République Algérienne Démocratique et Populaire

Ministère de L'Enseignement Supérieur et de la Recherche Scientifique

Université Mouloud Mammeri Tizi-Ouzou

Faculté Génie de la Construction

Département Electromécanique

Mémoire de Fin d'études

En vue de l'obtention du diplôme de **Master en Electromécanique**

Spécialité : Maintenance Industrielle

Thème :

Planification de Trajectoire d'un Robot Mobile

Réalisé par :

Mme.Oubaziz Houria

M. Boutalbi Mohamed

Encadré par :

M. OUELMOKHTAR Hand

Soutenu le 26 Juin 2024, Devant le jury composé de :

Mme.	Samira OUSSIDHOUM	MCB	à l'UMMTO	Présidente
M.	Hocine BELGAID	MCB	à l'UMMTO	Examinateur
M.	Hand OUELMOKHTAR	MCB	à l'UMMTO	Promoteur

Promotion : 2023/2024

Dédicace

je dédie ce travail à ma famille qui m'a soutenue et qui a sacrifié les plus belles années de leur vie pour me voir réussir.

A mes chers parents à qui je souhaite une bonne santé.

Ames chères sœurs et frères.

A tout mes amies et amis.

A toute personne qui m'a soutenue de prés ou de loi durant ce travail.

Remerciement

Si l'accomplissement de ce mémoire sollicite une bonne dose de détermination, de rigueur et de passion, ce travail de recherche a nécessité la contribution de plusieurs personnes que je tiens à remercier.

Nous remercions d'abord le Bon Dieu le tout Puissant de nous avoir accordé la santé et le courage pour accomplir ce travail.

Nous remercions notre promoteur M. OUELMOKHTAR Hand pour ses orientations et son encadrement.

Nous remercions également nos enseignants pour leur soutien fort remarqué durant notre formation académique.

Nous remercions les membres du jury pour le temps qu'ils nous ont consacré pour la lecture et l'évaluation de ce travail. Nos remerciements s'adressent aussi à toute personne qui a contribué de près ou de loin à l'élaboration de notre travail.

Abstract

The present manuscript deals with the problem of path planning for a ground robot. In this context, we used a global method that ensure optimal and safe movement of the robot within a given space. We began by modeling the environment using the cell decomposition method. Then, we transformed it into a graph G(V,E). Next, we employed a dynamic programming algorithm called Floyd-Warshall, which enabled us to: 1) calculate the shortest paths, and 2) establish the sequences of moves between vertices of V by traversing those arcs of E. Our approach was validated with one of the most realistic robot simulators, called ROS (Robot Operating System), which was applied to the TurtleBot3-waffle robot. Tests and simulations carried out in various navigation environments proved the effectiveness of our approach.

Résumé

Ce mémoire traite la problématique de planification de la trajectoire d'un robot terrestre. Dans ce cadre, nous avons choisi une méthode globale visant à assurer un déplacement optimal et sûr du robot dans une surface prédéfinie. Dans cette perspective, Nous avons commencé par modéliser l'environnement via la méthode de décomposition cellulaire. Puis, nous l'avons transformé en un graphe G(V,E), ou V sont les noeuds (points dans l'espace, et V sont les aretes reliant ces Noeuds). Ensuite, nous avons employé un algorithme à programmation dynamique dénommé Floyd-Warshall qui nous a permis de : 1) calculer les chemins les plus courts, et 2) établir les séquences des déplacements entre sommets de V en parcourant ces arcs de E. Notre approche a été validée avec un des simulateurs de robots les plus réalistes appelé ROS (Robot Operating System), que nous avons appliqué au robot TurtleBot3-burger. Les tests et les simulations effectués dans divers environnements de navigation ont démontré l'efficacité de notre approche.

ملخص

تتناول هذه الأطروحة مشكلة تخطيط مسار الروبوت الأرضي. وفي هذا السياق، اخترنا طريقة شاملة تهدف إلى ضمان الحركة المثلى والآمنة للروبوت في منطقة محددة مسبقاً. بدأنا بنمذجة البيئة باستخدام طريقة تحليل الخليوي. ثم قمنا بتحويلها إلى رسم بياني (ط،ز). بعد ذلك استخدمنا خوارزمية برمجة ديناميكية تسمى فلويد-وارشال والتي مكنتنا من ١) حساب أقصر المسارات، و٢) تحديد تسلسل الحركات بين رؤوس ط من خلال اجتياز هذه الأقواس من ز. تم التحقق من صحة منهجنا باستخدام أحد أكثر برامج محاكاة الروبوتات واقعية، والذي يسمى ن ت ر (نظام تشغيل الروبوت)، والذي طبقناه على روبوت السلحفاة الآلي ٣ الوافل. أثبتت الاختبارات والمحاكاة التي أجريت في بيئات ملاحة مختلفة فعالية نهجنا.

Table des matières

Table des figures **i**

Introduction Générale **1**

1 État de l'Art **3**
1.1 Introduction . . . 3
1.2 Définitions . . . 4
1.3 Historique . . . 4
1.4 La robotique mobile : . . . 9
1.4.1 Domaine d'application : . . . 10
1.4.2 Caractéristiques des robots mobiles . . . 12
1.4.3 classification . . . 13
1.4.4 Architecture d'un robot mobile : une synergie de composants . . . 16
1.4.5 Composants des robots les plus fréquents : diversité et spécialisation 17
1.4.6 Applications des robots mobiles : un éventail de possibilités . . . 18
1.5 planification de trajectoire . . . 19
1.5.1 type de planification de trajectoire . . . 19
1.5.2 Les modes de planification de trajectoire . . . 21
1.5.3 méthode de modélisation d'environnement . . . 22
1.5.4 Méthode de planification des chemins : . . . 24
1.6 Conclusion . . . 27

2 Environnement de simulation : ROS & Turtlebot3 **28**
2.1 Introduction . . . 28
2.2 Robot Operating System . . . 29
2.2.1 Le système de fichiers de ROS . . . 29
2.2.2 Les notions de base de ROS . . . 30
2.2.3 Les outils utiles dans ROS . . . 32
2.3 TurtleBot3 . . . 32

2.3.1 Description de turtlebot3 . . . 33
2.3.2 Les modèles de turtlebot3 . . . 34
2.3.3 Turtlebot3 Burger : caractéristiques techniques . . . 36
2.4 Conclusion . . . 37

3 Modélisation de l'environnement 38
3.1 Introduction . . . 38
3.2 Cartographie de l'environnement : . . . 39
3.3 Transformation de la map en image binaire : . . . 43
3.4 Création de graph G(V.E) : . . . 43
3.4.1 Création de l'ensemble des noeuds V : . . . 43
3.4.2 création des arcs E : . . . 45
3.4.3 Modélisation du Graphe G(V, E) . . . 46
3.5 Conclusion . . . 47

4 Calcul d'itinéraire : Simulation & Resultat 48
4.1 Introduction . . . 48
4.2 L'implémentation de l'Algorithme de Floyd-Warshell . . . 48
4.2.1 Algorithme de Planification "Floyd-Warshall" . . . 48
4.3 Mise en œuvre sur ROS . . . 49
4.3.1 Détermination de la trajectoire . . . 49
4.3.2 Serveur de Service ROS . . . 52
4.3.3 Contrôleur du Robot . . . 53
4.4 Résultat et Discussion . . . 54
4.4.1 Test et Simulation . . . 54
4.4.2 Comparaison des résultats . . . 58
4.5 Conclusion . . . 59

Conclusion Générale 60

Bibliographie 62

Table des figures

1.1 les disciplines dans un robot . . . 3
1.2 Chevalier mécanique . . . 5
1.3 Premier robot Breveté . . . 5
1.4 Premier Robot Mobile Shakey . . . 6
1.5 Standford Cart . . . 7
1.6 Mars Pathfinder . . . 7
1.7 le robot Stanley développé par l'université de Stanford . . . 8
1.8 le Robot Watson . . . 8
1.9 Le Robot Sophia . . . 9
1.10 Robot d'intervention . . . 10
1.11 Robot de service . . . 11
1.12 Robot à usage personnel . . . 12
1.13 Véhicules aériens sans pilote . . . 13
1.14 Véhicules de surface sans pilote . . . 14
1.15 Véhicules sous-marins sans pilote . . . 15
1.16 Véhicules terrestres sans pilote . . . 16
1.17 planification de trajectoire . . . 21
1.18 Graphique de visibilité . . . 22
1.19 Diagramme voronoïde . . . 23
1.20 Décomposition cellulaire . . . 23

2.1 Turtlebot3 modèle Burger et Symbole de ROS . . . 29
2.2 Schéma explicatif communication entre nœuds, topics et services . . . 31
2.3 Turtlebot3 modèle Burger et Waffle_pi . . . 33
2.4 Turtlebot3 modèle Waffle . . . 34
2.5 Turtlebot3 modèle Waffle_pi . . . 35
2.6 Turtlebot3 modèle Burger . . . 35

3.1 l'image au début de la modélisation et a la fin de la modélisation . . . 38

3.2 Exportation du modèle de Turtlebot3 en Burger 39
3.3 Déploiement de Turtlebot3 39
3.4 lancement de slam 40
3.5 lancement de teleop.key 41
3.6 la carte scannée 41
3.7 l'enregistrement de la map final 42
3.8 les nœuds utilisés 42
3.9 Première étape de modélisation 43
3.10 Division cellulaire 44
3.11 Schéma explicatif sur le teste d'existence 45
3.12 Schéma final de teste global de tous les points adjacents 46
3.13 figure montrant la génération graph G(V,E) 47

4.1 organigramme de l'algorithme de Floyd-Warshall 49
4.2 Position initiale du robot 50
4.3 transformation de coordonnées et obtention les coordonnées d'arrivée . . . 50
4.4 les deux repères du programme Op (xp, yp) et de gazebo Og (xb, yb) . . . 51
4.5 génération de la séquence de point de la trajectoire 51
4.6 Trajectoire du robot dans le repere OP 52
4.7 le serveur ROS a récupéré la séquence est prêt à envoyer s'il y a une requête 52
4.8 le deuxième nœud a reçu une requête les coordonnées sont bien envoyer . . 53
4.9 nœud contrôleur du robot : envoie la demande de requete 53
4.10 nœud contrôleur du robot : reception de la séquence 54
4.11 navigation dans un espace libre 55
4.12 programme de ROBOTIS 56
4.13 image montrant les nœuds générés par le programme 57
4.14 extrait de la matrice des prédécesseurs 57
4.15 génération de la séquence des points le nœuds 1 58
4.16 schéma explicatif de communication entre nœuds ros 58
4.17 mouvement du robot avec le programme de robotis (-c-) et notre programme (-d-) 59

Introduction Générale

La robotique mobile constitue un domaine pluridisciplinaire visant la conception, la fabrication et l'utilisation de robots capables de se déplacer de manière autonome. Ces robots, également appelés "véhicules sans pilote", sont classés selon leur environnement de navigation : aérien, terrestre, de surface, maritime et sous-marin.

Dans cette étude, nous nous concentrons sur les "Véhicules Terrestres Sans Pilote", communément désignés sous le terme anglais UGV (Unmanned Ground Vehicles). Les UGV sont des robots qui se déplacent à la surface terrestre, trouvant des applications diversifiées allant de l'usage domestique aux interventions dans des situations critiques requérant une autonomie totale. Dans ce contexte, la planification de trajectoire demeure une tâche primordiale.

La planification de trajectoire peut être divisée en trois niveaux : la planification de l'itinéraire, considérant le robot comme un point dans son environnement ; la planification de la trajectoire, prenant en compte sa géométrie globale ; enfin, la planification des mouvements, où l'on considère rigoureusement les modèles géométriques, cinématiques et dynamiques du robot. Ce dernier niveau est crucial pour l'asservissement et la régulation des mouvements, relevant davantage du suivi de chemin que de la recherche de chemin.

Notre travail se focalise sur l'amélioration d'un programme permettant le déplacement d'un robot, par une recherche de chemin off-line, visant à trouver en amont le trajet le plus court tout en évitant les collisions avec des obstacles statiques. Cette étude est mise en œuvre dans un simulateur de robots appelé ROS (Robot Operating System), reconnu pour sa performance, sa richesse en plateformes et ses exemples de simulations de plus en plus réalistes.

Nous utilisons comme exemple la plateforme robotique TurtleBot3_Burger, un robot non holonome à deux roues motorisées et une roue libre, intégrant plusieurs programmes sous forme de nœuds ROS permettant son déplacement. Parmi ces nœuds, Teleop_Key permet le contrôle du robot via le clavier, tandis que turtlebot3_pointop_key facilite le déplacement linéaire d'un point à un autre en utilisant des coordonnées cartésiennes. Cependant, ce dernier programme montre ses limites dès qu'il y a présence d'obstacles.

Notre objectif est d'améliorer ce programme pour le rendre efficace dans tout environnement statique. Pour cela, une planification de trajectoire globale est proposée, comprenant les étapes suivantes :

Cartographie de l'environnement de navigation du robot dans Gazebo (simulateur graphique intégré dans ROS), en utilisant le nœud Gmapping disponible sur la plateforme TurtleBot3, basé sur la méthode SLAM (Simultaneous Localization and Mapping).

Transformation de cet environnement (carte 2D) via la méthode de décomposition cellulaire, afin d'obtenir un modèle numérique exploitable par les programmes informatiques.

Application de la théorie des graphes pour transformer l'environnement numérisé en un graphe G(V,E), où V représente l'ensemble des sommets (points) et E l'ensemble des arcs (chemins unitaires entre les sommets).

Utilisation de l'algorithme de Floyd-Warshall pour obtenir une base de données des séquences optimales de points (way-points) de déplacement entre chaque paire de points dans G.

Mise en œuvre de notre approche en utilisant à nouveau le nœud turtlebot3_pointop_key, où les coordonnées des way-points déterminées dans l'étape précédente sont envoyées séquentiellement, permettant au robot de parcourir en lignes droites chaque deux way-points de la séquence.

Cette étude est validée à travers divers environnements et différents points de départ et d'arrivée, démontrant l'efficacité de notre approche proposée.

Ce mémoire est structuré en quatre chapitres : le premier chapitre explore l'état de l'art, le second introduit ROS et TurtleBot3, le troisième expose notre méthodologie de modélisation environnementale, et enfin, le quatrième présente l'application de l'algorithme de planification Floyd-Warshall, ainsi que les résultats des tests sur le simulateur ROS.

Chapitre 1

État de l'Art

1.1 Introduction

La montée en puissance de la robotique sera perçue comme une avancée tournante dans le panorama de la technologie et dans celui de l'industrie contemporaine. Celle-ci, qui est une conjonction de divers domaines de la mécanique, de l'électronique, de l'automatique et de l'informatique, constitue le résultat d'années d'innovations et des inspirations humaines. De la notion initiale d'assister des ouvriers malléables ayant des malaises spécifiques, à l'extension actuelle de la robotique en tant que domaine multidisciplinaire en charge de la conception, de la réalisation et du déploiement de divers types de robots.

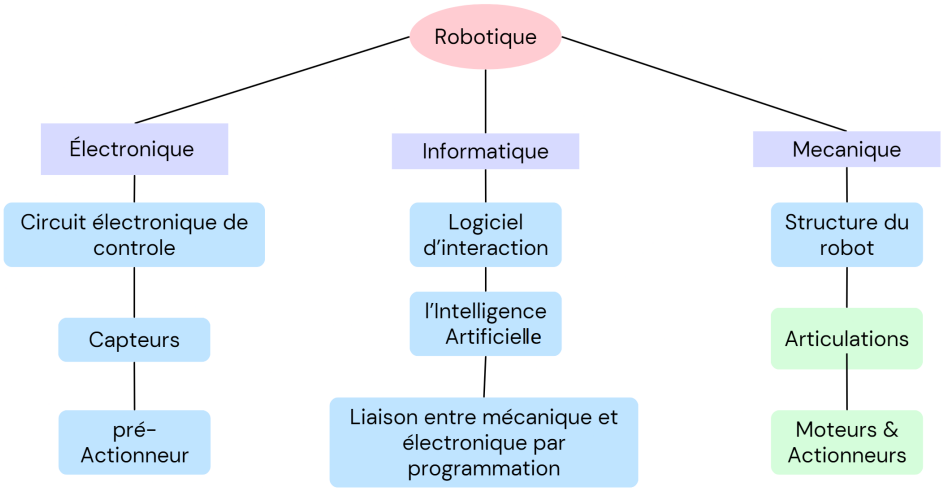

Figure 1.1: les disciplines dans un robot

1.2 Définitions

À l'origine, l'expression "robot" est issue du tchèque robota, mot qui peut signifier travail forcé ou labeur. Plus tard, il est devenu un terme générique pour désigner des machines autonomes capables d'effectuer un ensemble prédéterminé d'actions physiques.

D'après la norme ISO 8373 de l'Organisation Internationale de Normalisation, un robot est une machine polyvalente programmable apte à exécuter des tâches ou une séquence de tâches sur des pièces, des objets, des matériaux ou des milieux, sans intervention directe d'un opérateur humain dans l'exécution de la tâche proprement dite. [1]

Selon le Larousse (Dictionnaire Larousse), un robot est "une machine automatisée capable d'accomplir diverses tâches en remplaçant l'homme dans certaines opérations, notamment dans des milieux dangereux." [2]

Au Japon, un pionnier dans le domaine de la robotique, la définition est souvent plus large, voire culturellement nuancée. Pour certains, un robot est une machine capable d'effectuer des tâches physiques ou cognitives, souvent équipée de capteurs et de systèmes de contrôle avancés. [3]

Selon la "Robotics Industries Association" (RIA), un robot est "une machine programmable multifonctionnelle conçue pour déplacer des matériaux, des outils ou des dispositifs spécialisés grâce à des mouvements variables programmés en trois dimensions." [4]

1.3 Historique

L'histoire de la robotique mobile remonte à plusieurs décennies et a été marquée par des avancées significatives.

— Leonardo Da Vinci est le premier à avoir songé à réaliser un robot (figure 1.2) doté de membres coordonnant leurs mouvements, ce qui l'a amené à développer un automate nommé chevalier mécanique, capable de se tenir debout, de s'asseoir, d'agiter les bras, d'ouvrir et de refermer les mâchoires [5] [6]

Figure 1.2: Chevalier mécanique

— Au XXe siècle, les premiers robots électriques apparaissent, tout comme le robot ayant l'apparence d'un chien développé par Hammond et Miessner en 1915. [7]
— La figure 1.3 montre le premier robot breveté par George C. Devol en 1954 est commercialisé en 1961 par Unimation Inc, une compagnie de robotique créée en 1962 par Joseph F. Engelberger et George Devol. [8] [9]

Figure 1.3: Premier robot Breveté

— À l'Université de Stanford, le premier robot mobile autonome dénommé Shakey (Figure 1.4) est développé en 1969 avec la particularité de se déplacer, de percevoir son environnement et de résoudre des problèmes tels qu'évoluer dans un monde restreint avec des portes et des interrupteurs, ainsi que de planifier et d'exécuter des mouvements. [10]

Figure 1.4: Premier Robot Mobile Shakey

— En 1970, le premier robot mobile dénommé Stanford Cart(Figure1.5), capable de se déplacer sans intervention humaine grâce à une caméra et un ordinateur, est créé par l'Université du Michigan. [11]

Figure 1.5: Standford Cart

— Le premier robot mobile à roues est conçu en 1976 à l'Université de Tsukuba ; il peut se déplacer dans toutes les directions sans changer d'orientation. [12]
— La NASA a envoyé Mars Pathfinder (Figure 1.6), le premier robot mobile à explorer la planète Mars, en 1997. [13]

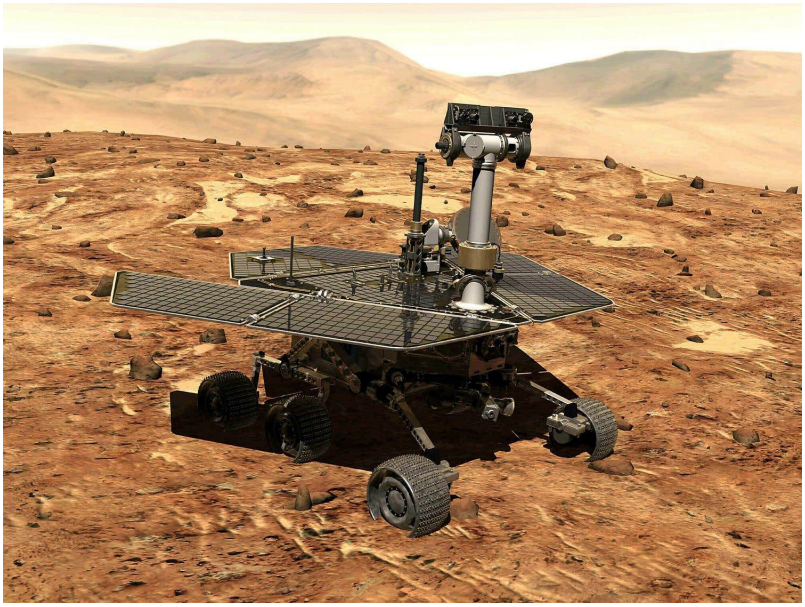

Figure 1.6: Mars Pathfinder

— En 2004, le robot Stanley (Figure1.7), développé par l'Université de Stanford, est le premier à avoir gagné le prix du Grand Challenge de la DARPA en parcourant 212 km dans le désert en 6 heures et 53 minutes. [14]

Figure 1.7: le robot Stanley développé par l'université de Stanford

— Watson est le premier robot mobile à avoir eu la chance de participer à un jeu télévisé, où il a battu les participants en répondant à des questions de culture générale.
[15]

Figure 1.8: le Robot Watson

— En 2015, Sophia, un robot possédant une apparence humaine, est le premier robot mobile à obtenir la citoyenneté ; il est fabriqué par Hanson Robotics. [16]

Figure 1.9: Le Robot Sophia

— Depuis la décennie 2010 jusqu'à aujourd'hui, grâce aux progrès notables de l'intelligence artificielle, les robots accroissent significativement leur autonomie et élargissent leurs domaines d'intervention, que ce soit dans des contextes industriels, agricoles ou domestiques. [17]

1.4 La robotique mobile :

La robotique mobile, domaine fascinant en plein essor, comprend l'élaboration, la fabrication et l'utilisation de robots capables de se déplacer de manière autonome dans leur environnement. Ces robots, munis d'un contrôleur d'intelligence artificielle et de cap-

teurs perfectionnés, explorent, interagissent et accomplissent des tâches dans des espaces extrêmement variés, repoussant les limites de l'automatisation et ouvrant un immense horizon d'opportunités dans de multiples domaines. L'application de la robotique mobile peut être classée en trois grandes catégories fondamentales : la robotique d'intervention, la robotique de service professionnel, et la robotique domestique.

1.4.1 Domaine d'application :

1.4.1.1 robotique d'intervention :

La robotique d'intervention est en général caractérisée par la téléopération du robot par un opérateur via des ordres directs (joysticks, bras maîtres, organes de commandes physiques ou virtuels). Dans cette approche, l'opérateur est donc nécessairement dans la boucle des commandes du robot pour l'aider à réaliser sa tâche. Ce type de robotique est couramment utilisé pour permettre aux robots d'accomplir une tâche dans des environnements difficiles d'accès ou hostiles (industries nucléaires, intervention sur des catastrophes naturelles, mais également exploration marine ou spatiale). [18]

Figure 1.10: Robot d'intervention

1.4.1.2 La robotique de service professionnel :

La robotique de service professionnelle est un domaine d'activité très populaire dans l'industrie, où le robot effectue une tâche à haut risque ou répétitive, ou est demandé pour des opérations nécessitant des capacités que l'opérateur humain ne pourrait pas maîtriser. [19]

Figure 1.11: Robot de service

1.4.1.3 La robotique domestique :

La robotique domestique ouvre un champ opérationnel aussi riche que celui de la robotique de service pour les professionnels. Ses applications possibles sont nombreuses, couvrant aussi bien la réalisation de tâches domestiques que l'accompagnement des personnes dépendantes, etc. [20]

Figure 1.12: Robot à usage personnel

1.4.2 Caractéristiques des robots mobiles

Les robots mobiles sont des dispositifs capables de naviguer dans divers environnements tout en accomplissant leurs tâches requises. Afin de mériter la dénomination de robot, ce dernier doit être capable de :

- Percevoir la scène qui l'entoure.
- Se localiser.
- Définir sa trajectoire.
- Contrôler ses actionneurs.
- Interagir avec d'autres robots ou avec des individus.

Les robots mobiles sont souvent répartis en fonction de leur mode de locomotion :

- Robots à roues.
- Robot aérien.
- Robots de surface.
- Robots sous-marins.
- Robots marcheurs (Humanoïde).

Selon leur degré d'autonomie, les robots mobiles peuvent être divisés en deux catégories :

- Les robots complètement autonomes, opérant d'une manière complètement indépendante.

— Les robots contrôlés à distance par le biais de la téléopération.

Les robots mobiles sont largement employés dans divers secteurs, notamment l'industrie, l'agriculture, la santé, la sécurité et la navigation spatiale, etc.

1.4.3 classification

Comme mentionné précédemment, plusieurs types de robots mobiles peuvent être rencontrés, y compris les drones et les humanoïdes. Dans le présent travail, nous nous concentrerons en particulier sur un groupe de robots mobiles connus sous le nom de « drones », soit des véhicules sans pilote (UV : Unmanned Vehicules).

1.4.3.1 Drones aériens (UAV : Unmanned Aerial Vehicles) :

Ce sont des véhicules sans pilote capables de naviguer dans l'air, de surveiller de vastes zones et d'opérer dans des environnements hostiles.

Les UAV varient en taille, poids, altitude opérationnelle et portée. Ils sont utilisés dans diverses applications telles que la surveillance en temps réel, la télédétection, la recherche et le sauvetage, la livraison de colis, l'agriculture de précision, l'inspection des infrastructures civiles, etc.

Ces drones sont confrontés à plusieurs défis, tels que : les conditions météorologiques changeantes, la présence d'obstacles, les altitudes variables, les limitations de charge utile et les influences environnementales susceptibles d'affecter leurs performances. [21]

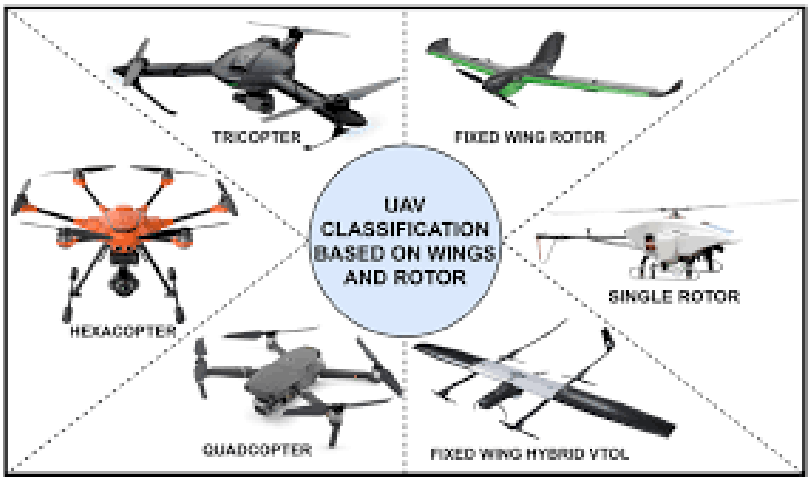

Figure 1.13: Véhicules aériens sans pilote

1.4.3.2 Drones de surface (USV : Unmanned Surface Vehicles) :

Les USV sont des navires autonomes opérant à la surface de l'eau, utilisés à l'origine à des fins navales et largement employés dans des applications civiles telles que la surveillance environnementale, le transport maritime autonome, la recherche et le sauvetage, la prospection dans les industries du pétrole et du gaz, et la cartographie des fonds marins.

Les USV doivent faire face à des conditions météorologiques et aquatiques défavorables, nécessitant une perception adéquate de l'environnement pour détecter et éviter les obstacles au-dessus et au-dessous de l'eau. [21]

Figure 1.14: Véhicules de surface sans pilote

1.4.3.3 Drones sous-marins (UUV : Unmanned Under-water Vehicles) :

Les UUV opèrent sous la surface de l'eau avec une intervention minimale de l'opérateur humain. Ils peuvent être télécommandés (ROV) ou autonomes (AUV).

Ils sont utilisés dans diverses applications telles que la surveillance persistante, la guerre anti-sous-marine, l'océanographie et la lutte contre les mines.

Les UUV font face à des défis tels que les conditions océaniques hostiles, la faible luminosité sous l'eau, les courants et la densité de l'eau, ainsi que la détection et l'évitement d'obstacles fixes ou mobiles. La navigation précise est un défi crucial, surtout en l'absence de connexion constante avec les opérateurs à distance dans les eaux profondes. [21]

Figure 1.15: Véhicules sous-marins sans pilote

1.4.3.4 Drones terrestres (UGV : Unmanned Ground Vehicles) :

Les UGV sont des systèmes sans pilote opérant sur le sol, utilisés pour diverses applications telles que l'exploration spatiale, la surveillance de l'environnement, et la recherche et le sauvetage.

ls varient en taille, poids et configuration selon leur mission dédiée, généralement équipés de capteurs pour scruter l'environnement et prendre des décisions autonomes ou les transmettre à distance à un opérateur humain.

Les UGV peuvent être entravés par des conditions environnementales telles que la poussière, la pluie, les obstacles et divers types de terrains. [21]

Figure 1.16: Véhicules terrestres sans pilote

1.4.4 Architecture d'un robot mobile : une synergie de composants

L'architecture d'un robot mobile repose sur une synergie de composants qui lui confèrent ses capacités motrices, sensorielles et cognitives :

1.4.4.1 Plateforme :

Le squelette du robot, supportant les autres composants et assurant la locomotion. Différents types existent, comme les robots à roues, à chenilles ou humanoïdes, chacun adapté à un terrain et à une tâche spécifiques.

1.4.4.2 Actionneurs :

Les muscles du robot, convertissant l'énergie en mouvement. Ils peuvent être des moteurs électriques, des servomoteurs, des actionneurs pneumatiques ou hydrauliques. Ces modèles sont les plus utilisés.

1.4.4.3 capteurs :

Les yeux et les oreilles du robot, percevant l'environnement. Ils peuvent être des caméras, des capteurs de distance, des gyroscopes, des accéléromètres, etc. Ces capteurs fournissent des informations essentielles au robot pour sa navigation.

1.4.4.4 Ordinateurs de bord :

C'est le cerveau du robot. Il coordonne les informations reçues par les capteurs et prend des décisions pour actionner les différents actionneurs en fonction de la tâche à accomplir. La complexité de ce cerveau dépend de la complexité du robot. Il peut être un simple microcontrôleur ou un système informatique d'une puissance considérable.

1.4.4.5 Logiciel : programme Informatique

Le programme informatique, l'âme du robot, doit être implémenté dans l'ordinateur de bord. C'est ce programme qui dictera le comportement du robot à un moment donné, sous certaines conditions spécifiées lors de la programmation.

1.4.5 Composants des robots les plus fréquents : diversité et spécialisation

La diversité des robots mobiles se reflète dans la variété de leurs composants, adaptés à des fonctions et environnements de navigation :

1.4.5.1 Roues :

Omniprésentes pour leur polyvalence sur des surfaces planes, les roues existent en versions motrices et directrices, permettant une grande maniabilité.

1.4.5.2 Chenilles :

Offrant une meilleure adhérence sur des terrains accidentés ou glissants, les chenilles sont idéales pour les robots tout-terrain ou d'exploration.

1.4.5.3 Jambes :

Inspirées de la locomotion humaine, les jambes confèrent aux robots humanoïdes une grande agilité et la capacité de s'adapter à des environnements complexes.

1.4.5.4 Bras manipulateurs :

Dotés d'articulations et de pinces, les bras manipulateurs permettent aux robots d'effectuer des tâches précises comme saisir des objets ou utiliser des outils.

1.4.5.5 Capteurs de vision :

Simulant la vue humaine, les caméras permettent aux robots de reconnaître des objets, suivre des trajectoires et cartographier leurs environnements.

1.4.5.6 Capteurs LiDar :

Utilisant des lasers pour mesurer la distance, les capteurs LiDAR génèrent des cartes 3D/2D précises de l'environnement, utiles pour la navigation et l'évitement d'obstacles.

1.4.5.7 Intelligence Artificielle :

Intégrant des algorithmes d'apprentissage automatique, l'IA dote les robots de capacités d'apprentissage, d'adaptation et de prise de décision autonome.

1.4.6 Applications des robots mobiles : un éventail de possibilités

Les robots mobiles trouvent leur application dans une multitude de domaines, révolutionnant la façon dont nous travaillons, vivons et interagissons avec le monde :

1.4.6.1 Exploration et recherche :

Les robots explorent des environnements dangereux ou inaccessibles aux humains, comme les fonds marins, les volcans ou les sites nucléaires contaminés.

1.4.6.2 Logistique et entreposage :

Dans les entrepôts, les robots transportent des marchandises, optimisent le stockage et automatisent les tâches répétitives, améliorant l'efficacité et la sécurité.

1.4.6.3 Agriculture :

Les robots surveillent les cultures, pulvérisent les pesticides, récoltent des produits et assistent à la gestion du bétail, contribuant à une agriculture plus précise et durable.

1.4.6.4 Médecine et Chirurgie

Les robots assistent les chirurgiens dans des opérations complexes, offrant une précision accrue et une réduction des risques pour les patients.

1.4.6.5 Sécurité et surveillance :

Les robots patrouillent dans les zones dangereuses ou sensibles, détectent les intrusions et dissuadent les activités criminelles, renforçant la sécurité et la protection.

1.4.6.6 Service à la clientèle :

Dans les magasins et les espaces publics, les robots répondent aux questions, guident les clients et fournissent une assistance, améliorant l'expérience client.

1.5 planification de trajectoire

La planification des chemins est un problème non déterministe à temps polynomial (NP-difficile ou NP-Hard). La complexité de ce problème croît avec le nombre de degrés de liberté du système et/ou les dimensions de la navigation. Le chemin optimal doit être déterminé en prenant en compte diverses contraintes et conditions. Par exemple, il peut s'agir du chemin le plus court entre les points de départ et d'arrivée, ou du temps minimum de déplacement tout en évitant les collisions. [22]

1.5.1 type de planification de trajectoire

La tâche de planification de trajectoire reste le problème le plus difficile lors du processus de mise en œuvre des robots mobiles. Elle peut être utilisée dans des environnements entièrement connus ou partiellement connus, ainsi que dans des environnements entièrement inconnus où les informations sont reçues à partir de capteurs embarqués et mettent à jour les cartes environnementales pour informer le mouvement souhaité du robot. Les algorithmes de planification de trajectoire sont différenciés en fonction de la connaissance environnementale disponible. Ainsi, la planification de trajectoire peut être globale ou locale. [22]

1.5.1.1 Planification globale

Les méthodes de planification de trajectoire globale, également appelées méthodes hors ligne, génèrent des trajectoires à partir d'informations environnementales totalement

connues, où la position des formes et des obstacles est prédéterminée sur le modèle de carte globale. Le modèle de carte globale consiste à transformer l'espace physique en configuration virtuelle en le divisant en plusieurs sous-espaces en tenant compte des contours géométriques tels que la longueur, la largeur, la surface, les obstacles, etc. Les techniques les plus utilisées sont : le graphe de visibilité, les diagrammes de Voronoï, la grille d'occupation régulière ou la décomposition cellulaire. Une fois la carte de l'environnement établie, la planification globale du trajet est effectuée à l'aide d'algorithmes de recherche de chemin. Il existe de nombreuses techniques utilisées dans la littérature pour résoudre les problèmes de planification de trajectoire globale, les plus utilisées étant : les méthodes basées sur la grille et les algorithmes de recherche intelligente. [22] Ce problème a été étudié dans un nombre important de travaux de recherche. Les algorithmes développés peuvent être divisés en trois catégories :

1. Les algorithmes basés sur la recherche de graphes : comprennent principalement l'algorithme de Dijkstra et l'algorithme A*.
2. Les algorithmes d'échantillonnage aléatoire.
3. Les algorithmes bioniques intelligents.

1.5.1.2 Planification locale

D'autre part, la méthode de planification de trajectoire locale suppose que la position de l'obstacle dans l'environnement est inconnue et que le robot mobile ne perçoit son environnement et son état qu'à travers des capteurs intégrés. Comme il est impossible d'obtenir des informations globales sur l'environnement, la planification locale des trajectoires se concentre sur les informations locales actuelles sur l'environnement du robot mobile et utilise les informations sur l'environnement local obtenues par les capteurs pour trouver une trajectoire optimale entre le point de départ et le point cible, qui ne touche pas l'obstacle dans l'environnement. La stratégie de planification de trajectoire doit être ajustée en temps réel. Les méthodes couramment utilisées pour la planification locale de la trajectoire comprennent la fenêtre roulante, le champ potentiel artificiel et divers algorithmes intelligents. [22]

— Il existe de nombreux algorithmes de planification locale pour obtenir un chemin optimal tels que :
— La logique floue.
— L'algorithme PSO chaotique.
— L'algorithme de recuit simulé.

1.5.2 Les modes de planification de trajectoire

Les modes de planification de trajectoire sont définis en fonction de l'échelle de visualisation de la carte, et ils se distinguent par la satisfaction de critères spécifiques appelés modalités. Trois modes principaux sont identifiés (figure 1.17) : [22]

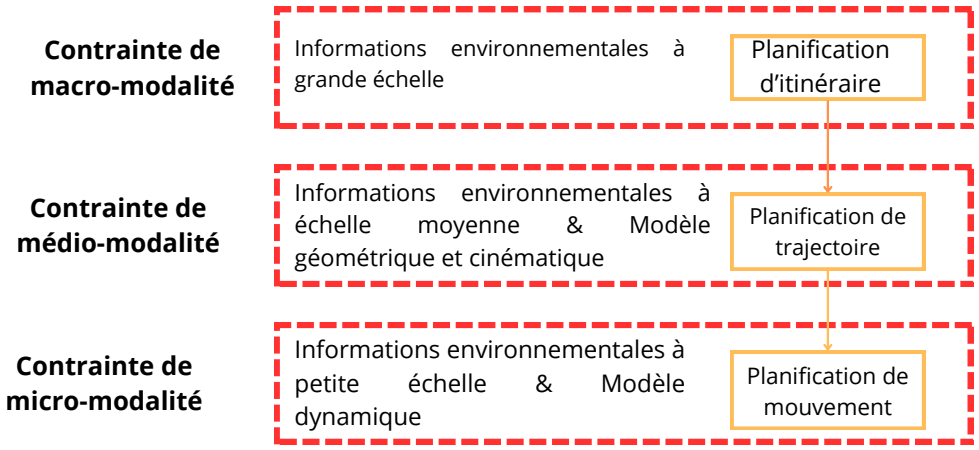

Figure 1.17: planification de trajectoire

1.5.2.1 Planification d'itinéraire :

Ce mode s'adresse aux contraintes de macro-modalité. Il vise à définir un itinéraire global, satisfaisant les contraintes à une échelle plus large. [22]

1.5.2.2 Planification de trajectoire :

Cette modalité se concentre sur les contraintes de moyenne modalité. Elle a pour objectif de déterminer une trajectoire plus détaillée, prenant en compte des contraintes à une échelle intermédiaire. [22]

1.5.2.3 Planification de mouvement :

Ce stade final de la planification de trajectoire vise à satisfaire les contraintes de micro-modalité. Il se concentre sur la planification précise des mouvements, prenant en compte des contraintes à une échelle fine. [22]

1.5.3 méthode de modélisation d'environnement

Pour permettre à un robot mobile de se déplacer d'un point initial à un point final, il est nécessaire de planifier des trajectoires. La planification de trajectoire est un sujet fréquemment traité dans le domaine scientifique. Il existe donc de nombreux algorithmes différents permettant de réaliser cette tâche. La recherche de chemin pour le robot mobile dans un environnement se transforme en un problème de recherche de chemin pour un point dans un "certain espace" caractéristique du problème. [22]

1.5.3.1 Graphique de visibilité :

La méthode du graphe de visibilité est une technique de planification de mouvement qui consiste à transformer l'espace de navigation en une cartographie à deux dimensions. Les obstacles doivent être représentés par des polygones convexes, et chaque nœud de polygone représente un sommet. Le graphe est généré en reliant chaque sommet à un autre sommet visible. Les sommets seront les candidats de l'algorithme de recherche du chemin le plus court lors de la planification du chemin optimal final. [22]

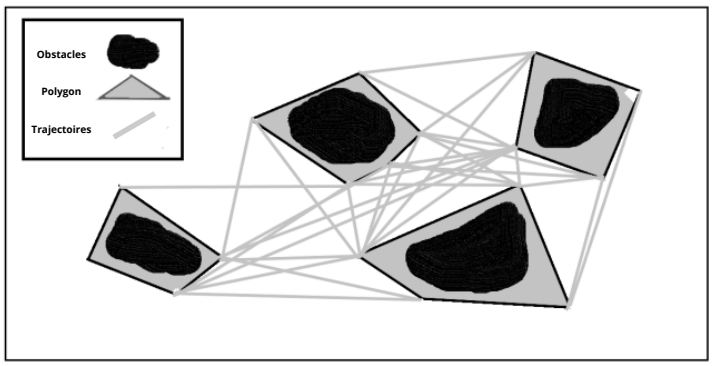

Figure 1.18: Graphique de visibilité

1.5.3.2 Diagramme voronoïde

Le diagramme de Voronoï est une technique de planification de mouvement qui permet de représenter l'espace de navigation en une cartographie à deux dimensions. Les diagrammes de Voronoï utilisent des polygones pour représenter les obstacles. Le graphe est construit en générant des arêtes équidistantes entre chaque paire de sommets de polygones les plus proches (arêtes de Voronoï), tandis que les points d'intersection sont les sommets du graphe. Le robot doit suivre les arêtes de Voronoï pour éviter les collisions, tandis que la marge entre les segments d'arc de Voronoï et les polygones permet au véhicule de maintenir une distance de sécurité par rapport aux obstacles. [22]

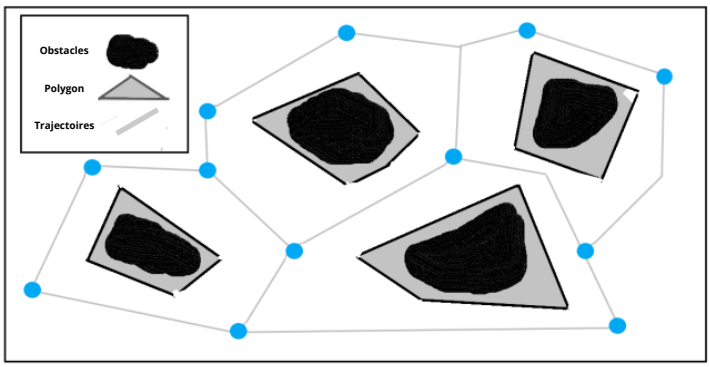

Figure 1.19: Diagramme voronoïde

1.5.3.3 Décomposition cellulaire :

La méthode de décomposition cellulaire est une technique de planification de mouvement qui consiste à représenter l'espace de recherche comme une superposition de grilles cartésiennes 2D appelées cellules. Le centre de chaque cellule représente un sommet et le bord reliant deux sommets adjacents non obstrués est considéré comme un arc. L'objectif est de fournir une séquence sans obstacle du point de départ au point d'arrivée en utilisant des techniques de recherche globale. [22]

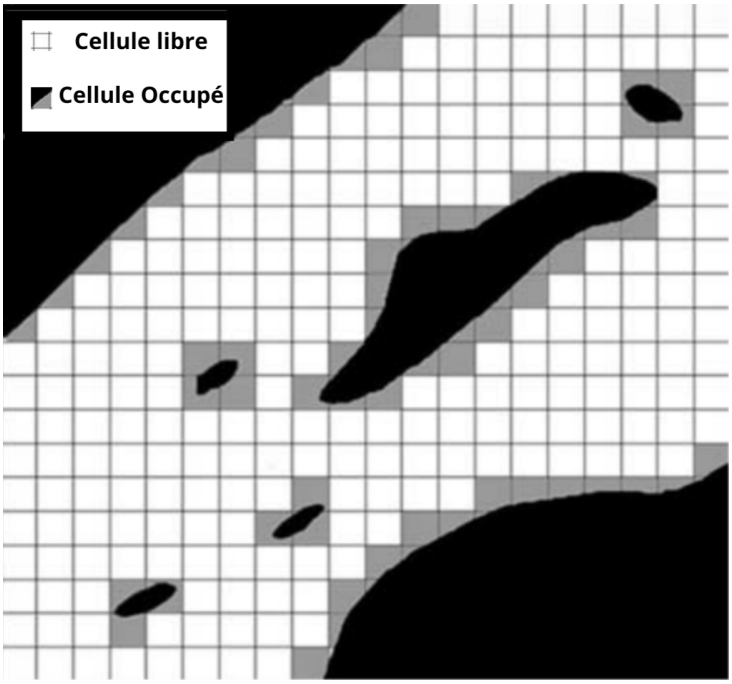

Figure 1.20: Décomposition cellulaire

1.5.4 Méthode de planification des chemins :

Les problèmes de planification de trajectoire sont connus sous le nom de NP-Hard, ce qui signifie que les techniques de résolution exactes ne parviennent généralement pas à évaluer l'optimalité dans des délais acceptables pour des problèmes de taille pratique. C'est pourquoi les techniques heuristiques sont devenues une priorité dans la résolution des problèmes de planification des trajets. Les algorithmes les plus utilisés sont les algorithmes basés sur les grilles, les algorithmes évolutionnaires et les algorithmes de recherche intelligente. [22]

1.5.4.1 Méthodes exactes :

Les méthodes exactes sont des algorithmes qui garantissent de trouver la solution optimale, c'est-à-dire le chemin le plus court ou le plus économique, selon certains critères. Parmi ces méthodes, on trouve :

1.5.4.2 Méthodes analytiques :

Les méthodes analytiques sont des approches mathématiques rigoureuses pour trouver des solutions exactes aux problèmes de planification de chemin.

— **Méthode de Newton :** Utilisée pour trouver les zéros d'une fonction dérivable, mais elle n'est pas typiquement utilisée pour la planification de chemin. Elle est plus souvent appliquée dans l'optimisation continue. [23]

1.5.4.3 Méthodes dynamiques :

Les méthodes dynamiques sont des algorithmes qui peuvent s'adapter à des changements dans l'environnement ou qui utilisent des structures de données et des techniques pour optimiser les calculs.

— **Algorithme de Bellman-Ford :** Résout le problème de recherche du plus court chemin depuis une source unique, même si des arcs ont des poids négatifs. Il permet de détecter l'existence de circuits absorbants (poids total strictement négatif). Cependant, s'il y a un circuit absorbant accessible depuis la source, il n'y a pas de solution. [24]

— **L'algorithme de Floyd Warshall :**
Algorithme de programmation dynamique utilisé pour résoudre tous les problèmes de plus court chemin par paire, similaire à l'algorithme de Dijkstra et à l'algorithme de Bellman-Ford. La seule différence est que les algorithmes de Dijkstra et de Bellman-Ford sont des algorithmes de plus court chemin à source unique, alors que

l'algorithme de Floyd-Warshall est un algorithme de plus court chemin pour toutes les paires. Il peut calculer le plus court chemin entre chaque paire de sommets dans le graphe. Floyd-Warshall est un algorithme basé sur la programmation dynamique qui décompose le problème en sous-problèmes plus petits, puis résout chaque sous-problème unitaire. Il combine ensuite les résultats pour résoudre le problème global tout en stockant le résultat de chaque sous-problème pour référence ultérieure. Il est également connu sous le nom d'algorithme de Floyd, d'algorithme de Roy-Floyd, d'algorithme de Roy-Warshall et d'algorithme WF [25]

l'utilisation de l'algorithme :

L'algorithme de Floyd Warshall est l'algorithme du plus court chemin pour toutes les paires, utilisé pour trouver le plus court chemin ou la plus courte distance entre toutes les paires de sommets d' ;un graphe donné. L'algorithme est applicable aux graphes dirigés et non dirigés, mais il échoue lorsque le graphe présente des cycles négatifs ; un cycle négatif signifie que la somme des arêtes d'un cycle est négatif.

fonctionnement :

Il initialise une matrice de distance D où D[i][j] représente la distance entre le sommet i et le sommet j. Si i est égal à j, alors D[i][j]=0. Si i et j sont directement connectés par une arête, alors D[i][j] est le poids de cette arête. Sinon, D[i][j] est initialisé à l'infini. Pour chaque sommet k du graphe, on met à jour la matrice de distance de manière à vérifier si le chemin passant par k entre les sommets i et j est plus court que le chemin déjà connu. Cela se fait en appliquant la formule : D[i][j]=min(D[i][j],D[i][k]+D[k][j]) Cette étape est répétée pour tous les triplets (i,j,k) ce qui permet d'améliorer progressivement les distances entre toutes les paires de sommets.

1.5.4.4 Méthodes heuristiques :

1. **Méthode basée sur une grille :**

   Les méthodes basées sur la grille transforment l'espace de recherche en une grille où chaque point de la grille représente une unité. Un objet peut se déplacer d'un point à un autre si le chemin est libre.

   Parmi ces méthodes, **l'algorithme de Dijkstra :** Populaire pour la recherche de chemin à coût minimal, il détermine tous les chemins possibles entre le point de départ et la destination.

   L'Algorithme A* : Est une autre méthode notable qui utilise une fonction heuristique pour estimer le meilleur coût d'itinéraire. Il prend en compte le coût de l'itinéraire et sa fonction heuristique pour déterminer le prochain nœud à visiter. Cette procédure est répétée jusqu'à atteindre la destination.

Ces algorithmes offrent de bonnes solutions, mais ils ont du mal à résoudre efficacement des environnements complexes en raison de leur consommation élevée de temps de calcul et de mémoire [22]

2. **Algorithme évolutif :**

   Les algorithmes évolutionnaires sont des méthodes d'optimisation inspirées de l'évolution biologique. Ils génèrent une population de solutions initiales, évaluées par une fonction d'aptitude. Les opérateurs génétiques (sélection, croisement, mutation) sont utilisés pour créer de nouvelles générations. Le processus est répété jusqu'à un critère d'arrêt, souvent le nombre de générations. Ces méthodes sont flexibles et adaptatives, mais ne garantissent pas toujours la solution optimale, nécessitant un ajustement des paramètres ou une hybridation avec des méthodes de recherche locale. [22]

3. **Méthode d'intelligence en essaim :**

   L'intelligence en essaim ou informatique bio-inspirée est une catégorie de l'intelligence artificielle qui a été identifiée pour la première fois par Gerardo Beni et Jing Wang en 1989 dans le contexte du développement de systèmes robotiques cellulaires. Contrairement aux algorithmes évolutionnaires, ces méthodes s'inspirent de comportements simples d'interactions auto-organisées entre agents, tels que les colonies de fourmis à la recherche de nourriture, les vols d'oiseaux, etc. Elles se caractérisent par leur flexibilité et leur multifonctionnalité, ainsi que par leur capacité d'auto-apprentissage et leur adaptabilité aux variations externes, sur la base du comportement collectif des agents, qui interagissent localement avec leur environnement. Les algorithmes matures de cette catégorie sont : l'optimisation par colonies d'abeilles (ACO), l'optimisation par essaims de particules (PSO), la colonie d'abeilles artificielle (ABC). [22]

1.6 Conclusion

Dans ce chapitre, nous avons exploré l'état de l'art en robotique, en mettant en lumière diverses méthodes et techniques de planification de trajectoire. Nous avons discuté des approches de modélisation d'environnements telles que les graphiques de visibilité, les diagrammes de Voronoï et la décomposition cellulaire. Nous avons également examiné différentes méthodes de planification, allant de la planification globale et locale à la planification de mouvements à différentes échelles de contraintes (macro, moyenne et micro-modalités). De plus, nous avons étudié les diverses méthodes de résolution des problèmes de planification, y compris les méthodes exactes, analytiques, dynamiques et heuristiques. En particulier, nous avons mis en évidence l'importance des méthodes basées sur les grilles, des algorithmes évolutionnaires et des techniques d'intelligence en essaim pour résoudre des problèmes complexes de manière efficace. En conclusion, ce chapitre illustre la diversité et la complexité des méthodes disponibles, offrant ainsi un aperçu précieux pour la conception et l'optimisation de systèmes robotiques robustes et efficaces dans divers environnements.

Chapitre 2

Environnement de simulation : ROS & Turtlebot3

2.1 Introduction

Ce chapitre aborde deux outils fondamentaux au cœur de la simulation et du développement robotique modernes : TurtleBot3 et ROS (Robot Operating System) (Figure 2.1). Ces deux ressources unies fournissent aux chercheurs et aux développeurs un ensemble de services considérables pour l'expérimentation, la simulation et le déploiement de nombreux types de robots autonomes et semi-autonomes.

Créé par la société sud-coréenne ROBOTIS, TurtleBot3 est une plateforme robotique open-source très répandue dans le milieu de la recherche et de l'enseignement. En raison de sa conception modulaire et modulable, les qualités de cet équipement s'accordent parfaitement à la réalisation de différentes configurations de dispositifs, matériels et logiciels.

Quant à ROS, il s'agit d'un framework open-source devenu incontournable et largement déployé dans la communauté des chercheurs en robotique. En déployant une infrastructure pour le développement logiciel de robots, ROS offre une diversité de services incluant la supervision de périphériques, la communication inter-processus, la gestion de packages, etc. Conçu sur une architecture distribuée, garant de sa modularité et de sa très vaste communauté opérationnelle, il facilite, par la forte recyclabilité du code, le prototypage et la mise en place rapide de projets robotiques.

Figure 2.1: Turtlebot3 modèle Burger et Symbole de ROS

2.2 Robot Operating System

Le ROS (Robot Operating System) est un méta système d'exploitation pour robot qu'on peut obtenir en code source ouvert. Il fournit les services attendus d'un système d'exploitation : abstraction de matériel, contrôle de périphérique de bas niveau, implémentation des fonctionnalités les plus courantes, communication inter-processus, gestion de paquets. Il fournit également des outils et des bibliothèques pour rechercher, construire, écrire et exécuter du code en utilisant plusieurs ordinateurs. Il est similaire, dans certains aspects, à des « cadres robotiques » tels que Player, YARP, Orocos, Carmen, Orca, MOOS et Microsoft Robotics Studio.

Bien que ROS ne soit pas un cadre temps/-réel, il est envisageable d'incorporer ROS dans un code temps-réel. Le robot Willow Garage PR2 utilise un mécanisme appelé « pr2_etherCAT ». Ce mécanisme est capable de transporter les messages ROS à l'intérieur et à l'extérieur d'un processus temps-réel. ROS s'intègre bien également au cadre temps-réel Orocos.

[26]

2.2.1 Le système de fichiers de ROS

Les ressources de ROS sont organisées dans une structure hiérarchique sur disque. Deux concepts importants se détachent [27] :

1. **Le package :** C'est l'unité principale d'organisation logicielle de ROS. Un package est un répertoire qui contient les nœuds (nous verrons ci-dessous ce qu'est un nœud), les bibliothèques externes, des données, des fichiers de configuration et un fichier de configuration XML nommé manifest.xml. [27]
2. **La stack :** Une stack est une collection de packages. Elle propose une agrégation de fonctionnalités telles que la navigation, la localisation, etc. Une stack est un répertoire qui contient les répertoires des packages ainsi qu'un fichier de configuration nommé stack.xml. [27]

En plus de ces deux notions très importantes, on relève également la notion de distribution. Une distribution, comme dans Linux, est un ensemble de stacks versionnées.

2.2.2 Les notions de base de ROS

Un des principes essentiels d'un système opératif robotique est de faire fonctionner en parallèle un nombre important de flux de contrôle devant pouvoir communiquer entre eux, que ce soit de façon synchrone ou asynchrone, indépendamment de la fréquence à laquelle chacun doit être exécuté. Par exemple, un système opératif robotique doit pouvoir à une fréquence définie interroger les capteurs du robot (capteur de distance à ultrason ou infrarouge, capteur de pression, capteur de température, gyroscope, accéléromètre, caméras, microphones...), traiter, c'est-à-dire fusionner ces informations, les passer à des algorithmes de traitement (traitement de la parole, vision artificielle, localisation et cartographie simultanée, ...), et enfin commander les moteurs en retour. Tout cela se fait de manière continue et en parallèle. D'un autre côté, un système opératif robotique doit assurer la gestion de la concurrence afin de gérer efficacement l'accès aux ressources du robot. Ce paragraphe décrit les concepts que regroupe ROS sous le nom de « ROS Computation Graph », qui permettent de relever ces défis. Ce sont les concepts utilisés par le système en cours de fonctionnement, tandis que le « ROS FileSystem » décrit dans le paragraphe précédent correspond aux concepts statiques. [27]

2.2.2.1 Les nœuds

un nœud : Un nœud est une instance d'un exécutable. Un nœud peut correspondre à un capteur, un moteur, un algorithme de traitement, de surveillance... Chaque nœud qui se lance se déclare au Master. Le système de nœuds permet de standardiser des fonctions basiques, et ainsi de développer rapidement des briques technologiques qui pourront être réutilisées, modifiées ou améliorées facilement. [27]

Le Master est un service de déclaration et d'enregistrement des nœuds qui permet ainsi à des nœuds de se connaître et d'échanger de l'information. Le Master est implémenté via XMLRPC. [27]

Le Master comprend une sous-partie très utilisée qui est le paramètre Server. Celui-ci, également implémenté sous forme de XMLRPC, comme son nom l'indique, est une sorte de base de données centralisée dans laquelle les nœuds peuvent stocker de l'information et ainsi partager des paramètres globaux. [27]

2.2.2.2 Les topics

L'échange de l'information s'effectue soit de manière asynchrone via un topic ou de manière synchrone via un service.

Un topic est un système de transport de l'information basé sur le système de l'abonnement / publication (subscribe / publish). Un ou plusieurs nœuds pourront publier de l'information sur un topic et un ou plusieurs nœuds pourront lire l'information sur ce topic. Le topic est en quelque sorte un bus d'information asynchrone, un peu comme un flux RSS. Cette notion de bus many-to-many asynchrone est essentielle dans le cas d'un système distribué. [27]

Le topic est typé, c'est-à-dire que le type d'information qui est publié (le message) est toujours structuré de la même manière. Les nœuds envoient ou reçoivent des messages sur des topics. [27]

2.2.2.3 Les messages

Un message est une structure de données composite. Il est composé d'une combinaison de types primitifs (chaînes de caractères, booléens, entiers, flottants...) et de messages (le message est une structure récursive). Par exemple, un nœud représentant un servomoteur du robot publiera certainement son état sur un topic (selon ce que vous aurez programmé) avec un message contenant par exemple un entier représentant la position du moteur, un flottant représentant sa température, un autre flottant représentant sa vitesse... [27]

2.2.2.4 Les bags

Les « bags » sont des formats pour stocker et rejouer les messages échangés. Ce système permet de collecter par exemple les données mesurées par les capteurs et les rejouer ensuite autant de fois qu'on le souhaite afin de faire de la simulation sur des données réelles. Ce système est également très utile pour déboguer un système a posteriori(Figure2.2). [27]

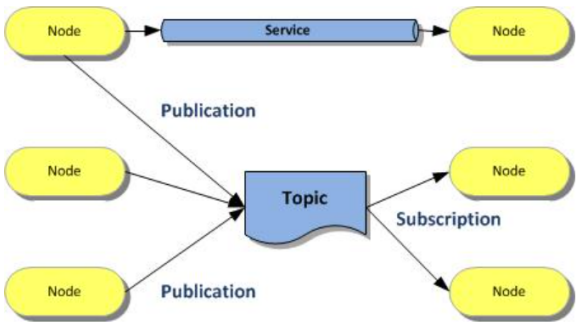

Figure 2.2: Schéma explicatif communication entre nœuds, topics et services

2.2.2.5 URDF

ROS propose d'autres notions que nous pourrons découvrir à l'occasion d'un nouvel article. Citons néanmoins une autre contribution intéressante de ROS à la robotique. Il s'agit de l'urdf (Unified Robot Description Format), un format XML permettant de décrire sous la forme d'un fichier standardisé un robot complet. Le robot ainsi décrit peut être statique ou dynamique et des propriétés physiques et de collision peuvent y être ajoutées.

Outre le standard, ROS propose plusieurs outils permettant de générer, parser ou valider ce format. L'URDF est utilisé par exemple par le simulateur Gazebo pour représenter le robot. [28]

2.2.3 Les outils utiles dans ROS

Comme nous l'avons dit plus haut, ROS est une collection d'outils et d'algorithmes. Certains sont très utilisés lors de la programmation, de la simulation ou de l'exécution des comportements des robots. Citons quelques outils ou algorithmes que le programmeur de ROS retrouvera souvent :

- Stage : un simulateur 2D
- Gazebo : un simulateur 3D
- Rviz : un système de visualisation 3D (contrairement à Gazebo, il n'inclut pas de moteur physique)
- Le package tf qui permet de manipuler des coordonnées et des transformations (si vous avez déjà eu à faire de la cinématique inverse et à manipuler des matrices, alors ce package va grandement vous faciliter la vie).
- Opencv : traitement d'images
- PointCloudLibrary : reconstruction d'environnement 3D à partir de mesures d'un laser

2.3 TurtleBot3

Le TurtleBot 3 est un équipement complet conçu pour débuter dans la construction et la programmation de robots mobiles(Figure 2.3). Il présente une conception modulaire et un logiciel OpenSource qui favorisent l'apprentissage et l'innovation en robotique. Du déplacement autonome à l'interaction avec l'environnement, ce robot dispose d'un large champ d'investigation pour les développeurs, les étudiants ou bien encore les amateurs de robotique. S'acheminant comme un support pédagogique incontournable en robotique mobile, le TurtleBot 3 a la particularité d'être évolutif et de disposer de fonctionnalités

avancées. Il se veut être un excellent socle d'exploration de la robotique avancée et de ROS. Que ce soit dans le cadre de l'éducation, la recherche ou le loisir, le TurtleBot 3 est une bonne alternative pour une expérience enrichissante et prenante pour les amateurs de robotique. [29]

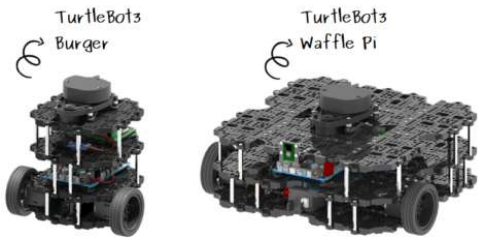

Figure 2.3: Turtlebot3 modèle Burger et Waffle_pi

2.3.1 Description de turtlebot3

Le robot turtlebot3 est répartie sur deux modèles principaux : turtlebot3 burger et turtlebot3 waffle (waffle_pi). dans les deux cas, les deux robots on une structure similaire qui se compose de : [30]

- Châssis compact : Le châssis du TurtleBot3 Burger est compact et léger, ce qui le rend facile à manœuvrer et adapté à des espaces restreints.
- Capteur : il est équipé de capteurs tels que des capteurs infrarouges, collision, un gyroscope et un accéléromètre. Ces capteurs permettent au robot de percevoir son environnement et d'interagir avec celui-ci.
- Plateforme open-source : le turtlebot3 est basé sur une plateforme open-source, ce qui signifie que son matériel et son logiciel sont disponibles au public. Cela favorise la collaboration, l'apprentissage et le développement de nouvelles fonctionnalités.
- Facilité d'utilisation : le turtlebot3 burger est conçu pour être convivial et facile à utiliser, ce qui en fait un excellent choix pour l'éducation et les projets de bricolage en robotique.
- Compatibilité avec ROS : le robot est compatible avec ROS, ce qui permet une programmation et une personnalisation avancées.
- Puissance de calcul : il dispose d'un mini-ordinateur embarqué pour le traitement des données et l'exécution d'algorithmes, généralement une Raspberry Pi.
- Connectivité : le turtleBot3 Burger peut être contrôlé via une connexion sans fil, ce qui facilite la programmation et le contrôle à distance.

— Modularité : il est modulaire, ce qui signifie que vous pouvez ajouter des capteurs supplémentaires ou des accessoires en fonction des besoins du projet.
— Une batterie Li-Po (ou batterie lithium-polymère) est un type de batterie rechargeable qui utilise la technologie lithium-ion. Contrairement aux batteries lithium-ion traditionnelles, les batteries Li-Po utilisent un électrolyte sous forme de polymère plutôt que de liquide. Cette conception offre plusieurs avantages, notamment une plus grande flexibilité dans la forme et la taille de la batterie.
— Le Dynamixel XL430 est un servomoteur intelligent développé par ROBOTIS, une entreprise spécialisée dans les composants pour robots éducatifs et de recherche. Les servomoteurs Dynamixel sont largement utilisés dans le domaine de la robotique pour leur précision, leur puissance et leur capacité de contrôle avancée.

• DYNAMIXEL est une marque de Smart Actuator développée par ROBOTIS pour une utilisation dans tout système robotique. Le nom “DYNAMIXEL” est dérivé de deux mots “Dynamic” et “Cell”, qui, combinés ensemble, deviennent des actionneurs intelligents tout-en-un.

2.3.2 Les modèles de turtlebot3

Il existe plusieurs modèles de TurtleBot3 chacun adapté à des besoins et des environnements spécifiques. Voici une liste des principaux modèles de TurtleBot3. [30]

2.3.2.1 Turtlebot3 waffle

C’est un modèle qui a une forme de gaufre et offre une meilleure stabilité que le Burger(figure2.4). Il est caractérisé par un grand espace de chargement, ce qui le rend adapté à des applications plus avancées nécessitant le transport d’équipements supplémentaires. [30]

Figure 2.4: Turtlebot3 modèle Waffle

2.3.2.2 Turtlebot3 Waffle Pi

Cette version intègre une Raspberry Pi dans sa structure(figure2.5), il est caractérisé par sa puissance de calcul et la possibilité d'exécution des tâches plus complexes. [30]

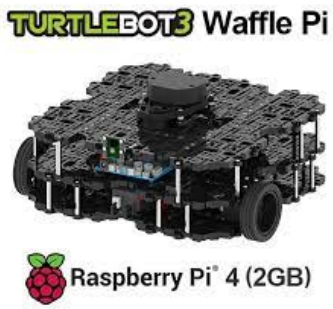

Figure 2.5: Turtlebot3 modèle Waffle_pi

2.3.2.3 Turtlebot3 Burger

C'est le modèle de base de turtlebot3(Figure2.6). il est caractérisé par la solidité de liaison de ses parties, la rapidité d'exécution de ses mouvements, ce que lui rond facile à l'utiliser. il est idéal pour l'apprentissage en robotique. [30]

Figure 2.6: Turtlebot3 modèle Burger

2.3.3 Turtlebot3 Burger : caractéristiques techniques

- Vitesse translationnelle max : 0,22 m/s
- Vitesse rotationnelle max : 162,72deg/s
- Charge max : 15 kg
- Dimensions : 138x178x192 mm
- Poids (avec SBC, batterie et capteurs) : 995 g
- Hauteur de seuil franchissable max : 10 mm
- Autonomie : 2h30
- Temps de chargement : 2h30
- Connexion PC : USB
- Centrale IMU : gyroscope 3 axes, accéléromètre 3 axes, magnétomètre 3 axes
- LiDAR LDS-02
- Connecteurs d'alimentation : 3,3 V/800 mA / 5 V/2 A / 12 V/ 1 A
- Pins de développement : 18 pins GPIO, 32 pins Arduino
- Plusieurs séquences de bips programmables
- 4 LED utilisateur programmables
- 1 LED de statut pour la carte
- 1 LED de statut pour la batterie
- 1 LED de statut Arduino
- 2 boutons poussoirs
- Batterie Lithium Polymère 11,1 V 1800 mAh/19.98 Wh 5C
- Mise à jour du firmware : via USB/JTAG
- Adaptateur pour recharge
- Entrée : 100-240 V, 50/60 Hz CA, 1,5 A @ max
- Sortie : 12 Vcc, 5 A

[30]

2.4 Conclusion

Ce chapitre a porté sur une étude approfondie du TurtleBot3, qui est un robot mobile modulable largement plébiscité dans les domaines éducatif et de la recherche en robotique. Nous avons entamé notre discussion en offrant un aperçu du TurtleBot3, soulignant son design, ses capacités diverses et son usage flexible. Puis, notre attention s'est orientée vers les différentes versions de TurtleBot3, avec un focus particulier sur le modèle TurtleBot3 Burger, en explorant ses spécifications techniques comme les différents capteurs, les éléments matériels et ses aptitudes en matière de navigation.

Le chapitre a aussi abordé le Robot Operating System (ROS), un cadre logiciel essentiel au bon fonctionnement du TurtleBot3. Nous avons exploré l'architecture du système de fichiers de ROS, détaillant la structure et comment s'organisent les fichiers et les paquets, ce qui simplifie le développement et la gestion de projets robotiques. Les principes fondamentaux de ROS ont aussi été examinés, mettant en avant des concepts cruciaux comme les nœuds, les topics, les services et les messages, qui facilitent la communication et la coordination au sein des systèmes robotiques.

Pour terminer, le chapitre a présenté plusieurs outils essentiels dans ROS, notamment rviz pour la visualisation et Gazebo pour la simulation, qui sont indispensables pour le développement, les tests et la validation des applications robotiques.

En résumé, ce chapitre offre une vue d'ensemble complète sur le TurtleBot3 et son intégration avec ROS, accentuant leur rôle et leur pertinence dans l'industrie de la robotique.

Chapitre 3

Modélisation de l'environnement

3.1 Introduction

Dans ce chapitre, nous explorerons l'approche que nous avons utilisée pour la modélisation de l'environnement, ainsi que les méthodes employées. Nous expliquerons les différentes étapes nécessaires pour transformer une carte 3D en une carte 2D en utilisant divers nœuds ROS. Ensuite, nous décrirons la méthode de division cellulaire pour l'analyse de la carte et la détection des obstacles, ainsi que la réalisation d'un graphe G(V, E) ou V représente les sommets (nœuds) et et E les arêtes (arcs).

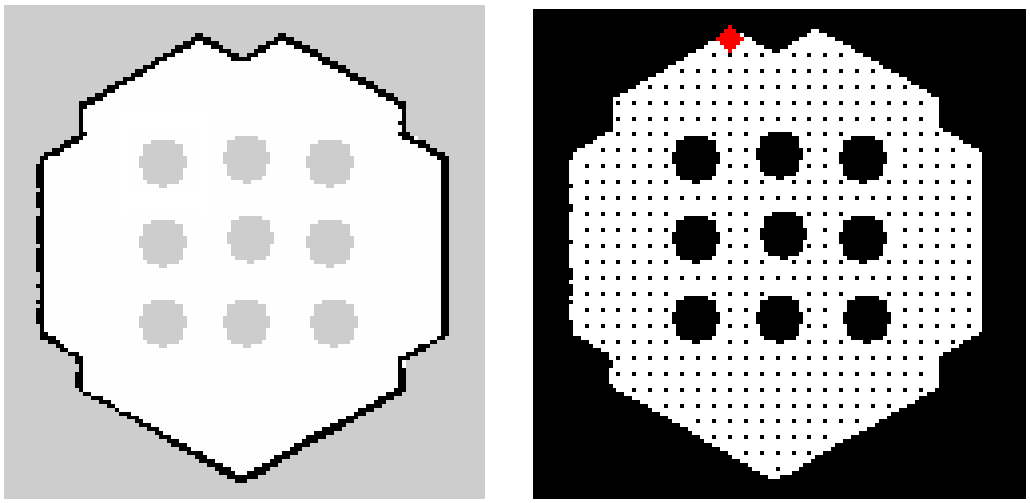

Figure 3.1: l'image au début de la modélisation et a la fin de la modélisation

3.2 Cartographie de l'environnement :

La transformation d'un environnement réel en une carte utilisable suit plusieurs étapes clés.

1. On exporte d'abord le modèle du robot en utilisant la commande export TURTLEBOT3_MODEL=burger. Cela permet au système de reconnaître le modèle de robot que nous utilisons, dans ce cas, le modèle 'burger'. (Figure 3.2)

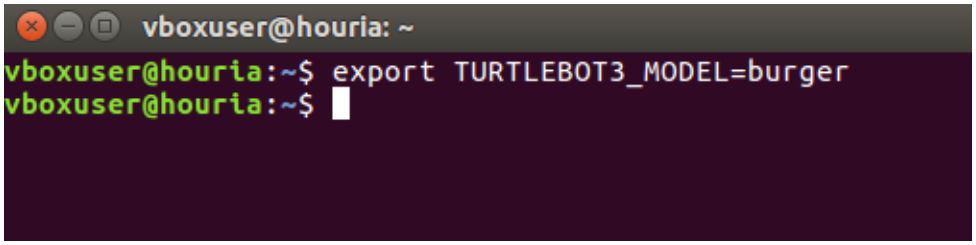

Figure 3.2: Exportation du modèle de Turtlebot3 en Burger

2. Après avoir exporté le modèle du robot, nous utilisons la commande roslaunch gazebo turtlebot3_world.launch pour déployer TurtleBot3 dans l'environnement de simulation Gazebo. Cette commande déploie notre robot avec la carte 3D 'world' sur le logiciel Gazebo. (Figure 3.3)

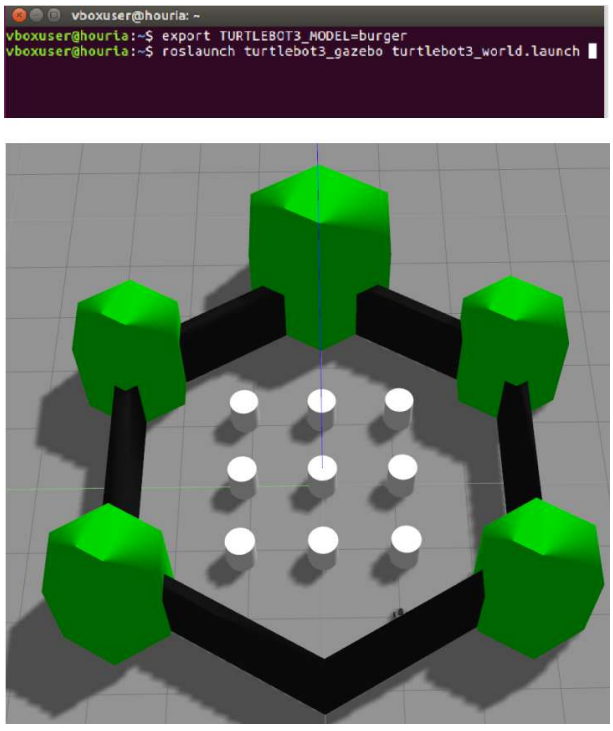

Figure 3.3: Déploiement de Turtlebot3

3. Après le déploiement du robot, l'étape suivante consiste à scanner la carte en utilisant le SLAM avec la méthode Gmapping. Pour cela, nous exécutons la commande roslaunch turtlebot3_slam turtlebot3_slam.launch slam_methods :=gmapping. Cette méthode permet de cartographier l'environnement et de fournir une image 2D de la carte, nous donnant ainsi une idée des obstacles présents. (Figure 3.4)

Figure 3.4: lancement de slam

4. Nous utilisons ensuite le nœud de téléopération en exécutant roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch pour déplacer le robot autour de la carte. Cela nous permet de contrôler le robot à l'aide du clavier. (Figure 3.5) [1]

1. La commande teleop_key nous permet d'avancer avec la touche W, reculer avec la touche X tourner à droite avec la touche D et à gauche avec la touche A, et arrêt avec la touche S. Ce nœud nous permet d'agir directement sur la vitesse linéaire et angulaire

```
/home/vboxuser/catkin_ws/src/turtlebot3/turtlebot3_teleop/launch/turtlebot3_teleop_key.launch htt
vboxuser@houria:~$ roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch
... logging to /home/vboxuser/.ros/log/6e90e66e-1415-11ef-8e44-08002711d3d6/roslaunch-houria
-4712.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://houria:43963/

SUMMARY
========

PARAMETERS
 * /model: burger
 * /rosdistro: kinetic
 * /rosversion: 1.12.17

NODES
  /
    turtlebot3_teleop_keyboard (turtlebot3_teleop/turtlebot3_teleop_key)

ROS_MASTER_URI=http://localhost:11311

process[turtlebot3_teleop_keyboard-1]: started with pid [4729]

Control Your TurtleBot3!
---------------------------
Moving around:
        w
   a    s    d
        x
```

Figure 3.5: lancement de teleop.key

5. Après avoir téléopéré TurtleBot3 autour de la carte 'World' et cartographié l'ensemble de l'environnement, nous obtenons le résultat illustré dans la figure suivante : (Figure 3.6)

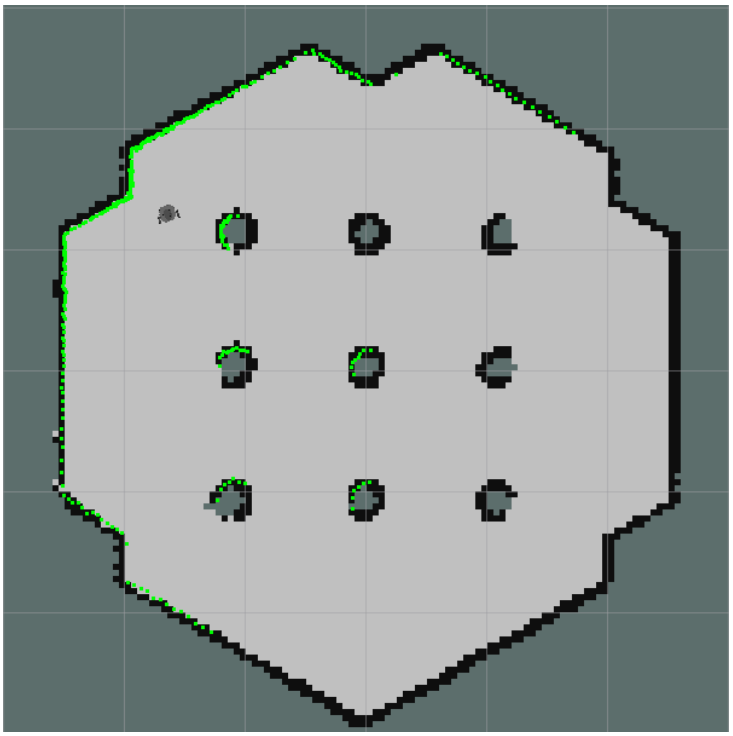

Figure 3.6: la carte scannée

6. Une fois que la carte est cartographiée avec succès, nous enregistrons cette carte 2D en exécutant la commande rosrun map_server map_saver -f /map, où /map est le nom du fichier. (Figure 3.7)

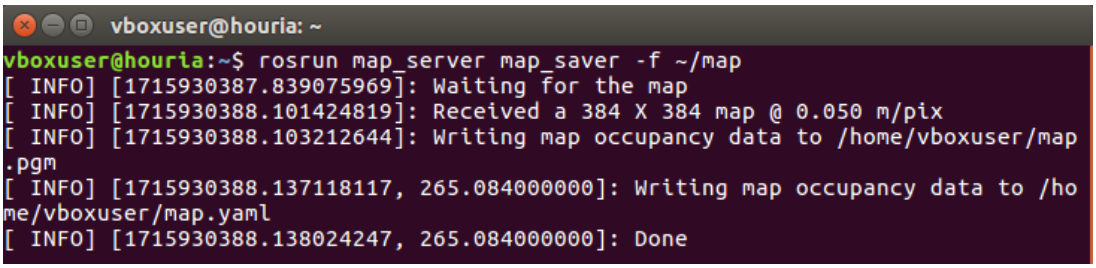

Figure 3.7: l'enregistrement de la map final

Pour assurer le bon fonctionnement des différentes étapes de la cartographie, plusieurs nœuds sont utilisés pour exécuter diverses tâches. Parmi ces nœuds (figure 3.8), on trouve :

1. Nœud Teleop Key : Permet le contrôle manuel du TurtleBot3 à l'aide du clavier de l'ordinateur, facilitant ainsi les mouvements de robot dans l'environnement simulé.
2. Nœud SLAM : Responsable de la technique SLAM (Simultaneous Localization and Mapping), il est chargé d'estimer la position du robot et de créer une carte de l'environnement en temps réel.
3. Nœud RViz : permet de visualiser le résultat de slam.
4. Nœud Gazebo : est un simulateur 3D, utilisé pour simuler l'environnement dans lequel le TurtleBot3 évolue, offrant ainsi un terrain de test réaliste pour le robot.

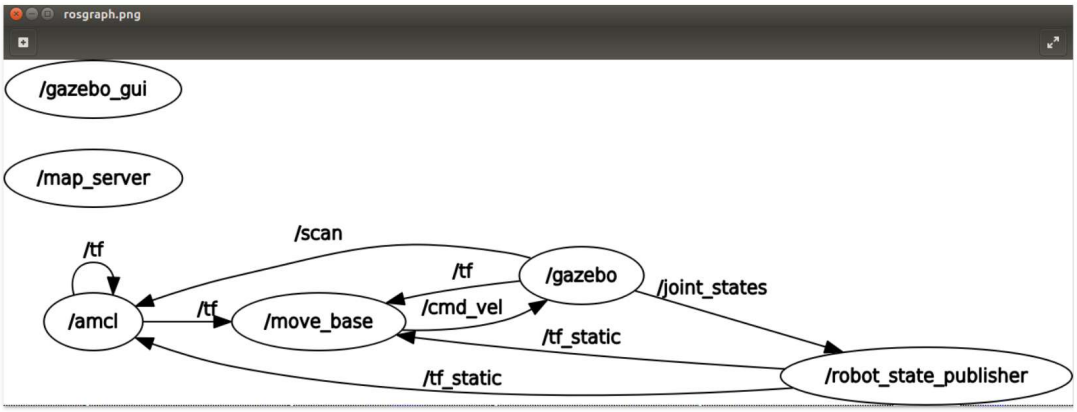

Figure 3.8: les nœuds utilisés

3.3 Transformation de la map en image binaire :

La carte obtenue à partir de SLAM est initialement en couleur. La première étape consiste à convertir cette image en binaire (Figure 3.9), où le noir représente les obstacles (zones non traversables) et le blanc représente les zones libres (traversables).

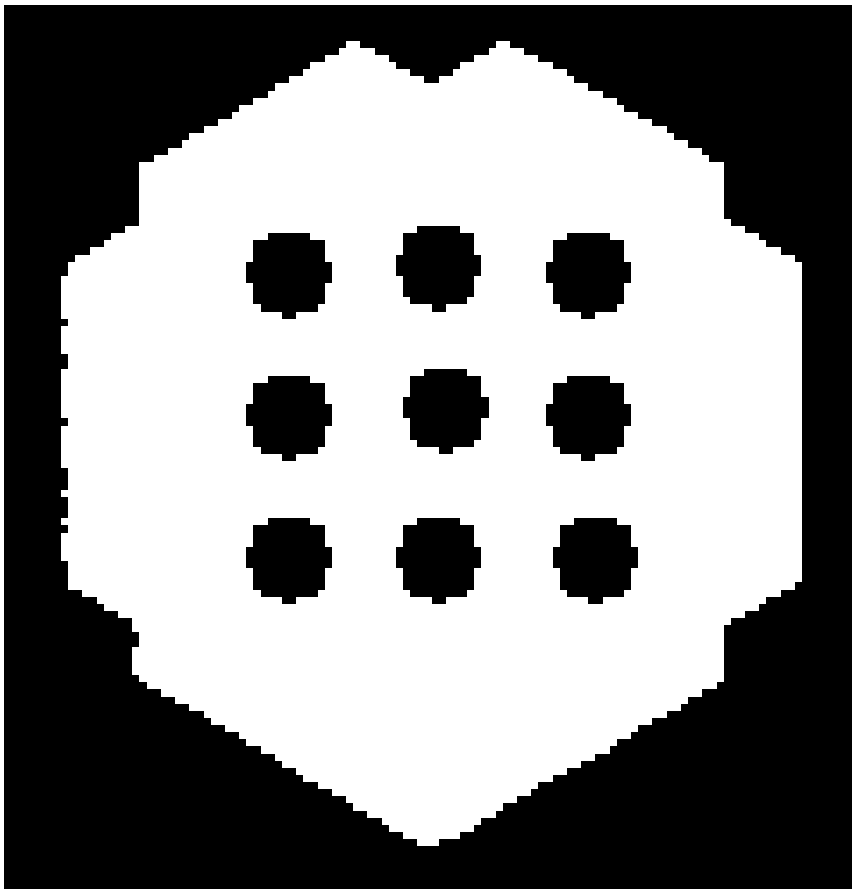

Figure 3.9: Première étape de modélisation

3.4 Création de graph G(V.E) :

Pour construire un graphe basé sur une carte générée par SLAM, plusieurs étapes sont nécessaires :

3.4.1 Création de l'ensemble des noeuds V :

3.4.1.1 Division cellulaires :

La carte est divisée en petites cellules régulières, formant une grille. Chaque cellule devient un nœud potentiel dans le graphe. Les points centraux de ces cellules sont utilisés

pour éviter l'encombrement visuel et simplifier les calculs (Figure 3.10).

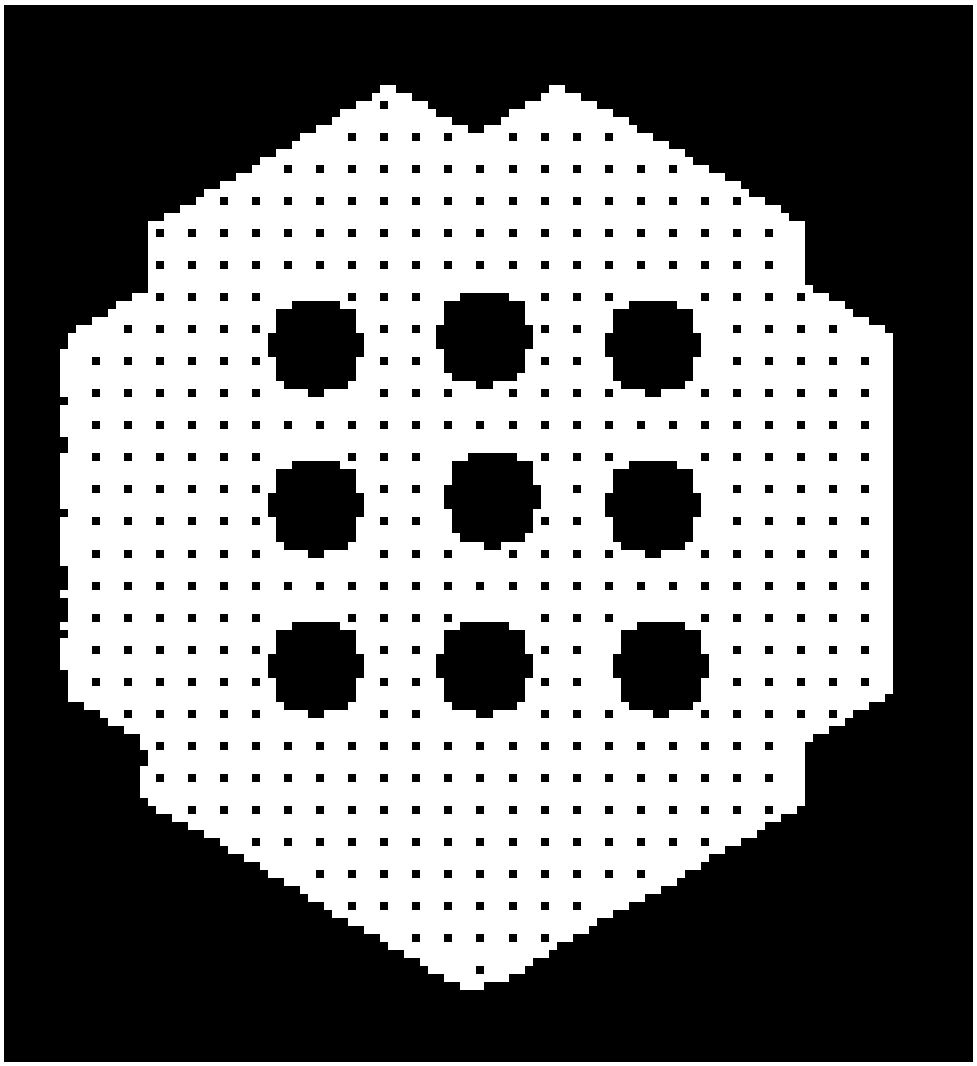

Figure 3.10: Division cellulaire

3.4.1.2 Attribution de coordonnées aux cellules :

Chaque cellule se voit attribuer des coordonnées spécifiques correspondant à son emplacement précis sur la carte. Ces coordonnées facilitent la localisation et la navigation.

3.4.1.3 Identification des cellules non traversables :

Les cellules contenant des obstacles sont identifiées et marquées comme non traversables. Une matrice binaire est utilisée pour cette représentation, où 1 indique une cellule non traversable et 0 une cellule traversable.

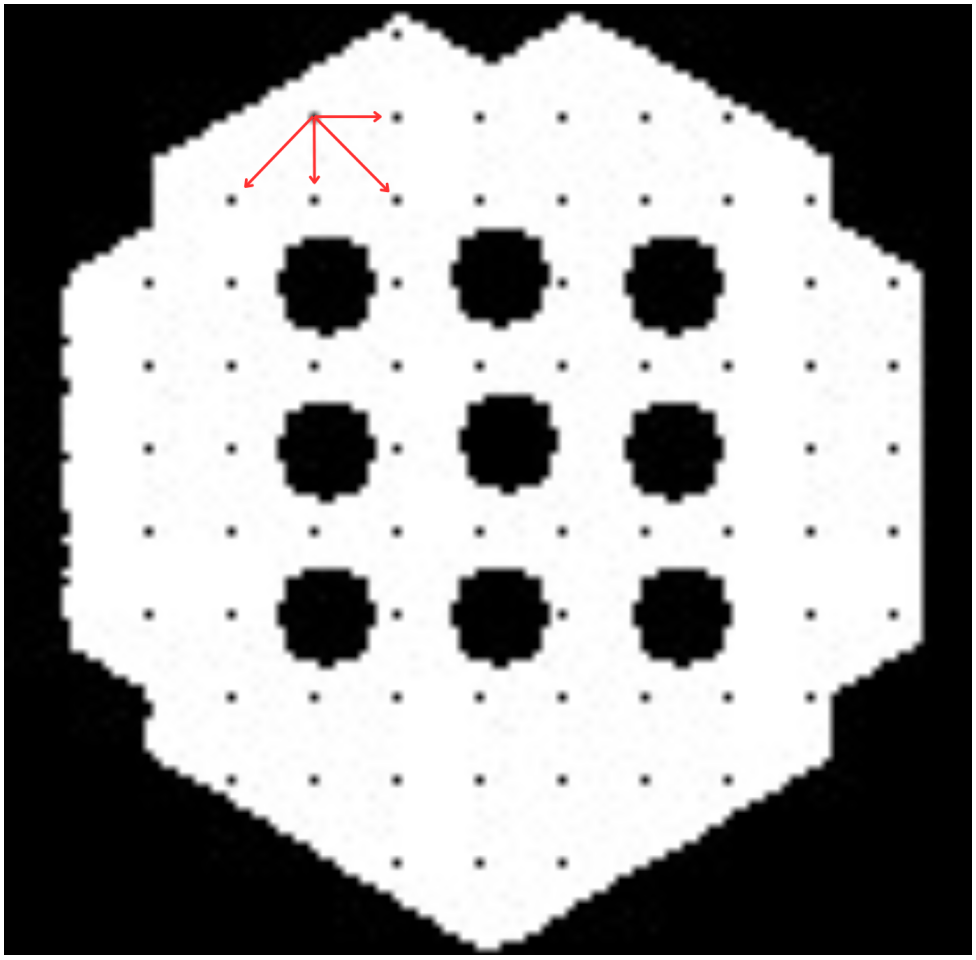

Figure 3.11: Schéma explicatif sur le teste d'existence

3.4.1.4 Transformation des cellules traversables en nœuds :

Les cellules traversables sont transformées en nœuds dans le graphe. Chaque nœud est associé à ses coordonnées, ce qui permet une représentation précise de l'environnement pour une navigation efficace.

3.4.2 création des arcs E :

3.4.2.1 Teste de voisinage : Cardinales et Diagonales

La visibilité entre les cellules adjacentes est évaluée. On considère les directions cardinales (haut, bas, gauche, droite) et diagonales. Cela permet de déterminer l'accessibilité entre les cellules.

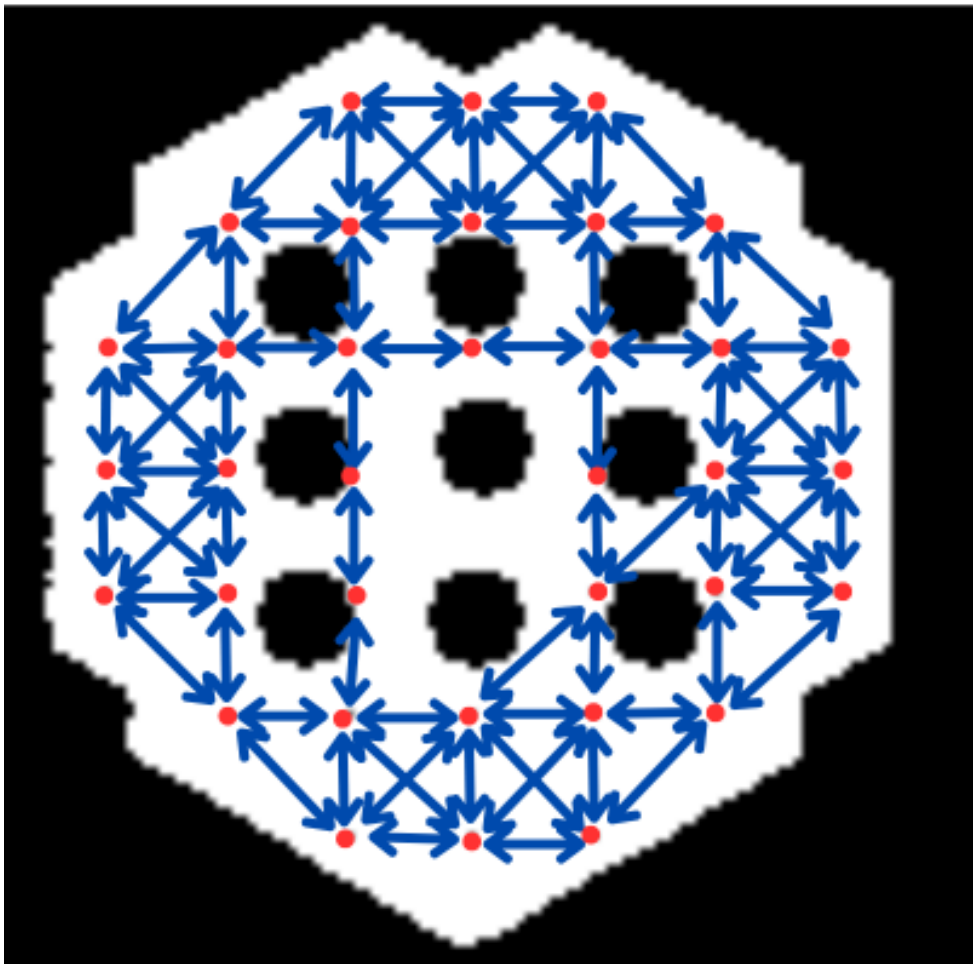

Figure 3.12: Schéma final de teste global de tous les points adjacents

3.4.2.2 Attribution des chemins :

Des arcs sont créés pour représenter les trajets sans obstacles entre les cellules adjacentes. Une matrice de visibilité est utilisée pour tester la possibilité de passage entre les 8 points adjacents. Une fois le chemin créé, on calcule la distance euclidienne entre les deux points, de sorte que chaque arc est caractérisé par sa distance.

3.4.3 Modélisation du Graphe G(V, E)

Nous avons modélisé un graph G(V, E) où V représente les nœuds (points dans l'espace) et E les arêtes (connexions entre les points). Les connexions incluent à la fois des connexions cardinales et diagonales, permettant au robot de se déplacer dans huit directions différentes.

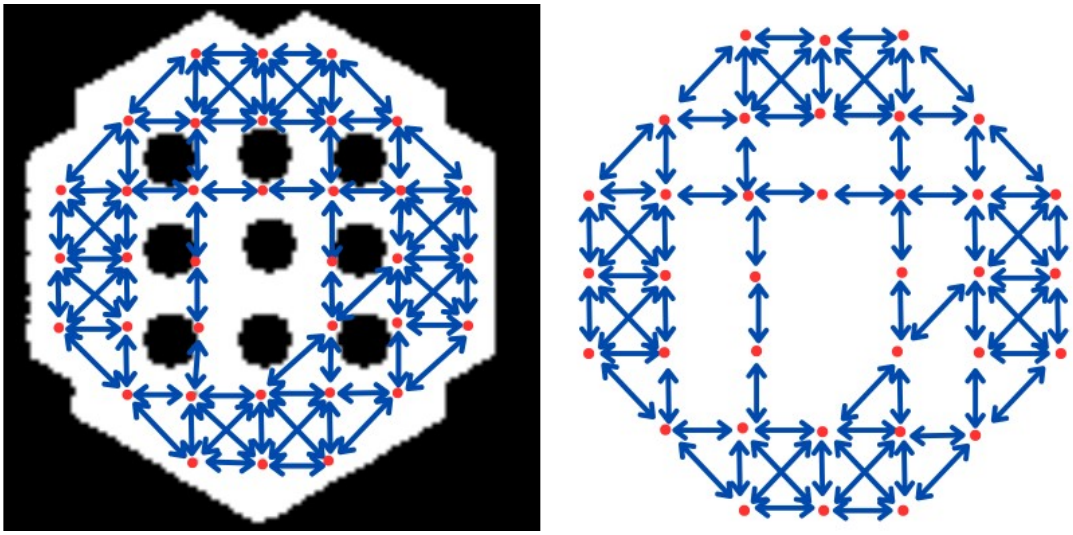

Figure 3.13: figure montrant la génération graph G(V,E)

3.5 Conclusion

La transformation d'un environnement réel en une carte puis en un graphe G(V,E) nécessite plusieurs étapes essentielles. Dans la première partie, pour créer une carte en 2D, nous avons utilisé la méthode SLAM sur le logiciel Rviz et avons déployé le robot sur Gazebo pour pouvoir naviguer avec celui-ci.

Dans la deuxième partie, la conversion de la carte en un graphe G(V, E) en divisant la carte en petites cellules adjacentes, chaque cellule devenant un nœud. Les connexions entre les cellules sont établies en tant qu'arêtes. Les cellules contenant des obstacles sont marquées comme non traversables. La visibilité entre les cellules adjacentes est testée pour permettre de planifier un chemin.

Chapitre 4

Calcul d'itinéraire : Simulation & Resultat

4.1 Introduction

La planification de trajectoire pour un robot mobile est une tâche essentielle en robotique, particulièrement dans des environnements complexes avec des obstacles. Ce chapitre décrit l'approche utilisée pour la recherche de chemin optimal en employant l'algorithme de Floyd-Warshall. Ainsi, sa mise en œuvre sur le simulateur ROS.

4.2 L'implémentation de l'Algorithme de Floyd-Warshell

4.2.1 Algorithme de Planification "Floyd-Warshall"

L'algorithme de Floyd-Warshall est utilisé pour trouver les plus courts chemins entre toutes les paires de nœuds du graphe. Initialement, la matrice des distances est définie avec les distances directes entre les nœuds connectés.

La matrice des prédécesseurs est générée pour reconstruire les chemins les plus courts. Elle est initialisée de sorte que le prédécesseur de chaque nœud j dans le chemin de i à j soit i. L'algorithme procède par une série d'itérations où, pour chaque nœud intermédiaire, il vérifie si un chemin passant par k est plus court que le chemin direct. Si c'est le cas, il met à jour la matrice des distances et la matrice des prédécesseurs.

— Pour chaque point, on crée une liaison avec le point le plus proche dans les différentes directions.

— À la fin, on obtient une matrice carrée de taille "nombre de points (nœuds)" générée par le programme précédent.

— La ligne i de la matrice des prédécesseurs représente le numéro du point de départ (position du robot), et la colonne j représente le numéro du point d'arrivée.
— Pour récupérer la séquence des points, il suffit de visiter la case [i][j] pour trouver le point précédent. Ce point-là deviendra le prochain [j] dans la matrice.
— On répète cette opération jusqu'à obtenir la valeur i.

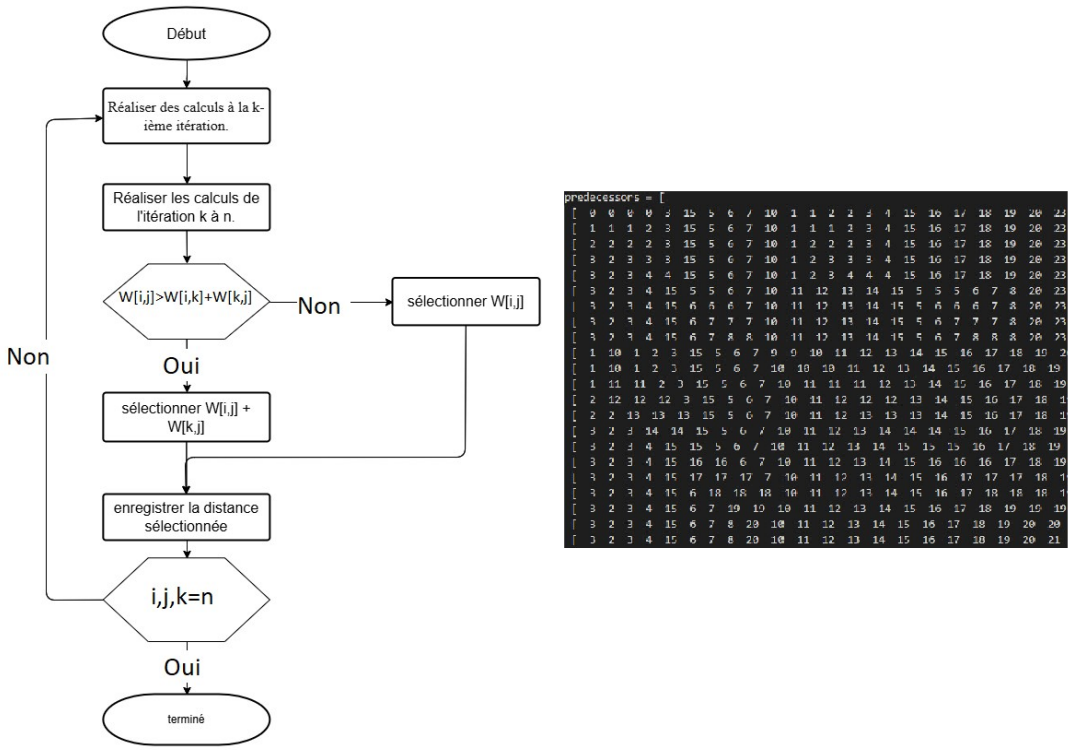

Figure 4.1: organigramme de l'algorithme de Floyd-Warshall

4.3 Mise en œuvre sur ROS

Nous avons créé trois programmes Python (nœuds ROS) pour planifier et exécuter la trajectoire du robot :

4.3.1 Détermination de la trajectoire

Ce premier programme permet de déterminer la séquence des points à visiter pour passer du point de départ au point d'arrivée. Il faut passer par :

4.3.1.1 Récupération de la position du Robot

La récupération de la position du robot se fait par l'odométrie. Cette fonction utilise les informations personnelles du robot, telles que la position initiale du robot, la vitesse linéaire (ou angulaire) du robot, et le diamètre des roues pour estimer la distance parcourue depuis la position de son déploiement. Ces coordonnées sont générées en fonction de l'origine du logiciel Gazebo.

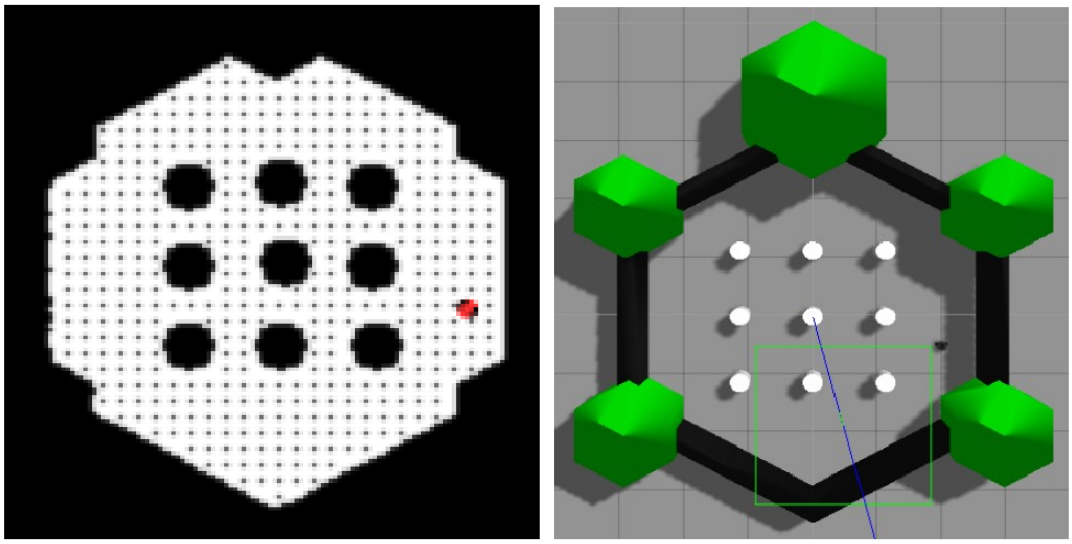

Figure 4.2: Position initiale du robot

4.3.1.2 Transformations des coordonnées

Afin que ces coordonnées soient valables dans notre programme, une transformation est nécessaire en termes d'unité et de repère.

```
boutalbi@boutalbi-VirtualBox:~/catkin_ws/src/turtlebot3/memoire/src$ ./create_coordonnee.py
extraire info
extraction terminer
Obtention des coordonnees du robot...
('x en mettre =', -0.5000296281926159)
('y en mettre =', -1.999931934123057)
('x=', 50)
('y=', 57)
Position de depart obtenue : (50, 57)
Veuillez entrer les coordonnee du point d'arrivee (format x,y) :
Point d'arrivee : 35,43
```

Figure 4.3: transformation de coordonnées et obtention les coordonnées d'arrivée

— Dans notre cas, l'origine de Gazebo est différente de l'origine de notre programme. Un changement de repère est nécessaire afin de pouvoir localiser correctement la position initiale du robot sur Gazebo par rapport à l'origine considérée par notre programme. D'autre part, la séquence de la trajectoire générée par notre programme doit être adaptable à l'environnement de simulation "GAZEBO".

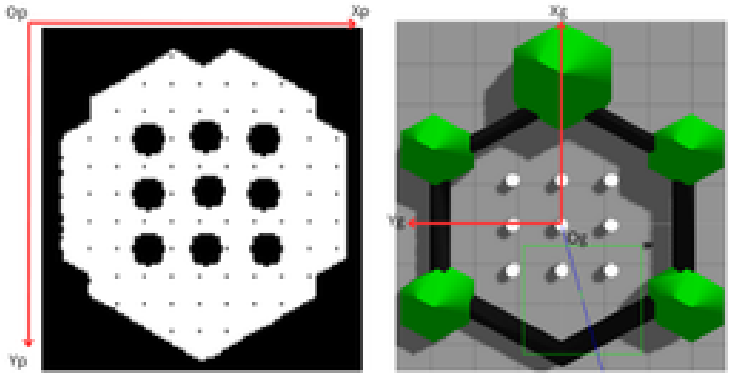

Figure 4.4: les deux repères du programme Op (xp, yp) et de gazebo Og (xb, yb)

4.3.1.3 Génération de la séquence

Une fois les coordonnées récupérées, on procède à la récupération de la séquence des points de trajectoire stockés dans la base de données de l'algorithme Floyd-Warshall (matrice des prédécesseurs). Puis, on détermine les coordonnées cartésiennes de chaque point.

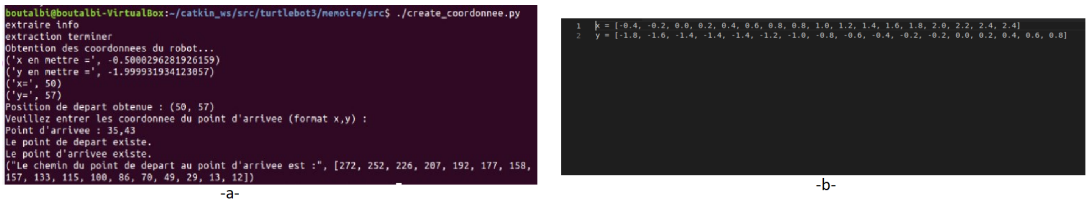

Figure 4.5: génération de la séquence de point de la trajectoire

— Le terminal montre le numéro des points selon l'algorithme de Floyd-Warshall.

— Les coordonnées enregistrées seront récupérées par le prochain nœud.

À la fin, on aura un ensemble de waypoints caractérisés par leurs coordonnées cartésiennes x et y, l'une dans le repère OG et l'autre dans le repère OP. .

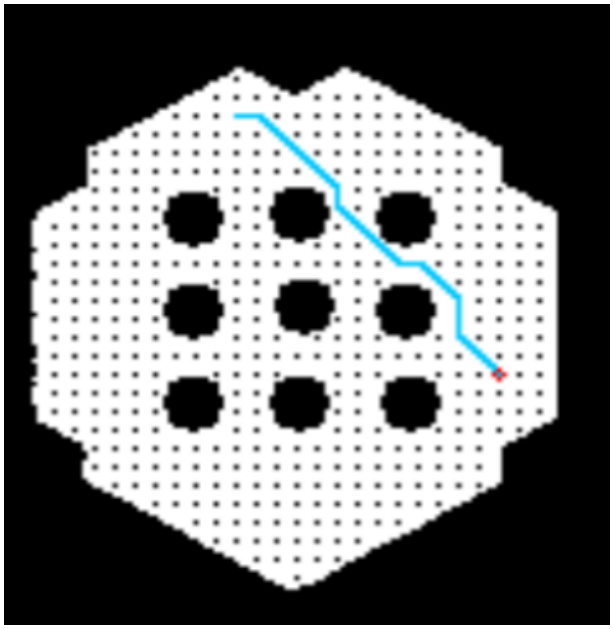

Figure 4.6: Trajectoire du robot dans le repere OP

4.3.2 Serveur de Service ROS

Ce nœud sert de serveur de service ROS, permettant le transfert des coordonnées vers le nœud contrôlant le robot.

4.3.2.1 Récupération des coordonnées

Ce nœud récupère les coordonnées enregistrées précédemment et affiche un message indiquant qu'il est prêt à envoyer les données si une requête est faite.

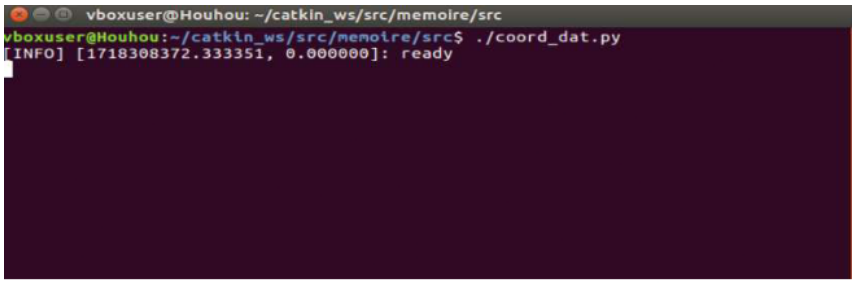

Figure 4.7: le serveur ROS a récupéré la séquence est prêt à envoyer s'il y a une requête

4.3.2.2 demande de requête

Si une requête est demandée par le troisième nœud (figure 4.9), il envoie les coordonnées formatées en liste pour le troisième nœud (figure 4.10).

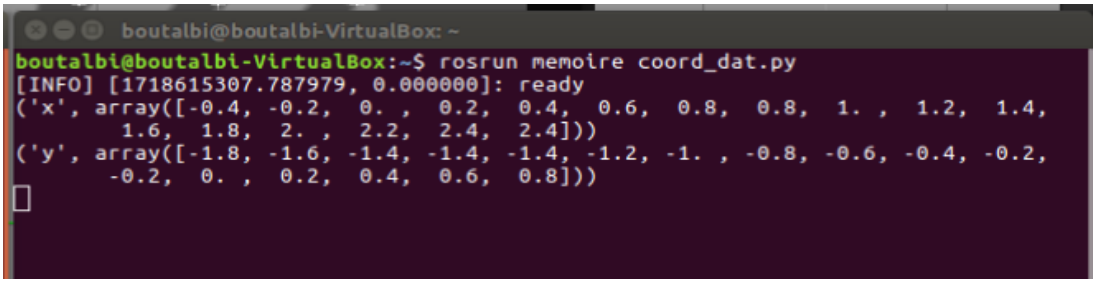

Figure 4.8: le deuxième nœud a reçu une requête les coordonnées sont bien envoyer

4.3.3 Contrôleur du Robot

Le rôle de ce dernier nœud est de contrôler le robot et de le déplacer aux différents points de la liste reçue par la requête.

4.3.3.1 la demande de la requete :

Afin de recevoir la liste des points à visiter, le troisième nœud doit envoyer une requête au deuxième nœud comme le montre la figure suivante.

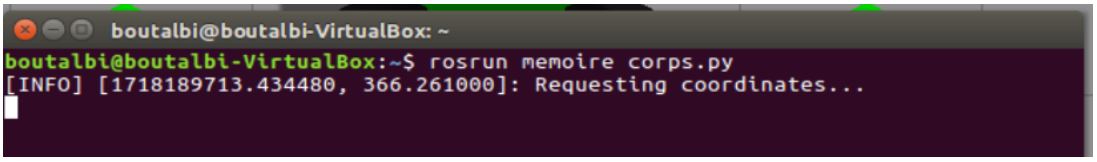

Figure 4.9: nœud contrôleur du robot : envoie la demande de requete

4.3.3.2 Réception de la liste des points :

Une fois la demande de la requête bien envoyée (figure 4.9), nous recevrons la liste des coordonnées des points (figure 4.8) comme le montre la figure suivante.

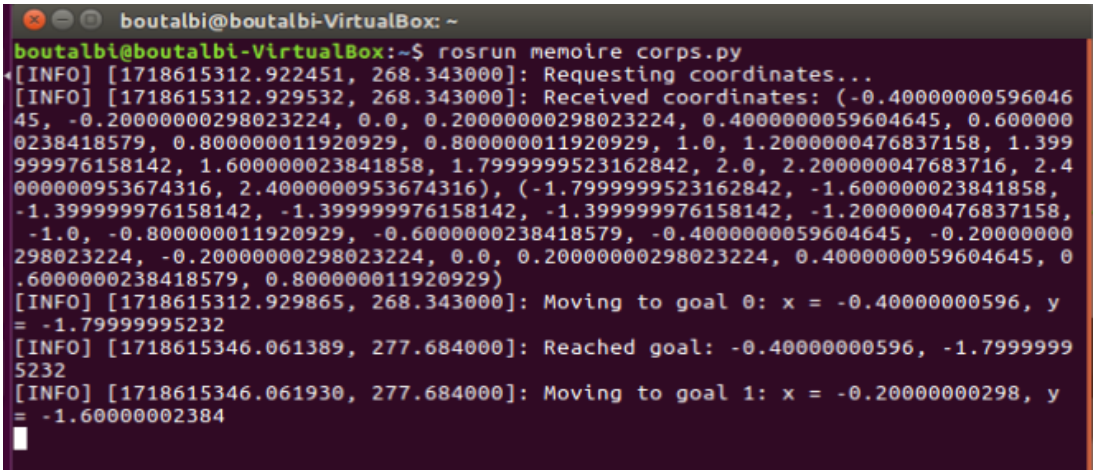

Figure 4.10: nœud contrôleur du robot : reception de la séquence

Dès la réception des coordonnées, le robot se met à bouger et visite les points un par un tout en indiquant les coordonnées du prochain point à visiter. Lorsqu'il atteint un point, il affiche le message "Reached goal".

4.4 Résultat et Discussion

4.4.1 Test et Simulation

À la base, la plateforme Turtlebot3 contient un nœud ROS permettant le déplacement du robot d'un point A à un point B par une ligne droite. Ce nœud est fonctionnel uniquement dans des environnements sans obstacles, comme illustré dans la figure 4.11.

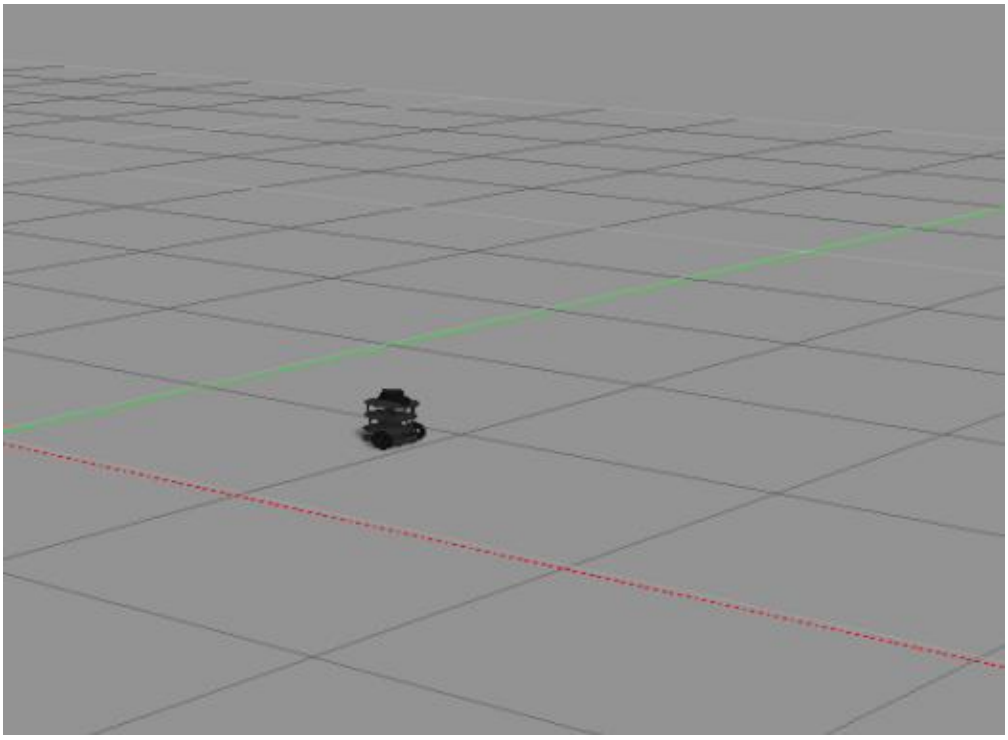

Figure 4.11: navigation dans un espace libre

Dans le cas des environnements contenant des obstacles, ce nœud n'est plus fonctionnel. Notre étude se focalise donc sur l'amélioration de ce dernier, permettant son utilisation même dans des environnements contenant des obstacles statiques.

4.4.1.1 Programme de Robotis

Le programme fourni par les développeurs de Turtlebot3 de Robotis nous permet de saisir les coordonnées du point souhaité et trace une ligne directe entre la position du robot et la destination. Cependant, ce programme présente l'inconvénient de ne pas prendre en considération les obstacles, ce qui fait que le robot les heurte.

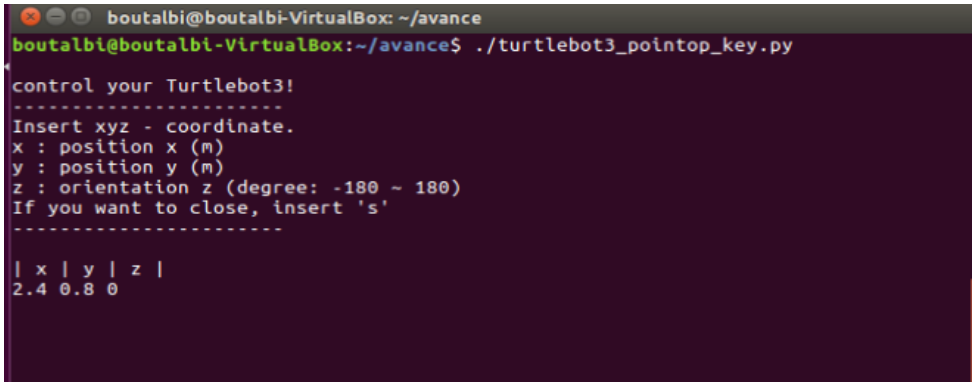

Figure 4.12: programme de ROBOTIS

4.4.1.2 Notre approche proposée

Afin de mettre en œuvre notre programme, nous avons réparti cela en cinq nœuds fonctionnant en coopération :

1. CSP : Le premier programme consiste à générer les nœuds et les arcs pour tester la visibilité entre les différents nœuds. Cela crée un certain nombre de points, ces points sont enregistrés et seront utilisés pour les prochaines étapes

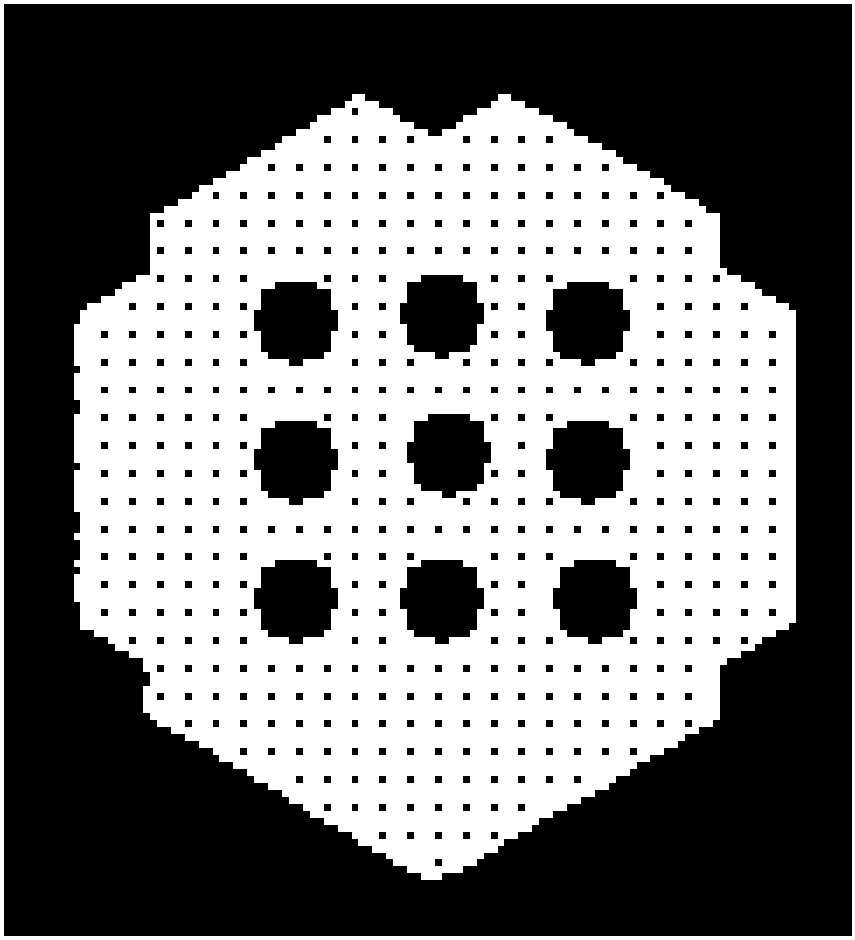

Figure 4.13: image montrant les nœuds générés par le programme

2. Floyd_Warshall : Le deuxième programme effectue une planification de trajectoire en utilisant l'algorithme de Floyd-Warshall. Ce programme nous permet d'obtenir la matrice des prédécesseurs, qui sera enregistrée dans un fichier "tsp-moo.dat".

```
predecessors = [
[  0  0  0  0  3  15  5  6  7  10  1  1  2  2  3  4  15  16  17  18  19  20  23  9  10  10  11  12  12  13  14  15  16  17  18  19  20  21
[  1  1  1  2  3  15  5  6  7  10  1  1  1  2  3  4  15  16  17  18  19  20  23  9  10  10  11  11  12  13  14  15  16  17  18  19  20  21
[  2  2  2  2  3  15  5  6  7  10  1  2  2  2  3  4  15  16  17  18  19  20  23  9  10  11  11  12  12  13  14  15  16  17  18  19  20  21
[  3  2  3  3  3  15  5  6  7  10  1  2  3  3  3  4  15  16  17  18  19  20  23  9  10  11  12  12  13  13  14  15  16  17  18  19  20  21
[  3  2  3  4  4  15  5  6  7  10  1  2  3  4  4  4  15  16  17  18  19  20  23  9  10  11  12  13  13  14  14  15  16  17  18  19  20  21
[  3  2  3  4  15  5  5  6  7  10  11  12  13  14  15  5  5  5  6  7  8  20  23  9  10  11  12  13  14  15  15  16  16  17  18  19  20  21
[  3  2  3  4  15  6  6  6  7  10  11  12  13  14  15  5  6  6  6  7  8  20  23  9  10  11  12  13  14  15  16  16  17  17  18  19  20  21
[  3  2  3  4  15  6  7  7  7  10  11  12  13  14  15  5  6  7  7  7  8  20  23  9  10  11  12  13  14  15  16  17  17  18  18  19  20  21
[  3  2  3  4  15  6  7  8  8  10  11  12  13  14  15  5  6  7  8  8  8  20  23  9  10  11  12  13  14  15  16  17  18  18  19  19  20  21
[  1  10  1  2  3  15  5  6  7  9  9  10  11  12  13  14  15  16  17  18  19  20  23  9  9  9  10  11  12  13  14  15  16  17  18  19  20
[  1  10  1  2  3  15  5  6  7  10  10  10  11  12  13  14  15  16  17  18  19  20  23  9  10  10  10  11  12  13  14  15  16  17  18  19
[  1  11  11  2  3  15  5  6  7  10  11  11  11  12  13  14  15  16  17  18  19  20  23  9  10  11  11  11  12  13  14  15  16  17  18  19
[  2  12  12  12  3  15  5  6  7  10  11  12  12  12  13  14  15  16  17  18  19  20  23  9  10  11  12  12  12  13  14  15  16  17  18  19
[  2  2  13  13  13  15  5  6  7  10  11  12  13  13  13  14  15  16  17  18  19  20  23  9  10  11  12  13  13  13  14  15  16  17  18  19
[  3  2  3  14  14  15  5  6  7  10  11  12  13  14  14  14  15  16  17  18  19  20  23  9  10  11  12  13  14  14  14  15  16  17  18  19
[  3  2  3  4  15  15  5  6  7  10  11  12  13  14  15  15  15  16  17  18  19  20  23  9  10  11  12  13  14  15  15  15  16  17  18  19
[  3  2  3  4  15  16  16  6  7  10  11  12  13  14  15  16  16  16  17  18  19  20  23  9  10  11  12  13  14  15  16  16  16  17  18  19
[  3  2  3  4  15  17  17  17  7  10  11  12  13  14  15  16  17  17  17  18  19  20  23  9  10  11  12  13  14  15  16  17  17  17  18  19
[  3  2  3  4  15  6  18  18  18  10  11  12  13  14  15  16  17  18  18  18  19  20  23  9  10  11  12  13  14  15  16  17  18  18  18  19
[  3  2  3  4  15  6  7  19  19  10  11  12  13  14  15  16  17  18  19  19  19  20  23  9  10  11  12  13  14  15  16  17  18  19  19  19
[  3  2  3  4  15  6  7  8  20  10  11  12  13  14  15  16  17  18  19  20  20  20  23  9  10  11  12  13  14  15  16  17  18  19  20  20
[  3  2  3  4  15  6  7  8  20  10  11  12  13  14  15  16  17  18  19  20  21  21  23  9  10  11  12  13  14  15  16  17  18  19  20  21
```

Figure 4.14: extrait de la matrice des prédécesseurs

3. Determination_de_trajectoire : Le troisième programme, après avoir obtenu les coordonnées du robot, demande à l'utilisateur de saisir les coordonnées des points d'arrivée. Un test d'existence est effectué et une séquence est créée seulement si les points existent.

```
boutalbi@boutalbi-VirtualBox:~/catkin_ws/src/turtlebot3/memoire/src$ ./create_coordonnee.py
extraire info
extraction terminer
Obtention des coordonnees du robot...
('x en mettre =', -0.5000296281926159)
('y en mettre =', -1.999931934123057)
('x=', 50)
('y=', 57)
Position de depart obtenue : (50, 57)
Veuillez entrer les coordonnee du point d'arrivee (format x,y) :
Point d'arrivee : 35,43
Le point de depart existe.
Le point d'arrivee existe.
("Le chemin du point de depart au point d'arrivee est :", [272, 252, 226, 207, 192, 177, 158,
157, 133, 115, 100, 86, 70, 49, 29, 13, 12])
```

Figure 4.15: génération de la séquence des points le nœuds 1

4. Server et Controleur_robot : Le quatrième et le cinquième programme utilisent le serveur de service de ROS pour communiquer entre eux. Le quatrième récupère les coordonnées enregistrées précédemment, attend la demande du cinquième et les lui transmet.
5. La figure 4.16 montre les trois nœuds ROS en action : le nœud determination_de_la_trajectoire reçoit les coordonnées du robot par le topic /odom, qui reçoit les informations d'odométrie par /gazebo. Le nœud controleur_robot envoie les commandes au topic cmd_vel pour contrôler le mouvement du robot, et ce dernier envoie les commandes à /gazebo. Le nœud server permet de récupérer et envoyer les données nécessaires au controleu

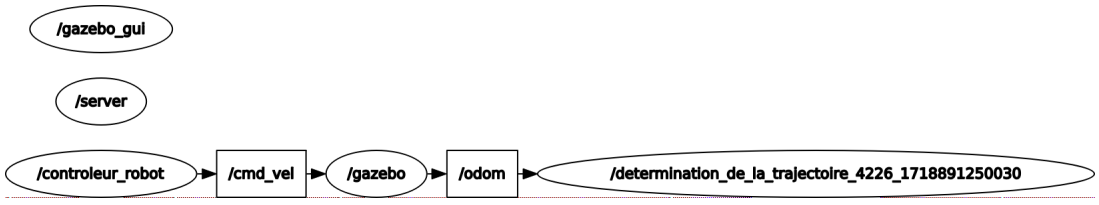

Figure 4.16: schéma explicatif de communication entre nœuds ros

4.4.2 Comparaison des résultats

Lorsque nous donnons les mêmes coordonnées aux deux programmes (celui sans l'algorithme de Floyd-Warshall et celui avec cet algorithme), nous remarquons que le premier heurte les obstacles, contrairement au second qui les évite.

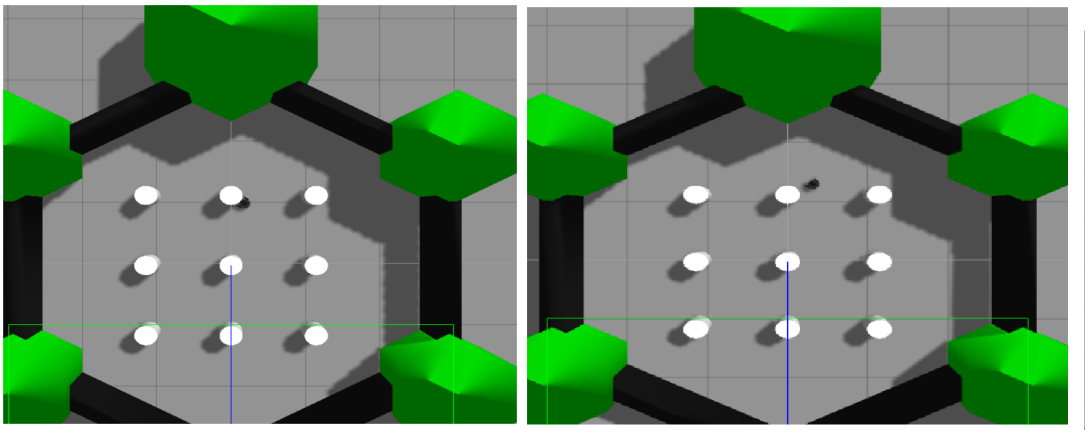

Figure 4.17: mouvement du robot avec le programme de robotis (-c-) et notre programme (-d-)

— Le point de départ du robot est la position initiale de déploiement du robot. Selon le repère de Gazebo, les coordonnées de la position de départ sont (-2, -0,5), et selon notre programme, elles sont (50, 57).
— Le point d'arrivée du robot est la position finale. Selon le repère de Gazebo, les coordonnées de ces points sont (2,8, 0,4), et selon notre programme, elles sont (35, 42).

4.5 Conclusion

En résumé, notre approche utilise l'algorithme de Floyd-Warshall pour planifier des trajectoires optimisées pour un robot mobile dans un environnement avec des obstacles. Les nœuds ROS implémentés permettent une communication efficace entre les différentes parties du système, assurant une planification et une exécution précises. Grâce à cette approche, nous pouvons garantir que le robot navigue de manière sûre et efficace, en évitant les obstacles et en atteignant les points de destination définis.

Conclusion Générale

Ce mémoire a parcouru les différentes phases de la planification de trajectoire pour les robots mobiles, de la modélisation initiale de l'environnement à la mise en œuvre d'une méthode basée sur la programmation dynamique pour la recherche de chemins optimaux, illustrée par l'utilisation de l'algorithme de Floyd-Warshall. Cette approche exacte permet de déterminer les chemins optimaux entre chaque paire de points dans un graphe.

L'application de cette méthode dans l'environnement de simulation appelé ROS a démontré son efficacité en tant que technique de planification globale. Les résultats obtenus attestent que l'approche proposée est non seulement théoriquement robuste, mais aussi pratique et applicable dans divers environnements statiques.

Pour les futures recherches, plusieurs pistes sont envisageables. Tout d'abord, l'utilisation des algorithmes de planification pour des environnements plus complexes et dynamiques. L'intégration de techniques de machine learning représente une autre voie prometteuse pour renforcer l'adaptabilité des robots aux conditions changeantes de leur environnement. Enfin, l'exploration de nouvelles plateformes robotiques permettrait d'étendre l'applicabilité de cette méthode à différents types de robots et d'applications.

En conclusion, ce mémoire a posé des bases solides pour la planification de trajectoire dans le domaine de la robotique mobile, tout en ouvrant la voie à des développements futurs qui pourraient rendre cette technologie encore plus performante et polyvalente.

Bibliographie

[1] *Organisation internationale de normalisation, Norme : ISO 8373 :1994 - Manipulateurs et robots industriels - Vocabulaire.*

[2] larousse.fr/dictionnaires/francais/robot/69647. *LAROUSSE.*

[3] Robotics Society of Japan. History of robotics research and development of japan.

[4] Robotics Industries Association. https ://www.robotics.org.

[5] Future-Sciences. sciences.com/tech/dossiers/robotique-robotique-a-z-178/page/2/.

[6] Clifford A.pickover. *La fabuleuse histoire de l'inteligence artificielle.* 2021.

[7] Robotics : A very short introduction (very short introductions.

[8] Jean-Claure Heudin. une histoire de la machine a penser. 2009.

[9] La robotique aux cycles 3 et 4. *https ://atelier-canope-95.canoprof.fr/*, 1959-1961.

[10] Shakey the robot. *sri.com/hoi/shakey-the-robot.*

[11] Les Earnest. Stanford cart. *Stanford University*, December 2012.

[12] Yoshihiko Nakamura. *The Development of Mobile Robots*, 1997.

[13] NASA. Mars pathfinder. *science.nasa.gov*, 1996-1997.

[14] Hendrik Dahlkamp David Stavens Andrei Aron James Diebel Philip Fong John Gale Morgan Halpenny Gabriel Hoffmann Kenny Lau Celia Oakley Mark Palatucci Vaughan Pratt Sebastian Thrun, Mike Montemerlo and Pascal Stang. Stanley : The robot that won the darpa grand challenge. *Journal of Field Robotics*, 2006.

[15] Bastien L. Bm watson : tout savoir sur le superordinateur ia et big data. Juillet 2019.

[16] Sophia. *Hanson Robotics : https ://www.hansonrobotics.com/.*

[17] Greian April Pangilinan Nazim Hussain1. Automation in robotics with artificial intelligence. *Aptisi Transactions on Technopreneurship (AT*, 2023.

[18] James G.Bellingham Et al. Robotics in remote ans hostile environments. *Science*, 318,1098 (2007).

[19] IFR. *Service Robots*. Fédération Internationale de la Robotique, 2021.

[20] Melissa Heikkila. Three reasons robots are about to become way more useful. *MIT Technologu Review*, 2024.

[21] Unmanned systems. *steatite-communications.co.uk/applications/unmanned-systems/*.

[22] thèse Doctorat : Ouelmokhtar Hand. *USV covering path planning under energy constraints*. PhD thesis, University M'Hamed Bougara of Boumerdes-Faculty of technology-Department of mechanical engineering, 2022.

[23] Philippe fraisse Sebastien Lengagne, Nacim Ramdani. Méthode pour la planification de trajectoires garanties. Juin 2008 -Mets- France.

[24] Heike Ripphausen-Lipa J.M.Adam. Algorithmes de plus court chemin. Beuth University of Applied Science- Berlin and Université de grenoble Alpes-Grenoble.

[25] geeksforgeeks.org. Floyd warshall algorithm. 04 Avril 2024.

[26] Amanda Dattalo. Ros introduction. In *ROS WIKI :https ://wiki.ros.org/*. 2018.

[27] Ammar Azab. Ros concepts. In *ROS WIKI : https ://wiki.ros.org/*. 2022.

[28] Tully Foote. Urdf overview. In *ROS WIKI : https ://wiki.ros.org/*. 2023.

[29] David Park. Turtlebot3. In *Robotis E-Manual*. GitHub, 2021.

[30] David Park. Turtlebot3. In *Robotis E-Manual*. Features, 2021.