niveau : extraction de géométrie plus complexe), notam-

ment lorsque la scène change dynamiquement (en raison de la météo, de la saison, des conditions d'éclairage, etc.) ou lorsque d'autres dynamiques les agents partagent le même environnement (piétons, autres robots, véhicules, etc.).

1.8 Différence entre un AGV et un AMR

L'automatisation de la logistique interne est une problématique récurrente pour les grandes comme pour les petites entreprises. Pourquoi utiliser les ressources des employés pour déplacer les matériaux alors que nous pouvons automatiser ces tâches et que les employés se concentre sur des activités à plus forte valeur ajoutée ? En automatisant le transport des matériaux, les organisations peuvent optimiser la productivité et planifier les livraisons plus efficacement afin de réduire les difficultés de la production.

À ce jour, les véhicules guidés automatisés (AGV) traditionnels étaient la seule option pour automatiser les tâches de transport interne. Les AGV sont un appareil familier dans les grandes installations fixes où il y a un besoin de livraisons de matériel répétitives et cohérentes, où des dépenses initiales élevées et un long retour sur investissement (ROI) peuvent être tolérés. Cependant, les AGV sont concurrencés par la technologie plus sophistiquée, flexible et rentable des robots mobiles autonomes (AMR). Alors que les AGV et les AMR déplacent tous les deux des matériaux d'un endroit à un autre, c'est là que les similitudes s'arrêtent.

1.8.1 Type de navigation

Un AGV a une intelligence embarquée minimale et ne peut obéir qu'à des instructions de programmation simples. Pour naviguer, il doit être guidé par des fils, des bandes magnétiques ou des capteurs, qui nécessitent généralement des mises à jour importantes (et coûteuses) des installations à installer, pendant lesquelles la production peut être interrompue. L'AGV est limité à ces itinéraires fixes, qui nécessitent des coûts supplémentaires et des perturbations si des changements sont nécessaires à l'avenir. L'AGV peut détec-

ter les obstacles devant lui, mais il n'est pas capable de les contourner, il s'arrête donc simplement sur sa trajectoire jusqu'à ce que l'obstacle soit supprimé.

En revanche, l'AMR navigue via des cartes que son logiciel construit sur site ou via des dessins d'installation préchargés. Cette capacité peut être comparée à une voiture avec un GPS et un ensemble de cartes préchargées. Lorsqu'il apprend l'adresse du domicile et du lieu de travail du propriétaire, il génère le chemin le plus direct en fonction de positions simples sur la carte. Ceci est similaire à la façon dont l'AMR apprend les emplacements pour ramasser et déposer des pièces. L'AMR utilise les données des caméras et des capteurs intégrés et des scanners laser ainsi que des logiciels sophistiqués qui lui permettent de détecter son environnement et de choisir l'itinéraire le plus efficace vers la cible d'une manière intelligente. Il fonctionne de manière complètement autonome et si des chariots élévateurs, des palettes, des personnes ou d'autres obstacles se présentent devant lui, l'AMR manœuvrera en toute sécurité autour d'eux, en utilisant le meilleur itinéraire alternatif. Cela optimise la productivité en garantissant que le flux de matériaux respecte le calendrier. [11]

1.8.2 Réutilisabilité dans les différentes applications

Ce fonctionnement autonome rend également un AMR beaucoup plus flexible qu'un AGV. Les AGV sont limités à suivre un itinéraire strict intégré à l'installation - généralement installé dans le sol. Cela signifie que les applications sont limitées et qu'un AGV effectue la même tâche de livraison tout au long de sa durée de vie. Les changements sont tout simplement trop coûteux et perturbateurs pour être rentables.

Cependant, l'AMR n'a besoin que de simples ajustements logiciels pour changer ses missions, de sorte que le même robot peut effectuer une variété de tâches différentes à différents endroits, en effectuant automatiquement des ajustements pour répondre à l'évolution des environnements et des exigences de production. Les tâches AMR peuvent

être contrôlées via l'interface du robot ou configurées par un logiciel de contrôle de flotte pour plusieurs robots qui hiérarchise automatiquement les commandes et le robot le mieux adapté à une tâche donnée en fonction de la position et de la disponibilité. Une fois qu'une mission est établie, les employés n'ont pas à passer du temps à coordonner le travail des robots, ce qui leur permet de se concentrer sur un travail de grande valeur qui contribue au succès de l'entreprise.

1.8.3 Modèles de commercialisation

La flexibilité des AMR est cruciale pour les environnements de fabrication modernes qui nécessitent agilité et flexibilité s'il est nécessaire de modifier les produits ou la chaîne de production. Les AMR sont hautement adaptables pour une production agile dans des installations de toutes tailles. Si les cellules de production sont déplacées ou si de nouvelles cellules ou de nouveaux processus sont ajoutés, une nouvelle carte du bâtiment peut être téléchargée rapidement et facilement où l'AMR peut re-cartographier sur place, de sorte qu'elle peut être utilisée immédiatement pour de nouvelles tâches. Cette capacité donne aux organisations la pleine propriété du robot et de ses fonctions. Plutôt que d'être contraints par une infrastructure AGV inflexible, les propriétaires peuvent facilement redéployer le robot eux-mêmes à mesure que leurs besoins commerciaux évoluent pour les aider à optimiser la production, même dans des environnements hautement dynamiques.

1.8.4 Coût d'exploitation

Bien qu'un AMR soit constitué d'une technologie beaucoup plus avancée qu'un AGV, il s'agit généralement d'une solution moins coûteuse. Un AMR n'a pas besoin de fils, de bandes magnétiques ou d'autres modifications coûteuses de l'infrastructure du bâtiment, il est donc plus rapide et moins coûteux de mettre en place et de faire fonctionner les AMR, et sans interruption coûteuse de la production dans le processus. Vu que les AMR peuvent être déployés rapidement et facilement, ils ajoutent de nouvelles efficacités presque immédiatement. Avec des coûts initiaux faibles et une optimisation rapide des processus,

ils offrent un retour sur investissement remarquablement rapide, souvent en moins de six mois. Au fur et à mesure que les entreprises se développent, la mise en œuvre des AMR peut se développer simultanément avec des coûts supplémentaires minimes.

Les environnements de fabrication modernes ne peuvent plus dépendre de technologies héritées coûteuses et rigides. Ils ne peuvent pas non plus se permettre de continuer le transport manuel improductif de matériaux, en particulier sur le marché du travail actuel. Les robots mobiles autonomes sont supérieurs aux AGV en termes de flexibilité, de rentabilité, de retour sur investissement et d'optimisation de la productivité.

1.9 Similitudes entre les AGV et les AMR

— Les deux peuvent être utilisés pour déplacer des matériaux d'un point à un autre dans une installation.

— Évitez les collisions avec des objets qui pourraient se trouver sur leur chemin.

— Peuvent fonctionner en collaboration et en toute sécurité avec les humains dans leur espace de travail (ou les chemins prévus).

— Transporter des marchandises entre 2 kg et 1500 kg.

1.10 Robot à pattes

Un robot à pattes est un robot mobile qui utilise des membres mécaniques pour se déplacer. Ils sont plus polyvalents que les robots à roues et peuvent traverser de nombreux terrains différents, bien que ces avantages nécessitent une complexité et une consommation d'énergie accrues. Dans un exemple de biomimétisme, les robots à pattes imitent souvent des animaux à pattes, comme les humains ou les insectes.

1.10.1 Les différents types

Les robots à pattes peuvent être classés en fonction du nombre de membres qu'ils utilisent, ce qui détermine les allures disponibles. Les robots à plusieurs pieds ont tendance

à être plus stables, tandis que moins de jambes aident à améliorer la maniabilité. Citons ses différents types :

1.10.1.1 Unijambiste

Les robots à une jambe ou à pogo stick utilisent des mouvements de saut pour la navigation. En 1980, l'Université Carnegie Mellon a développé un robot unijambiste pour étudier l'équilibre. La figure 1.2 représente le robot Berkeley SALTO.

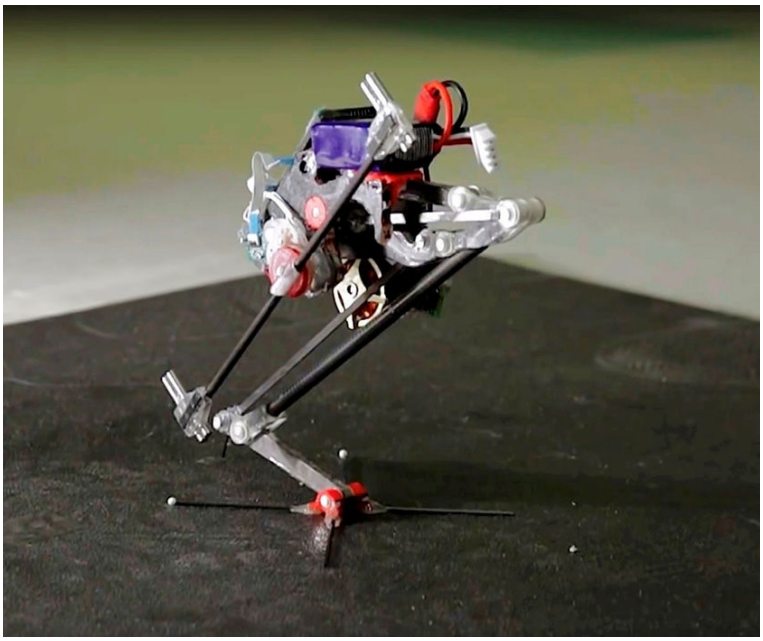

FIGURE 1.2: Robot Unijambiste.

1.10.1.2 À deux pattes

Les robots à deux pattes présentent un mouvement bipède. Ils sont donc confrontés à deux problèmes principaux :

Contrôle de stabilité, qui fait référence à l'équilibre du robot, et contrôle de mouvement, qui fait référence à la mobilité du robot.

Le contrôle de la stabilité est particulièrement difficile pour un système bipède, qui doit être équilibré dans les deux sens, même au repos. Certains robots, en particulier les

jouets, résolvent ce problème avec de grands pieds, offrant une plus grande stabilité tout en réduisant la mobilité. Alternativement, des systèmes plus avancés utilisent des capteurs tels que des accéléromètres ou des gyroscopes pour fournir une rétroaction dynamique d'une manière qui se rapproche de l'équilibre humain. Ces capteurs sont également utilisés pour le contrôle de mouvement et la marche. La complexité de ces tâches facilite l'apprentissage automatique.

Le mouvement bipède simple peut être approximé en faisant rouler un polygone, où la longueur de chaque côté correspond à la longueur d'un pas simple. Lorsque le pas diminue et que le nombre de côtés augmente, le mouvement se rapproche d'un cercle. Cela relie le mouvement bipède au mouvement à roues en tant que limiteur de foulée (voir 1.3).

FIGURE 1.3: Robot à deux jambes.

1.10.1.3 À quatre pattes

Les robots à quatre pattes présentent un mouvement quadrupède (voir 1.4). Par rapport aux robots bipèdes, ils bénéficient d'une plus grande stabilité, notamment lors des déplacements. À basse vitesse, le robot quadrupède ne peut bouger qu'une jambe à la fois, assurant la stabilité du trépied. Les robots quadrupèdes bénéficient également d'un centre de gravité plus bas que les systèmes bipèdes.

Les robots à quatre pattes comprennent :

— La série TITAN, développée depuis 1980 par le laboratoire Hirose-Yoneda.

— Le BigDog, dynamique et stable, développé en 2005 par Boston Dynamics, le Jet Propulsion Laboratory de la NASA, et la station de recherche de l'Université Harvard de Concord.

— Le successeur de BigDog, LS3.

FIGURE 1.4: Robot à quatre pattes.

1.10.1.4 À six pattes

La force motrice des robots à six pattes ou hexapodes est plus stable que celle d'un bipède ou d'un quadrupède (voir 1.5). Leurs conceptions finales imitent souvent la mécanique des insectes et leurs démarches peuvent être classées de la même manière. Ceux-ci inclus :

Marche de la vague : La démarche la plus lente, où les jambes se déplacent d'arrière en avant dans une "vague".

Marche sur trépied : Un pas légèrement plus rapide dans lequel les trois jambes bougent en même temps. Les trois pieds restants fournissent un trépied stable pour le robot.

Les robots hexapodes comprennent :

— L'Odex, un hexapode de 375 livres développé par Odetics en 1980, se distinguait par son ordinateur de bord qui contrôlait chaque jambe.

— Gengis était l'un des premiers robots autonomes à six pattes, développé par Rodney Brooks au MIT en 1980.

— Une ligne de jouets moderne, Hexbug.

FIGURE 1.5: Robot à six pattes.

1.10.1.5 À huit pattes

Le robot à huit pattes a été inspiré par les araignées et autres arachnides, ainsi que par certains marcheurs sous-marins. Ils offrent la plus grande stabilité à ce jour, ce qui a conduit à un certain succès précoce avec les robots à pattes (voir 1.6).

Les robots à huit pattes comprennent :

— Dante, Projet d'exploration du mont Erebus de l'Université Carnegie Mellon.

— T8X, un robot disponible dans le commerce conçu pour imiter l'apparence et les mouvements d'une araignée.

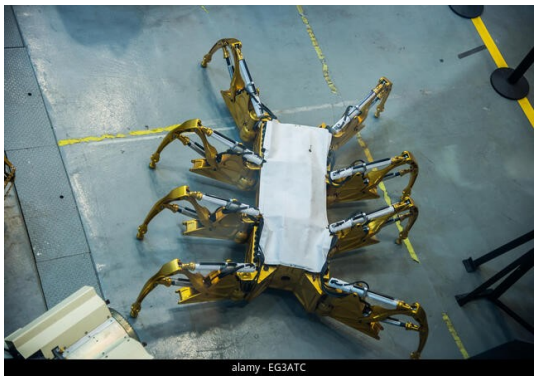

FIGURE 1.6: Robot à huit pattes.

1.10.1.6 Hybride

Certains robots utilisent une combinaison de jambes et de roues. Cela donne à la machine la vitesse et l'efficacité énergétique du mouvement des roues, et la maniabilité de la navigation avec les jambes. Poignée de Boston Dynamics, un robot bipède à roues avec deux jambes, en est un exemple.

1.10.2 Comportement de la marche

1.10.2.1 Marche statique

La marche statique se produit lorsque le centre de gravité du robot est toujours plus haut que ses pieds, de sorte qu'il ne bascule pas sans qu'une force externe agisse sur lui.

1.10.2.2 Marche et course dynamiques

La marche dynamique se produit lorsque le centre de gravité du robot peut également se trouver en dehors de la zone des pieds sans que le robot ne tombe. En fait, on peut parler de "chute contrôlée" car le robot tombera lorsque son mouvement s'arrêtera brusquement. Elle se produit aussi lorsque le mouvement requis pour maintenir la vitesse fait en sorte qu'aucune des jambes du robot ne touche le sol.

1.11 Conclusion

Ce chapitre passe en revue les concepts de base de la robotique mobile et présente certains types d'applications. Ainsi on a concentré sur l'environnement d'un robot mobile et ses différents milieu navigation visuelle, ensuite nous allons voir quel que types des robots à pattes et en fin finalisée ce chapitre par une petite comparaison entre un AGV et un AMR. Il y a donc perception d'un côté et commandement de l'autre. Deux axes de recherche majeurs ont ainsi permis d'obtenir des robots mobiles totalement autonomes. Parmi les problèmes liés à la commande, le problème de navigation joue un rôle important, qui consiste à déterminer la trajectoire à suivre par le robot pour évoluer correctement dans un environnement plein d'obstacles.

Chapitre 2

Concepts de Navigation ROS

2.1 Introduction

Ce chapitre vise à présenter les principaux concepts théoriques et à expliquer l'utilisation de ROS Navigation Stack. Il s'agit d'une boîte à outils puissante pour la planification de trajectoires, la localisation et la cartographie simultanées (SLAM : Simultaneous Localization And Mapping) mais son application n'est pas triviale en raison du manque de compréhension des concepts associés. Ce chapitre présentera la théorie à l'intérieur de cette pile et expliquera de manière simple comment effectuer le SLAM dans n'importe quel robot. Nous présenterons les prérequis qui nous permettrons de régler l'odométrie, d'établir des référentiels et ses transformations, de configurer les capteurs de perception, de régler les contrôleurs de navigation et de planifier le chemin sur nos propres robots virtuels ou réels.

2.2 Arrière plan(Background)

ROS dispose d'un ensemble de ressources utiles pour qu'un robot soit capable de naviguer dans un environnement connu, partiellement connu ou inconnu, en d'autres termes, le robot est capable de planifier et de suivre un chemin lorsqu'il s'écarte des obstacles qui apparaissent sur son chemin tout au long du parcours. Ces ressources se trouvent sur la pile de navigation.

L'une des nombreuses ressources nécessaires pour accomplir cette tâche et qui est présente sur la pile de navigation sont les systèmes SLAM (également appelés systèmes de localisation), qui permettent à un robot de se localiser, qu'il y ait une carte statique disponible ou que le SLAM soit requis. Le AMCL est un outil qui permet au robot de se repérer dans un environnement grâce à une carte statique, une carte préalablement créée. Toute la zone dans laquelle le robot pourrait naviguer devrait être cartographiée de manière métriquement correcte pour utiliser des cartes statiques, ce qui présente un gros inconvénient de cette ressource. Selon l'environnement qui entoure le robot, ces cartes statiques sont capables d'augmenter ou de diminuer la confiance des systèmes de localisation.

Pour contourner le manque de flexibilité des cartes statiques, deux autres systèmes de localisation sont proposés par ROS pour fonctionner avec la Pile de Navigation : Gmapping et hector_mapping.

Gmapping et hector_mapping sont des implémentations de SLAM, une technique qui consiste à cartographier un environnement en même temps que le robot se déplace, en d'autres termes, pendant que le robot navigue dans un environnement, il recueille des informations de l'environnement à travers ses capteurs et génère une carte. De cette façon, nous disposons d'une base mobile capable non seulement de générer une carte d'un environnement inconnu ainsi que de mettre à jour la carte existante, permettant ainsi l'utilisation de l'appareil dans des environnements plus génériques, non à l'abri des changements.

La différence entre Gmapping et hector_mapping est que le premier prend en compte les informations d'odométrie pour générer et mettre à jour la carte et la position du robot. Cependant, le robot doit disposer de capteurs proprioceptifs, ce qui rend son utilisation difficile pour certains robots (par exemple, les robots volants). Les informations d'odométrie sont intéressantes car elles peuvent aider à la génération de cartes plus précises, car en comprenant la cinématique du robot, nous pouvons estimer sa position.

La cinématique est essentiellement influencée par la façon dont les dispositifs qui garantissent le mouvement du robot sont assemblés. Quelques exemples de caractéristiques mécaniques qui influencent la cinématique sont : le type de roue, le nombre de roues, le positionnement de la roue et l'angle auquel elles sont disposées.

Cependant, aussi utile que puisse être l'information d'odométrie, elle n'est pas à l'abri des défauts. Les défauts sont causés par le manque de précision sur la capture des données, le frottement, le glissement, la dérive et d'autres facteurs. Ces facteurs accumulés peuvent conduire à des données incohérentes et nuire à la formation des cartes, qui ont tendance à être déformées dans ces circonstances.

D'autres données indispensables pour générer une carte sont les relevés de distance

des capteurs, car ils sont responsables de la détection du monde extérieur et, de cette façon, servent de référence au robot. Néanmoins, les données recueillies par les capteurs doivent être ajustées avant d'être utilisées par l'appareil. Ces ajustements sont nécessaires car les capteurs mesurent l'environnement par rapport à eux-mêmes et non par rapport au robot, autrement dit, une conversion géométrique est nécessaire. Pour simplifier cette conversion, ROS propose l'outil tf qui permet d'ajuster les positions des capteurs par rapport au robot et ainsi d'adapter les mesures à la navigation du robot.

2.3 ROS Environment

Avant de commencer à configurer l'environnement dans lequel nous allons travailler, il est très important d'être conscient des limites de la pile de navigation, afin que nous soyons en mesure d'adapter notre matériel. Il existe quatre limitations principales concernant le matériel :

— La pile a été construite dans le but de ne traiter que les robots à entraînement différentiel et holonomiques, bien qu'il soit possible d'utiliser certaines fonctionnalités avec d'autres types de robots, qui ne seront pas abordés ici.

— La pile de navigation suppose que le robot reçoit un message de type torsion avec des vitesses X, Y et thêta et capable de contrôler la base mobile pour atteindre ces vitesses. Si notre robot n'est pas en mesure de le faire, nous pouvons adapter notre matériel ou simplement créer un nœud ROS qui convertit le message de torsion fourni par la pile de navigation au type de message qui correspond le mieux à nos besoins.[**ros_org**]

— Les informations sur l'environnement sont recueillies à partir d'une rubrique de type de message LaserScan. De plus, il est possible d'utiliser d'autres capteurs, à condition de pouvoir convertir leurs données au type LaserScan.

— La pile de navigation fonctionnera mieux avec des robots carrés ou circulaires, alors qu'il est possible de l'utiliser avec des formes et des tailles arbitraires. Des tailles

et des formes uniques peuvent causer des problèmes au robot dans des espaces restreints.

2.3.1 Configuration du capteur Kinect

Le capteur Kinect de Microsoft est un équipement multi-capteurs, équipé d'un capteur de profondeur, d'une caméra RGB et de microphones. Ce dernier est intéressant dans le cadre de la robotique d'intérieur car il fournit une description 3D de l'environnement à très faible coût. Mais si on le compare à un télémètre laser avec une plage de détection horizontale supérieure à 200°, le Kinect a un champ de vision très limité (environ 57°). De même, la portée du Kinect est limitée à 500 centimètres, alors que certains télémètres laser sont capables de fournir des données jusqu'à des dizaines de mètres. Par conséquent, il serait intéressant de combiner les données de Kinect avec les données des télémètres laser 2D dans la plupart des applications de robotique mobile d'intérieur. [2]

2.3.2 Capteur laser Sick S300

Le S300 est un système de mesure laser compact qui scanne l'environnement en deux dimensions à une hauteur de 150 mm au-dessus du sol au moyen de faisceaux laser infrarouges. Chaque laser a une portée de balayage de 270°, couvrant un côté long et un côté court du véhicule, et une résolution de 0,5°.

La fonction principale des capteurs S300 est de fonctionner comme équipement de sécurité du système en surveillant des zones prédéfinies autour du véhicule. Par défaut, les capteurs S300 sont configurés avec un champ de protection et un champ d'avertissement. La taille des champs surveillés dépend de la vitesse du véhicule. La conséquence d'une violation des deux champs varie selon le mode de fonctionnement. Généralement, une brèche dans un champ provoque une réduction de la vitesse maximale de déplacement ou déclenche un arrêt de sécurité du véhicule. Les scrutateurs laser ne sont pas actifs pour des vitesses inférieures à 0,13 m/s dans les modes de fonctionnement manuels. [6]

2.3.3 Transformations

Les systèmes robotiques ont souvent besoin de suivre les relations spatiales pour diverses raisons : entre un robot mobile et un cadre de référence fixe pour la localisation, entre les différents cadres de capteurs et de manipulateurs, ou pour placer des cadres sur des objets cibles à des fins de contrôle.Les transformations sont une nécessité pour la pile de navigation pour comprendre où se trouvent les capteurs par rapport au centre du robot (base_link).

Pour simplifier et unifier le traitement des cadres spatiaux, un système de transformation a été écrit pour ROS, appelé tf. Le système tf construit un arbre de transformation dynamique qui relie tous les référentiels du système. Au fur et à mesure que les informations proviennent des différents sous-systèmes du robot (encodeurs conjoints, algorithmes de localisation, etc.), le système tf peut produire des flux de transformations entre les nœuds de l'arbre en construisant un chemin entre les nœuds souhaités et en effectuant les calculs nécessaires. [10]

2.3.4 Création d'un paquet

A ce stade, Nous avons déjà quelques fichiers XML pour lancer des nœuds contenant l'initialisation de nos capteurs et pour initialiser certains packages liés à notre plateforme (mise sous tension du robot, lecture d'odométrie, etc.). Pour organiser ces fichiers et faciliter la recherche de nos lanceurs avec les commandes ROS, nous avons créé un package contenant tous nos lanceurs. Pour ce faire, accédons à notre dossier src, dans notre espace de travail catkin, et créons un package avec les commandes suivantes (commandes valables pour les versions catkinisées de ROS) :

1. $ cd /home/user/catkin_ws/src
2. $ catkin_create_pkg packageName std_msgs rospy roscpp move_base_msgs

Ces deux commandes suffisent pour créer un dossier contenant tous les fichiers qui permettront à ROS de trouver le paquet par son nom. Copions tous nos fichiers de

lancement dans le nouveau dossier, homonyme de notre package, puis compilons le package en accédant au dossier de notre espace de travail et en exécutant la commande compile :

1. $ cd /home/user/catkin_ws.
2. $ catkin_make

C'est tout ce que nous devons faire. À partir de maintenant, pensons à mettre nos fichiers de lancement et de configuration dans ce dossier. Nous pouvons également créé un dossier, tel que launch et config, à condition de spécifier ces sous-chemins lors de l'utilisation de trouver pour inclure des lanceurs dans d'autres lanceurs.

2.3.5 La pile de navigation(The Navigation Stack)

Maintenant, Nous commencons à étudier les concepts de base de la pile de navigation, ainsi que leur utilisation. Étant donné la vue d'ensemble du système, qui est faite à la Figure 2.1. Les éléments seront analysés dans les sections suivantes bloc par bloc. Nous pouvons voir sur la figure 2.1 qu'il existe trois types de nœuds : les nœuds fournis, les nœuds fournis facultatifs et les nœuds spécifiques à la plate-forme.

— Les nœuds à l'intérieur de la boîte, nœuds fournis, sont au cœur de la pile de navigation et sont principalement responsables de la gestion des cartes de coûts et des fonctionnalités de planification de chemin.

— Les nœuds facultatifs fournis, amcl et map_server, sont liés aux fonctions de carte statique, et puisque l'utilisation d'une carte statique est facultative, l'utilisation de ces nœuds est également facultative.

— Les nœuds spécifiques à la plate−forme sont les nœuds liés à notre robot, tels que les nœuds de lecture de capteur et les nœuds de contrôleur de base.

De plus, nous avons les systèmes de localisation, non représentés sur la figure 2.1. Si la source d'odométrie était parfaite et qu'aucune erreur n'était présente sur les données d'odométrie, nous n'aurions pas besoin de systèmes de localisation. Cependant, dans les applications réelles, ce n'est pas le cas, et nous devons prendre en compte d'autres types de

données, telles que les données IMU (Inertial Measurement Unit), afin de pouvoir corriger les erreurs d'odométrie. Les systèmes de localisation abordés dans ce chapitre sont : amcl, gmapping et hector_mapping. Ci−dessous, un tableau est disponible pour relier un nœud à un mot−clé, afin que vous puissiez mieux comprendre la relation des nœuds de la pile de navigation.

Localisation	Interaction avec l'environnement	Cartes statiques	Trajectory planning	Cartographie
amcl	sensor_sources	amcl	global_planner	local_costmap
gmapping	base_controller	map_server	local_planner	global_costmap
hector_mapping	odometry_source		recovery_behaviors	

TABLE 2.1: Les mots-clés de différents nœuds

2.3.5.1 Amcl et Map_server

Les deux premiers blocs sur lesquels nous pouvons nous concentrer sont les blocs facultatifs, responsables de l'utilisation de la carte statique : amcl et map_server. Map_server contient deux nœuds : map_server et map_saver. Le premier, homonyme du package, comme son nom l'indique, est un nœud ROS qui fournit des données cartographiques statiques en tant que service ROS, tandis que le second, map_saver, enregistre une carte générée dynamiquement dans un fichier. amcl ne gère pas les cartes, c'est en fait un système de localisation qui s'exécute sur une carte connue. Il utilise la transformation base_footprint ou base_link vers la carte pour fonctionner, il a donc besoin d'une carte statique et il ne fonctionnera qu'après la création d'une carte. Ce système de localisation est basé sur l'approche de localisation de Monte Carlo : il distribue aléatoirement des particules dans une carte connue, représentant les emplacements possibles du robot, puis nous utilisons un filtre à particules pour déterminer la position réelle du robot.

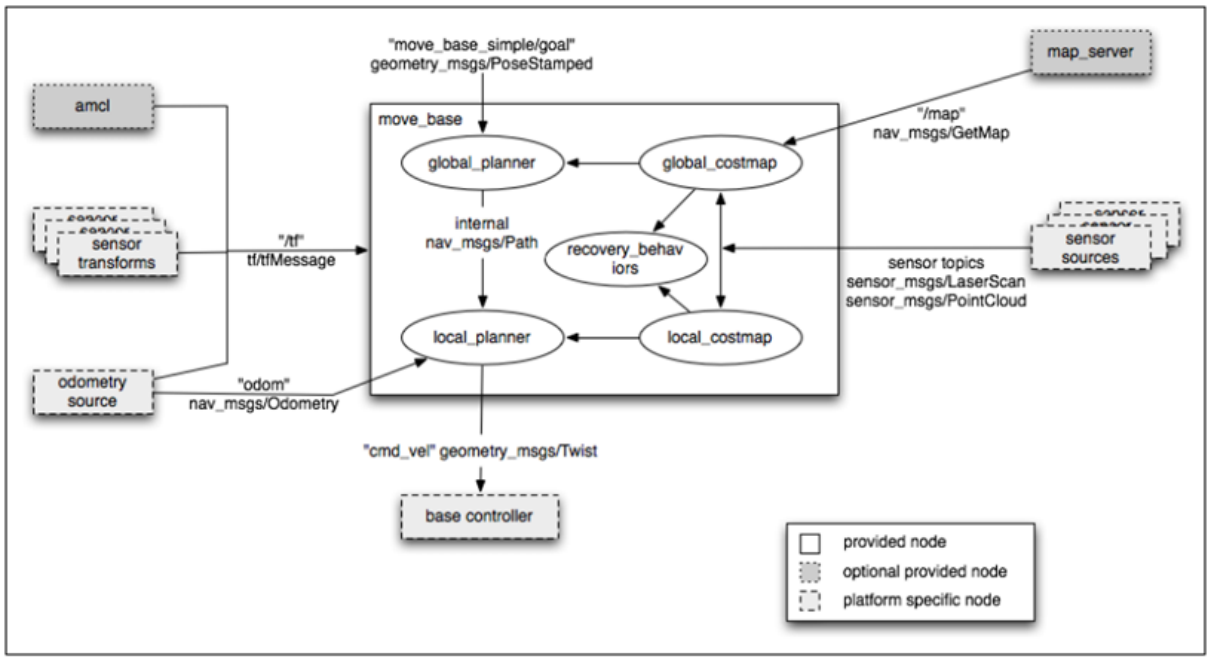

FIGURE 2.1: Vue d'ensemble d'un système typique exécutant la pile de navigation [16]

2.3.5.2 Gmapping

Gmapping, ainsi qu'amcl, est un système de localisation, mais contrairement à amcl, il s'exécute dans un environnement inconnu, effectuant une localisation et une cartographie simultanées (SLAM). Il crée une carte de grille d'occupation 2D à l'aide de la position du robot et des données laser (ou des données converties, c'est-à-dire des données Kinect). Il fonctionne sur la transformation odométrique en carte, il n'a donc pas besoin de la carte ni des informations IMU, il n'a besoin que de l'odométrie.

2.3.5.3 Hector_mapping

Hector_mapping peut être utilisé à la place de gmapping. Il utilise le base_link pour mapper la transformation et il n'a pas besoin de l'odométrie ni de la carte statique, il utilise simplement le balayage laser et les informations de l'IMU pour se localiser.

2.3.5.4 Capteurs et Contrôleur

Ces blocs de la vue d'ensemble du système concernent l'interaction matériel-logiciel et, comme indiqué, sont des nœuds spécifiques à la plate-forme. La source de l'odométrie et les blocs du contrôleur de base sont spécifiques au robot que nous utilisons, car le premier est généralement publié à l'aide des données des encodeurs de roue et le second est chargé de prendre les données de vitesse du sujet cmd_vel et de s'assurer que la reproduction du robot produit ces vitesses. Il est à noter que l'ensemble du système ne fonctionnera pas si les transformées du capteur ne sont pas disponibles, car elles seront utilisées pour calculer la position des lectures du capteur sur l'environnement.

2.3.5.5 Cartes de coûts locales et globales

Les costmaps 2D locales et globales sont les sujets contenant les informations qui représentent la projection des obstacles dans un plan 2D (le sol), ainsi qu'un rayon d'inflation de sécurité, une zone autour des obstacles qui garantit que le robot n'entrera pas en collision avec tout objet, quelle que soit son orientation. Ces projections sont associées à un coût, et l'objectif du robot est d'atteindre l'objectif de navigation en créant un chemin au moindre coût possible. Alors que la carte des coûts globale représente l'ensemble de l'environnement (ou une grande partie de celui-ci), la carte des coûts locale est, en général, une fenêtre de défilement qui se déplace dans la carte des coûts globale par rapport à la position actuelle du robot.

2.3.5.6 Planificateurs locaux et globaux

Les planificateurs locaux et globaux ne fonctionnent pas de la même manière. Le planificateur global prend la position actuelle du robot et l'objectif et tracer la trajectoire de moindre coût par rapport à la carte de coût globale. Cependant, le planificateur local a une tâche plus intéressante, il travaille sur la carte des coûts locale et, comme la carte des coûts locale est plus petite, elle a généralement plus de définition donc elle est capable de détecter plus d'obstacles que la carte des coûts globale. Ainsi, le planificateur local

est responsable de la création d'un déploiement de trajectoire sur la trajectoire globale, capable de revenir à la trajectoire d'origine avec le moindre coût alors qu'il s'écarte des obstacles nouvellement insérés ou des obstacles que la définition de la carte de coût globale n'a pas été en mesure de détecter. Juste pour être clair, move_base est un package qui contient les planificateurs locaux et globaux et est chargé de les relier pour atteindre l'objectif de navigation.

2.3.6 Cartes de coûts en couches (Layered Costmaps)

Les informations sur l'environnement utilisées par les planificateurs de chemin sont stockées dans une carte des coûts. La structure de données de la carte des coûts en couches contient une grille de coûts à deux dimensions qui est utilisée pour la planification du chemin. La carte des coûts en couches maintient une liste ordonnée de couches, dont chacune suit les données liées à une fonctionnalité spécifique. Les données de chacune des couches sont ensuite accumulées dans la carte principale des coûts, qui effectue deux passages dans la liste ordonnée des couches. [9]

2.4 Conclusion

Naviguer dans des environnements inconnus est une tâche très difficile, mais l'utilisation de la pile de navigation peut nous donner d'excellents résultats. Cet ensemble d'outils peut nous aider à obtenir d'excellents résultats, permettant au robot de créer une carte générée dynamiquement et d'atteindre des objectifs sans plantage.

Chapitre 3

ROS Navigation : Simulation

3.1 Introduction

L'objectif de ce chapitre est de donner les outils et les connaissances de base pour être en mesure de comprendre et de créer tout projet de base lié à la navigation ROS. Nous pourrons créer des cartes d'environnements, localiser le robot dans l'environnement, faire effectuer aux robots une planification de trajectoire, visualiser les données des différents processus de navigation et déboguer les erreurs à l'aide de la plate-forme ROSDS (ROS Development Studio), configurer les différents nœuds de navigation,etc.

Nous apprendrons par la suite, les principaux concepts ROS qui sont au cœur de la navigation ROS. Ce sont les concepts les plus importants que nous devons maîtriser.

Partie 1 : Fonctionnement de la structure de base de la navigation ROS.

Partie 2 : La création d'une carte d'environnement.

Partie 3 : La localisation d'un robot dans un environnement.

Partie 4 : La planification d'une trajectoire dans cet environnement.

Partie 5 : L'exécution d'une trajectoire et l'évitement des obstacles.

3.2 ROSDS (ROS Development Studio)

ROSDS est l'outil Web de construction permettant de programmer des robots ROS en ligne. Il nécessite aucune installation sur notre ordinateur. Par conséquent, nous pouvons utiliser n'importe quel système d'exploitation pour travailler dessus (Windows, Linux ou Mac). De plus, des comptes gratuits sont disponibles. Nous pouvons créer un compte ROSDS gratuit ici : http ://rosds.online

Nous pouvons utiliser n'importe quels ROSjects disponibles afin d'appliquer tout ce que nous avons appris. Il suffit de coller le lien ROSject dans l'URL de notre navigateur et la simulation sera automatiquement préparée dans notre espace de travail ROSDS. [15]

3.3 Partie 1 : Fonctionnement de la structure de base de la navigation ROS.

Pour effectuer la navigation du robot avec ROS nous avons besoin tout d'abord d'une carte d'environnement dans laquelle nous souhaitons que notre robot navigue.

Une carte n'est qu'une représentation d'un environnement créé à partir des lectures des capteurs du robot (par exemple, du laser). Ainsi, en déplaçant le robot dans l'environnement, nous pouvons créer une carte adéquate. En termes de navigation ROS, cela s'appelle le mappage.

Maintenant, nous avons besoin d'une carte pour naviguer de manière autonome avec notre robot, mais ce n'est pas suffisant. Pour effectuer une navigation correcte, notre robot doit savoir dans quelle position de la carte il se trouve et avec quelle orientation (c'est-à-dire dans quelle direction le robot fait face) à chaque instant. En termes de navigation ROS, cela s'appelle la localisation.

Dans la prochaine étape nous commençons à naviguer de manière autonome, Pour cela, nous avons besoin d'une sorte de système qui indique au robot où aller, au début, et comment y aller. Dans ROS, nous appelons ce système la planification de trajectoire (Path Planning), cette dernière prend essentiellement en entrée l'emplacement actuel du robot et la position où le robot veut aller, et nous donne en sortie le meilleur chemin et le plus rapide pour atteindre ce point.

Enfin, nous devons éviter les obstacles déjà prévus dans l'environnement ou ceux qui aparaissent soudainement dans la trajectoire du robot.

Fondamentalement, le système d'évitement d'obstacles divise la grande image (carte) en plus petits morceaux, qui se mettent à jour en temps réel en utilisant les données qu'il obtient des capteurs. De cette façon, il s'assure qu'il ne sera pas surpris par tout changement soudain dans l'environnement, ou par tout obstacle qui apparaît sur le chemin.

La figure 3.1 illustre le lien entre les trois parties de la navigation.

Dans les parties suivantes nous allons donner plus de détails sur la navigation ROS. [**giorgio_grisetti**]

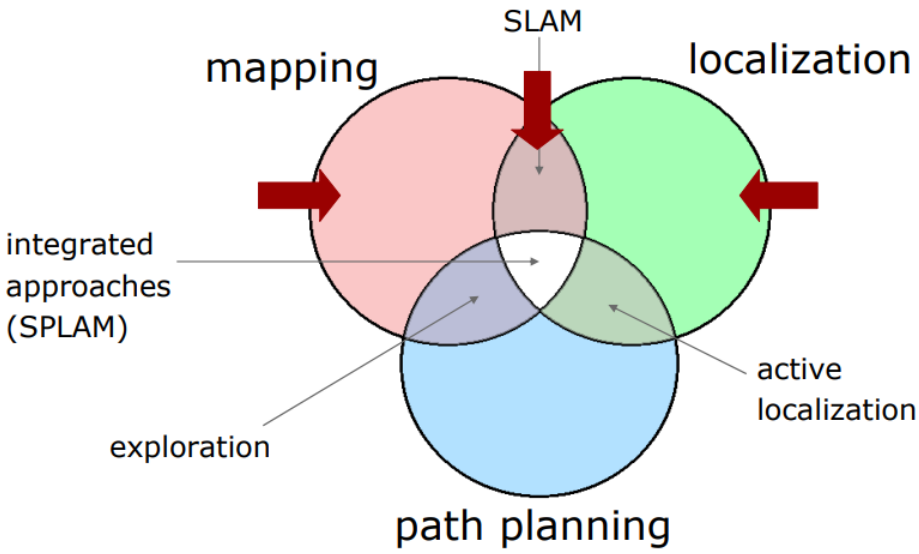

FIGURE 3.1: Le lien entre les trois parties de la navigation.

3.4 Partie 2 : La création d'une carte d'environnement.

Dans cette partie, nous allons voir comment nous pouvons créer une carte à partir de zéro. Avant de commencer nous allons présenter le logiciel Rviz car c'est un outil très important dans le processus de navigation mondiale, il nous permet de visualiser certains paramètres tels que le processus de cartographie, comment la carte est construite, à quoi ressemble la carte et les données aquises par les capteurs laser, etc.

3.4.1 Logiciel Rviz

Rviz, abréviation de visualisation ROS, c'est un puissant outil de visualisation 3D pour ROS. Il permet à l'utilisateur de visualiser le modèle de robot, d'afficher et/ou d'enregistrer les informations des capteurs du robot et de relire les informations enregistrées des capteurs. En visualisant ce que le robot voit, pense et fait, l'utilisateur peut déboguer une application robotique depuis les entrées de capteur jusqu'aux actions planifiées (ou

non planifiées).

Rviz affiche les données des capteurs 3D des caméras stéréo, des lasers des kinects et d'autres appareils 3D sous la forme de nuages de points ou d'images de profondeur. Les données de capteur 2D des webcams, des caméras RGB et des télémètres laser 2D peuvent être visualisées par Rviz sous forme d'image.

Si un robot réel communique avec un poste de travail qui exécute Rviz, ce dernier affichera la configuration actuelle du robot sur le modèle de robot virtuel. Par exemple, si un vrai robot à deux bras comme Baxter a ses bras dans une certaine position, alors le modèle de robot affichera cette position dans Rviz. La rubrique ROS contenant des informations sur la configuration du bras ainsi que toute rubrique ROS publiée pour déplacer les articulations du bras peuvent être affichées dans les informations sur l'écran Rviz. Les rubriques ROS peuvent également afficher des représentations en direct, basées sur les données des capteurs publiées par les caméras, les capteurs infrarouges et les scanners laser qui font partie du système du robot. Cela peut être utile pour développer et déboguer des systèmes de robots et des contrôleurs. Rviz fournit une interface utilisateur graphique (GUI) configurable pour permettre à l'utilisateur d'afficher uniquement les informations pertinentes pour la tâche actuelle. [3]

Comme nous l'avons déjà vu, nous pouvons lancer Rviz et ajouter des affichages afin de suivre la progression de la cartographie. Pour cette dernière , nous aurons essentiellement besoin d'utiliser 2 affichages de Rviz :

— Affichage laserScaner.

— Affichage de la carte.

3.4.2 Simulation

Tout d'abord nous allons lancer notre nœud de cartographie dans notre premier terminal en utilisant la commande suivante :

roslaunch turtlebot_navigaton_gazebo gmapping_demo.launch comme

le montre la figure 3.2

```
user:~$ roslaunch turtlebot_navigation_gazebo gmapping_demo.launch
... logging to /home/user/.ros/log/4307d57e-de76-11ec-bbd6-0242ac180007/roslaunch-7_xterm-27
97.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://7_xterm:43029/

SUMMARY
========

PARAMETERS
 * /rosdistro: noetic
 * /rosversion: 1.15.9
 * /slam_gmapping/angularUpdate: 0.436
 * /slam_gmapping/astep: 0.05
 * /slam_gmapping/base_frame: base_footprint
 * /slam_gmapping/delta: 0.05
 * /slam_gmapping/iterations: 5
 * /slam_gmapping/kernelSize: 1
 * /slam_gmapping/lasamplerange: 0.005
 * /slam_gmapping/lasamplestep: 0.005
 * /slam_gmapping/linearUpdate: 0.5
 * /slam_gmapping/llsamplerange: 0.01
 * /slam_gmapping/llsamplestep: 0.01
```

FIGURE 3.2: Commande de lancement du noeud et démarge du serveur ROS.

Puis, nous allons lancer notre logiciel Rviz en exécutons dans le deuxième terminal par la commande illustré dans la figure 3.3 :

rosrun rviz rviz

```
user:~$ rosrun rviz rviz
QStandardPaths: XDG_RUNTIME_DIR not set, defaulting to '/tmp/runtime-user'
[ INFO] [1655203461.735201499]: rviz version 1.14.4
[ INFO] [1655203461.735253641]: compiled against Qt version 5.12.8
[ INFO] [1655203461.735279415]: compiled against OGRE version 1.9.0 (Ghadamon)
[ INFO] [1655203461.752929971]: Forcing OpenGl version 0.
[ INFO] [1655203462.353756992, 57.159000000]: Stereo is NOT SUPPORTED
[ INFO] [1655203462.353869527, 57.159000000]: OpenGl version: 3.1 (GLSL 1.4).
```

FIGURE 3.3: Commande de démmarage du logiciel Rviz.

L'exécution de la commande précédante permet d'ouvrir l'interface graphique directement sous le navigateur web comme le montre la figure 3.4.

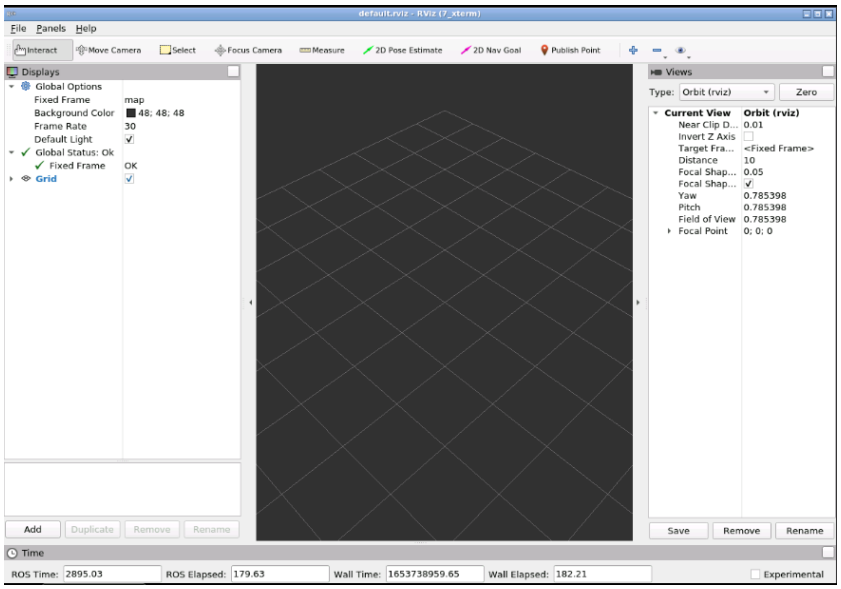

FIGURE 3.4: Interface graphique du logiciel Rviz.

Alors la première chose, nous allons ajouter notre modèle de robot Kobuki en appuyant sur le bouton **"add"** et sélectionner robotModel dans la liste des éléments voir figure 3.5

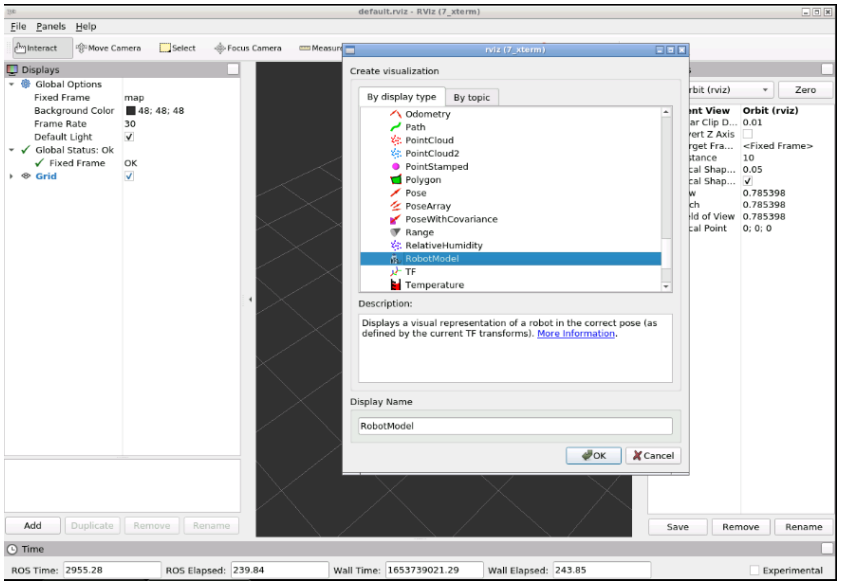

FIGURE 3.5: Insertion d'un nouveau modèle robot dans le projet Rviz.

Pour visualiser les lectures du laser(voir la figure 3.6), nous allons ajouter **"La-**

serScan" avec la même façon précédante, puis aller au **"Topic"** où les données laser sont publiées. A fin de lister tous les sujets nous allons écrire la commande suivante dans le troisième terminal.

rostopic list

Le **"Topic"** de notre laser est :

/kobuki/laser/balayage

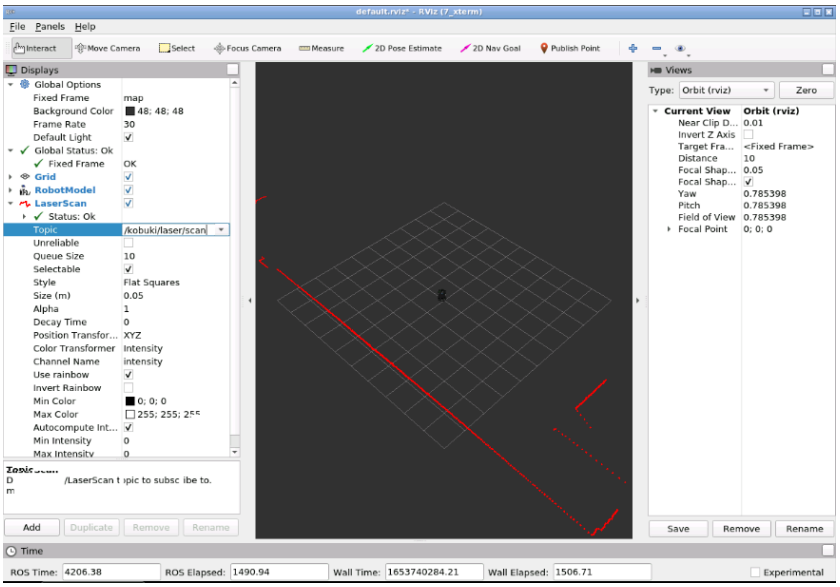

FIGURE 3.6: Visualisation des lectures du laser.

Donc ici nous pouvons visualiser ce que le laser détecte par les lignes rouge. Mais pour le processus de cartographie nous voudrons visualiser la carte, donc pour ça il faut revenir au bouton **"add"** et sélectionner l'option **"Map"** illustrée dans la figure 3.7 et indiquer le sujet où la carte à été publiée.

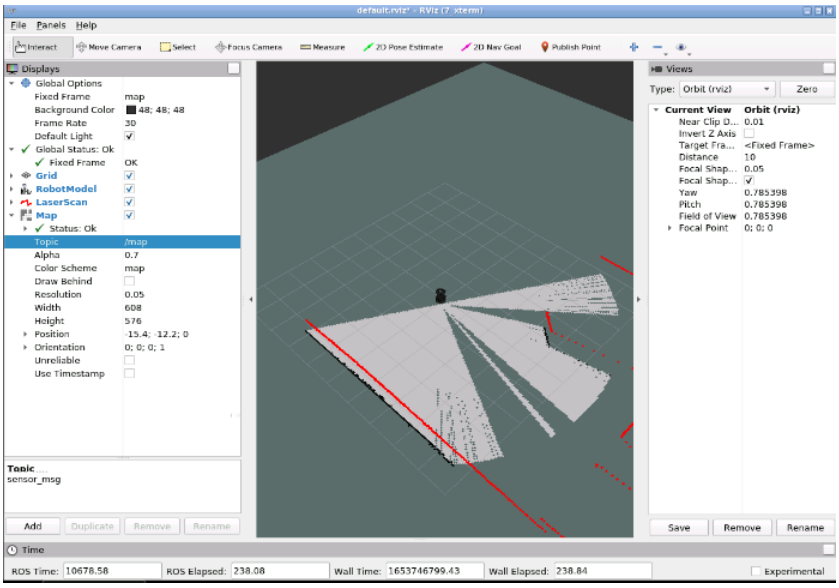

FIGURE 3.7: Visualisation de la carte.

La figure 3.7 représente aussi l'état initial de la carte avant le balayage par le robot. Pour déplacer le robot nous allons lancer le programme teleop du clavier par la commande montrée dans la figure 3.8 afin de contrôler le robot Kobuki via le clavier.

```
user:~$ roslaunch turtlebot_teleop keyboard_teleop.launch
... logging to /home/user/.ros/log/ce8ce070-ebce-11ec-b932-0242ac1c0007/roslaunch-4_xterm-1515.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://4_xterm:36285/
```

FIGURE 3.8: Démarrage du programme teleop (contrôle par le clavier).

Le résultat de le balayage du robot donne la carte cartographique illustrée par la figure 3.10 qui représente une copie numérique de l'environnement simulé (conçu sous ROS) voir figure 3.9.

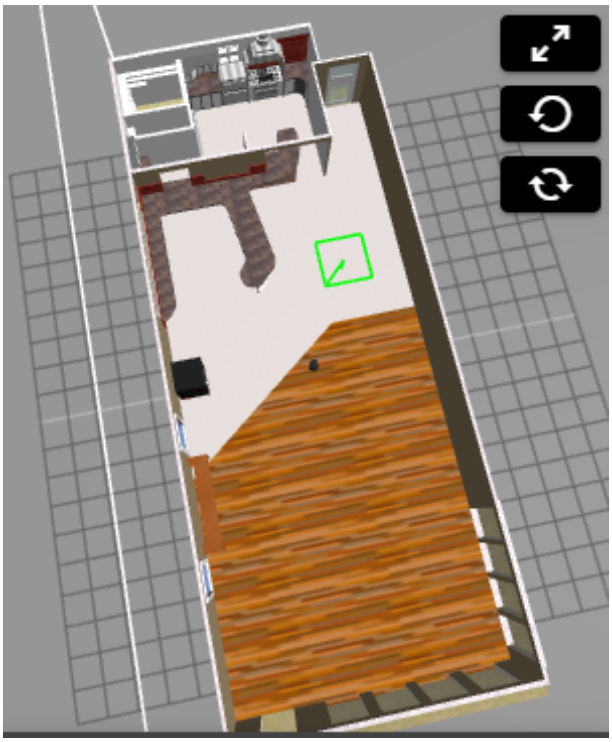

FIGURE 3.9: L'environnement d'un cafeteria.

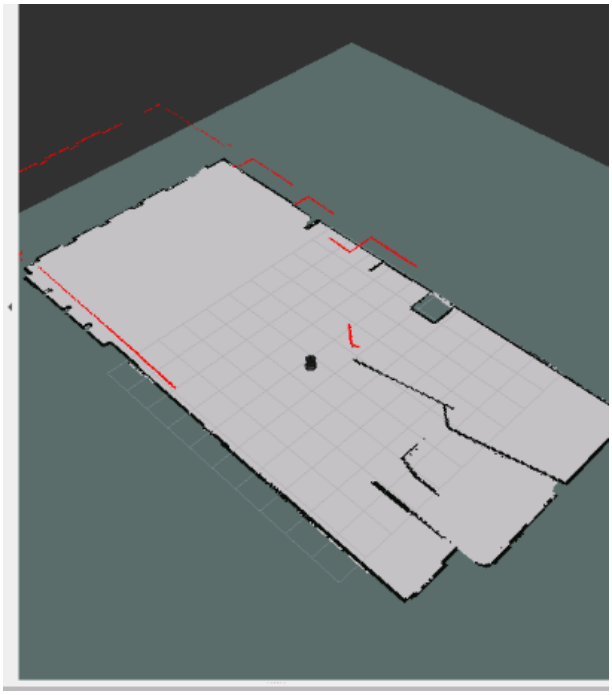

FIGURE 3.10: Visualisation de l'environnement cafeteria.

Ce que nous faisons est d'effectuer un SLAM (Simultaneous Localization and Mapping) est le nom qui fait référence à la construction d'une carte d'un environnement inconnu tout en localisant simultanément le robot dans la carte en cour de construction, c'est un problème de robotique qui s'appelle SLAM et qui se résout dans ROS en utilisant le package gmapping. Ce package de cartographie ça tâche principale est d'implémenter un algorithme SLAM appelé aussi gmapping , ce nœud nous permet de créer une carte 2D en utilisant les données du laser et la position du robot, puis lire essentiellement les données du laser et les transformation du robot, afin de transformer ces données en une carte de grille d'occupation (OGM).

Donc, fondamentalement, ce que nous venons de faire dans la simulation précédent était le suivant :

1. Nous avons utilisé un fichier de lancement de configuration créé précédemment (**gmapping_demo.launch**) pour lancer le package gmapping avec le robot Kobuki.
2. Ce fichier de lancement démarre un nœud **slam_gmapping** (à partir du package gmapping). Ensuite, nous avons déplacé le robot dans notre environnement (la cafeteria).
3. Le nœud **slam_gmapping** lancé s'est pour le Laser ik (/**kobuki**/**laser**/**scan**) et aux Transform Topics (/**tf**) afin d'obtenir les données dont il a besoin, et construire une carte.
4. La carte générée est publiée pendant tout le processus dans le sujet (Topic) /**map**, c'est la raison pour laquelle nous pouvons voir le processus de construction de la carte avec Rviz (car Rviz ne fait que visualiser les sujets).

Le sujet (/**map**) utilise un type de message de **nav_msgs**/**OccupancyGrid**, puisqu'il s'agit d'une carte de grille d'occupation (OGM). L'occupation est représentée par un nombre entier dans la plage {0, 100}. Avec 0 signifiant complètement libre, 100

signifiant complètement occupé, et la valeur spéciale de -1 pour complètement inconnu.

3.4.3 Enregistrement de la carte

Un autre des packages disponibles dans la pile de navigation ROS est **map_server_package**. Ce package fournit le **map_saver_node**. Qui nous permet d'accéder aux données cartographiques d'un service ROS et de les enregistrer dans un fichier.

Lorsque nous demandons le **map_saver** pour enregistrer la carte actuelle, les données de la carte sont enregistrées dans deux fichiers : le premier est le fichier YAML, qui contient les métadonnées de la carte et le nom de l'image, et le second est l'image elle-même, qui contient les données encodées de plan quadrillé d'occupation.

Nous pouvons sauvegarder la carte construite à tout moment en utilisant la commande suivante :

rosrun map_server map_saver–f nam_of_map comme le montre la figure 3.11

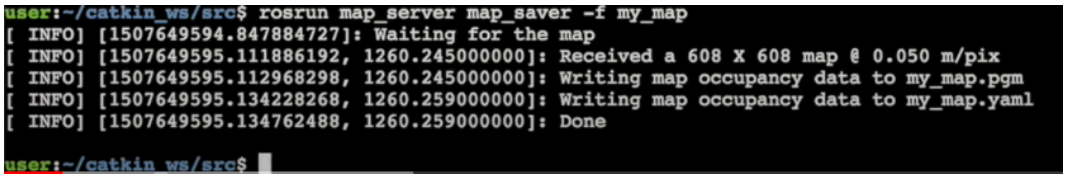

FIGURE 3.11: Enregistrement de la carte.

Après l'exécution de cette commande, il apparait deux nouveaux fichiers (**my_map.pgm**) et (**my_map.yaml**), ce sont les fichiers qui définissent notre carte.

Après avoir montré comment créer une carte d'un environnement avec un robot, nous devons comprendre ce qui suit :

— La carte que nous avons créé est une carte statique. Cela signifie que la carte restera toujours telle qu'elle était lorsque nous l'avons créé. Ainsi, lorsque nous créons une carte, elle capture l'environnement tel qu'il est au moment exact où le processus de cartographie est en cours d'exécution. Si pour une raison quelconque,

l'environnement change dans le futur, ces changements n'apparaîtront pas sur la carte, donc il ne sera plus valide (ou il ne correspondra pas à l'environnement réel).

— La carte que nous avons créé est une carte 2D. Cela signifie que les obstacles qui apparaissent sur la carte n'ont pas de hauteur. Donc si, par exemple, nous essayons d'utiliser cette carte pour naviguer avec un drone, elle ne sera pas valide. Il existe des packages qui nous permettent de générer des mappigs 3D, mais ce problème ne concerne pas notre cas.

3.5 Partie 3 : Localisation d'un robot dans un environnement.

Dans cette partie nous allons parler de la localisation de robot. Lorsque nous avons une carte de notre environnement, il est obligatoire que notre robot ait besoin de savoir quelle est sa position et sont orientation dans la carte à chaque instant. Ceci est crucial, car si nous n'avons pas ces informations, la carte est complètement inutile. C'est comme aller dans une nouvelle ville et acheter une carte de cette ville, mais si nous ne savons pas où somme nous dans la carte, cette dernière sera complètement inutile pour. Donc la localisation du robot est fondamentale.

3.5.1 Visualiser la localisation dans Rviz

Comme nous l'avons déjà vu. Nous pouvons lancer Rviz et ajouter des affichages afin de surveiller la localisation du robot. Dans cette partie, nous avons essentiellement besoin d'utiliser 3 éléments de Rviz :

— LaserScan Affichage (illustré dans la partie 2).

— Affichage de la carte (illustré dans la partie 2).

— Affichage du tableau de position.

Donc pour cela nous allons commencer par cette simulation en exécutant la commande suivante afin de lancer le nœud amcl préconfiguré :

roslaunch husky_navigation amcl_demo.launch

Après, nous allons lancer le logiciel Rviz par la commande illustré dans la figure 3.12 :

```
user:~$ roslaunch turtlebot_teleop keyboard_teleop.launch
... logging to /home/user/.ros/log/ce8ce070-ebce-11ec-b932-0242ac1c0007/roslaunch-4_xterm-1515.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://4_xterm:36285/
```

FIGURE 3.12: Commande de lancement du programme teleop du clavier.

Ensuite, nous allons faire la configuration comme la partie précédente et nous pouvons visualiser les données laser et la carte (voir la figure 3.13).

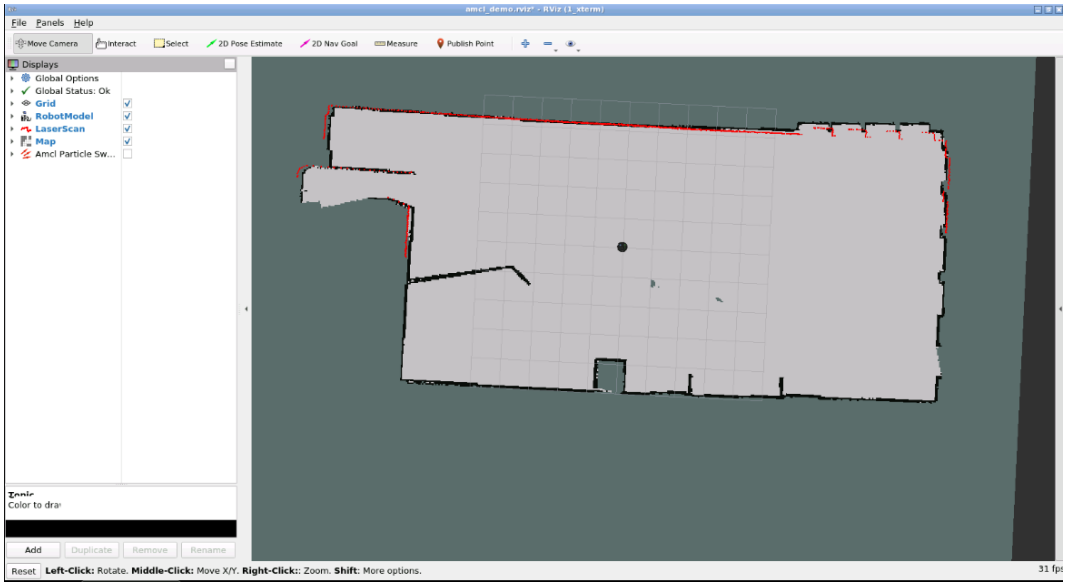

FIGURE 3.13: Visualisation des données laser et la construction de la carte.

Maintenant pour visualiser la localisation nous devons ajouter l'affichage du tableau de poste qui se trouve à travers le bouton (**"add"**), ensuite configurer le sujet (**"Topic"**), ainsi nous pouvons visualiser ces flèches caractéristiques que nous l'utilisons pour visualiser la localisation (voir la figure 3.14).

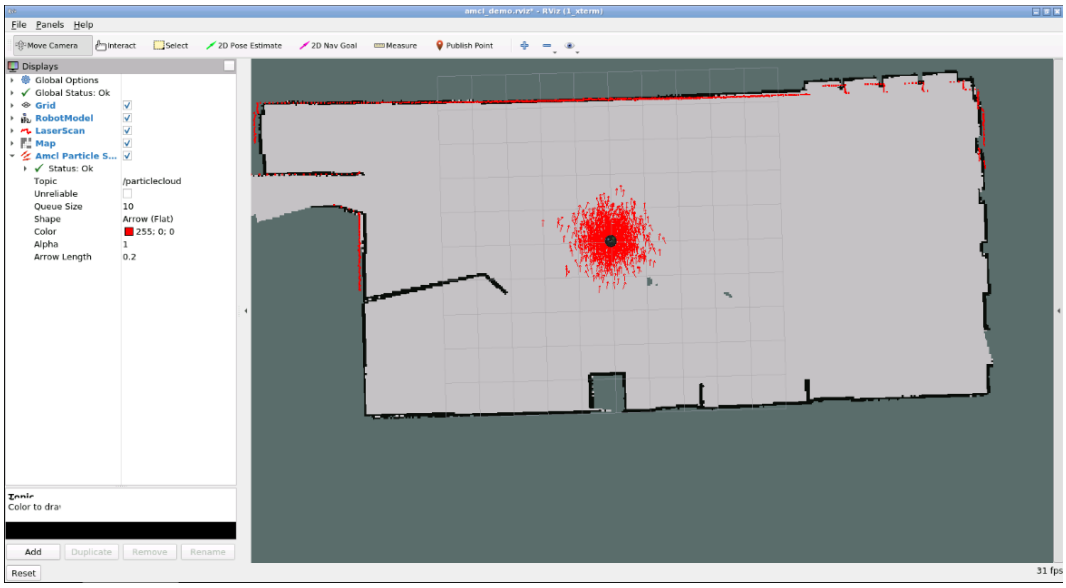

FIGURE 3.14: Visualisation de la localisation.

Dans l'étape suivante, nous allons voir la façon dont ROS traite le problème de localisation du robot.

1. Ajouter les affichages laser et cartographiques afin de voir la position du robot dans la pièce via Rviz.
2. À l'aide de l'outil d'estimation de position 2D, définir une position et une orientation initiales dans Rviz pour le robot kobuki. Il n'a pas besoin d'être exact, il faut simplement voir la position et l'orientation du Kobuki dans la simulation et essayer de le définir de la même manière dans Rviz.
3. Faire bouger le robot dans la pièce en utilisant le clavier teleop.

Ce n'est pas très précis mais ça sera corrigé lorsque le robot se déplace un peu autour de l'environnement où nous sommes à l'aide du clavier (voir la figure 3.15).

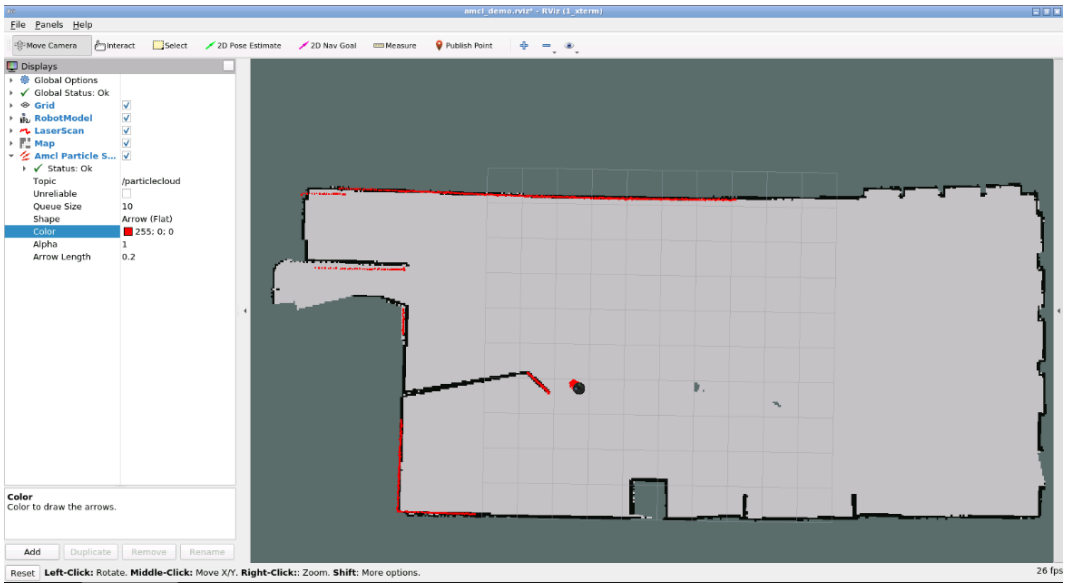

FIGURE 3.15: Localisation du robot.

Remarque 1 : La localisation dans ROS est visualisée à travers des éléments appelés particules.

Remarque 2 : La propagation du nuage représente l'incertitude du système de localisation sur la position du robot.

Remarque 3 : Au fur et à mesure que le robot se déplace dans l'environnement, ce nuage devrait diminuer de taille en raison de données de numérisation supplémentaires permettant à amcl d'affiner son estimation de la position et de l'orientation du robot.

Quelles étaient ces flèches étranges que nous visualisions dans Rviz ?

Introduisons d'abord quelques notions afin de mieux comprendre ce que nous venons de faire.

3.5.2 Localisation de Monte-Carlo (MCL)

Parce que le robot ne se déplace pas toujours comme prévu, il génère de nombreuses suppositions aléatoires quant à l'endroit où il va se déplacer ensuite. Ces suppositions sont connues sous le nom de particules. Chaque particule contient une description complète d'une future position possible. Lorsque le robot observe l'environnement dans

lequel il se trouve (via des lectures de capteurs), il rejette les particules qui ne correspondent pas à ces lectures et génère plus de particules proches de celles qui semblent plus probables. De cette façon, à la fin, la majeure partie de la convergence des particules se trouve dans la position la plus probable dans laquelle se trouve le robot. Ainsi, plus nous nous déplaçons, plus nous obtiendrons des données de nos capteurs, d'où la localisation sera plus précise. Ces particules sont les "flèches" que nous avons vu dans Rviz dans l'exécution précédente.
Ceci est connu sous le nom d'algorithme de localisation de Monte Carlo (MCL), ou encore localisation de filtre à particules. [12]

3.5.3 Le pack AMCL

Le package AMCL (Adaptive Monte Carlo Localization) fournit le nœud amcl, qui utilise le système MCL afin de suivre la localisation d'un robot qui se déplace dans un espace 2D. Ce nœud s'abonne aux données du laser, à la carte laser et aux transformations du robot, et publie sa position estimée dans la carte. Au démarrage, le nœud amcl initialise son filtre à particules en fonction des paramètres fournis.
REMARQUE : Pour nommer ce package ROS (et ce nœud), le mot Adaptive a été ajouté à l'algorithme de localisation de Monte Carlo. En effet, dans ce nœud, nous pourrons configurer (adapter) certains des paramètres utilisés dans cet algorithme.
Donc, en gros, nous pouvons résumer les étapes de l'application précédente comme suit :

- Lancer un nœud amcl à l'aide du fichier **amcl_demo.launch** préconfiguré.
- Configurer une position initiale à l'aide de l'outil d'estimation de position 2D (qui a publié cette position dans le sujet /**initialposition**).
- Déplacer le robot dans la pièce, ensuite le nœud amcl commence à lire les données publiées dans le sujet laser (/**scan**), le sujet de la carte (/**map**) et le sujet de transformation (/**tf**), ont publié la position estimée où le robot correspond à /**amcl_position** et aux sujets /**particlecloud**.

— L'accé aux données publiées via Rviz par ce nœud dans le sujet /**particlecloud**, les visualiser ensuite , grâce au nuage de "flèches", qui indique la position la plus probable dans laquelle se trouve le robot, et son orientation.

3.6 Partie 4 : La planification d'une trajectoire dans cet environnement.

Cette partie consiste à faire en sorte que le robot navigue de manière autonome.

Pour l'instant, nous avons vu comment créer une carte d'un environnement et comment y localiser le robot. Donc, à ce stade, nous avons tout ce dont nous avons besoin pour effectuer la navigation. Autrement dit, nous sommes maintenant prêts à planifier des trajectoires afin de déplacer le robot de la position A à la position B.

Dans cette partie, nous apprendrons comment fonctionne le processus de planification de chemin dans ROS, et tous les éléments qui s'y déroulent.

3.6.1 Visualisez la planification de chemin dans Rviz

Comme nous l'avons déjà vu dans les parties précédentes, nous pouvons également lancer Rviz et ajouter des affichages afin de surveiller le processus de planification de trajectoire du robot. Pour cette partie, nous avons essentiellement besoin d'utiliser trois éléments de Rviz :

— Affichage de la carte (Costmaps).

— Affichages de chemin (plans).

— Outils 2D.

Effectuons alors la simulation sur Rviz. Tout d'abord lançons la commande illstrée dans la figure 3.16 qui va réellement lancer le système de planification de trajectoire.

```
user:~$ roslaunch turtlebot_navigation_gazebo move_base_demo.launch
... logging to /home/user/.ros/log/e7ef6c66-e0e5-11ec-bfe3-0242ac160008/roslaunch-1_xterm-14
59.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://1_xterm:44205/
```

FIGURE 3.16: Commande pour lancer le système de planification de trajectoire.

Ensuite, lançons le logiciel Rviz par la commande montrée dans la figure 3.17.

```
user:~$ rosrun rviz rviz
QStandardPaths: XDG_RUNTIME_DIR not set, defaulting to '/tmp/runtime-user'
[ INFO] [1654004142.942088393]: rviz version 1.14.4
[ INFO] [1654004142.942156276]: compiled against Qt version 5.12.8
```

FIGURE 3.17: Commande de lancement de logiciel Rviz.

Donc nous avons déjà une carte, dupliquons cela car nous allons utiliser deux, ajouter ensuite au moins un élément de chemin, et nous avons également ajouter un élément de tableau post dans l'ordre pour visualiser la localisation, ensuite configurer correctement ces éléments. Dans la première carte, nous allons définir le sujet par la rubrique /**move_base/global_costmap/costmap** afin de visualiser les coûts globales illustrés dans la figure 3.18.

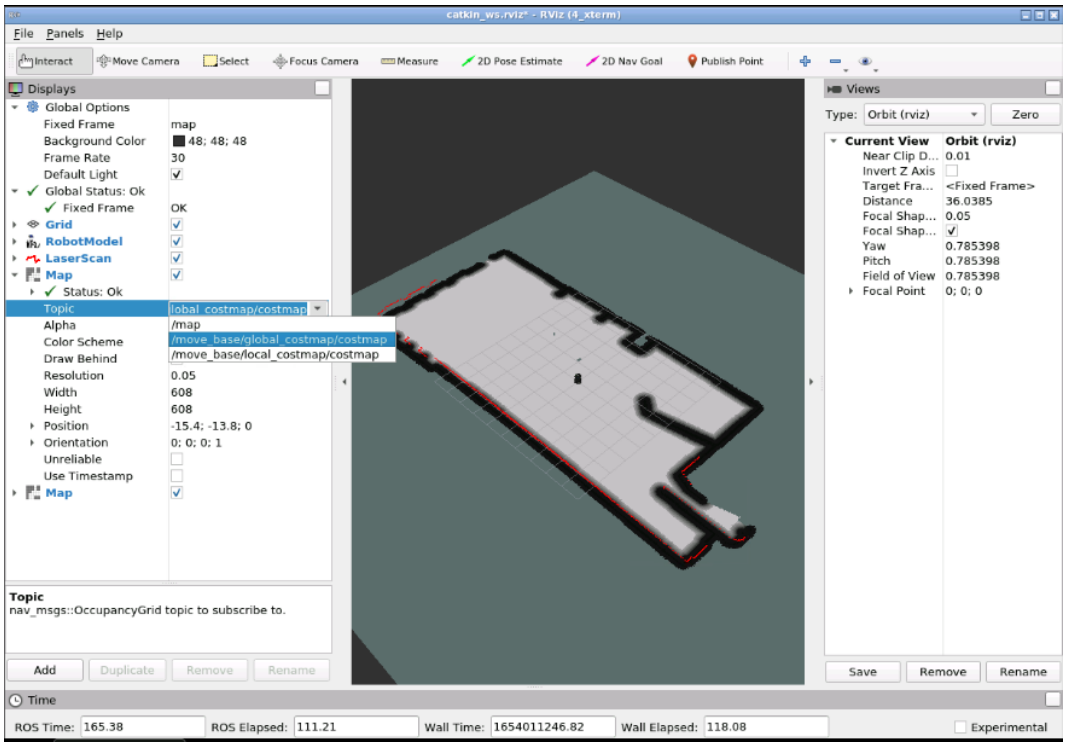

FIGURE 3.18: L'affichage de la carte des coûts globale.

Dans la deuxième carte nous changeons la rubrique en **/move_base/local_costmap/costm** afin de visualiser la carte des coûts locale montrés dans la figure 3.19.

Nous pouvons avoir deux affichages de carte, un pour chaque carte de coût.

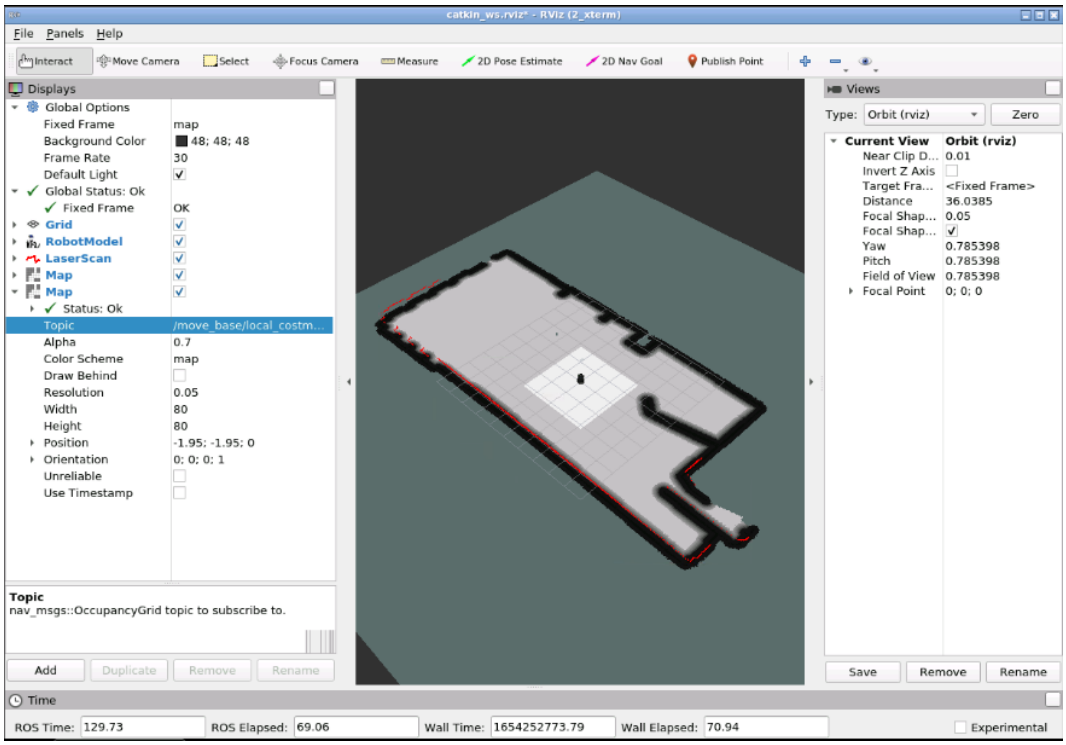

FIGURE 3.19: L'affichage de la carte des coûts locale.

— Cliquer sur le bouton **"add"** sous Affichages et choisir l'élément Chemin.

— Définir le sujet par la rubrique **/move_base/NavfnROS/plan** afin de visualiser le plan global.

— L'affichage du chemin, pour notre plan est illustré dans la figure 3.20.

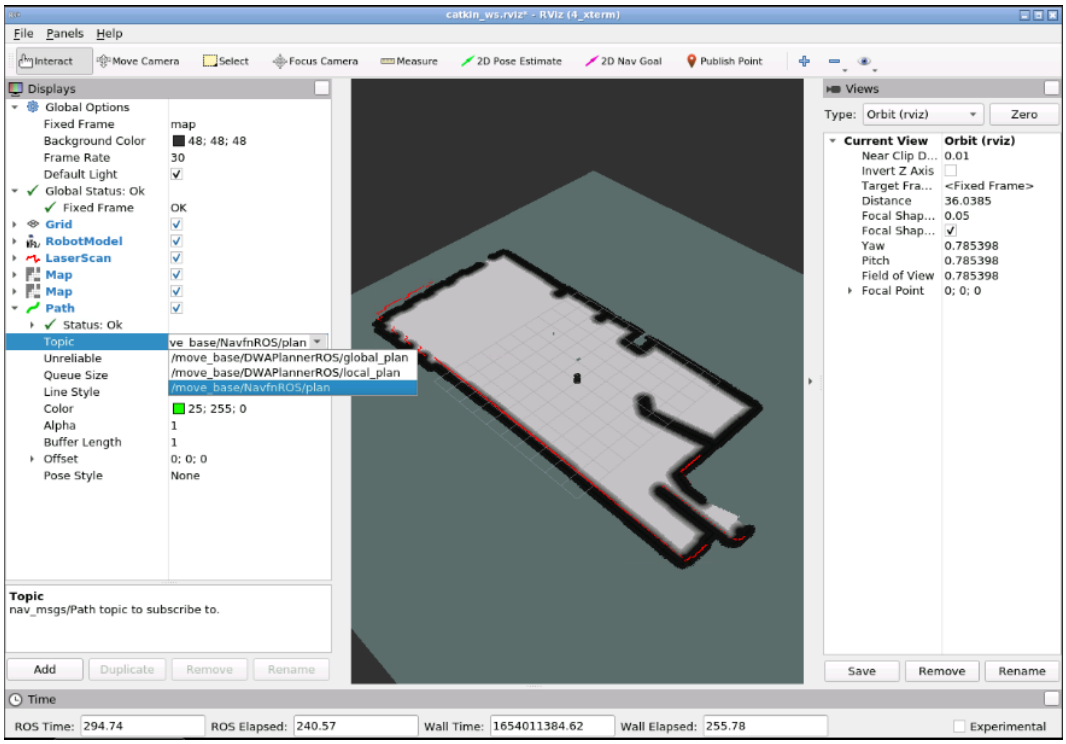

FIGURE 3.20: Affichage du plan gobal.

Pour l'instant rien n'apparait encore cela sera affiché lorsque le chemin sera calculé.

Ensuite, nous allons ajouter le nuage de particules et le configurer(voir la figure 3.21).

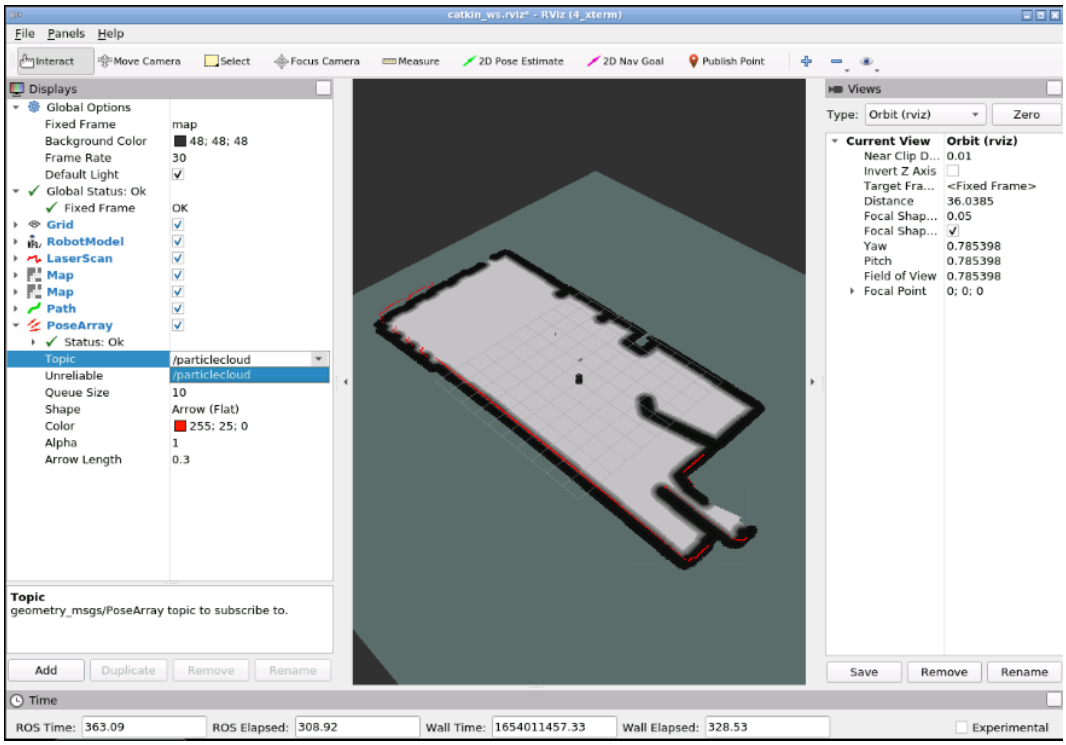

FIGURE 3.21: Commande de nuage de particules.

Alors la première chose à faire est localiser notre robot afin que nous puissions voir notre vrai robot dans la simulation.

Nous allons utilisez l'outil d'estimation de position 2D afin de fournir une position initiale pour le robot (voir la figure 3.22).

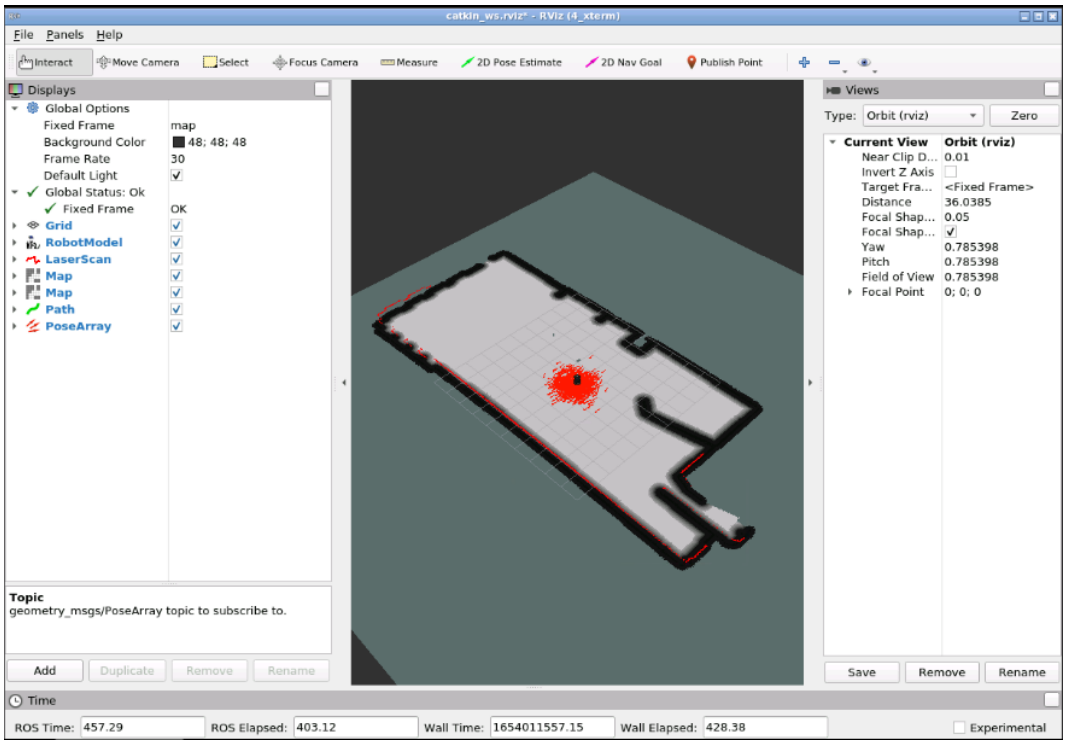

FIGURE 3.22: Position initiale du robot.

Maintenant nous allons envoyer un objectif à notre robot en utilisant l'outil **2D Nav Goal** afin d'envoyer une position cible au robot (voir les figures 3.23 et 3.24).

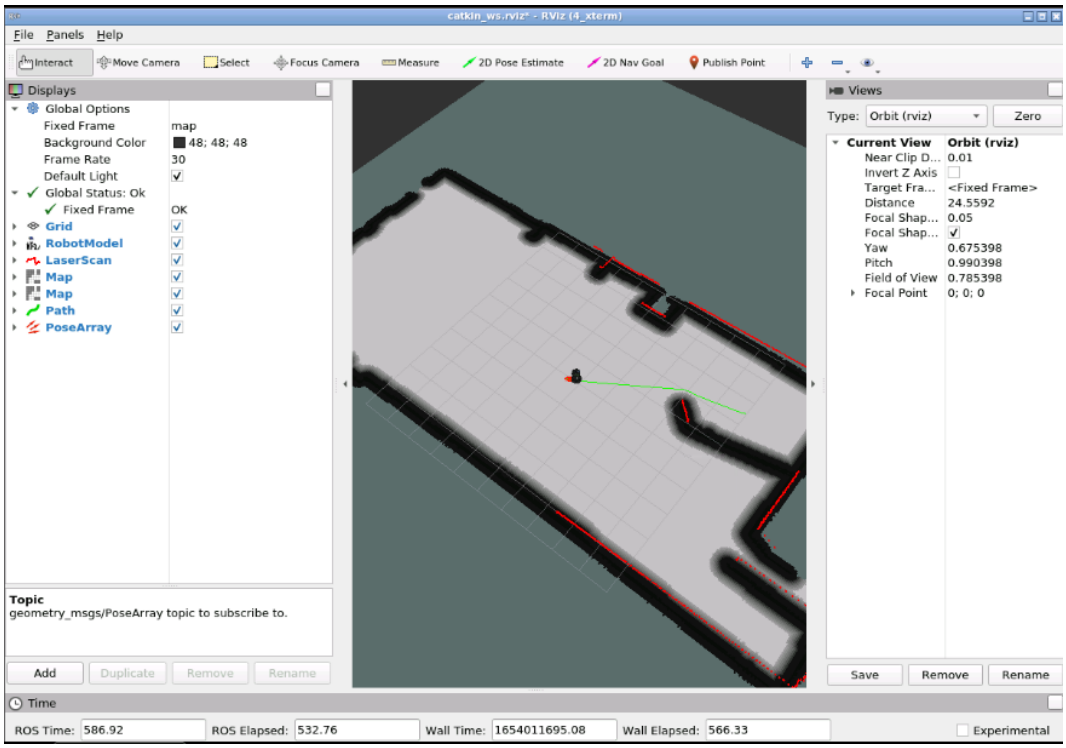

FIGURE 3.23: L'envoi d'une position cible au robot.

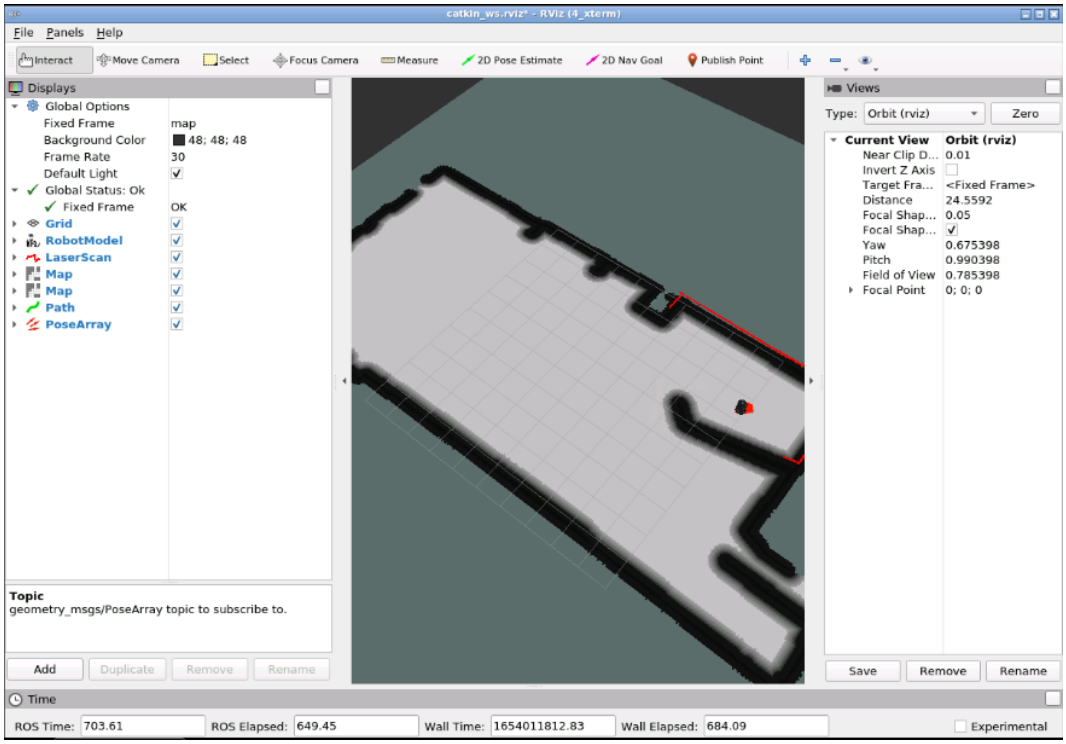

FIGURE 3.24: L'arrivée du robot à la destination finale.

Nous allons détailler par la suite le fonctionnement de l'ensemble du processus.

3.6.2 Le paquet move_base

Le package de base de déplacement contient le nœud **move_base**. Le nœud de base de déplacement est l'un des principaux éléments de la pile de navigation ROS, car il relie tous les éléments qui ont lieu dans le processus de navigation. Sans ce nœud, la pile de navigation ROS n'aurait aucun sens. Nous comprenons que le nœud de base de déplacement est très important, mais quelques questions se positionnt. Qu'est-ce que c'est exactement ? Qu'est-ce qu'il fait ?

La fonction principale du nœud **move_base** est de déplacer le robot de sa position actuelle vers une position cible. Fondamentalement, ce nœud est une implémentation d'un **SimpleActionServer**, qui prend une position cible avec le type de message **geometry_msgs/positionStamped**. Par conséquent, nous pouvons envoyer des objectifs de position à ce nœud en utilisant un **SimpleActionClient**.

Ce serveur d'action fournit la rubrique **move_base/goal**, qui est l'entrée de **Navigation_Stack**. Ce sujet est ensuite utilisé pour fournir la position de but.

3.6.3 La planification globale

Lorsqu'un nouvel objectif est reçu par le nœud de base de déplacement, cet objectif est immédiatement envoyé au planificateur global. Ensuite, le planificateur global est chargé de calculer un chemin sûr afin d'arriver à cette position cible. Ce chemin est calculé avant que le robot ne commence à se déplacer, il ne prendra donc pas en compte les lectures que les capteurs du robot font pendant le déplacement.

Chaque fois qu'un nouveau chemin est planifié par le planificateur global, ce chemin est publié dans la rubrique **/plan**.

Nous remarquons que lorsque nous envoyons un objectif afin de visualiser le plan de chemin réalisé par le planificateur global, le robot démarre automatiquement l'exécution de ce plan. Cela se produit parce qu'en envoyant cette position cible, tout le processus de

navigation démarre. Dans certains cas, nous pouvons s'intéresser à la simple visualisation du plan global, mais pas par l'exécution de ce plan. Dans ce cas, le nœud de base de déplacement fournit un service nommé /**make_plan**. Ce service permet de calculer un plan global sans que le robot n'exécute le parcours.

Ainsi, nous savons maintenant que la première étape de ce processus de navigation consiste à calculer un plan de sécurité afin que notre robot puisse arriver à la position cible spécifiée par l'utilisateur. Mais comment ce chemin est-il calculé ?

Il existe différents planificateurs globaux. Selon notre configuration (le robot que nous utilisons, l'environnement dans lequel il navigue, etc.), nous utiliserons l'un ou l'autre.

3.6.3.1 Navfn

Le planificateur Navfn est probablement le planificateur global le plus couramment utilisé pour la navigation ROS. Il utilise l'algorithme de Dijkstra pour calculer le chemin le plus court entre la position initiale et la position cible.

3.6.3.2 Planificateur de carottes

Le planificateur de carottes prend la position du but et vérifie si ce but se trouve dans un obstacle. Ensuite, s'il se trouve dans un obstacle, il recule le long du vecteur entre le but et le robot jusqu'à ce qu'un point de but qui n'est pas dans un obstacle soit trouvé. Il transmet ensuite ce point cible sous forme de plan à un planificateur ou contrôleur local. Par conséquent, ce planificateur ne fait pas de planification de chemin globale. C'est utile si nous avons besoin que notre robot se rapproche de l'objectif donné, même si l'objectif est inaccessible. Dans les environnements intérieurs compliqués, ce planificateur n'est pas très pratique.

Cet algorithme peut être utile si, par exemple, nous souhaitons que notre robot se déplace le plus près possible d'un obstacle (une table, par exemple).

3.6.3.3 Planificateur global

Le planificateur global disposition également de ses propres paramètres afin de personnaliser son comportement. Les paramètres du planificateur global se trouvent également dans un fichier YAML. Selon le planificateur global que nous utilisons, les paramètres à définir seront différents. Dans cette partie, Nous allons voir les paramètres du navfn planner car c'est celui qui est le plus couramment utilisé.

Paramètres Navfn

- **/allow_unknown (défaut : true)** : spécifie s'il en faut, ou non autoriser navfn à créer des plans qui traversent un espace inconnu.
- **/planner_window_x (défaut : 0.0)** : spécifie la taille x d'une fenêtre facultative à laquelle le planificateur est limité. Cela peut être utile pour limiter NavFn pour qu'il fonctionne dans une petite fenêtre d'une grande carte de coûts.
- **/planner_window_y (défaut : 0.0)** : spécifie la taille y d'une fenêtre facultative à laquelle le planificateur est limité. Cela peut être utile pour limiter NavFn pour qu'il fonctionne dans une petite fenêtre d'une grande carte de coûts.
- **/default_tolerance (défaut : 0.0)** : une tolérance sur le point cible pour le planificateur. NavFn tentera de créer un plan aussi proche que possible de la cible spécifié, mais pas plus éloigné que le default_tolerance.

Jusqu'à présent, nous avons vu qu'il existe un planificateur global chargé de calculer un chemin sûr afin de déplacer le robot d'une position initiale à une position cible. Nous avons également vu qu'il existe différents types de planificateurs globaux et que nous pouvons choisir le planificateur global que nous souhaitons utiliser. Enfin, chaque planificateur a ses propres paramètres, qui modifient le comportement du planificateur. Mais Lorsque nous planifions une trajectoire, cette dernière doit être planifiée selon une carte. Un chemin sans carte n'a aucun sens. Alors quelle carte le planificateur global utilise pour calculer son chemin ?

Nous pouvons penser que la carte utilisée est la carte que nous avons créé dans la partie Cartographie (partie 2) de ce chapitre, mais, ce n'est pas tout à fait vrai.

Il existe un autre type de carte :

La costmap :

Une costmap est une carte qui représente les lieux qui sont sûrs pour le robot d'être dans une grille de cellules. Généralement, Les valeurs de la carte des coûts sont binaires, représentant soit l'espace libre, soit les endroits où le robot serait en collision.

Chaque cellule d'une carte de coûts a une valeur entière dans la plage 0,255. Certaines valeurs spéciales sont fréquemment utilisées dans cette plage, qui fonctionne comme suit :

- **255 (NO_INFORMATION)** : réservé aux cellules où les informations sont insuffisantes.
- **254 (LETHAL_OBSTACLE)** : indique qu'un obstacle provoquant une collision a été détecté dans cette cellule.
- **253 (INSCRIBED_INFLATED_OBSTACLE)** : n'indique aucun obstacle, mais déplacer le centre du robot à cet emplacement entraînera une collision.
- **0 (ESPACE LIBRE)** : Cellules où il n'y a pas d'obstacles et, par conséquent, déplacer le centre du robot à cette position n'entraînera pas de collision.

Il existe 2 types de costmaps : costmap global et costmap local. La principale différence entre eux est, fondamentalement, la façon dont ils sont construits :

— La carte des coûts globale est créée à partir d'une carte statique.

— La carte des coûts locale est créée à partir des relevés des capteurs du robot.

Pour l'instant, concentrons-nous bien sur la carte des coûts globale puisque c'est celle utilisée par le planificateur global. Ainsi, le planificateur global utilise la carte des coûts globale afin de calculer le chemin à suivre.

3.6.4 Carte des coûts globale

La carte des coûts globale est créée à partir d'une carte statique générée par l'utilisateur (comme celle créée dans la partie de la cartographie). Dans ce cas, la carte de coût est initialisée pour correspondre aux informations de largeur, hauteur et d'obstacle fournies par la carte statique. Cette configuration est normalement utilisée en conjonction avec un système de localisation, tel que amcl. C'est la méthode que nous utiliserons pour initialiser une carte de coûts globale.

La carte des coûts globale a également ses propres paramètres, qui sont définis dans un fichier YAML.

3.6.5 Les paramètres Costmap

Les paramètres Costmap sont définis dans trois fichiers différents :

— Un fichier YAML qui définit les paramètres de la carte des coûts globals. Appelons ce fichier **global_costmap_params.yaml**.

— Un fichier YAML qui définit les paramètres de la carte des coûts locales. Appelons ce fichier **local_costmap_params.yaml**.

— Un fichier YAML qui définit les paramètres des cartes des coûts globales et locales. Appelons ce fichier **common_costmap_params.yaml**.

Maintenant, nous concentrons sur les paramètres globaux de la carte des coûts puisque c'est elle qui est utilisée par le planificateur global.

3.6.6 Paramètres de carte de coût globale

Les paramètres que nous devons connaître sont les suivants :

— **global_frame (par défaut : "/map")** : le cadre global dans lequel la carte de coûts doit fonctionner.

— **static_map (défaut : true)** : s'il faut ou non utiliser une carte statique pour initialiser la costmap.

— **rolling_window (défaut : false)** : s'il faut ou non utiliser une version à fenêtre glissante de la carte des coûts. Si le paramètre de la carte statique est défini sur true, ce paramètre doit être défini sur false.

— **plugins** : Séquence de spécifications de plugins, une par couche. Chaque spécification est un dictionnaire avec un nom et des champs de type. Le nom est utilisé pour définir l'espace de nommage du paramètre pour le plugin. Ce nom sera ensuite défini dans le fichier **common_costmap_parameters.yaml**.

3.7 Partie 5 : L'exécution d'une trajectoire et l'évitement des obstacles

Jusqu'à présent, nous avons vu comment ROS planifie une trajectoire afin de déplacer un robot d'une position de départ à une position d'arrivée. Dans cette partie, nous apprendrons comment ROS exécute cette trajectoire et évite les obstacles rencontrés. Nous apprendrons également d'autres concepts importants concernant la planification de chemin, que nous avons manqués dans la partie précédente. Enfin, nous ferons un résumé afin que nous puissions mieux comprendre l'ensemble du processus.

3.7.1 Le planificateur local

Une fois que le planificateur global a calculé le chemin à suivre, ce chemin est envoyé au planificateur local. Ce dernier exécutera alors chaque segment du plan global (imaginons le plan local comme une plus petite partie du plan global). Alors, donné un plan à suivre (fourni par le planificateur global) et une carte, le planificateur local fournira des commandes de vitesse afin de déplacer le robot.

Contrairement au planificateur global, le planificateur local surveille l'odométrie et les données laser, et choisir un plan local sans collision pour le robot. Ainsi, le planificateur local peut recalculer la trajectoire du robot afin d'empêcher le robot de heurter des objets, tout en lui permettant d'atteindre sa destination.

Une fois le plan local calculé, il sera publié dans une rubrique nommée /**local_plan**. Le planificateur local publie également la partie du plan global qu'il tente de suivre dans la rubrique /**plan_global**. Faisons une simulation pour que nous puissions mieux voir cela à l'aide des étapes suivantes :

— Ouvrez Rviz et ajoutez des affichages afin de pouvoir visualiser les sujets /**global_plan** et /**local_plan** du planificateur local.

Nous avons déjà configurer le plan global /**move_base/NavfnROS/plan**, et maintenant nous allons dupliquer cet élément du plan puis nous allons le configurer pour un autre sujet qui sera le plan local /**move_base/DWAPlannerROS/local_plan** illustrée dans la figure 3.25.

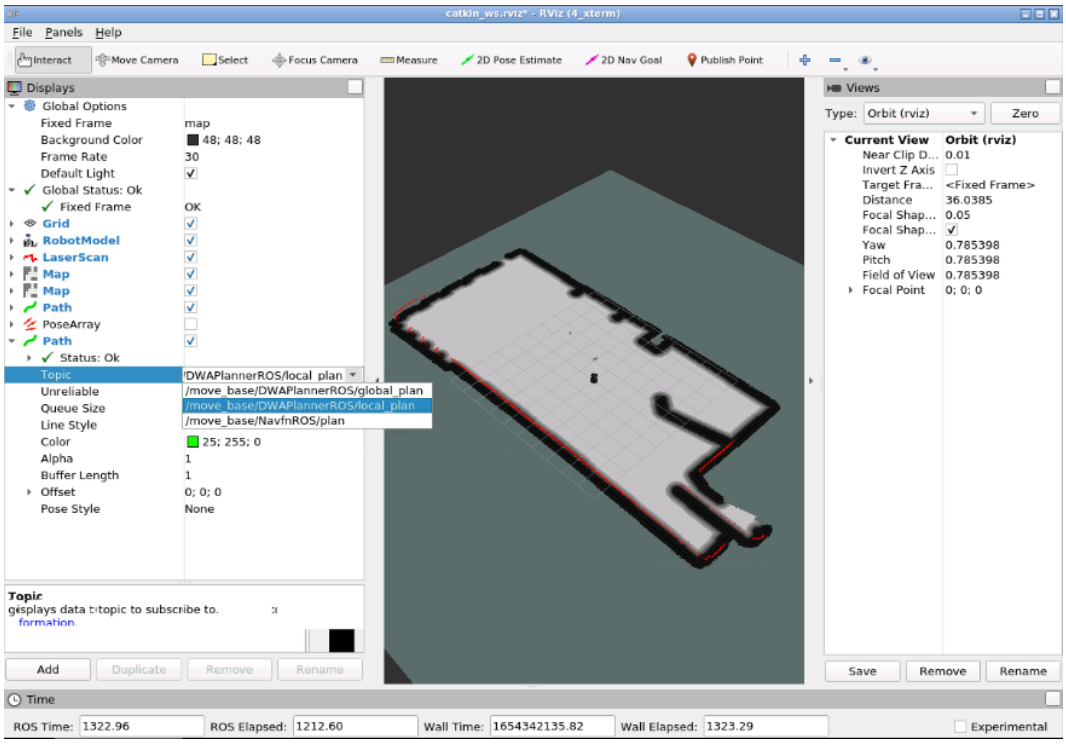

FIGURE 3.25: La configuration de plan local.

— Rapprochons-nous de la simulation afin de pouvoir la visualiser.

Nous allons mettre le tracé du plan global en rouge et celui du plan local en vert, en suite nous envoyons une position cible au robot puis nous visualisons les deux sujets illustrée

dans la figure 3.26.

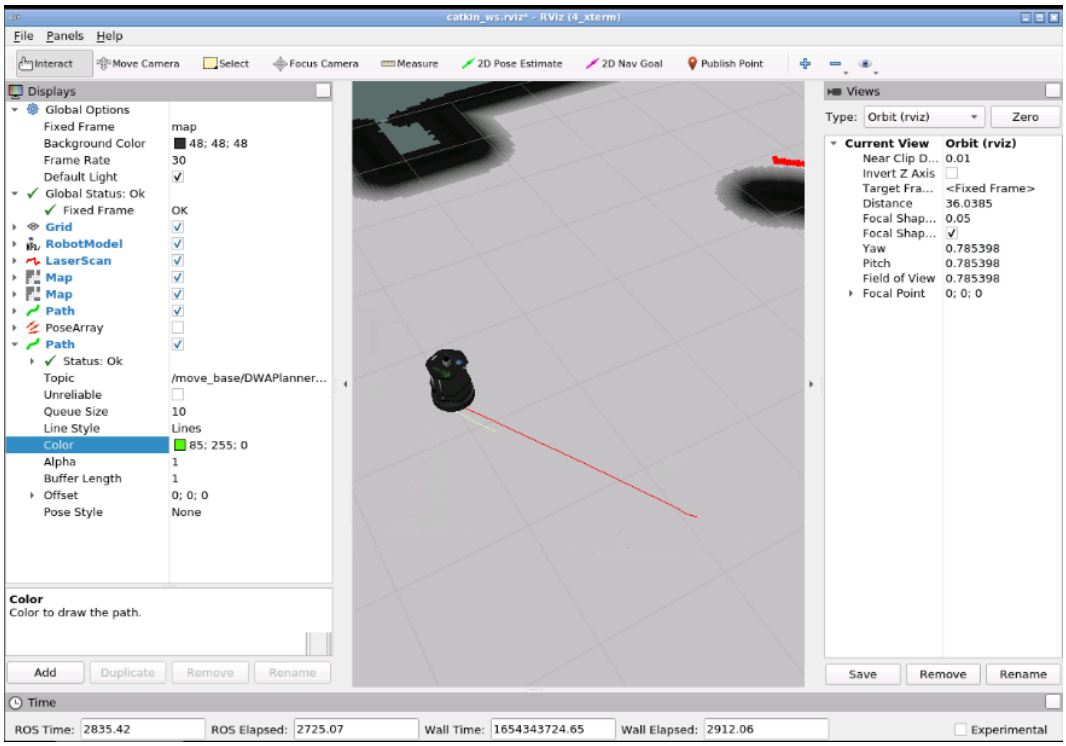

FIGURE 3.26: Visualisation de la trajectoire de plan global et plan local.

Dans la figure 3.26 nous remarquons que le plan global est tracé en rouge, mais nous avons aussi un petit plan qui est en vert représentant le plan local. Donc, nous pouvons dire que le plan local divise le plan global en plus petites parties et il continue d'exécuter ces petites partie du plan.

Comme pour le planificateur global, différents types de planificateurs locaux existent également. Selon notre configuration (le robot utilisé, l'environnement dans lequel il navigue, etc.) et le type de performances que nous souhaitons, nous utiliserons l'un ou l'autre.

3.7.1.1 Planificateur local de base (Base_local_planner)

Le planificateur local de base fournit des implémentations des algorithmes Déploiement de la trajectoire (Trajectory Rollout) et Approche de fenêtre dynamique (Dynamic Window Approach (DWA)) afin de calculer et d'exécuter un plan global pour le robot. En résumé, l'idée de base du fonctionnement de cet algorithme est la suivante :

— Échantillonnez discrètement depuis l'espace de contrôle du robot.

— Pour chaque vitesse échantillonnée, effectuez des simulations en avant à partir de l'état actuel du robot pour prédire ce qui se passerait si la vitesse échantillonnée était appliquée.

— Evaluer chaque trajectoire issue de la simulation directe.

— Jetez les trajectoires illégales.

— Choisissez la trajectoire la plus performante et envoyez les vitesses associées à la base mobile.

— Répéter.

DWA diffère du déploiement de la trajectoire (TR) dans la façon dont l'espace du robot est échantillonné. Les échantillons de déploiement de trajectoire proviennent de l'ensemble des vitesses réalisables sur toute la période de simulation vers l'avant compte tenu des limites d'accélération du robot, tandis que les échantillons DWA proviennent de l'ensemble des vitesses réalisables pour une seule étape de simulation compte tenu des limites d'accélération du robot.

DWA est un algorithme plus efficace car il échantillonne un espace plus petit, mais peut être surpassé par le déploiement de trajectoire pour les robots de faibles limites d'accélération car DWA ne simule pas les accélérations constantes. En pratique, DWA et le déploiement de trajectoire fonctionnent de la même manière, il est donc recommandé d'utiliser DWA en raison de ses gains d'efficience.

L'algorithme DWA du planificateur local de base a été amélioré dans un nouveau planificateur local séparé de celui-ci. C'est le planificateur local DWA.

3.7.1.2 Le planificateur local DWA (dwa_local_planner)

Le planificateur local DWA fournit une implémentation de l'algorithme DWA. Il s'agit essentiellement d'une réécriture de l'option DWA du planificateur local de base, mais le code est beaucoup plus propre et plus facile à comprendre, en particulier dans la

manière dont les trajectoires sont simulées.

Ainsi, pour les applications qui utilisent l'approche DWA pour la planification locale, le planificateur local DWA est probablement le meilleur choix. C'est l'option la plus couramment utilisée.

Paramètres du planificateur local DWA

Les paramètres du planificateur local sont définis dans un autre fichier YAML. Les paramètres les plus importants pour le planificateur local DWA sont les suivants :

Paramètres de configuration du robot :

- **/acc_lim_x (par défaut : 2.5)** : la limite d'accélération x du robot en m/s^2.
- **/acc_lim_th (par défaut : 3.2)** : la limite d'accélération de rotation du robot en rad/s^2.
- **/max_trans_vel (par défaut : 0.55)** : la valeur absolue de la vitesse de translation maximale du robot en m/s.
- **/min_trans_vel (par défaut : 0.1)** : la valeur absolue de la vitesse de translation minimale du robot en m/s.
- **/max_vel_x (par défaut : 0.55)** : La vitesse maximale x pour le robot en m/s.
- **/min_vel_x (par défaut : 0.0)** : La vitesse minimale x du robot en m/s, négative pour le mouvement vers l'arrière.
- **/max_rot_vel (par défaut : 1.0)** : la valeur absolue de la vitesse de rotation maximale du robot en rad/s.
- **/min_rot_vel (par défaut : 0.4)** : la valeur absolue de la vitesse de rotation minimale du robot en rad/s.

3.7.1.3 Le planificateur local Eband (eband_local_planner)

Le planificateur local Eband implémente la méthode Elastique (Elastic Band) afin de calculer le plan local à suivre.

3.7.1.4 Planificateur local TEB (teb_local_planner)

Le planificateur local TEB implémente la méthode Bande élastique chronométrée (Timed Elastic Band) afin de calculer le plan local à suivre.

3.7.2 Carte des coûts locale (local costmap)

La première chose que nous devons savoir est que le planificateur local utilise la carte des coûts locale afin de calculer les plans locaux.

Contrairement à la carte des coûts globale, la carte des coûts locale est créée directement à partir des lectures des capteurs du robot. Étant donné une largeur et une hauteur pour la carte des coûts (qui sont définies par l'utilisateur), il maintient le robot au centre de la carte des coûts lorsqu'il se déplace dans l'environnement, supprimant les informations sur les obstacles de la carte lorsque le robot se déplace.

3.7.3 Détection et évitement d'obstacles

Dans cette partie nous allons ajouter un objet qui n'est pas dans la carte statique, puis nous allons exécuter la commande suivante dans notre terminal afin de démarrer la démo d'évitement d'obstacles.

roslaunch turtlebot_navigation_gazebo move_base_demo.launch

Ensuite nous allons exécuter la commande suivante dans un deuxième terminal afin de pouvoir copier dans notre espace de travail le fichier URDF de l'obstacle que nous avons générer par la simulation.

cp /home/simulations/public_sim_ws/srs/all/turtlebot_navigation_gazebo/urd /home/user/catkin_ws/srs

Puis, nous allons exécuter la commande suivante dans le deuxième terminal afin de générer l'obstacle dans la pièce.

rosrun gazebo_ros spawn_model -file /home/user/catkin_ws/srs/object.urdf -urdf -x 0 -y 0 -z 1 -model my_object

Nous devrions voir apparaître une boîte bleue dans la simulation. (Voir la figure

3.27)

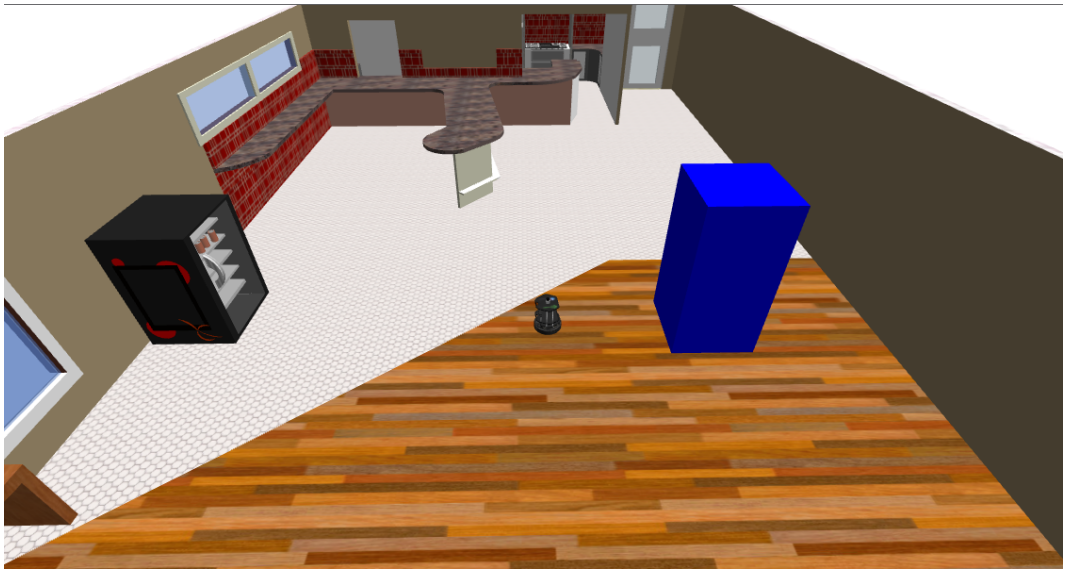

FIGURE 3.27: L'obstacle dans l'environnement.

Exécutons maintenant la commande suivante dans le deuxième terminal afin de démarrer Rviz avec une configuration prédéfinie.

roslaunch turtlebot_rviz_launchers view_planning.lauch

Comme nous allons voir, cet objet n'apparait pas (voir la figure 3.28).

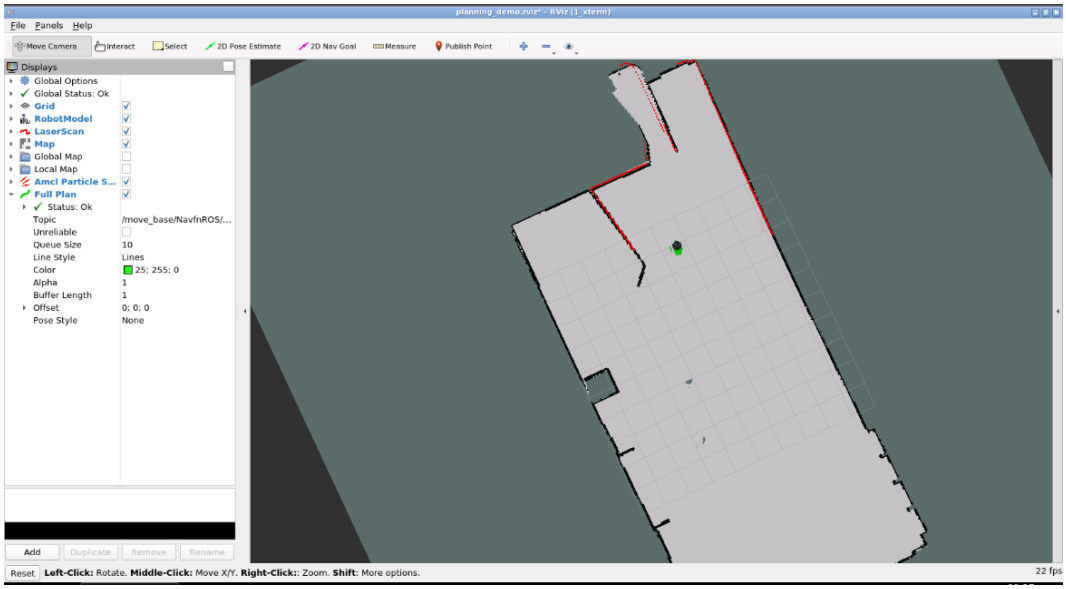

FIGURE 3.28: Aucun obstacle détecté.

Nous allons utiliser les outils **2D position Estimate** et **2D Nav Goal** dans Rviz (comme dans la simulation précédente) afin de dire au robot où il se trouve et où nous voulons qu'il aille. Fournissons au robot un emplacement qui oblige le robot à éviter l'obstacle pour vérifier le fonctionnement du système (voir la figure 3.29).

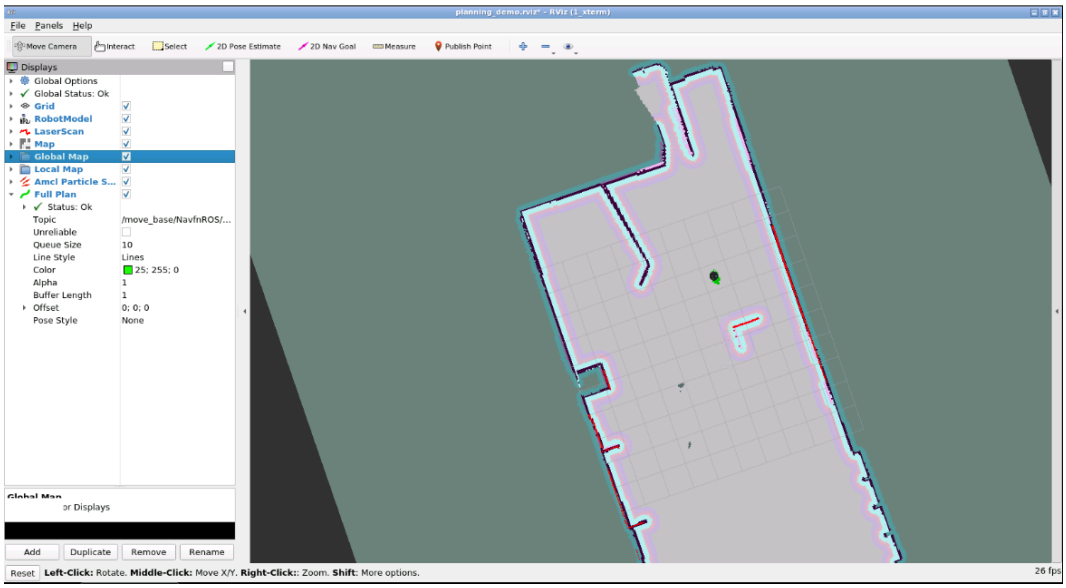

FIGURE 3.29: Obstacle détecté.

Maintenant le laser commence à détecter cet objet car la carte des couts locale est calculée à partir des lectures laser.

Maintenant nous allons définir une position cible d'une manière à imposé au robot l'évitement d'obstacle que nous avons créé dans la simulation.

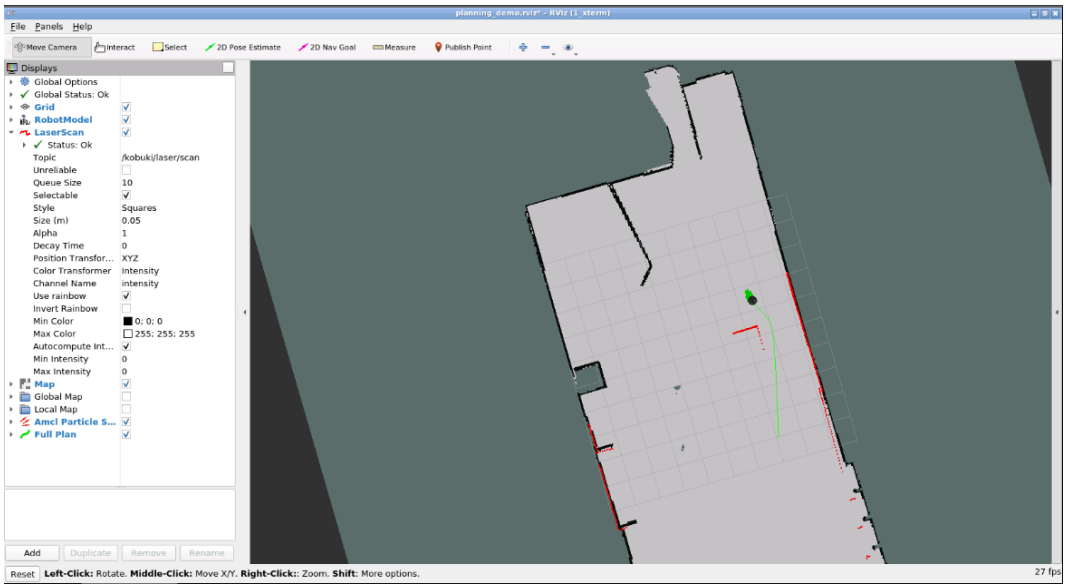

FIGURE 3.30: Évitement d'obstacle.

Ainsi, comme nous l'avons vu dans l'exécution précédente (voir la figure 3.30), la carte des coûts locale détecte les nouveaux objets qui apparaissent dans la simulation, contrairement à la carte des coûts globale.

Cela se produit, comme nous l'avons peut-être déjà déduit, car la carte des coûts globale est créée à partir d'un fichier de carte statique. Cela signifie que la carte des coûts ne changera pas, même si l'environnement change. La carte des coûts locale, à la place, est créée à partir des lectures des capteurs du robot, elle sera donc mise à jour d'une manière permanente par les nouvelles lectures des capteurs.

Étant donné que la carte des couts globale et la carte des couts locale n'ont pas le même comportement, le fichier de paramètres doit également être différent.

3.7.4 Reconfiguration dynamique

Jusqu'à présent, nous avons vu comment changer les paramètres en les modifiant dans les fichiers de paramètres. Mais, ce n'est pas la seule façon de modifier les paramètres, nous pouvons également modifier les paramètres dynamiques à l'aide de l'outil **rqt_reconfigure**, par l'exécution de la commande suivante :

/home/simulations/public_sim_ws/srs/all/turtlebot_navigation_gazebo$ rosrun rqt_reconfigure rqt_reconfigure

Voici l'interface **rqt_reconfigure** illustrée dans la figure 3.31

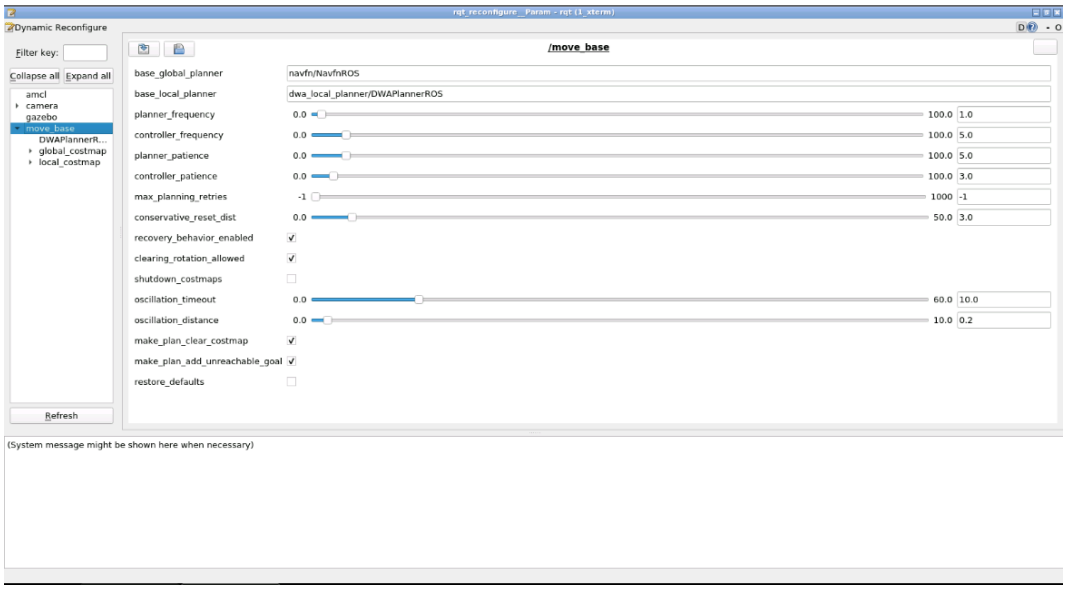

FIGURE 3.31: L'interface **rqt_reconfigure**.

3.7.5 Synthèse

3.7.5.1 Le nœud move_base

Le nœud move_base est, fondamentalement, le nœud qui coordonne tout le système de planification de chemin. Il prend une position cible en entrée et génère les commandes de vitesse nécessaires pour déplacer le robot d'une position initiale à la position cible spécifiée. Pour ce faire, le nœud de base de déplacement gère tout un processus interne où il se déroule pour différentes parties :

— Planificateur global.

— Planificateur local.

— Cartes de coûts.

— Comportements de récupération.

3.7.5.2 Le planificateur global

Lorsqu'un nouvel objectif est reçu par le nœud move_base, il est immédiatement envoyé au planificateur global. Le planificateur global calculera alors un chemin sûr que le robot utilisera pour arriver à l'objectif spécifié. Le planificateur global utilise les données de la carte de coûts globale pour calculer ce chemin.

Il existe différents types de planificateurs globaux. Selon notre configuration, nous utiliserons l'un ou l'autre.

3.7.5.3 Le planificateur local

Une fois que le planificateur global a calculé une trajectoire pour le robot, celle-ci est envoyée au planificateur local. Ce dernier exécutera alors ce chemin, le divisant en blocs plus petits (locaux). Alors, étant donné un plan à suivre et une carte, le planificateur local fournira des commandes de vitesse afin de déplacer le robot. Le planificateur local fonctionne sur une carte de coûts locale.

Il existe différents types de planificateurs local. Selon le type de performance dont nous avons besoin, nous utiliserons l'un ou l'autre.

3.7.5.4 Cartes de coûts

Les cartes de coûts sont essentiellement, des cartes qui représentent les points de la carte où le robot peut être en sécurité et ceux qui ne le soit pas. Il existe 2 types de cartes de coûts :

— Carte des coûts globale.

— Carte des coûts locale.

Fondamentalement, la différence entre eux est que la carte des coûts globale est construite à l'aide des données d'une carte statique précédemment construite, tandis que la carte des coûts locale est construite à partir des lectures des capteurs du robot.

3.7.5.5 Comportements de récupération

Les comportements de récupération fournissent des méthodes au robot en cas de blocage. La pile de navigation fournit 2 comportements de récupération différents :

— Récupération par rotation.

— La suppression d'une carte des coûts (clear costmap).

3.7.5.6 Configuration

Comme de nombreux nœuds différents travaillent ensemble, le nombre de paramètres disponibles pour configurer les différents nœuds est également très élevé.

Les fichiers de paramètres dont nous aurons besoin sont les suivants :

— **move_base_params.yaml**

— **global_planner_params.yaml**

— **local_planner_params.yamls**

— **common_costmap_params.yaml**

— **global_costmap_params.yaml**

— **local_costmap_params.yaml**

Outre les fichiers de paramètres présentés ci-dessus, nous aurons également besoin d'un fichier de lancement afin de lancer l'ensemble du système et de charger les différents paramètres.

3.7.6 Conclusion

En résumé, voici comment se déroule toute la méthode de planification de parcours :

Après avoir obtenu la position actuelle du robot, nous pouvons envoyer une position cible au nœud **move_base**. Ce nœud enverra ensuite cette position cible à un planificateur global qui planifiera un chemin de la position actuelle du robot à la position cible. Ce plan concerne la carte des coûts globale, qui est alimentée par le serveur de carte.

Le planificateur global enverra alors ce chemin au planificateur local, qui exécute chaque partie du plan global. Le planificateur local obtient l'odométrie et les valeurs des données laser et trouve un plan local sans collision pour le robot. Le planificateur local est associé à la carte des coûts locale, qui peut surveiller le ou les obstacles autour du robot. Le planificateur local génère les commandes de vitesse et les envoie au contrôleur de base qui convertira ensuite ces commandes en mouvement réel du robot.

Si le robot est coincé quelque part, les nœuds de comportement de récupération, telles que la récupération de suppression de la carte des coûts ou la récupération par rotation, seront appelées.

Conclusion générale

Dans ce mémoire, nous avons introduit les notions génerales sur les robots mobiles tels que les types d'applications, l'environnement d'un robot mobile et ses différents milieuxs de navigation visuels. Par la suite, nous avons vu des concepts théoriques concernant la navigation ROS. Enfin, nous avons appliqué les différents concepts de la navigation ROS à travèrs une simulation.

ROS est un système de contrôle très puissant. Il existe une tendance croissante pour prendre en charge une plus grande variété de robots par le système ROS. Nous avons utilisé le système de navigation ROS pour simuler un robot dans notre environnement et pour nous apporter la sensation de démonstrations réelles. Dans ce mémoire, nous avons présenté le système de navigation ROS et expliqué comment utiliser la navigation ROS dans la plateforme ROSDS. Le système ROS peut être intégré sur n'importe quel robot avec des modifications minimes et peut fonctionner indépendamment. Nous avons utlisé la pile de navigation pour visualiser les données publiées du robot et créer une carte de notre espace de navigation. Une fois la cartographie et la localisation effectuées avec succès, la navigation peut être facilement réalisée. La pile de navigation ROS utilise des plugins pour la planification locale et globale, ce qui rend ROS très utile et très simple à naviguer dans des environnements indéfinis. Dans ce projet, le package move_base a été utilisé pour implémenter la planification de chemin, ce qui a permis une navigation autonome. Le sous-

système de construction et de localisation de la carte est basé sur l'algorithme gmapping SLAM, qui crée progressivement une carte de l'environnement pendant le mouvement du système.

Dans des travaux futurs, nous pouvons intégrer ce système de navigation ROS dans un véritable robot. Nous pouvons améliorer ce système de navigation en utilisant des données 3D pour appliquer sur les véhicules réels. Nous pouvons réduire le temps de balayage de notre environnement par l'utilisation de plusieurs robots en même temps. Nous pouvons améliorer notre navigation où un robot doit être conscient de la signification des objets rencontrés dans l'environnement et préciser correctement laquelle doit le robot prendre en compte lorsqu'il prend des décisions de navigation. L'interprétation de scène pour les robots mobiles est un problème difficile pour les robots. Avec la vision, de tels robots pourraient éventuellement s'engager dans une navigation plus intelligente et une interaction plus intelligente avec les personnes et les objets dans l'environnement.

Bibliographie

[1] Ahmed Akili. "etat de l'art sur la robotique et solution adoptée". In : *Conception et réalisation d'un robot mobile sans Fil À base d'arduino*. Éditions universitaires européennes, 2017, p. 5-7.

[2] Jean-Clément Devaux, Hicham Hadj-Abdelkader et Etienne Colle. "Toolbox d'étalonnage pour Kinect: Application à la fusion d'une Kinect et d'un télémètre laser". In : *Reconnaissance de Formes et Intelligence Artificielle (RFIA) 2014*. 2014.

[3] Carol Fairchild et Thomas L Harman. *ROS Robotics By Example: Learning to control wheeled, limbed, and flying robots using ROS Kinetic Kame*. Packt Publishing Ltd, 2017.

[4] BOUSIF Fares. "Conception, réalisation et implémentation d'un robot mobile dans un environnement de travail". Thèse de doct. Université Abou-Bekr Belkaid, 2020.

[5] Georges Giralt, Ralph Sobek et Raja Chatila. "A multi-level planning and navigation system for a mobile robot: a first approach to Hilare". In : *Proceedings of the 6th international joint conference on Artificial intelligence-Volume 1*. 1979, p. 335-337.

[6] Charlotte Heggem, Nina Marie Wahl et Lars Tingelstad. "Configuration and Control of KMR iiwa Mobile Robots using ROS2". In : *2020 3rd International Symposium on Small-scale Intelligent Manufacturing Systems (SIMS)*. IEEE. 2020, p. 1-6.

[7] Luc Jaulin. *La robotique mobile: cours et exercices*. ISTE Group, 2015.

[8] Frédéric Large. "Navigation autonome d'un robot mobile en environnement dynamique et incertain". Thèse de doct. Université de Savoie, 2003.

[9] David V Lu, Dave Hershberger et William D Smart. "Layered costmaps for context-sensitive navigation". In : *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2014, p. 709-715.

[10] Rajesh Kannan Megalingam, Anandu Rajendraprasad et Sakthiprasad Kuttankulangara Manoharan. "Comparison of planned path and travelled path using ROS navigation stack". In : *2020 International Conference for Emerging Technology (INCET)*. IEEE. 2020, p. 1-6.

[11] Emmanuel A Oyekanlu et al. "A review of recent advances in automated guided vehicle technologies: Integration challenges and research areas for 5G-based smart manufacturing applications". In : *IEEE access* 8 (2020), p. 202312-202353.

[12] Fagner Pimentel et Plinio Aquino. "Performance evaluation of ROS local trajectory planning algorithms to social navigation". In : *2019 Latin American Robotics Symposium (LARS), 2019 Brazilian Symposium on Robotics (SBR) and 2019 Workshop on Robotics in Education (WRE)*. IEEE. 2019, p. 156-161.

[13] Francisco Rubio, Francisco Valero et Carlos Llopis-Albert. "A review of mobile robots: Concepts, methods, theoretical framework, and applications". In : *International Journal of Advanced Robotic Systems* 16.2 (2019), p. 1729881419839596.

[14] Roland Siegwart, Illah Reza Nourbakhsh et Davide Scaramuzza. *Introduction to autonomous mobile robots*. MIT press, 2011.

[15] Ricardo Tellez, Alberto Ezquerro et Miguel Angel Rodriguez. *Ros navigation in 5 days: Entirely practical robot operating system training*. Independently published, 2017.

[16] Sebastian THRUN. "Probabilistic robotics". In : *Communications of the ACM* 45.3 (2002), p. 52-57.