République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Amar Thelidji- Laghouat

Faculté: De Technologie
Departement : D'électronique

MEMOIRE DE MASTER

Présenté par :

- **Boussouri Masseauda**
- **Zaza Ahlam**

Domaine : Science Et Technologie
Filière : Electronique
Option : Electronique Des Systèmes Embarqués

Thème

Planification de trajectoire et commande d'un robot mobile en Utilisant l'interface MATLAB-ROS

Devant le jury composé de :

Nom	Grade	Qualité
Chouireb Fatima	*Prof*	*Encadreur*
Ben chrif Aissa	*prof*	*Co-encadreur*
Rougab Ilyas	*MCA*	*Président*
Taaba Kheira	*MA*	*Examinateur*

Promotion : 2021/2022

Remerciement

Avant tous, Nous remercions en premier lieu notre Dieu de nous avoir Donné la santé et la patience pour avoir terminé ce travail.

Un très grand merci à l'endroit de notre promotrice Mme. CHOUIREB FATIMA pour son soutien et ses conseils tout au long de ce travail. Ses Encouragements et ses conseils ont toujours été une source de motivation. Ce fut Un honneur pour nous de travailler sous sa supervision

Nous tenons à exprimer notre sincère gratitude à monsieur AISSA BENCHRIF pour son aide et ses Conseils durant l'élaboration de ce projet.

Aussi, nous tenons à présenter nos remerciements munis d'expression de reconnaissance et de considération à tous les professeurs et au cadre administratif de Département d'Electronique de l'université de AMAR THELIDJI-Laghouat

Nous tenons également à remercier les membres du jury qui superviseront le jugement de nos travaux et apporteront leurs idées et suggestions scientifiques

Nos derniers remerciements à tous ceux qui ont contribué de près ou De loin à la réalisation de ce projet.

Dédicace

J'ai le gronde plaisir de dédie ce modeste travail :

A me très chère grande mère, qui me donne toujours l'espoir de vivre et qui n'a jamais cessé de prier pour moi

A Mes chers parents que je ne remercierai jamais assez pour tous les sacrifices, Leur confiance, leur soutien et toutes les valeurs qu'ils ont su m'inculquer.

A Mes très chères sœurs ***« Khadija, Zohra, Chaima, Nourelhouda, Nadjwa»*** *et Mes chers frères* ***« Bilal, Mohamed, Abd elrezak, Taher, Sliman, Khaled »****pour leurs appui et leur précieux encouragement.*

Et A toutes mes amies et surtout ***« Mounir»*** *pour le soutenir et rester à mes côtés pendant mes moments les plus difficiles tout au long de cette année universitaire*

A l'âme de ma chère tonte ***« Fatna »,*** *que dieu lui fasse miséricorde, qui a toujours été un bon modèle pour moi dans cette vie*

Et A toute ma chère famille sans exception

Tous ceux que j'aime, et tous ceux qui m'aiment

Masseauda

Dédicace

J'ai le gronde plaisir de dédie ce modeste travail :

A la plus belle créature que Dieu a créée pour moi sur terre, A mon père, la source de mon bonheur et de mon énergie, il a été la raison de mon succès et des prières pour moi. A mon bien-aimé, mon cher père, que Dieu lui fasse miséricorde et fasse de lui un des gens du Paradis

A ma source de tendresse, de patience et de générosité, A ma maman qui m'a soutenu et encouragé durant ces années d'études

*A mes frères et mes modèles "**Gharbi, Taher, Mohammed, Abdelfattah** "*

A mes chères sœurs sans exception et leurs enfants,

A mon mari et ma deuxième famille

Ahlam

ملخص

الروبوتات هي المجال الذي تتقاطع فيه معظم علوم الهندسة والتكنولوجيا،، لتحل محل السلوك البشري أو تكرره من خلال إنتاج آلات تسمى الروبوتات, إنه مجال يتطلب معرفة ومهارات تقنية متعددة من تخصصات مختلفة حتى يتمكن الباحثون من أداء بتنفيذ عمل تجريبي مهم، مثل التحكم والمحاكاة و تتبع الأخطاء و التنبؤ لتسهيل البحث و التطوير في هذا المجال المهم للغاية.

الهدف من عملنا هو إيجاد المسار الأمثل للروبوت المتحرك بالانتقال من نقطة البداية الى نقطة الوجهة في بيئة معروفة واتباع المسار مع تجنب العوائق الثابتة و الديناميكية. للقيام بذلك، درسنا أولا خوارزميات التخطيط الاكثر شهرة،و هي طرق PRM وRRT وDstar Lite Astar. بعد ذلك، في الخطوة الثانية، قمنا بدراسة و تنفيذ قوانين التحكم PurePursuit و نهج يعتمد على وظيفة Lyapunov .تم تنفيذ كل هذه الخوارزميات تحت Matlab وواجهة ROS- MATLAB بالإضافة الى محاكي GAZEBO النتائج التي تم الحصول عليها *حاسمة للغاية*

الكلمات المفتاحية: الروبوتات، الروبوت المتحرك، التحكم، تتبع المسار،ROS.

Résumé

La robotique est le domaine où la plupart des sciences de l'ingénierie et de la technologie se croisent, remplaçant ou reproduisant le comportement humain par la production des machines appelées robots. C'est un domaine qui exige de multiples connaissances et compétences techniques de différentes disciplines afin que les chercheurs puissent effectuer d'importants travaux expérimentaux, tels que le contrôle, la planification de trajectoire, le suivi et la prévision des erreurs, pour faciliter la recherche et le développement dans ce domaine très important.

Le but de notre travail est de trouver le chemin optimal pour permettre à un robot mobile de se déplacer d'un point de départ à un point destination dans un environnement connu et de suivre ce chemin tout en évitant les obstacles statiques et dynamiques. Pour ce faire, nous avons étudié dans un premier temps, les algorithmes de planification les plus connus, à savoir les méthodes PRM, RRT, Astar et Dstar Lite. Ensuite, dans un deuxième temps, nous avons étudié et implémenté deux lois de commande : PurePursuit et une approche basée sur la fonction de Lyapunov. Tous ces algorithmes ont été implémentés sous Matlab et l'interface MATLAB-ROS ainsi que le simulateur Gazebo. Les résultats obtenus sont très concluants.

Mot clés : robotique, ROS, robot mobile, contrôle, suivi de trajectoire

Abstract

Robotics is the area where most of the sciences of engineering and technology intersect, replacing or replicating human behavior through the production of machines called robots. It is an area that requires multiple knowledge and technical skills from different disciplines so that researchers can perform important experimental work, such as control, trajectory planning, error tracking and prediction, to facilitate research and development in this very important area.

The goal of our work is to find the optimal path to allow a mobile robot to move from a starting point to a destination point in a known environment and to follow this path while avoiding static and dynamic obstacles. To do this, we first studied the best-known planning algorithms, namely the PRM, RRT, Astar and Dstar Lite methods. Then, in a second step, we studied and implemented two control laws: PurePursuit and an approach based on the Lyapunov function. All these algorithms have been implemented under Matlab and the MATLAB-ROS interface as well as the Gazebo simulator. The results obtained are very conclusive.

Keywords : Robotics, mobile robot, control, trajectory traking, ROS

Liste des Abréviations

RRT : Probabilistic Roadmap

PRM : Rapidly-exploring Random Tree

LPA* : Lifelong planning A star

Rhs : Right Hand Side

ROS : Robot Operating System

SLAM : Simultaneous Localization And Mapping

Table des Matières

Chapitre I : Robot Mobile, Modélisation Et Commande

Table des Matières

Chapitre II : Planification de trajectoire

Chapitre III : Résultats et interprétations

Table des Matières

Liste des Figures

Chapitre I : Robot Mobile, Modélisation Et Command

Chapitre II: planification de trajectoire

Chapitre III : Résultats et interprétations

Liste des Figures

Liste des Tableaux

Chapitre I : Robot Mobile, Modélisation Et Commande

Introduction générale

Introduction générale

Le besoin humain de la machine était l'émergence d'une nouvelle science, la robotique. La science robotique est apparue au début du milieu du siècle dernier et reste une science de la future. La robotique est un très bon exemple de domaine pluridisciplinaire qui implique de nombreuses thématiques telles que la mécanique, la mécatronique, l'électronique, l'automatique, l'informatique ou l'intelligence artificielle.

Le développement des moyens informatiques dont disposent les chercheurs en robotique rend de plus en plus facile l'utilisation d'algorithmes puissants pour donner aux robots une capacité d'autonomie.

Dans ce mémoire, on s'est intéressé à la planification de trajectoire et la commande du robot mobile unicycle. L'idée consiste à étudier et implémenter des méthodes et des algorithmes permettant de piloter le robot d'un point de départ à un point d'arrivée selon une planification de trajectoire bien définie grâce aux logiciels de simulation MATLAB-ROS et GAZEBO. L'objectif de ce travail est d'étudier et d'appliquer des méthodes globales A star et D star Lite et la comparaison de celle-ci à d'autres méthodes probabilistes à savoir : la méthode RRT, RPM. Après la planification du chemin optimal du robot mobile, on a besoin d'une méthode de commande permettant au robot de suivre ce chemin avec le maximum de précision. Pour ce faire, on a choisi deux méthodes : une méthode basée sur la fonction de Lyapunov et la méthode PurePursuit. Ainsi, le suivi de trajectoire est le processus qui consiste à déterminer les paramètres de vitesse et de direction à chaque instant afin que le robot suive le chemin désiré.

Notre mémoire est organisé en trois chapitres qui peuvent être ainsi résumés :

- ***Chapitre 1: Robot mobile, Modélisation et Commande***

  Le chapitre 1 présente une introduction sur la robotique en donnant quelques notions principales sur les robots mobiles : leurs définitions, classifications, historique, caractéristiques, Navigation, modélisation, etc. La commande est une tâche importante et nécessaire dans l'étude des robots. Par conséquent, nous proposerons plusieurs techniques pour contrôler un robot mobile à la fin de ce chapitre.

- ***Chapitre2 : planification de trajectoire***

Le deuxième chapitre décrit les différentes méthodes de planification de trajectoire, ainsi que leurs avantages et inconvénients.

- ***Chapitr3: Résultats et discussions***

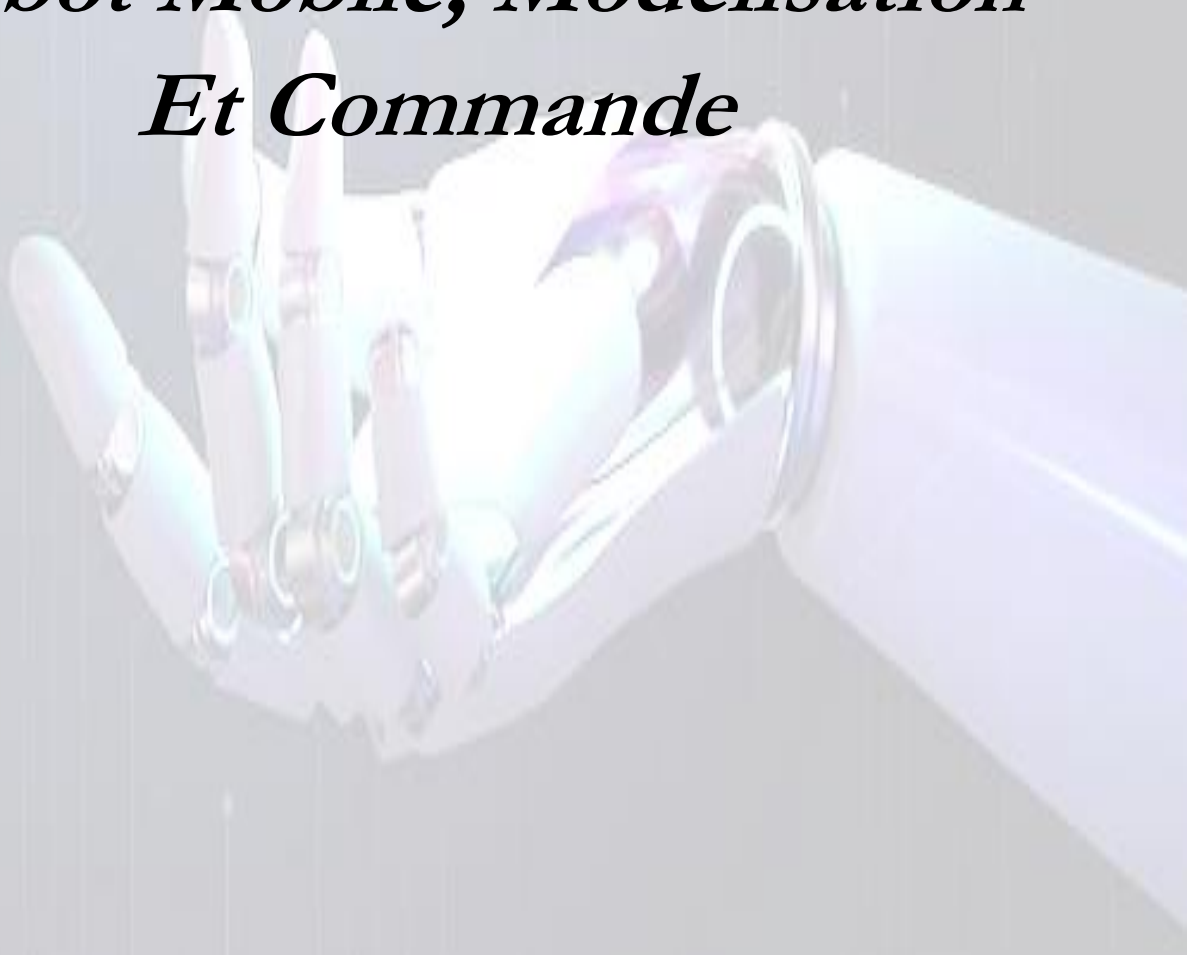
Le dernier chapitre donne les outils et interfaces qu'on a utilisés pour la planification des trajectoires. Il illustre également les différentes étapes et les résultats d'implémentation des méthodes de planification et de commande étudiées aux chapitres précédents.

Enfin, nous terminerons ce manuscrit par une conclusion générale récapitulant ce qui a été fait et exposant les perspectives de ce travail.

.

Chapitre I

Robot Mobile, Modélisation Et Commande



I.1. Introduction

Les robots mobiles à roues ont fait l'objet de beaucoup de recherches dans différents domaines de la robotique. Ces robots ont un champ d'application très large, que ce soit pour des applications civiles comme les transports urbains automatisés, scientifique dans le cas d'exploration de milieux naturels hostiles ou encore militaire.

L'étude de ces robots se fait suivant plusieurs domaines, on cite en premier, la modélisation des robots en donnant une représentation mathématiques du robot mobile. En second, la planification de la trajectoire pour le robot à commander. Et enfin, la commande des robots mobiles. Les robots mobiles à roues existants sont des systèmes complexes à cause de leur non- linéarité et non holonomie. En plus, ce sont des systèmes sous actionnés.

Dans ce chapitre, on présente certaines notions importantes liées à la robotique mobiles et à ces applications. On introduit en premier ce que signifie la robotique. On présente juste après les caractéristique liées aux différents types des robots existants dans la littérature ainsi que les différentes formes mathématiques existantes et pouvant les représenter. En enfin, on présente les différentes tâches de commande pouvant être appliquées sur les robots mobiles à roues de type unicycle.

I.2. Généralités sur la robotique

I.2.1. Définition de la robotique

La robotique est la science qui étudie les systèmes électromécaniques actionnés et contrôlés par le biais d'un ensemble de logiciels leur conférant une intelligence dite artificielle; c'est aussi l'ensemble des techniques permettant la conception et la réalisation de machines automatiques ou de robots.

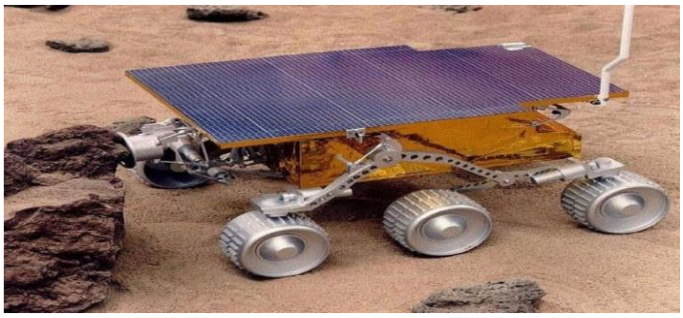
La robotique est pluridisciplinaire implique de nombreuses thématiques telles que la mécanique, l'électronique, l'automatique, l'informatique ou l'intelligence artificielle.

I.2.1. Définition d'un robot mobile

Un robot mobile est un dispositif mécanique (alliant mécanique, électronique et informatique) doté de capteurs et d'effecteurs lui donnant une capacité d'adaptation et de déplacement proche de l'autonomie accomplissant automatiquement soit des tâches qui sont

Généralement dangereuses, pénibles, répétitives ou impossibles pour les humains, soit des tâches plus simples mais en les réalisant mieux que ce que ferait un être humain [1].

D'une manière générale, on peut définir un robot comme étant un agent physique réalisant des tâches dans l'environnement dans lequel il évolue [2]. La figure (I.1) ci-dessous montre une photo du robot mobile "Sojourner20" de la NASA.



FigureI. 1: Le robot mobile "Sojourner20" de la NASA.

I.3. Historique de la robotique

I.3.1. Les origines de la robotique

Ancêtres des robots sont les automates. Un automate très évolué fut présenté par Jacques de Vaucanson en 1738 : il représentait un homme jouant d'un instrument de musique à vent. Jacques de Vaucanson créa également un automate représentant un canard mangeant et refoulant sa nourriture après ingestion de cette dernière [3].

I.3.2. Les premiers robots

Unimate est le premier robot industriel créé. Il fut intégré aux lignes d'assemblage de Général Motors en 1961.

En 1970, le robot lunaire Lunokhod 1, envoyé par l'union soviétique, a voyagé sur une distance de 10 Km et a transmis plus de 20 000 images.

I.3.3 .Dates marquantes de la robotique

La robotique est passée par plusieurs générations comme suit [4] :

- 1947 : Premier manipulateur électrique téléopéré.
- 1948 : Grey Walter invente le premier robot mobile autonome, une tortue se dirigeant vers les sources de lumière qu'elle perçoit. Cependant, ce robot n'est pas programmable.

- 1954 : Premier robot programmable.
- 1961 : Premier robot industriel mis en place dans une usine de General Motors. Qui a fait le premier robot avec contrôle en effort.
- 1963 : Utilisation de la vision pour commander un robot.
- 1972 : Nissan ouvre la première chaine de production complètement robotisée.
- 1973 : Premier robot mobile à roues.
- 1977 : Premier robot mobile français HILARE au LAAS (CNRS Toulouse).
- 1979: Les chercheurs de L.A.A.S. DE TOULOUSE (France) étudièrent la planification des trajectoires pour le robot HILARE dans un environnement totalement connu.
- 1981 : Le robot VESA. Ce robot, construit à l'I.N.S.A(France) de Rennes, est équipée d'un arceau de sécurité pour réaliser la détection d'obstacles dans un environnement totalement inconnu.
- 1992 : Mise en place de la compétition annuelle AAAI sur la robotique mobile.
- 1995 : Mise en place de la Roba Cup.
- 1997 : Premier robot mobile extra planétaire sur Mars.

Depuis 2000 : Exploration

- 2003 : Projet « *Mars Exploration Rover* » (Spirit &Opportunity).
- 2009 : Projet « *Mars Science Laboratoire* » succédant au projet Rover, envoi prévu de Curiosity fin 2001.

I.4. Différents types de robots mobiles

Le robot mobile est doté de moyens qui lui permettent de se déplacer dans son espace de travail. Suivant son degré d'autonomie ou degré d'intelligence, il peut être doté de moyens de perception et de raisonnement. Certains sont capables sans contrôle humain de réduire, de modéliser leur espace de travail et de planifier un chemin dans un environnement qu'ils ne connaissent pas forcément d'avance [5].

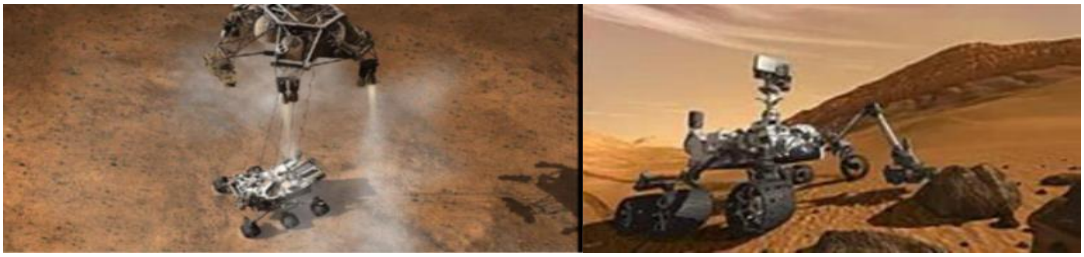
I.4.1. Robots militaires

Les robots militaires sont principalement utilisés pour la surveillance aussi bien dans les airs que dans la mer. Les drones sont une sous-classe des robots militaires. Des systèmes sont déjà actuellement en service dans un certain nombre de forces armées avec des succès remarquables, tel que le drone "Predator" qui est présenté dans la figure (I.2) [6].

FigureI. 2: Drone "Predator".

I.4.2. Robots explorateurs

Les robots explorateurs remplacent l'homme dans des environnements difficiles. Nous pouvons citer comme exemple: *Mars Science Laboratory* (MSL) qui est une mission d'exploration de la planète Mars à l'aide d'une automobile « Rover» (voir la figure I.3), développée par l'agence spatiale américaine de la NASA [7].



FigureI. 3:Représentation artistique d'un rover martien de la NASA [7].

I.4.3. Robots humanoïdes

Le terme humanoïde signifie « de forme humaine », il évoque principalement la bipédie, la présence de deux bras et d'une tête. Il s'agit donc uniquement d'un critère phénotypique plus précisément morphologique comme illustre la figure 1.4 [8].

FigureI. 4: Robot humanoïde [8].

Il existe d'autres catégories de robots: robots sous-marins, volants, manipulateurs, ... etc.

I.4.4. Classifications des robots mobiles

La classification des robots mobiles se fait suivant plusieurs critères (degré d'autonomie, système de locomotion, énergie utilisée, ...etc).

La classification la plus intéressante et la plus utilisée est selon leur degré d'autonomie. Un robot mobile autonome est un système automoteur doté de capacités décisionnelles et de moyens d'acquisition et de traitement de l'information qui lui permettent d'accomplir sans contrôle humain un certain nombre de tâches dans un environnement non complètement connu. Nous pouvons citer quelques types:

- ✓ Véhicule télécommandé par un opérateur (ces robots sont commandés par un opérateur qui leur impose chaque tâche élémentaire à réaliser).
- ✓ Véhicule télécommandé au sens de la tâche à réaliser (le véhicule contrôle automatiquement ses actions).
- ✓ Véhicule semi-autonome (ce dernier réalise des tâches avec une intervention partielle de l'opérateur)
- ✓ Véhicule autonome (ce type de véhicule réalise des tâches sans 1'aide de 1'opérateur) [9].

Selon leur système de locomotion, les robots mobiles peuvent aussi être classés comme dans la figure(I.5).

FigureI. 5: La classification des robots mobiles selon le type de locomotion.

I.5. Caractéristiques des robots

Un robot doit être choisi en fonction de l'application à laquelle il est réservé à faire. Voici quelques paramètres à prendre éventuellement en compte :

- ✓ ***La charge maximale transportable***

  De quelques kilos à quelques tonnes, à déterminer dans les conditions les plus défavorables.

- ✓ ***La répétabilité***

  Ce paramètre caractérise la capacité du robot à retourner vers un point (position, orientation) donné [10].

- ✓ ***La vitesse de déplacement***

  Vitesse maximale en élongation maximum ou accélération.

Il existe d'autres caractéristiques comme: la masse du robot, le coût du robot, la maintenance, …etc.

I.6. Domaines d'application des robots

La robotique est un domaine en plein essor depuis quelques années. Les évolutions technologiques dépassant sans cesse nos espérances, permettent maintenant de réaliser des solutions technologiques s'adaptant au moindre problème. Par conséquent, la robotique est utilisée dans des domaines extrêmement rigoureux et exigeants.

Le domaine d'application des robots est vaste, nous présentons quelques applications dans le tableau (1.1) [11].

Domaines	**Applications**
Industrie nucléaire	✓ Surveillance de sites ✓ Manipulation de matériaux radioactifs
Sécurité civile	✓ Neutralisation d'activité terroriste ✓ Déminage ✓ Pose d'explosif ✓ Surveillance de munitions

Militaire	✓ Surveillance ✓ Pose d'explosifs ✓ Manipulation de munitions
Chimique	✓ Surveillance de site ✓ Manipulation de matériaux toxiques
Médecine	✓ Assistance d'urgence ✓ Aide aux handicapés physiques

Tableau I.1 : Applications des robots

I.7. Modélisation d'un robot mobile unicycle

I.7.1. Objectif de la modélisation du robot

La modélisation d'un robot, considéré comme étant un système mécanique articulé, actionné et commandé, consiste à en établir un modèle mathématique. Outre une fonction générale d'aide à la conception, elle a de multiples utilisations pour, la prédiction des mouvements, l'adaptation des actionneurs, la planification des tâches, l'établissement des lois de commande, l'incorporation du robot dans des simulations informatiques...etc.

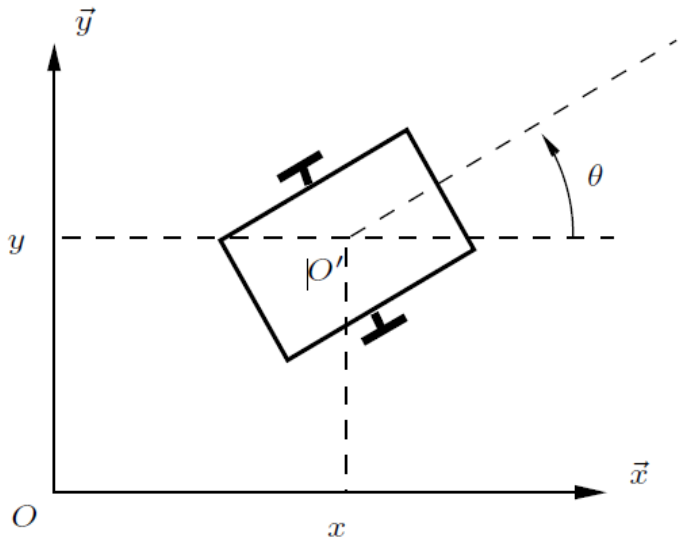
I.7.2. Définitions

- ***Définition***

La modélisation consiste à mettre au point un ensemble d'équations ou de règles pour décrire un phénomène de façon reproductible et simulable.

I.7.3. Description

On désigne par unicycle un robot actionné par deux roues indépendantes et possédant éventuellement un certain nombre de roues folles assurant sa stabilité. Le schéma des robots de type unicycle est donné à la figure (I.6). On y a omis les roues folles, qui n'interviennent pas dans la cinématique, dans la mesure où elles ont été judicieusement placées.



FigureI. 6: Robot mobile de type Unicycle.

Ce type de robot est très répandu en raison de sa simplicité de construction et de ses propriétés cinématiques intéressantes. La figure (I.7) présente deux robots de type unicycle.

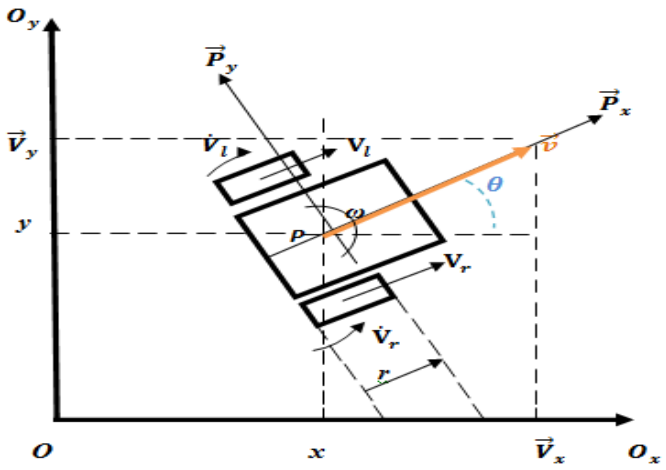
(a): Robot mobile MIABOT (b): Robot mobile Pioneer P3-DX

FigureI. 7: Robots mobiles unicycle

I.7.4. Modèle Cinématique du Robot mobile unicycle :

La modélisation cinématique traite des relations géométriques qui contrôlent le robot mobile sans tenir compte des forces qui l'affectent. Par exemple, il fait référence à l'évolution de la position et des vitesses du système de robot mobile, sans se référer à sa masse et à son couple. On considère le modèle représenté sur la figure (I.8).

Les paramètres de configuration sont donnés par les coordonnées du point *P(x, y, θ)* représentant sa position par rapport à un référentiel fixe et son orientation θ par rapport à l'axe des abscisses Ox.



FigureI. 8: Modélisation cinématique d'un robot mobile de type unicycle

Il reste maintenant à voir les équations cinématiques du robot, c'est-à-dire le lien entre les vitesses linéaires et angulaires du robot et la dérivée de la position du robot. Ceci permet de Comprendre comment les vitesses linéaires et angulaires induisent le changement de position du robot.

Nous définissons pour cela un deuxième repère lié au robot : **(P ,P_x, P_y).** Plus précisément, nous considérons que l'origine ***P*** du repère se trouve au centre de l'axe des roues motrices du robot.

Ainsi, le modèle cinématique du robot est donnée par [12]:

$$\begin{aligned} \dot{x} &= v.\,\mathrm{Cos}\,\theta. \\ \dot{y} &= v.\mathrm{Sin}\theta. \\ \dot{\theta} &= \omega. \end{aligned} \qquad \textbf{(I.1)}$$

- ✓ Les différents paramètres de ce modèle sont définis dans le tableau (I.2).
- ✓ Le modèle du robot unicycle (I.1) peut être réécrit sous forme matricielle comme suit:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} . \begin{bmatrix} v \\ \omega \end{bmatrix} \qquad \textbf{(I.2)}$$

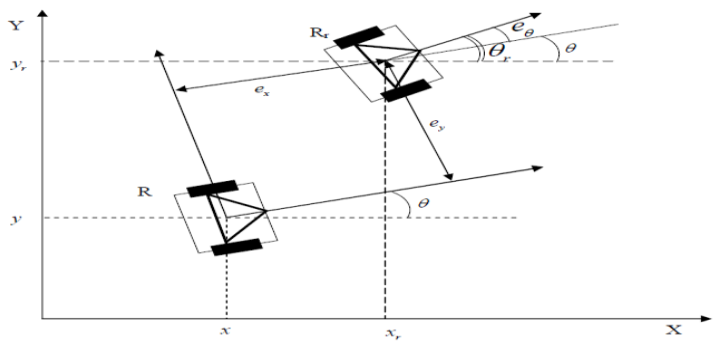
Où $v = \frac{(v_r + v_l)}{2}$ et $\omega = \frac{(v_r - v_l)}{d}$

Symboles	**Unités**	Paramètres
O		Un point immobile dans le domaine d'évolution du robot
P		Un point fixe sur le châssis
[O, Ox, Oy]		Un repère immobile dans le plan d'évolution du robot
$[P, \overrightarrow{P_x}, \overrightarrow{P_y}]$		Un repère attaché au châssis
$\dot{v}_r$,$\dot{v}_l$	rad/s	Vitesses des roues droite et gauche
v_r,v_l	m/s	Vitesses linéaires des roues droite et gauche
r	m	Rayon des roues droite et gauche
(x, y)	m	Position du robot dans le plan (les coordonnées de P dans [O, Ox, Oy])
θ	rad	Orientation du robot dans le plan (l'orientation du repère $[P, \overrightarrow{P_x}, \overrightarrow{P_y}]$par rapport au repère [O, Ox, Oy])
$(\dot{x}, \dot{y})$	m/s	Vitesses du robot dans le plan
$\vec{v}$	m/s	Vitesse linéaire du robot
ω	rad/s	Vitesse angulaire du robot
d	m	Distance entre les deux points de contact au sol des roues droite et gauche

Tableau I.2 : Paramètres du modèle du robot

I.7.5. Erreur de Posture :

Dans un problème de poursuite de trajectoire pour un robot mobile, nous nous intéressons à déterminer l'erreur entre la position du robot et la position désirée comme le montre la figure (I.9), selon le référentiel de base. Nous réalisons ensuite un changement de référentiel pour calculer cette erreur selon le référentiel lié au robot, pour implémenter la commande de ce dernier.



FigureI. 9 : Les coordonnées de l'erreur du robot unicycle

Voici l'équation de la trajectoire de référence (désirée) :

$$\dot{q}_r = \begin{bmatrix} \dot{x}_r \\ \dot{y}_r \\ \dot{\theta}_r \end{bmatrix} = \begin{bmatrix} v_r \cos\theta_r \\ v_r \sin\theta_r \\ w_r \end{bmatrix} \quad \textbf{(I.3)}$$

Selon la figure précédente, l'erreur de posture se calcule comme suite :

$$qr - q = \begin{bmatrix} e_x \\ e_y \\ e_\theta \end{bmatrix} = \begin{bmatrix} x_r - x \\ y_r - y \\ \theta_r - \theta \end{bmatrix} \quad \textbf{(I.4)}$$

En appliquant une transformation de coordonnés, nous aurons :

$$\begin{bmatrix} x_e \\ y_e \\ \theta_e \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} x_r - x \\ y_r - y \\ \theta_r - \theta \end{bmatrix} \quad \textbf{(I.5)}$$

Alors

$$x_e = e_x \cos(\theta) + e_y \sin(\theta)$$
$$y_e = e_y \cos(\theta) - e_x \sin(\theta)$$
$$\theta_e = e_\theta$$

Après dérivation du système d'équations (I.5) et en utilisant l'équation (I.3) ainsi que la contrainte de non holonomie $\dot{x}\sin\theta_r = \dot{y}\cos\theta_r$, nous obtenons les dynamiques de l'erreur comme suit :

$$\begin{bmatrix} \dot{x}_e \\ \dot{y}_e \\ \dot{\theta}_e \end{bmatrix} = \begin{bmatrix} w y_e - v + v_r \cos\theta_e \\ -w x_e + v_r \sin\theta_e \\ w_r - w \end{bmatrix} \quad \textbf{(I.6)}$$

I.8. Autonomie du robot mobile :

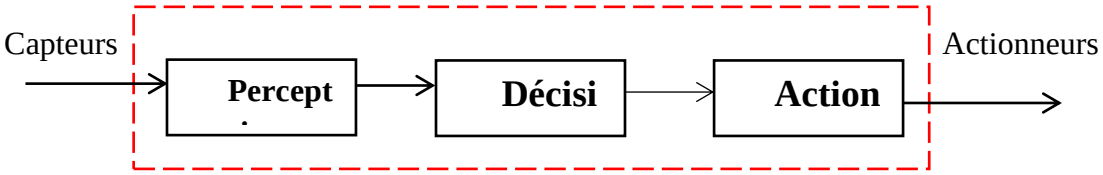
L'autonomie d'un robot mobile réside dans sa capacité de faire appel à une stratégie intelligente pour lui permettre de planifier ses actions à long terme, et qu'il est capable d'accomplir sans intervention humaine les objectifs pour lesquels il a été conçue.

Un robot autonome peut être défini comme étant l'association des capacités de perception, de modélisation de son environnement et de son propre état, et de capacités d'actions sur son propre état et sur son environnement. Pour cela, le robot doit suivre le schéma correspondant au paradigme (Perception-Décision-Action) [13].Pour que le robot mobile exécute

correctement ses tâches, il doit fournir à son système de contrôle un modèle interne de l'environnement dans lequel il interagit comme le montre la figure (I.10) [13].

Les diverses activités se amènent aux tâches suivantes :

- **Perception** : le robot doit acquérir des informations sur l'environnement dans lequel il évolue par l'intermédiaire de ses capteurs, qui peuvent être utilisés directement comme des entrées.
- **Décision** : le robot doit définir des séquences d'actions résultant d'un raisonnement appliqué sur un modèle de l'environnement ou répondant de manière réflexe à des stimulis étroitement liés aux capteurs.
- **Action :** il doit exécuter les séquences d'actions élaborées en envoyant des consignes aux actionneurs par l'intermédiaire des boucles d'asservissement.



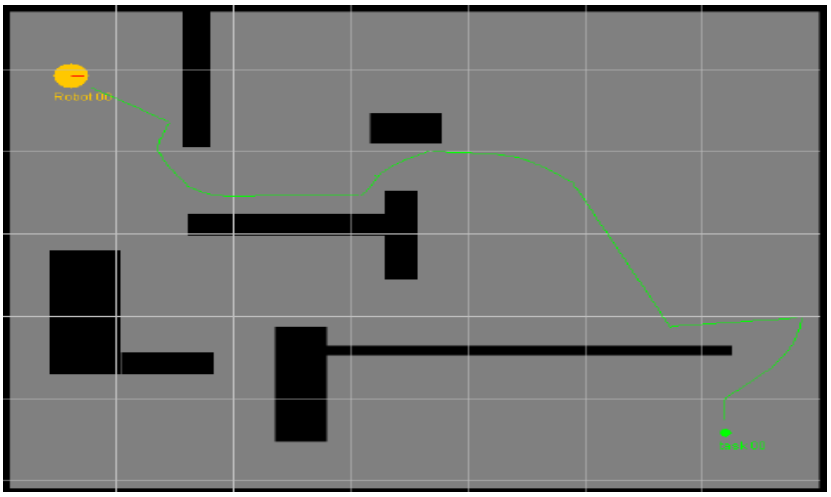
FigureI. 10: Etapes de traitement automatique

I.9. Navigation Autonome d'un robot mobile

La navigation d'un robot mobile est un des problèmes clés dans la communauté de la robotique. Le problème consiste à intégrer sur un robot réel, toutes les fonctions nécessaires pour qu'il puisse exécuter des déplacements à travers un environnement donné, en utilisant l'information perçue par ses capteurs.

Généralement, la navigation d'un robot mobile est une tâche qui consiste à trouver un mouvement libre dans l'espace de configuration sans collision avec les obstacles proche du robot. Ce mouvement amène le robot d'une configuration initiale, vers une position finale désirée.

Il existe plusieurs approches pour la navigation d'un robot mobile, mais entre toutes ces approches, les plus connues et aussi les plus utilisées sont : la planification et le suivi des trajectoires et l'évitement d'obstacles (Figure I.11).



FigureI. 11: Navigation d'un robot mobile [14]

- ***Planification de trajectoire***

Parmi les objectifs principaux du robot mobile est de pouvoir évoluer dans un environnement complexe encombré d'obstacles pour atteindre son but final. Il a besoin de construire une trajectoire définie comme une séquence de déplacement sans collision avec ces obstacles entre la position initiale (point de démarrage) et le point but ou cible.

Plusieurs approches sont proposées pour la planification de trajectoire. Cependant, les plus utilisées sont la planification globale et locale. Nous allons détailler quelques méthodes de planification de trajectoires dans le chapitre II de ce mémoire.

- ***Suivi de trajectoire***

Le suivi de trajectoires est une fonction essentielle des robots mobiles, elle consiste à calculer les commandes envoyées aux actionneurs permettant de réaliser le mouvement planifié pour guider le robot mobile dans un chemin prédéfini, entre une configuration de départ et une configuration d'arrivée données, dans un modèle géométrique de l'environnement.

- ***Evitement d'obstacles***

Éviter les obstacles est une fonction principale qui doit être disponible dans tous les robots mobiles, et c'est la capacité du robot à éviter les collisions avec des objets ou des obstacles grâce aux capteurs de distance.

I.10. Méthode de suivi de trajectoire

Dans cette partie, nous nous intéresserons dans un premier temps à donner une description théorique des méthodes utilisées pour commander le robot mobile de type

unicycle afin de suivre une trajectoire prédéfini, nous entendons par Loi de commande basée fonction de Lyapunov et la commande Pure Pursuit.

I.10.1. Loi de commande basée fonction de Lyapunov

a) Définition de la commande

L'approche de la commande par backstepping a été développée par P.V.Kokotovic. Cette méthode est principalement destinée aux systèmes non linéaires avec des incertitudes paramétriques. L'algorithme de cette commande permet de calculer pas à pas l'expression de la loi de commande en utilisant le principe de stabilité de Lyapunov (fonction de Lyapunov) afin d'obtenir une commande qui garantit la stabilité su système au sens de Lyapunov et qui permet de réaliser la tâche de poursuite de trajectoire.

Dans cette méthode, on considère certains variables d'états comme étant des commandes virtuelles qui sont des commandes intermédiaires pour la stabilisation de certains états du système. Ces commandes engendrent de nouvelles variables d'états qui sont considérées comme des commandes virtuelles jusqu'à apparition de la commandes réelle de façon explicite. Le calcul récursif de la fonction de Lyapunov permet par la suite de calculer les véritables commandes appliquées au système.

b) Description de la commande

Nous proposons d'utiliser une fonction de Lyapunov de la forme suivante pour calculer la commande [15] :

$$V = \frac{1}{2}x_e^2 + \frac{1}{2}y_e^2 + \frac{1}{c_2}(1 - \cos\theta_e)) \quad / \quad c_2 > 0 \qquad \textbf{(I.7)}$$

Où $[x_e, y_e, \theta_e]'$ est le vecteur d'erreur de posture entre la trajectoire désirée et la trajectoire actuelle du robot calculée dans un référentiel lié au robot (voir la section (I.7.5) de ce chapitre)).

La dérivée de cette fonction est :

$$\dot{V} = x_e\dot{x}_e + y_e\dot{y}_e + \frac{\sin(\theta_e)}{c_2}\dot{\theta}_e \qquad \textbf{(I.8)}$$

En remplaçant $\dot{x}_e, \dot{y}_e$ et $\dot{\theta}_e$ par leurs expressions à partir de l'équation (I.6) dans l'équation (I.8), nous obtenons :

$$\dot{V} = x_e(v_r cos(\theta_e) - v) + \frac{\sin(\theta_e)}{c_2}(w_r - w + c_2 v_r y_e) \qquad \textbf{(I.9)}$$

En posant

$$v = v_r \cos(\theta_e) + c_1 xe \qquad \textbf{(I.10)}$$

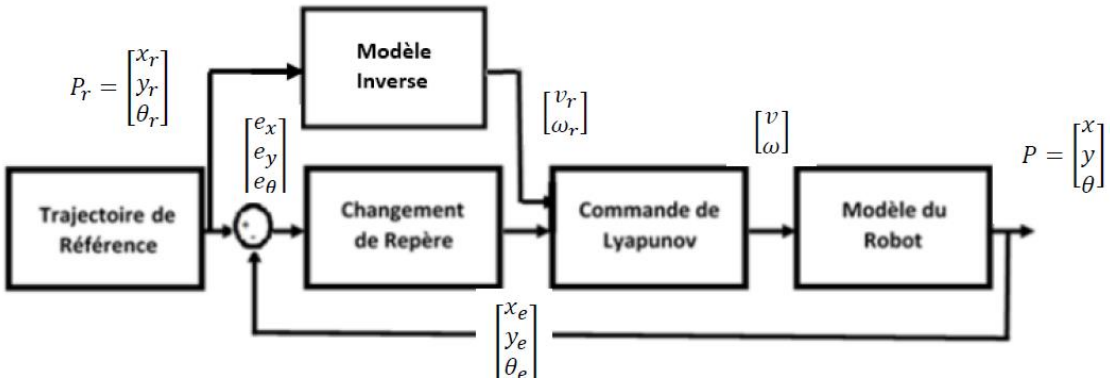
Et

$$\omega = \omega_r + c_2 v_r y_e + c_2 \sin(\theta_e) \quad \textbf{(I.11)}$$

Nous aurons alors :

$$\dot{V} = -x_e^2 c_1 - \frac{c_3}{c_2} \sin^2 \theta_e < 0 \qquad \forall [x_e, \theta_e] \neq [0,0] \quad \textbf{(1.12)}$$

Le schéma bloc de cette commande est donné dans la figure qui suit :



FigureI. 12: Schéma Bloc de la Commande de Backstepping

I.10.2. Commande Pure Pursuit

a) définition de la commande

L'approche de poursuite pure (PurePursuit) est une méthode de détermination géométrique de la courbure qui conduira le véhicule à un point choisi de la trajectoire, appelé point de but. Il calcule la commande de vitesse angulaire qui déplace le robot de sa position actuelle pour atteindre un point de vue (look-ahead point) en avant du robot.

b) Description de la commande

Le principe de la mise en œuvre de cet algorithme est illustré sur la figure (I.13). Sur cette figure, le point (x, y) qui est à une distance d'observation (distance look-ahead) l du robot, est l'un des points sur le chemin prédéfini. A partir de cette figure, on déduit les deux équations (I.13) et (I.14). La première provient de la géométrie du plus petit triangle rectangle de la figure (I.13).La seconde provient de la somme des segments sur l'axe X.

$$x^2 + y^2 = l^2 \quad \textbf{(I.13)}$$

$$x + d = r \quad \textbf{(I.14)}$$

Nous pouvons démontrer facilement les équations suivantes qui relient la courbure de l'arc à la distance look-ahead l.

$$d = r - x \quad \textbf{(I.15)}$$

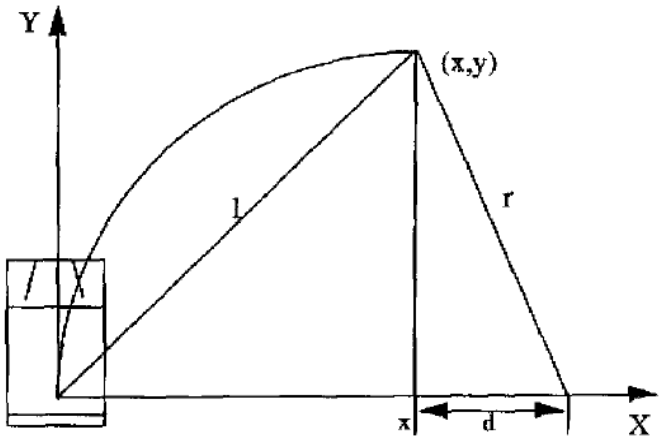
$$(r - x)^2 + y^2 = r^2 \quad \textbf{(I.16)}$$

$$r^2 - 2rx + x^2 + y^2 = r^2 \quad \textbf{(I.17)}$$

$$2rx = l^2 \quad \textbf{(I.18)}$$

$$r = \frac{l^2}{2x} \quad \textbf{(I.19)}$$

$$y = \frac{2x}{l^2} \quad \textbf{(I.20)}$$



FigureI. 13: Illustration du principe de l'algorithme Pure Pursuit.

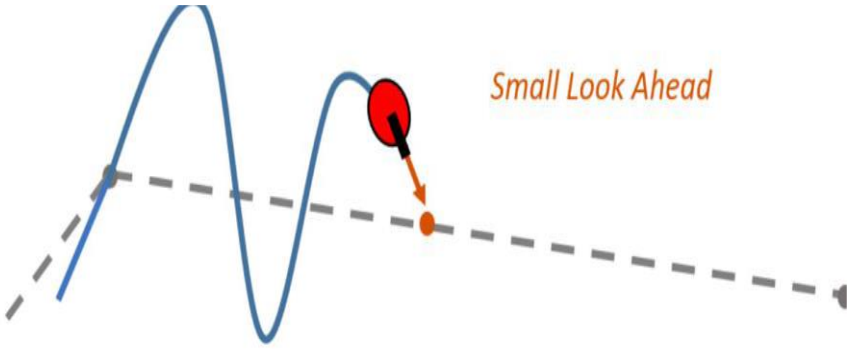
La valeur détermine le rayon de l'arc que le véhicule doit suivre. La courbure de ce rayon est évidemment sa valeur réciproque $\gamma = \frac{1}{r}$. Cette courbure peut être utilisée pour déterminer la constante de proportionnalité entre la vitesse linéaire et la vitesse angulaire du robot comme l'équation suivante

$$w = yv \quad \textbf{(I. 21)}$$

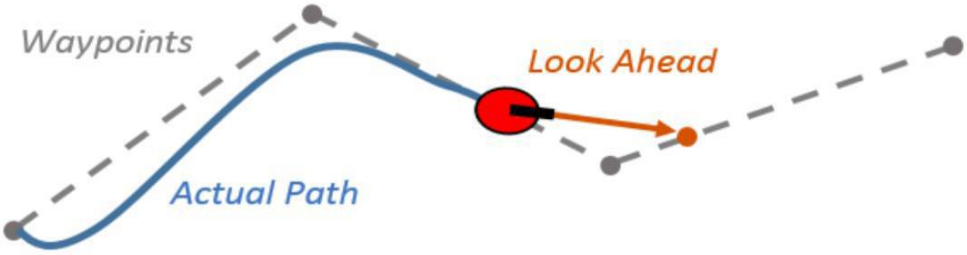
La vitesse linéaire est supposée constante, nous pouvons donc modifier la vitesse linéaire du robot à tout moment. L'algorithme déplace ensuite le point d'observation sur la trajectoire en fonction de la position actuelle du robot jusqu'au dernier point de la trajectoire.

La distance d'observation (Look-ahead distance) l est le paramètre de réglage principal du contrôleur. La figure ci-dessous montre le robot et le point d'observation pour différentes

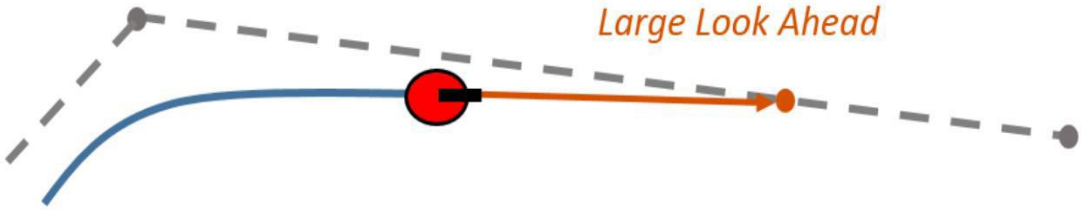
valeurs de la distance d'observation. Comme indiqué sur cette image (figure. I.14 (a)), nous notons que le chemin réel approxime mais ne correspond pas exactement à la ligne directe entre les points du chemin (waypoints) prédéfini.



(a)



(b)



(c)

FigureI. 14: Exemple de Suivi de chemin par l'algorithme Pure Pursuit pour différentes valeurs de l

La modification de ce paramètre peut changer la façon dont notre robot suit le chemin et il ya deux objectifs principaux : retrouver le chemin et maintenir le chemin. Afin de retrouver rapidement le chemin entre les waypoints, une petite distance d'observation fera avancer notre robot rapidement vers le chemin. Cependant, comme le montre (la figure I.14 (b)), le robot dépasse la trajectoire et oscille le long de la trajectoire souhaitée. Afin de réduire

les oscillations le long de la trajectoire, une plus grande distance d'anticipation peut être choisie, cependant, cela peut entraîner des courbures plus grandes près des coins (voir figure I.14 (c)).

Lors de la mise en œuvre de l'algorithme, les positions du robot et du point d'observation (look-ahead point) sont définies en fonction de coordonnées globales. Donc, tout d'abord, nous devons les transformer du référentiel global au référentiel lié au robot.

A travers la figure (I. 15) ,on remarque que lorsque la valeurs de la distance d'observation (Look-ahead distance) l est moyenne (voir la figure(I. 16.(b)) ,on obtient un suivi de trajectoire précise .

I.11. Conclusion

Dans ce chapitre nous avons pu définir quelques concepts de base liés à la robotique et aux robots mobiles, impliquant généralement les différents domaines de la modélisation, de la planification de trajectoire et de commande.

Un robot est un système programmable multifonctionnel qui imite les actions de créatures intelligentes. Au début, l'homme a combiné tous les moyens nécessaires pour réaliser une machine capable de remplacer l'homme dans diverses fonctions en plus de tâches complexes et irréalisable par l'homme. La robotique a parcouru un long chemin avec la progression de la technologie, mais il existe encore des problèmes à résoudre dans la robotique. Pour cela, l'objectif principal de notre travail est de contribuer à la planification de trajectoire et à la commande du robot mobile pour améliorer son autonomie et sa capacité de navigation.

Chapitre II

Planification de trajectoire

II.1. Introduction

Le domaine de la planification de mouvement consiste à trouver une trajectoire réalisable qui relie le point de départ au point d'arrivée tout en évitant la collision avec les obstacles. Le domaine de la planification de mouvement et de la navigation a acquis une popularité et une importance immenses ces dernières années en raison du fait que les tendances actuelles de la recherche en robotique pour les besoins industriels et domestiques se concentrent sur l'automatisation intelligente.

Depuis les années 1970, de nombreux algorithmes de planification de chemin, y compris des algorithmes géométriques, des algorithmes basés sur des grilles, des algorithmes de champ potentiel, des réseaux de neurones, des algorithmes génétiques et des algorithmes basés sur l'échantillonnage,…etc, ont été proposés pour divers environnements statiques et dynamiques. Chacun de ces algorithmes à ses propres avantages et inconvénients pour trouver la solution de planification de chemin la plus efficace en termes de complexité spatiale et temporelle et d'optimisation de chemin.

II.2. Problématique et Définitions

Etant donné une configuration initiale et une autre finale, l'objectif est de trouver une trajectoire qui mène le robot de façon autonome à la position souhaitée (finale) tout en évitant les obstacles?

II.2.1. Définitions [17]

- **La planification** sert à donner les moyens au robot de trouver une suite d'actions à appliquer sur un environnement connue, pour le faire passer de l'état initial à un état qui satisfait le but à atteindre.
- **Trajectoire ou plan** est un ensemble structuré d'actions qui mène au but. La détermination de cette trajectoire se fait tout en respectant un certain nombre de contraintes et de critères [18]:
  - ✓ ***Contraintes relatives au robot :*** Concernant sa géométrie, sa cinématique et sa dynamique et même l'architecture (type) du robot pouvant correspondre à un système articulé d'objets rigides un robot mobile, une main à plusieurs doigts, ou encore à plusieurs systèmes de robots à coordonner tels que des bras manipulateurs ou des robots mobiles.
  - ✓ ***Contraintes de l'environnement*** **:** Concernant essentiellement la non-collision aux obstacles, la prise en compte d'interactions de contact avec le robot.

D'autres critères peuvent être également considérés tels que la prise en compte de la distance de sécurité aux obstacles pour un robot manipulateur ou robot mobile, ou encore la "**qualité**'' et la "**stabilité**'' des prises pour une main articulée.

II.3. Les différents aspects de la planification

La planification présente plusieurs aspects importants [17] :

1. La résolution automatique du problème : La solution est l'assemblage d'un ensemble d'actions élémentaires élaboré lors de la planification et qui permet au robot d'atteindre l'objectif.

2. La prévision : Une trajectoire ou un plan est une prévision d'une séquence d'actes à accomplir. Lors de l'exécution, la surveillance du monde permet de vérifier que les actions sont bien exécutées et que le plan mène vers la solution.

3. La création d'une nouvelle connaissance : Une fois le plan (la trajectoire) est élaboré, il peut être conservé pour le réutiliser lorsqu'un problème similaire se posera (domaine d'apprentissage).

II.4. La planification de chemin (trajectoire)

La planification de trajectoire est utilisée dans la robotique mobile pour permettre au robot de se déplacer d'un point initial à un point final en suivant un chemin planifié. Comme nous l'avons évoqué auparavant, le planificateur prend en considération plusieurs types de contraintes, qu'elles soient liées au robot mobile ou à l'environnement de ce dernier. Plusieurs méthodes de planification ont été élaborées pour s'adapter aux différents environnements dans lesquels évolue le robot, comme l'environnement avec connaissance a priori, sans connaissance a priori ou encore l'environnement dynamique.

Dans un environnement avec connaissance a priori, on suppose que les obstacles sont connus et statiques, parmi les méthodes utilisées pour la planification on cite : le Roadmap, les potentiels, la décomposition cellulaire [19]. Le travail effectué grâce à l'une de ces trois méthodes se résume en trois tâches : la modélisation de l'environnement, la recherche de tous les chemins possibles que peut emprunter le robot mobile et enfin le choix du chemin optimal

Dans un environnement sans connaissance a priori, on ne connait que la position du robot mobile ainsi que la position de la cible, on ne connait pas la position des obstacles mais on suppose qu'ils sont statiques.

Les algorithmes de planification de trajectoire peuvent être divisés en deux grandes catégories : la planification globale de trajectoire et la planification locale de trajectoire.

II.4.1. Planification globale et planification locale

a) La planification globale de trajectoire

Le planificateur global est utilisé dans les environnements avec connaissance a priori. En général, c'est un algorithme itératif destıné à planıfier le chemin du robot mobile dans un environnement connu et statique et qui permet au robot d'atteindre la position finale à partir de sa position initiale. La première étape consiste à faire la modélisation de l'environnement, et la seconde étape consiste à planifier un chemin optimal qui évite tout type de collision avec les obstacles statiques. Comme exemples de planificateurs, on cite : Dijksta, A*, D*, Fil d'Ariane [20], décomposition cellulaire.

Les méthodes se basant sur une approche globale ont l'avantage de générer un chemin optimal tandis qu'elles ont l'inconvénient de consommer beaucoup de temps de calcul et d'espace mémoire.

b) La planification locale de trajectoire

Les méthodes locales se caractérisent par la connaissance locale de l'environnement. A chaque pas de déplacement l'algorithme détermine l'existence ou non de collision, s'il y a une collision, la trajectoire est modifiée localement. Ce type de méthodes permet de résoudre des problèmes dans un temps raisonnable. Leur principe consiste à déterminer les déplacements du robot en ne considérant qu'une représentation locale de l'environnement et à prévoir la planification comme un problème d'optimisation. Plusieurs applications sont possibles dans la planification locale comme par exemple :

- Des planificateurs pour l'évitement d'obstacle : On utilise généralement les méthodes probabilistes pour la planification. Les informations issues des capteurs extéroceptifs sont nécessaires pour générer une trajectoire qui va permettre au robot d'éviter une collision avec un obstacle statique ou dynamique. Ces planificateurs sont généralement utilisés dans les environnements sans connaissance a priori et les environnements dynamiques.
- Des planificateurs pour la génération de chemins réalisables par les robots mobiles [21]. Ces planificateurs prennent en considération certaines contraintes de non holonomie et cinématiques propres au robot mobile. Il existe plusieurs manières pour générer une trajectoire. La génération dépend des contraintes propres du robot mais aussi des contraintes qui lui sont imposées, comme par exemple, la trajectoire à réaliser par le robot

mobile peut obliger ce dernier à effectuer des manœuvres. Dans la littérature on peut trouver plusieurs méthodes comme par exemple : la déformation des chemins holonomes pour les rendre admissibles pour les robots non-holonomes. L'utilisation des chemins de Dubins [22] et des chemins de Reeds and Shepp [23]. La génération des chemins à courbure continue ou encore l'utilisation de la méthode de guidage.

L'avantage des méthodes utilisant cette approche réside dans leurs efficacités en termes de temps de calcul qui permet leurs utilisations dans des applications temps réel. En revanche elles présentent l'inconvénient de tomber dans des minima locaux et de générer une trajectoire souvent non optimale.

II.5. Algorithmes de planification

La planification de trajectoire est un sujet souvent traité dans beaucoup d'applications. Il existe donc de nombreux algorithmes pour effectuer une telle tâche. Dans cette section, nous présenterons quelques méthodes les plus utilisées pour planifier une trajectoire en expliquant brièvement leurs principes, ainsi que leurs avantages et leurs inconvénients.

II.5.1. Méthodes probabilistes

Les planificateurs probabilistes font partie d'une famille plus large des méthodes d'échantillonnage (″sampling-based methods″, en anglais). Très efficaces, notamment pour les problèmes définis dans des configurations de très grande dimension.

Une méthode de planification appelée « Roadmap » est basée sur l'utilisation de la théorie des graphes. Le principe est de construire un graphe reliant la configuration initiale à la configuration finale tout en évitant les obstacles. Ces techniques sont parmi les plus utilisées et développées ces dernières années car ce sont des planificateurs globaux.

Dans ce travail, nous détaillons deux principales approches probabilistes :

- ***PRM (Probabilistic Roadmap)***
- ***RRT (Rapidly-exploring Random Tree)***

II.5.1.1. Méthode RRT

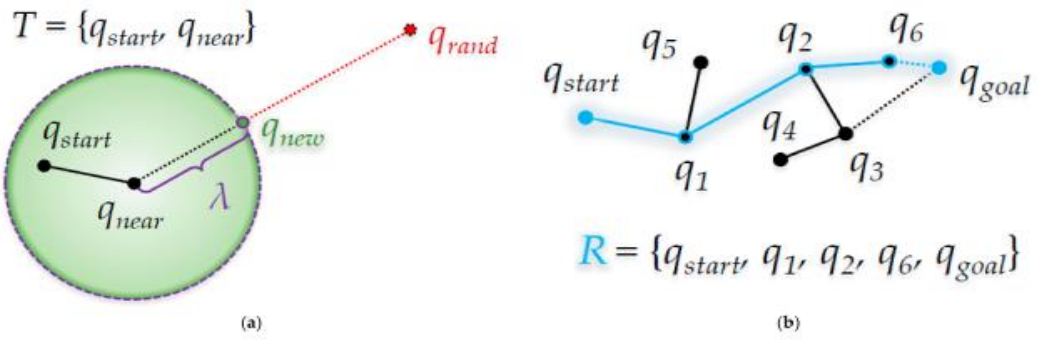
Initialement proposée par Lavalle [24], la RRT est l'une des méthodes les plus populaires ces dernières années. Elle consiste à construire un arbre, à partir de la position initiale du

robot, qui est le nœud racine de l'arbre. L'algorithme RRT explore progressivement l'espace de configuration pour trouver un chemin vers l'emplacement cible. Le principe de la méthode RRT est le suivant :

Le planificateur RRT développe l'arborescence de recherche enracinée à l'état de démarrage $\boldsymbol{q_{start}}$ suivant ces étapes :

1. Le planificateur échantillonne un état aléatoire $\boldsymbol{q_{rand}}$ dans l'espace d'état.
2. Le planificateur trouve un état $\boldsymbol{q_{near}}$ qui est déjà dans l'arbre de recherche et est le plus proche de $\boldsymbol{q_{rand}}$ dans l'arbre existant.
3. Essayer d'étendre l'arbre à partir de $\boldsymbol{q_{near}}$ dans la direction de $\boldsymbol{q_{rand}}$ d'une longueur $\boldsymbol{\lambda}$.
4. Le nouvel état $\boldsymbol{q_{new}}$ est ajouté à l'arbre de recherche.

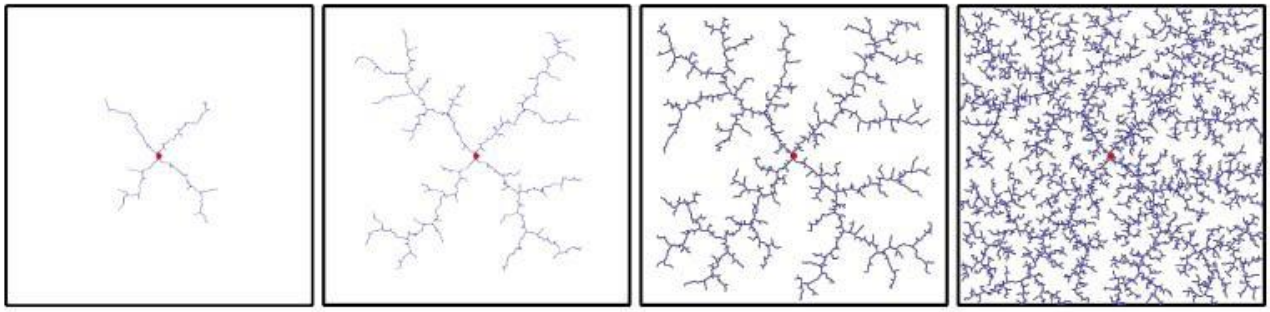
Ce processus est répété jusqu'à ce que l'arbre atteigne q_{goal}. Chaque fois qu'un nouveau nœud q_{new} est échantillonné, sa connexion avec d'autres nœuds est vérifiée pour qu'il n'y ait pas de collisions afin de réaliser un chemin pilotable de q_{start} à q_{goal}. Ce principe est illustré dans la figure (II.1 (a)). La Figure (II.1 (b)) montre le chemin résultant ″ R " parmi plusieurs chemins candidats vers la position de départ q_{start} et la position but q_{goal} .



FigureII. 1 : RRT algorithme : (a) Illustration du principe d'extension du RRT; (b) après la fin d'échantillonnage aléatoire**.**

La méthode d'exploration RRT se compose d'une phase de construction d'un arbre couvrant l'espace libre (voir la figure II.2) et d'une phase de requête. La performance de cette méthode vient du fait qu'elle ne nécessite pas de phase de pré-calcul. Une propriété intéressante inhérente de cette approche est que la croissance des arbres est fortement biaisée en faveur des zones inexplorées de l'espace de configuration, l'exploration se fait rapidement. Malgré ces avantages, la RRT comme toute approche peut avoir des lacunes; elle ne prend pas

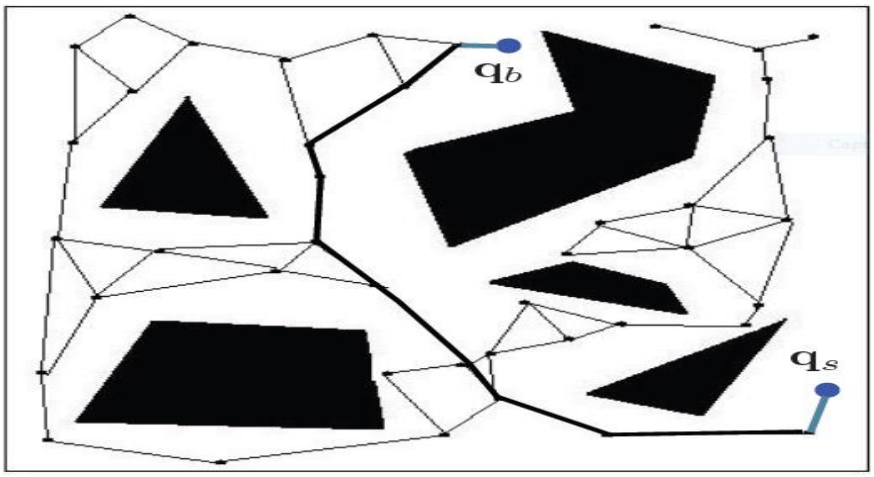
en compte le coût de la solution lors de la recherche. Ainsi, les chemins qu'elle produise sont parfois loins d'être optimaux.



FigureII. 2: Évolution d'un arbre couvrant l'espace libre par la méthode RRT.

II.5.1.2. Méthode PRM :

La planification de trajectoire par échantillonnage aléatoire a été présentée pour la première fois par Kavraki [24] sous le nom de Probabilistic Roadmap (PRM). Le principe de cette méthode est d'échantillonner l'espace des configurations C pour trouver suffisamment de configurations sans collisions pour permettre de représenter l'espace des configurations libres. L'algorithme utilise déterminer si les configurations échantillonnées sont en collision ou non grâce à un détecteur de collision. Celles qui sont dans l'espace libre sont gardées et l'algorithme tente de les relier par une arête au graphe contenant les autres nœuds. Le lien se fait à l'aide d'une méthode locale permettant de relier un nœud à un autre le long d'un espace libre. Le graphe comporte initialement deux nœuds, la configuration initiale et la configuration finale. . Lorsque l'on souhaite trouver un chemin entre deux nœuds du graphe, celui-ci est parcouru à l'aide d'un algorithme de type A*, voir figure (II.3) où est indiqué la configuration de départ $\boldsymbol{q_s}$ et la configuration d'arrivée $\boldsymbol{q_b}$.



FigureII. 3: Planification par PRM.

- **Les avantages de la méthode PRM**
  - Elle trouve un chemin sans collisions très rapidement.
  - Dans des espaces à très haute dimensionnalité, la méthode PRM est très efficace pour trouver rapidement une solution.
  - Simplicité de mise en œuvre.
- **Inconvénients de la méthode PRM**

  Son désavantage est celui de toutes les méthodes de planification de mouvement probabilistes, il faut que le graphe soit suffisamment représentatif de l'espace libre C_{libre} avant qu'il ne soit utilisable pour des requêtes de planification de mouvement. Cette étape est donc généralement effectuée hors ligne car la génération d'une Roadmap est coûteuse en temps de calcul; le graphe de l'environnement obtenu est conservé.

II.5.2. Algorithme A*

a) Présentation

L'algorithme de recherche A* (qui se prononce A étoile, ou A star à l'anglaise) est un algorithme de recherche globale permettant de trouver le chemin dans un graphe entre un nœud initial et un nœud final tous deux donnés. De par sa simplicité il est souvent présenté comme exemple typique d'algorithme de planification dans le domaine d'intelligence artificielle. L'algorithme A* a été créé pour que la première solution trouvée soit l'une des meilleurs, c'est pourquoi il est célèbre dans des applications comme les vidéos privilégiant la vitesse de calcul sur l'exactitude des résultats. Cet algorithme a été proposé pour la première fois par Peter E. Hart, Nils Nilsson et Bertram Raphael en 1968 [25].

L'algorithme A* est un algorithme qui est très similaire à l'algorithme de Dijkstra de 1959 [26]. Les deux fonctionnent exactement de la même façon, mais le coût des points est calculé de façon différente. Contrairement à l'algorithme de Dijkstra, le A* va évaluer le coût des points comme étant le coût parcouru pour s'y rendre plus une prédiction du coût minimal qu'il reste à faire pour arriver à la solution. Cette prédiction est surnommée l'Heuristique. Dans la plupart des cas, puisque l'on utilise souvent le déplacement comme étant le coût parcouru, c'est souvent la distance euclidienne (à vol d'oiseau) entre un point et le point d'arrivée qui est utilisée comme heuristique de ce point. Le coût total d'un point sera donc le déplacement parcouru plus la distance euclidienne [27]. Cette méthode tente donc de prédire quel chemin emmènera plus rapidement à la solution. Pour le reste, celui-ci fonctionne exactement comme l'algorithme de Dijkstra, toutefois, cette différence de coût réduit énormément le nombre de points examinés.

b) Les listes de A*

A* utilise deux listes, ces listes contiennent des nœuds d'un graphe :

- ✓ La première liste, appelée ***liste ouverte***, va contenir tous les nœuds étudiés. Dès que l'algorithme va se pencher sur un nœud du graphe, il passera dans la liste ouverte (sauf s'il y est déjà).
- ✓ La seconde liste, appelée ***liste fermée***, contiendra tous les nœuds qui, à un moment où à un autre, ont été considérés comme faisant partie du chemin solution. Avant de passer dans la liste fermée, un nœud doit d'abord passer dans la liste ouverte, en effet, il doit d'abord être étudié avant d'être jugé comme bon.

c) Déroulement de l'algorithme

- ✓ Pour déterminer si un nœud est susceptible de faire partie du chemin solution, il faut pouvoir quantifier sa qualité. Pour cela, on calcule la distance entre le point étudié et le dernier point qu'on a jugé comme bon. Et on calcule aussi la distance entre le point étudié et le point de destination. La somme de ces deux distances nous donne la qualité du nœud étudié. Plus un nœud a une qualité faible, meilleur il est.
- ✓ Le point qui pourra nous rapprocher de la solution est un point voisin du point que nous étudions. On va donc étudier chacun des nœuds voisins du nœud courant pour déterminer celui qui a le plus de chances de faire partie du chemin solution.
- ✓ La recherche du chemin commence par le premier de départ, en étudiant tous ses voisins, en calculant leur qualité, et en choisissant le meilleur pour continuer. Chaque point étudié est mis dans la liste ouverte et le meilleur de cette liste passe dans la liste fermée, il va servir de base pour la recherche suivante.
- ✓ Ainsi, à chaque itération on va regarder parmi tous les nœuds qui ont été étudiés (et qui n'ont pas encore été choisis) celui qui a la meilleure qualité. Et il est tout à fait possible que le meilleur ne soit pas un voisin direct du point courant. Cela signifiera que le point courant nous éloigne de la solution et qu'il faut corriger le tir. L'algorithme s'arrête quand la destination a été atteinte ou bien lorsque toutes les solutions mises de côté ont été étudiées et qu'aucune ne s'est révélée bonne, c'est le cas où il n'y a pas de solution.

Nous commençons la recherche de chemin selon les étapes suivantes :

1. On commence par le nœud de départ, c'est le nœud courant.

2. On regarde tous les nœuds voisins du nœud courant.
3. Pour chaque nœud voisin :
   - ✓ S'il fait partie d'un obstacle, on l'oublie.
   - ✓ S'il est déjà dans la liste fermé, on l'oublie.
   - ✓ S'il est déjà dans la liste ouverte, on met à jour la liste ouverte si le nœud a une moins bonne qualité (et on n'oublie pas de mettre à jour son parent).
   - ✓ Sinon, on ajoute le nœud dans la liste ouverte avec comme parent le nœud courant.
4. On cherche le meilleur nœud de toute la liste ouverte (Si la liste ouverte est vide, il n'y a Pas de solution, fin de l'algorithme).
5. On le met dans la liste fermée et on le retire de la liste ouverte.
6. On réitère avec ce nœud comme nœud courant jusqu'à ce que le nœud courant soit le nœud de destinations

La liste ouverte contient toutes les pistes de trajet qui sont encore possibles. La liste fermée contient toutes les solutions étudiées, bonnes et mauvaises.

Le "bon chemin" est déterminé en remontant le long des parents des nœuds.

d) Qualité d'un nœud

La qualité d'un nœud est calculée par l'équation suivante :

$$F = G + H \qquad \textbf{(II.1)}$$

Où :

- ✓ **G** représente le coût de déplacement pour aller du point de départ au nœud considéré, en suivant le chemin généré pour y arriver.
- ✓ **H** représente le coût de déplacement pour aller du nœud considéré à la destination finale.

Prenons l'exemple de la figure (II.4) où nous allons chercher le plus court chemin entre le nœud A1 de départ et le nœud E4 d'arrivée.

	1	2	3	4	5
A	Départ ∞	∞	∞	∞	∞
B	∞	∞	∞	∞	∞
C	∞	∞	∞	∞	∞
D	∞	∞	∞	∞	∞
E	∞	∞	∞	Arrivé ∞	∞

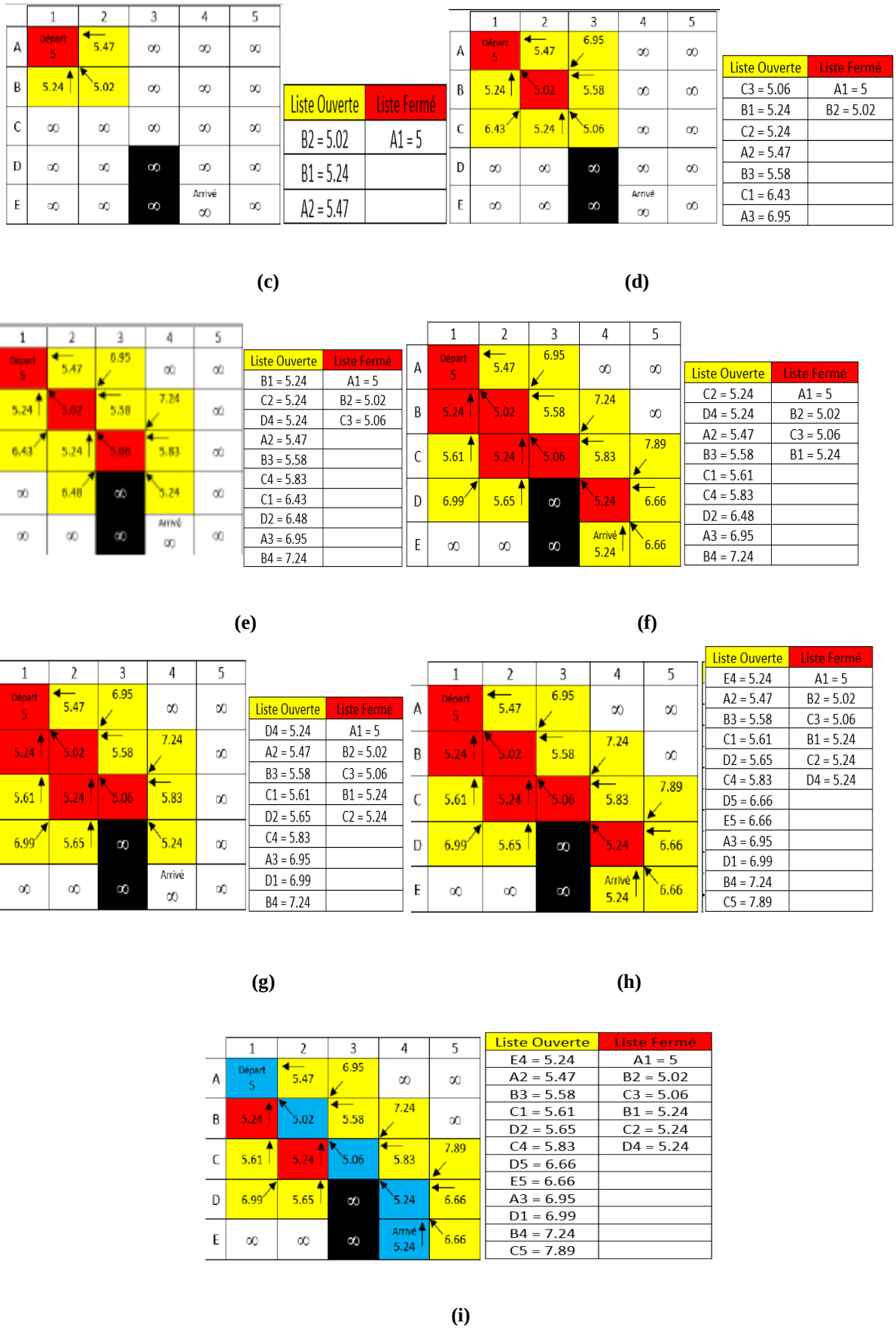
Liste Ouverte	Liste Fermé

(a)

	1	2	3	4	5
A	Départ 5	∞	∞	∞	∞
B	∞	∞	∞	∞	∞
C	∞	∞	∞	∞	∞
D	∞	∞	∞	∞	∞
E	∞	∞	∞	Arrivé ∞	∞

Liste Ouverte	Liste Fermé
A1 = 5	

(b)



FigureII. 4: Exemple illustrant l'algorithme A*

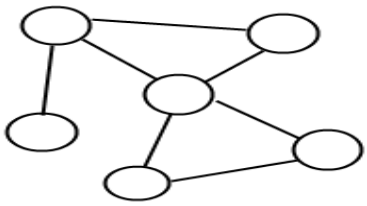
d) Les avantages et les inconvénients

- ***Avantage :*** A* est facilement réalisé et calculé en temps réduit.
- ***Inconvénient :*** A* dépend fortement de la fonction heuristique, ce qui influence les performances de l'algorithme.

II.5.3. Lifelong Planning A*

L'algorithme Lifelong planning A*[28**]**, connu sous de le nom LPA*est un algorithme de recherche heuristique incrémental basé sur A*. Il a été décrit pour la première fois par Sven Koenig et Maxim Likhachev en 2001. Il peut s'adapter aux changements du graphe sans recalculer l'ensemble du graphe, en mettant à jour les valeurs g (distance depuis le début) de la recherche précédente pendant la période de la recherche actuelle pour les corriger si nécessaire. Il réutilise les informations des exécutions précédentes pour réduire considérablement le nombre des nœuds qui doivent être examinés, par rapport à A*, en mettant à jour uniquement les valeurs g pertinentes pour le calcul du chemin le plus court.

En fait, le problème de navigation d'un robot peut être modélisé comme un problème de parcours de graphe sur une grille à huit connexions avec des arêtes qui sont soit traversables (avec un coût un) soit infranchissables comme le montre la figure(II.5).



FigureII. 5: Graphe représentant le problème de navigation d'un robot constitué de nœuds et d'arêtes

- **Lifelong planning A* : les variables**
  - $\boldsymbol{g(s)}$: la valeur g précédemment calculée (distance pour aller du point de départ au nœud s considéré communément appelée distance de départ) comme dans A*.
  - $\boldsymbol{rhs(s)}$: Cette valeur est connue en anglais par the Right Hand Side (rhs) value. La valeur rhs d'un nœud s est basée sur les valeurs g de ses prédécesseurs plus le coût de déplacement vers ce nœud. Elle vérifie toujours la relation suivante :

$$rhs = \begin{cases} 0 & si \quad s = s_{start} \\ min_{\acute{s} \in P_{red(s)}}(g(\acute{s}) + c(\acute{s}, s)) & sinon \end{cases} \qquad \textbf{(II.2)}$$

Où:

- ✓ s' est un prédécesseur de s

- ✓ $c(s', s)$ est le coût de l'arête reliant s' et s
- ✓ $Pred(s)$ est un nœud prédécesseur de s.

Un nœud est dit localement cohérent (ou consistant) si $g(s) = rhs(s)$ sinon, il est appelé localement incohérent.

- $\boldsymbol{h(s)}$ est l'heuristique estimée pour aller du nœud considéré à la destination finale.
- $\boldsymbol{Key(s)}$: L'algorithme LPA* maintient une file d'attente prioritaire qui contient toujours les nœuds localement inconsistants. Ce sont les nœuds dont les valeurs de g doivent potentiellement changer pour les rendre localement consistants. La clé $Key(s)$ du nœud s dans la file prioritaire est un vecteur à deux composantes :

$$Key(s) = [min\,(g(s), rhs(s)) + h(s);\ min\,(g(s), rhs(s)] \qquad \textbf{(II.3)}$$

Ce vecteur est utilisé pour trier la liste ouverte qui contient tous les nœuds inconsistants. Les clés $key(s)$ de tous les nœuds s dans la liste ouverte sont comparées selon un ordre lexicographique, c.à.d. si on a deux vecteurs a deux dimensions u et v, on dit que : u < v si (u.first < v.first OR u.first == v.first AND u.second < v.second).

- $\boldsymbol{Succ}(\,)$ désigne l'ensemble des successeurs du nœud considéré dans le graphe.

L'algorithme LPA* est montré sur la figure II.6 où S est l'ensemble fini de tous les nœuds du graphe. U représente la file d'attente prioritaire du graphe et les fonctions suivantes qui permettent de la gérer :

- ✓ $\boldsymbol{U.Top}(\,)$ renvoie le nœud ayant la plus petite priorité de tous les nœuds dans la file d'attente prioritaire U.
- ✓ $\boldsymbol{U.TopKey}(\,)$ renvoie la plus petite priorité de tous les nœuds dans la file d'attente prioritaire U (si U est vide alors $U.TopKey(\,)$ renvoie $[\infty; \infty]$.
- ✓ $\boldsymbol{U.Pop}(\,)$ supprime le nœud avec la plus petite priorité dans la file d'attente prioritaire U et renvoie le nœud.
- ✓ $\boldsymbol{U.Insert(s, k)}$ insère le nœud s dans la file d'attente U avec la priorité k.
- ✓ $\boldsymbol{U.Apdate(s, k)}$ change la priorité du nœud s dans la file d'attente U à la priorité k.
- ✓ $\boldsymbol{U.Remove(s)}$ supprime le nœud s de la file d'attente U.

- **Lifelong planning A* : Algorithme**

```
procedure CalcKey(s)
{01} return [min(g(s), rhs(s)) + h(s, s_goal); min(g(s), rhs(s))];
procedure Initialize()
{02} U = ∅;
{03} for all s ∈ S rhs(s) = g(s) = ∞;
{04} rhs(s_start) = 0;
{05} U.Insert(s_start, CalcKey(s_start));
procedure UpdateVertex(u)
{06} if (u ≠ s_start) rhs(u) = min_{s'∈Pred(u)}(g(s') + c(s', u));
{07} if (u ∈ U) U.Remove(u);
{08} if (g(u) ≠ rhs(u)) U.Insert(u, CalcKey(u));
procedure ComputeShortestPath()
{09} while (U.TopKey() < CalcKey(s_goal) OR rhs(s_goal) ≠ g(s_goal))
{10}    u = U.Pop();
{11}    if (g(u) > rhs(u))
{12}       g(u) = rhs(u);
{13}       for all s ∈ Succ(u) UpdateVertex(s);
{14}    else
{15}       g(u) = ∞;
{16}       for all s ∈ Succ(u) ∪ {u} UpdateVertex(s);
procedure Main()
{17} Initialize();
{18} forever
{19}    ComputeShortestPath();
{20}    Wait for changes in edge costs;
{21}    for all directed edges (u, v) with changed edge costs
{22}       Update the edge cost c(u, v);
{23}       UpdateVertex(v);
```

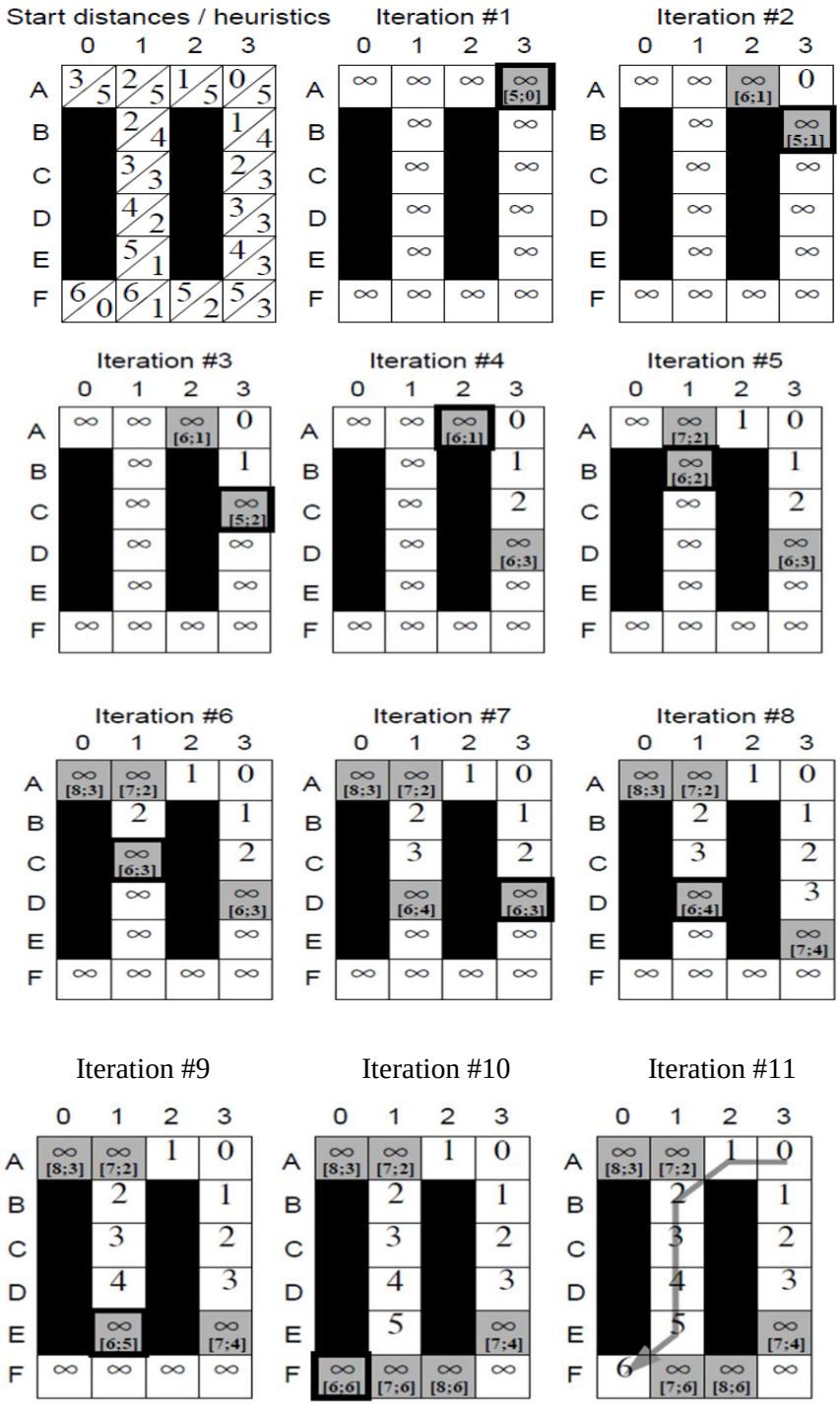
FigureII. 6: Algorithme LPA*

- **Lifelong planning A* : Exemple**

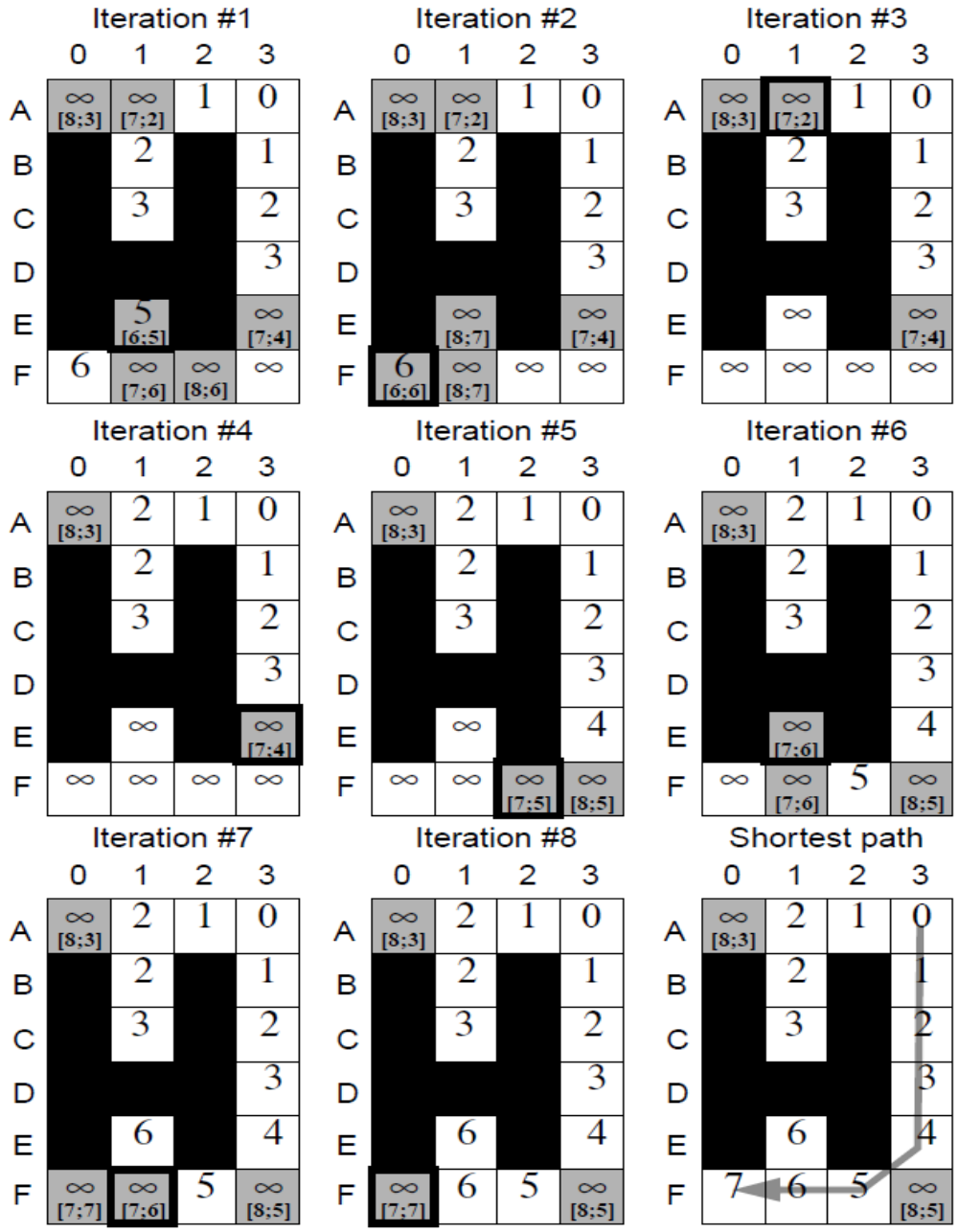
Dans cet exemple simple, on considère une grille (à 8 connexions) de taille 6x4, la grille originale est montrée sur la figure (II.7). La cellule A3 est la position de départ du robot et la cellule F0 représente la position cible. Les cellules en noir représentent des obstacles. Le robot doit planifier sa trajectoire la plus courte pour atteindre la cible tout en évitant les obstacles. Pour cela, on fait appel à l'algorithme de planification Lifelong A* ci-dessus (Figure II.6). La figure (II.7) montre les différentes itérations du premier appel de la fonction $ComputeShortestPath()$ de l'algorithme. A chaque itération, sont affichés les coûts g de chaque cellule de la grille et la clé $Key(s)$ de chaque nœud s de la file d'attente prioritaire. A la dernière itération (itération 11 dans cet exemple), l'algorithme a déterminé la trajectoire optimale entre le point de départ et le point destination sachant que la première recherche de Lifelong A* est la même que celle de A* mais toutes les recherches suivantes seront beaucoup plus rapides.

Maintenant l'algorithme doit avoir la capacité de mettre à jour le chemin le plus court à chaque fois que certains coûts changent suite à un changement dans l'environnement.

Dans la figure (II.8), on suppose que dans l'exemple précédent, une cellule traversable devient bloquée (à savoir la cellule D1). Un point fort de l'algorithme LPA* est qu'il tire parti des informations des recherches précédentes et par conséquent la première itération de la nouvelle recherche du chemin optimal utilise les informations de la recherche précédente. On obtient à la fin une autre trajectoire optimale qui évite le nouvel obstacle de la cellule D1 de façon plus rapide (9 itérations seulement).



FigureII. 7: Exemple simple de LPA* (1ère recherche : 1er Appel de la fonction ComputeShortestPath( ))



FigureII. 8): Exemple simple de LPA* (2ème recherche : 2ème Appel de la fonction ComputeShortestPath( ))

- **Inconvénients de l'algorithme Lifelong A***
  - ✓ L'algorithme ne recalcule qu'à partir d'un seul point de départ et un seul point cible.
  - ✓ La question qui se pose : Que se passe-t-il si nous avons déjà avancé lorsque le graphe change ?

II.5.4. Algorithme D* Lite

L'algorithme D* Lite [28] est un algorithme de recherche heuristique incrémental basé sur LPA* en échangeant les sommets de départ et d'arrivée et en inversant algorithmiquement toutes les arêtes et en prenant en compte le chemin déjà parcouru. L'algorithme trouve le chemin le plus court entre le nœud cible et le nœud de départ dans un environnement dynamique inconnu en minimisant les valeurs $\boldsymbol{rhs}$ (right hand side) calculées à l'aide de l'équation (II.2). Les valeurs clé (Key) des sommets sont calculées et mise à jour à l'aide de l'équation (II.3) lorsqu'une connexion change non seulement avec les nouvelles données de connexion, mais avec le nouveau trajet parcouru par le robot.

Donc on peut dire que l'algorithme D* Lite détermine à plusieurs reprises les chemins les plus courts entre la position actuelle du robot et la position cible lorsque les coûts d'arête d'un graphe changent pendant que le robot se déplace vers le nœud cible.

II.6. Conclusion

Nous avons abordé dans ce chapitre la notion de la planification de trajectoire afin de générer un chemin optimal et sans collision avec les obstacles pour les robots en général, et nous avons présenté les principes de quelques méthodes et algorithmes de planification de trajectoire et d'optimisation ainsi que leurs avantages et inconvénients.

Chapitre III

Résultats et interprétations

III.1. Introduction

Dans ce chapitre nous allons présenter les résultats de simulation sous Matlab et Gazebo des méthodes de planification de trajectoires étudiées et des deux lois de commande : PurePursuit et la commande à base de fonction de Lyapunov appliquées au robot pour suivre les trajectoires désirées.

Avant de présenter et discuter les résultats obtenus, nous commençons tout d'abord par décrire les outils utilisés dans ce travail à savoir : le Système ROS, l'interface Matlab-ROS, le simulateur Gazebo et le robot Turtlebot.

III.2. Outils utilisés

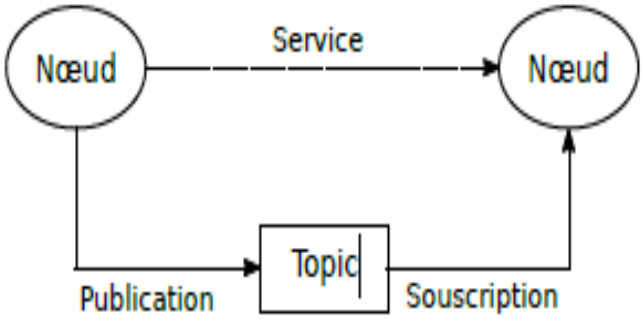
III .2.1. Description du système ROS

- **Présentation**

ROS (Robot Operating System) est un ensemble d'outils informatiques open source pour le développement d'applications robotiques. Son fonctionnement est distribué, composé de différents modules codés en C++ ou en Python. Il s'agit d'un méta-système d'exploitation qui peut fonctionner sur un ou plusieurs ordinateurs et qui fournit plusieurs fonctionnalités : abstraction du matériel, contrôle des périphériques de bas niveau, mise en œuvre de fonctionnalités couramment utilisées, transmission de messages entre les processus et gestions des packages installées [29].

FigureIII. 1: Logo de ROS [30].

Le fonctionnement de ROS repose sur un système de nœuds communiquant entre eux via des topics ou des services, s'échangeant des messages standardisés. Un nœud est un processus qui accompli une tâche spécifique. Il peut communiquer avec les autres nœuds en publiant des messages sur un topic. Tous les nœuds peuvent souscrire aux topics et accéder aux messages qui ont été publiés. Un nœud peut envoyer une requête directement à un autre nœud en utilisant un service. Le schéma (III.2) illustre le fonctionnement de ROS.

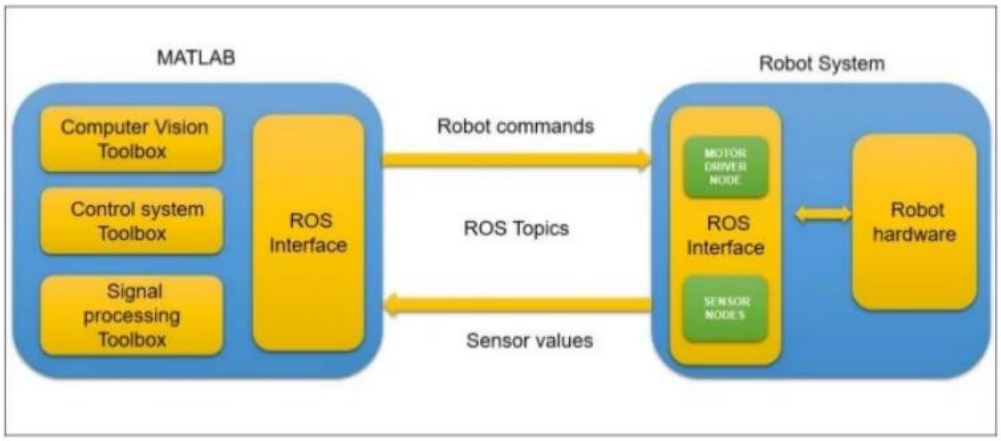


FigureIII. 2 : Le concept de fonctionnement de ROS.

III.2.2. Description de l'interface MATLAB-ROS

L'interface MATLAB-ROS **[31]** est une interface utile pour les chercheurs et les étudiants pour prototyper leurs algorithmes robotiques dans MATLAB et les tester sur des robots compatibles ROS. L'outil robotique toolbox fournit une interface entre MATLAB et ROS. Nous pouvons ainsi tester nos algorithmes sur des robots compatibles ROS ou des simulateurs de robots tels que Gazebo. Dans MATLAB, nous pouvons puplier ou souscrire à un sujet, par exemple un nœud ROS et nous pouvons en faire un maître ROS.

L'interface MATLAB-ROS possède la plupart des fonctionnalités ROS dont nous avons besoin. Voici un schéma fonctionnel montrant comment MATLAB communique avec un robot fonctionnant sous ROS [32].



FigureIII. 3: Schéma fonctionnel montrant comment MATLAB communique avec un robot fonctionnant sous ROS

III.2.3. Simulateur robot GAZEBO

- **Description de Gazebo**

Gazebo **[33]** est un simulateur multi-robots open source créé en 2004 par "*Andrew Howard* " et son étudiant "*Nate Koenig*" au laboratoire *USC Robotics Research Laboratory*. Il supporte des simulations pour l'odométrie, les télémètres laser et les capteurs de caméra. Il est capable de simuler un grand nombre de robots dans des environnements intérieurs ou extérieurs, complexes, réalistes et en 3D. Il peut simuler une dynamique corporelle très ridige du corps, ce qui nécessite une bonne carte graphique et un bon processeur CPU.

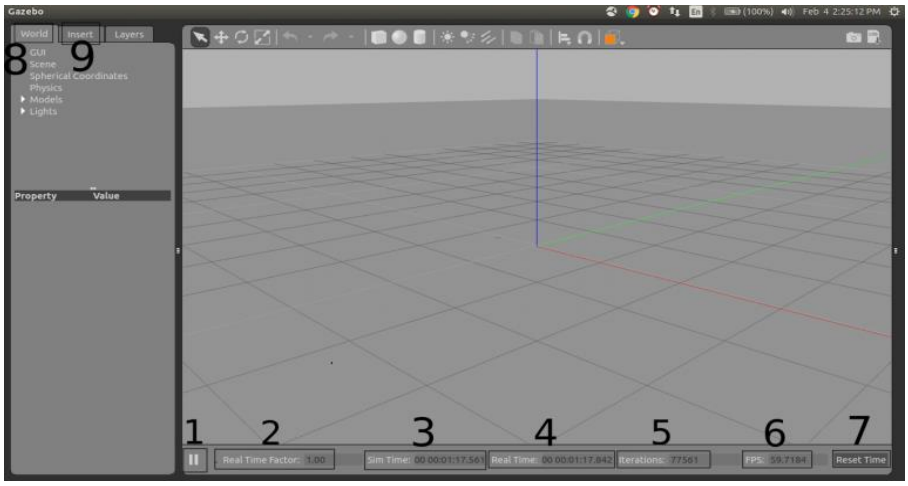
A l'origine, Gazebo a été conçu pour évaluer des algorithmes pour de nombreuses applications en robotique.

Gazebo est séparé en deux parties, un serveur et un client. Le serveur gère le moteur physique et la génération des données des capteurs, tandis que le client est responsable de la visualisation et de l'interface graphique.

FigureIII. 4: logo de Gazebo [29]

- **L'environnement de Gazebo**

Gazebo est un simulateur ***3D*** Soutenu *Par Open Source Robotics Foundation* (OSRF), créé à l'origine pour fonctionner comme over *Player*, il a été adapté pour également travailler sur *ROS.* La figure (III.5) représente l'interface de GAZEBO, dont nous allons expliquer les éléments.



FigureIII. 5: l'interface de GAZEBO

1- Arrêt ou redémarrer l'émulateur après l'avoir arrêté.

2- *Real Time Factor* il montre l'efficacité de l'émulateur. Si la valeur est de 0.25, cela signifie que l'émulateur a besoin de 4 secondes pour simuler 1 seconde dans la vie réelle.

3- Temps de simulation fait référence à la vitesse à laquelle le temps passe dans le simulateur lorsqu'une simulation est en cours d'exécution.

4- Temps réel fait référence au temps réel qui s'écoule dans la vie réelle pendant que le simulateur fonctionne.

5- L'état de l'environnement dans Gazebo est calculé une fois par itération. Nous pouvons voir le nombre d'itérations sur le côté droit de la barre d'outils inférieure.

6- Le nombre d'image par seconde (Frame Per Second).

7- Réinitialisez *Sim Time*, *Real Time*, *Itérations* à Zéro.

8- Il contient les propriétés du monde que nous simulons.

9- L'onglet Insertion est l'endroit où nous ajoutons de nouveaux objets (modèles) à la simulation.

III.2.4. Robot Turtlebot

- **Définition**

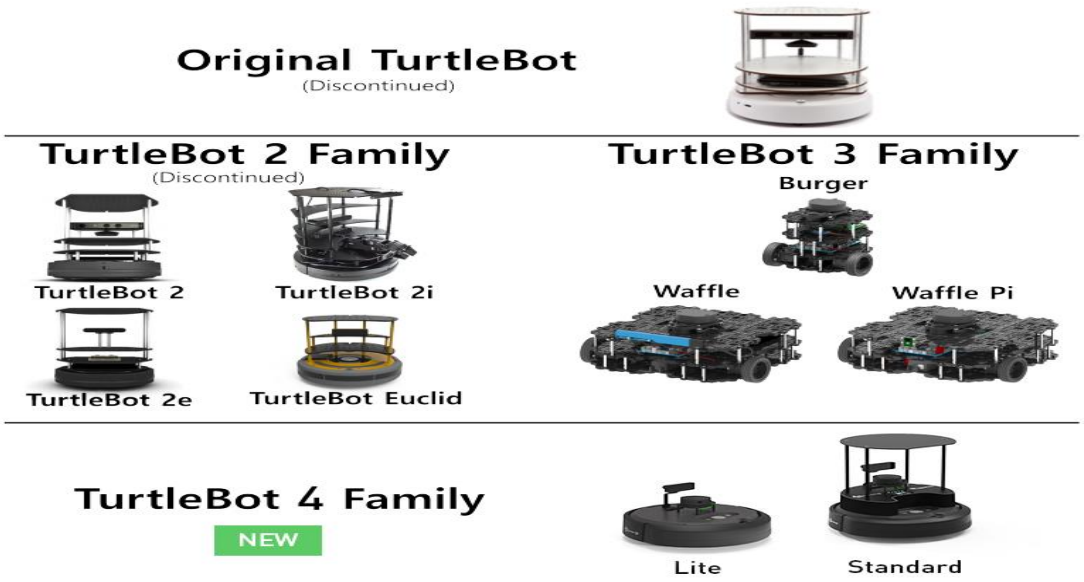
Turtlebot est un kit de robotique personnel à faible coût avec un logiciel open source. Turtlebot a été créé dans le laboratoire *Willow Garage* par *Melonee Wise* et *Tully Foote* en Novembre 2010. Avec Turtlebot, nous pouvons construire un robot qui peut conduire dans la maison, voir en 3D et avoir suffisamment de puissance pour créer des applications passionnantes [34].

FigureIII. 6: Turtlebot

- **Caractéristiques**
  - Les fonctionnalités de base du Turtlebot sont la cartographie et la localisation simultanées (SLAM) ainsi que la navigation.
  - Turtlebot dispose d'une importante communauté d'utilisation dans le monde.
  - Turtlebot est une marque sous licence qui est gérée par Open Source Robotics Foundation.
- **Familles de Turtlebot**

Il y a maintenant 4 générations de robots Turtlebot :

- Turtlebot 1 a été développé en 2010 par *Willow Garage* pour le déploiement de ROS, puis mise en vente en 2011.
- En 2012, Turtlebot 2 a été développée par *yujin robot* sur la base du robot de recherche iClebo kobuki.
- En 2017, Turtlebot 3 a été développée avec de nouvelles fonctionnalités pour compléter les fonctions manquantes de ses prédécesseurs et répondre aux demandes des utilisateurs. Le Turtlebot 3 adopte l'actionneur intelligent ROBOTIS Dynamixel pour le déplacement du robot [35].
- Turtlebot 4 est la nouvelle génération de la plate-forme robotique open source la plus populaire au monde pour l'enseignement et la recherche, offrant une meilleure puissance de calcul, de meilleurs capteurs et une expérience utilisateur de classe mondiale à un prix abordable.



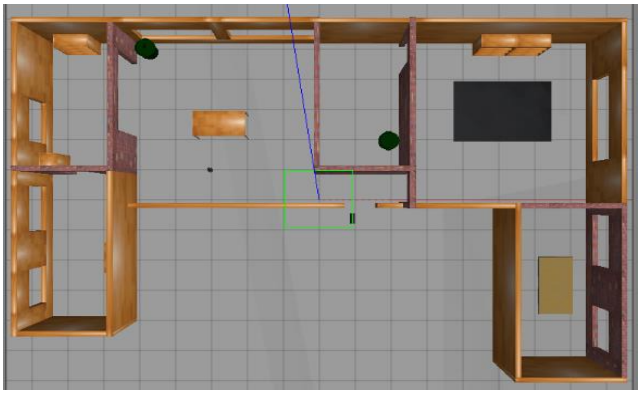
FigureIII. 7: Familles de Turtlebot

III.3. Résultats et Discussion

Dans cette section, nous allons tout d'abord présenté et discuté les résultats des différentes méthodes de planification et de commande obtenus en utilisant Matlab. Ensuite, nous allons passer à la simulation en utilisant l'interface Matlab-ROS, Gazebo et le robot Turtlebot simulé. Il est à noter qu'on peut commander le robot Turtlebot réel juste en changeant quelques instructions dans nos codes.

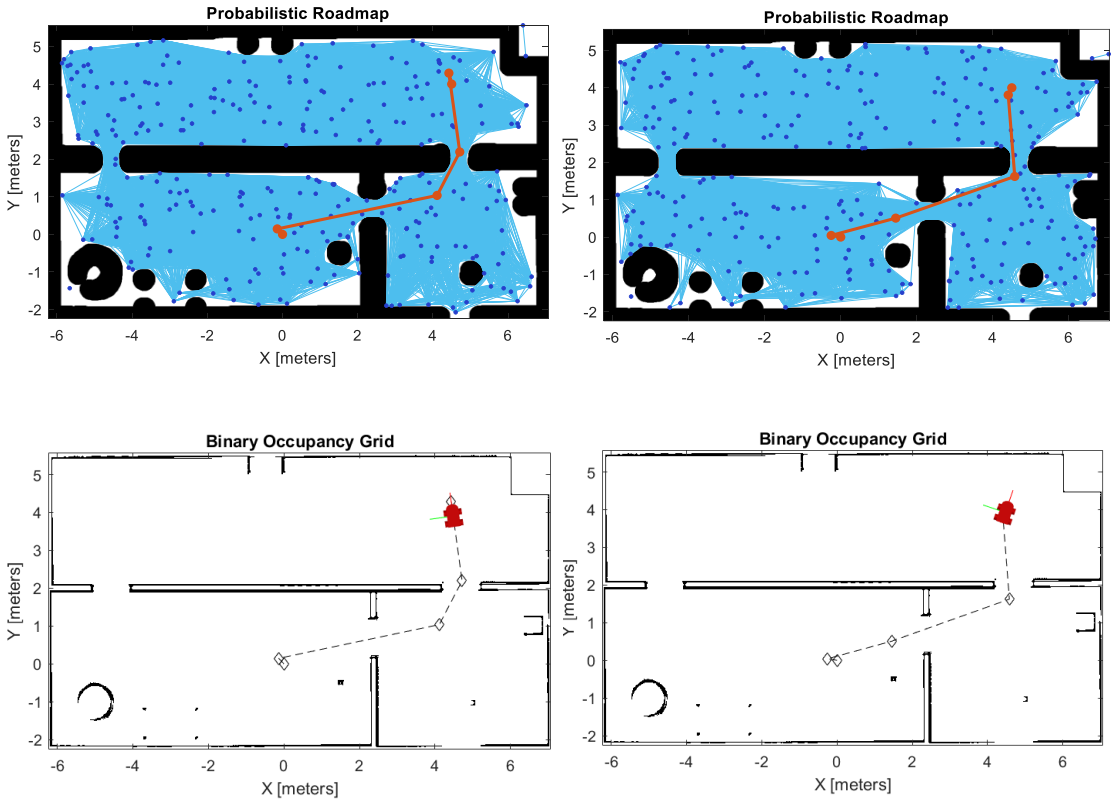
III.3.1. Simulation des algorithmes de planification et de la commande PurePursuit

Pour planifier la trajectoire d'un robot mobile, nous avons utilsé plusieurs environnement parmi lesquels l'environnement 'office'. La Figure (III.8) représente cet environnement dans GAZEBO

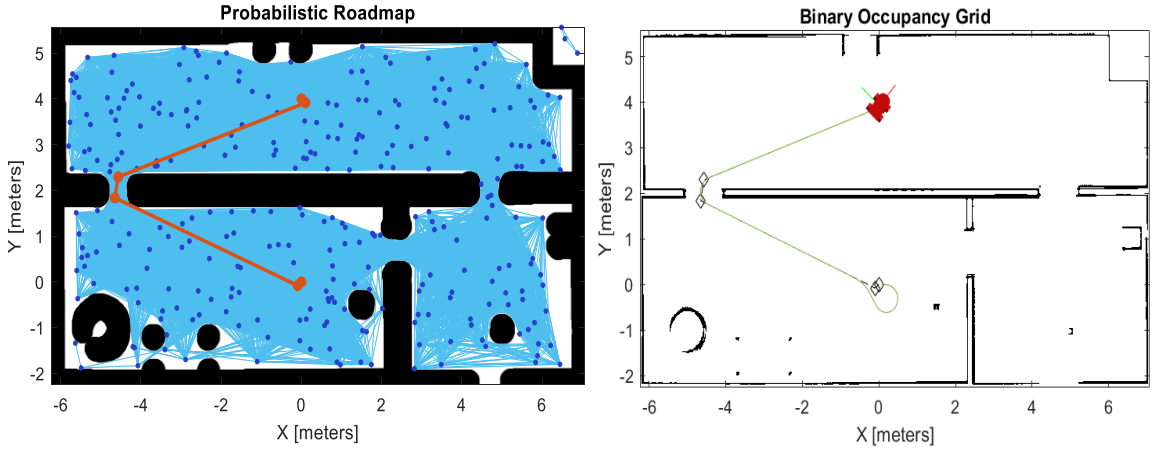


FigureIII. 8:L'environnement office

La Figure (III.9) et la figure (III.10) donnent les résultats de planification de trajectoire en utilisant l'algorithme PRM et la commande du robot par la méthode PurePursuit.



FigureIII. 9:Planification par PRM et commande par PurePursuit (1ère Exécution) à droite 1ère exécution, à gauche 2ème exécution



FigureIII. 10: Planification par PRM et commande par PurePursuit (Positions initiale et destination différentes)

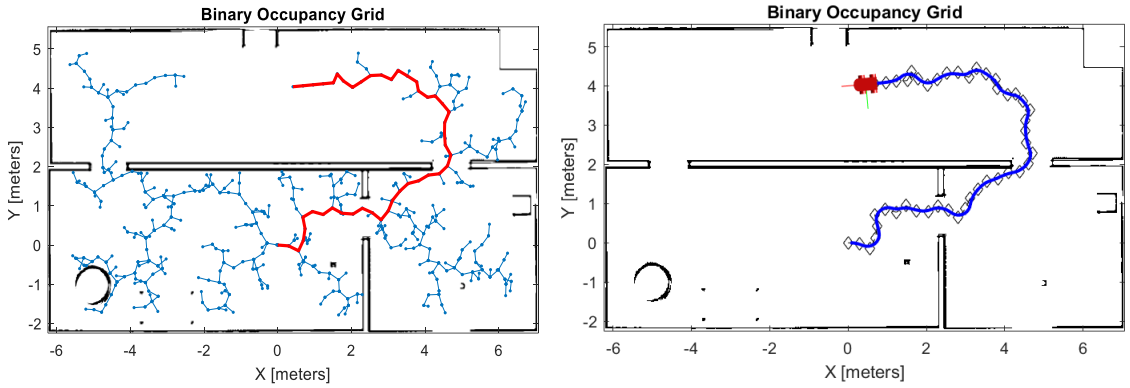
Commentaires

- Dans la figure (III.9), nous considérons que le robot doit démarrer de la position [0 0] vers la destination située à la position [4.5 4]. Les nœuds bleus représentent les

nœuds échantillonnés aléatoirement dans l'espace libre de la carte et la ligne rouge représente le chemin planifié par la méthode PRM pour le robot (la partie haute de la figure).

- On note la différence entre la 1ère exécution et la 2ème exécution du code ; c'est dû au fait que PRM est une méthode probabiliste. Ce qui signifie qu'on peut avoir des chemins différents pour aller du même point de départ au même point d'arrivée.
- A partir des courbes en bas de la figure (III.9), nous remarquons que la trajectoire réelle du robot suit parfaitement bien la trajectoire planifiée par PRM. A la fin de son trajet, le robot s'est arrêté exactement à la position [4.4790 3.9049] qui est très proche de la destination désirée.
- Le résultat d'une autre exécution pour des positions initiale et destination différentes est représenté à la figure (III.10). La trajectoire planifiée par PRM est montré à gauche de la figure et la trajectoire réelle du robot après l'avoir commandé par Pure Pursuit est montrée à droite en vert. Nous remarquons que le robot a fait tout un cercle au départ avant qu'il ne continue son chemin. Des fois on peut avoir une instabilité de la commande à la fin et le robot tourne sur lui-même à la position finale.

La Figure (III.11) montre les résultats de planification par l'algorithme RRT à gauche et de commande par PurePursuit à droite.



FigureIII. 11: Planification par RRT et commande par PurePursuit

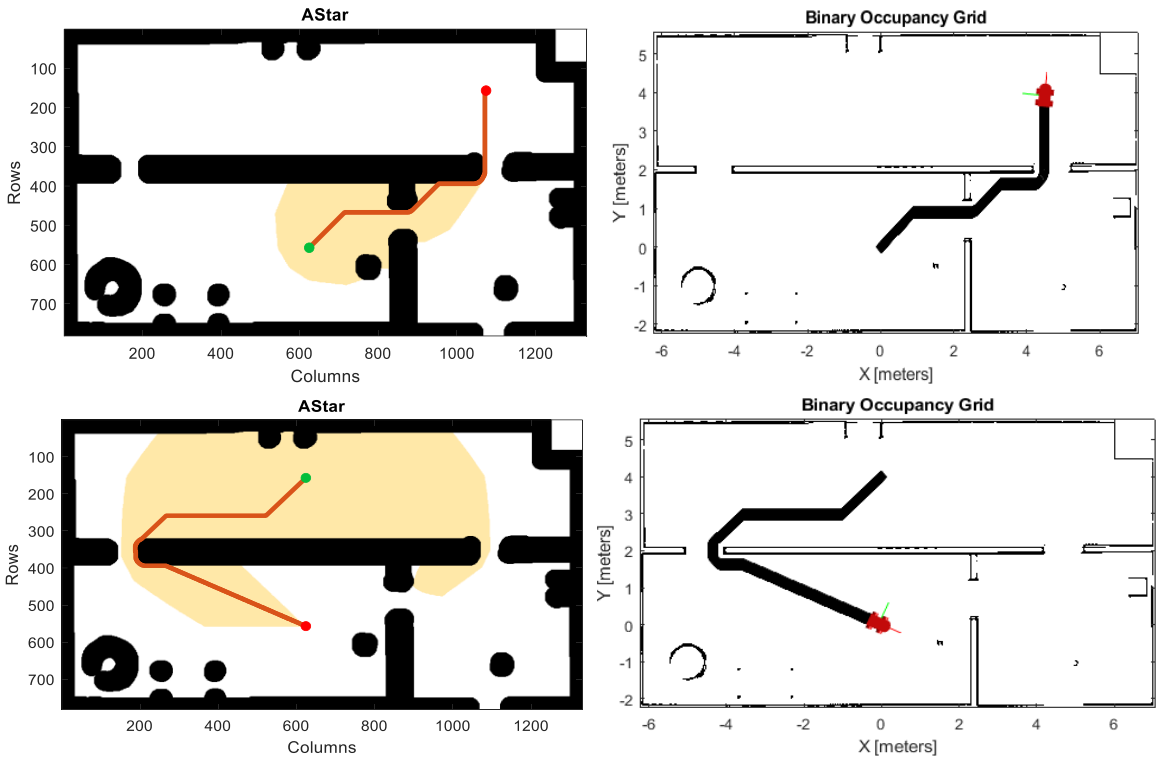
Commentaire

- Dans ce cas, le robot doit se déplacer de la position [0 0] à la position [0 4] en utilisant l'algorithme RRT et la commande PurePursuit. Un arbre tout d'abord est construit couvrant l'espace libre. Cet arbre, représenté en bleu à gauche de la figure (III.11), est constitué de nœuds générés aléatoirement de manière incrémentale. Ensuite, se fait la détermination du chemin entre les deux nœuds 'Départ' et 'Destination' qui font partie eux aussi de l'arbre, et ce en utilisant un algorithme RRT de de recherche de plus court

chemin. Ce chemin est montré en rouge. On remarque qu'on obtient toujours la même trajectoire quand on refait l'exécution et que le nombre de waypoints (points de cheminement) est élevé par rapports aux autres algorithmes. La position finale du robot est [0.4552 4.0412].

- Ensuite, on applique le contrôleur PurePursuit au robot. Nous remarquons que la trajectoire réelle du robot en bleu (à droite de la figure (III.11)) suit parfaitement bien la trajectoire planifiée par RRT.

La Figure (III.12) illustre les résultats de simulation de l'algorithme de planification Astar et la commande PurePursuit pour différentes positions initiales et différentes destinations. A gauche sont montrés les trajectoires planifiées par AStar en rouge et à droite, sont montrées les trajectoires réelles (en noir) du robot qui a été commandé par PurePursuit.



FigureIII. 12: Planification par Astar et commande par PurePursuit (pour différentes positions initiales et positions finales)

Commentaires

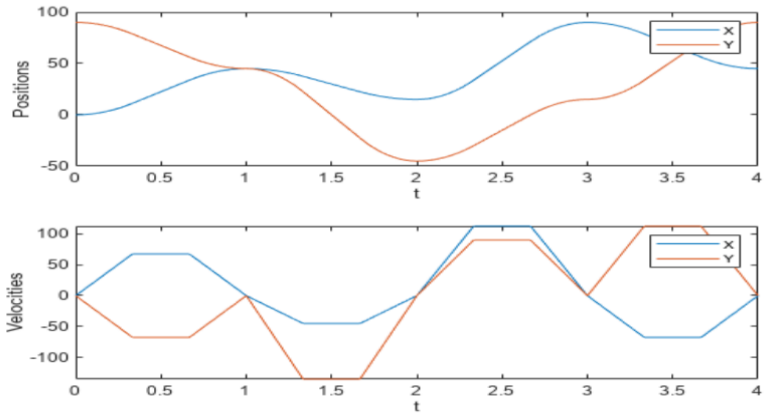
- Le robot doit démarrer de la position de 'Départ' (le nœud vert) pour aller à la position 'Destination' représenté par un nœud rouge, en évitant les obstacles présents dans l'environnement.

- Dans cette figure on note une zone jaune représentant toutes les cellules de la grille représentant l'environnement qui sont impliquées dans le calcul du chemin optimal.
- A partir du résultat obtenu, nous remarquons que le chemin planifié évite toute collision avec les obstacles ce qui permet au robot d'atteindre la cible sain et sauf.
- Notons que dans toutes nos simulations, nous avons gonflé la carte de l'environnement pour assurer une marge de sécurité pour le robot et éviter qu'il ne heurte un mur ou un obstacle lors de son déplacement.
- Après la planification de la trajectoire, nous avons appliqué l'algorithme de commande PurePursuit au robot, et nous remarquons que la trajectoire réelle du robot suit bien et précisément avec moins d'erreur la trajectoire prévue par AStar par rapport aux autres algorithmes.

III.3.2. Commande Non Linéaire à base de la fonction de Lyapunov

Pour commander un robot mobile nous devons implémenter un algorithme de contrôle approprié qui permet au robot de suivre avec précision la trajectoire désirée. Dans les résultats précédents nous avons utilisé la commande PurePursuit pour assurer cette tâche. Par contre dans cette section, nous allons implémenter la commande NL à base de fonction de Lyapunov que nous avons étudié au chapitre I et la comparé à la loi de commande PurePursuit en terme de précision dans le suivi de trajectoire désirée.

Pour se faire et dans un premier temps, nous devons générer une trajectoire de référence à travers l'ensemble de 'waypoints' obtenus à la sortie du planificateur utilisé qui suive un profil de vitesse trapézoïdal. La figure(III.13) montre un exemple de trajectoires pour les positions x_r et y_r désirées et le profil des vitesses trapézoïdales entre chaque paire de 'waypoints'.

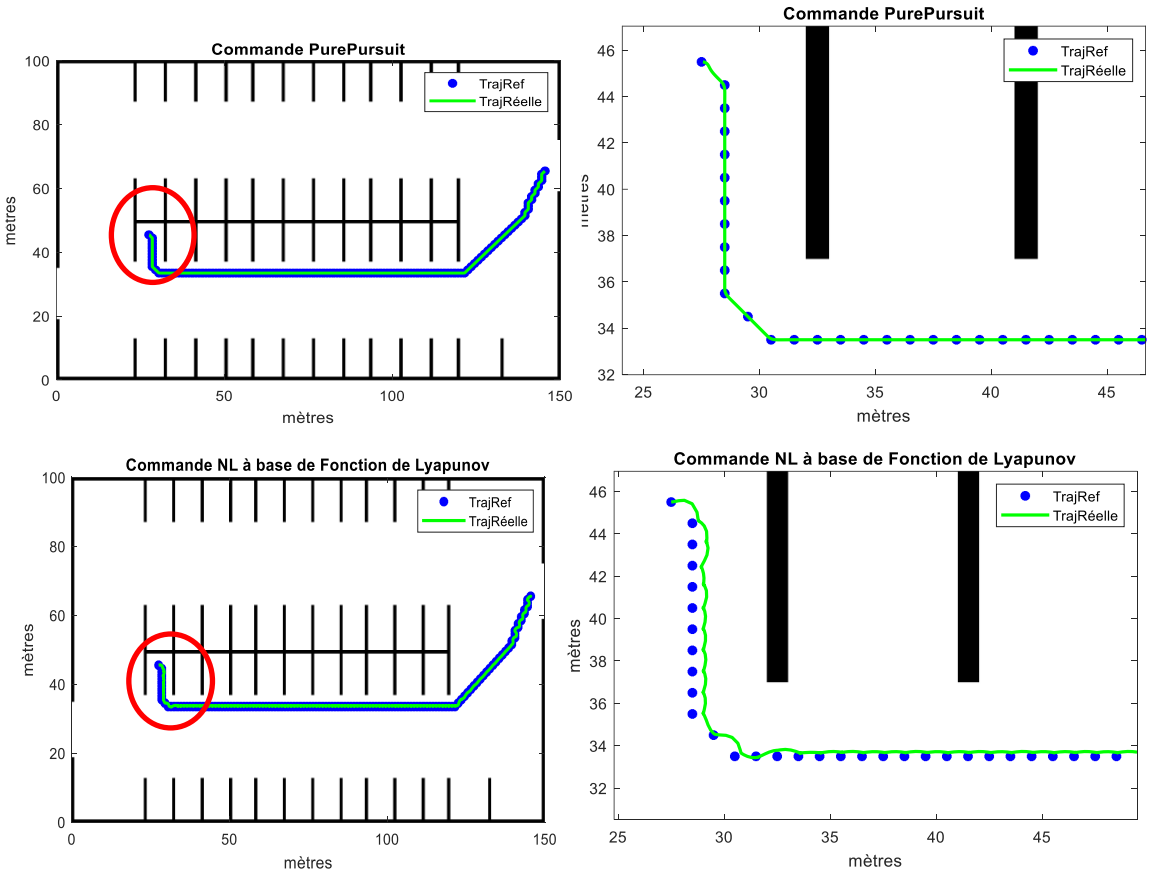


FigureIII. 13:Exemple de trajectoires pour les positions x_r et y_r et le profil des vitesses trapézoïdales entre chaque paire de 'waypoints'.

Ainsi, connaissant les coordonnées de la trajectoire de référence x_r, y_r nous pouvons déterminer l'orientation de référence et les vitesses linéaire et angulaires de références par :

$$\begin{cases} \theta_r = Atan2(\dot{y}_r(t), \dot{x}_r(t)) \\ v_r(t) = \sqrt{\dot{x}_r{}^2(t) + \dot{y}_r{}^2(t)} \\ \omega_r(t) = \frac{\ddot{y}_r(t)\dot{x}_r(t) - \ddot{x}_r(t)\dot{y}_r(t)}{\dot{x}_r{}^2(t) + \dot{y}_r{}^2(t)} \end{cases} \quad \textbf{(III.1)}$$

Les résultats de simulation de cette commande sont montrés sur la figure (III.14) et comparé aux résultats obtenus en utilisant la commande Pure Pursuit dans les mêmes conditions de simulation.



FigureIII. 14: Comparaison de la commande NL à base de la fonction de Lyapunov

Commentaire

- Dans cet exemple, l'objectif est de permettre au robot de type voiture de se déplacer, de façon autonome, de sa position initiale dans le parking vers la sortie. Nous avons obtenu la trajectoire de référence en bleu planifiée par l'algorithme Astar et la trajectoire réelle du robot en vert obtenue pour les deux types de commande : la

Commande PurePursuit et la commande NL à base de la fonction de Lyapunov dans un environnement de type parking. Nous remarquons que la trajectoire réelle est confondue avec la trajectoire de référence pour les deux lois de commande.

- Pour des raisons de comparaison, nous avons zoomé une partie du chemin (le cercle rouge) et remarqué que dans la commande PurePursuit la trajectoire réelle suit la trajectoire de référence avec précision et faible erreur. Alors que, dans la commande de Lyapunov on trouve un petit décalage entre les deux trajectoires.

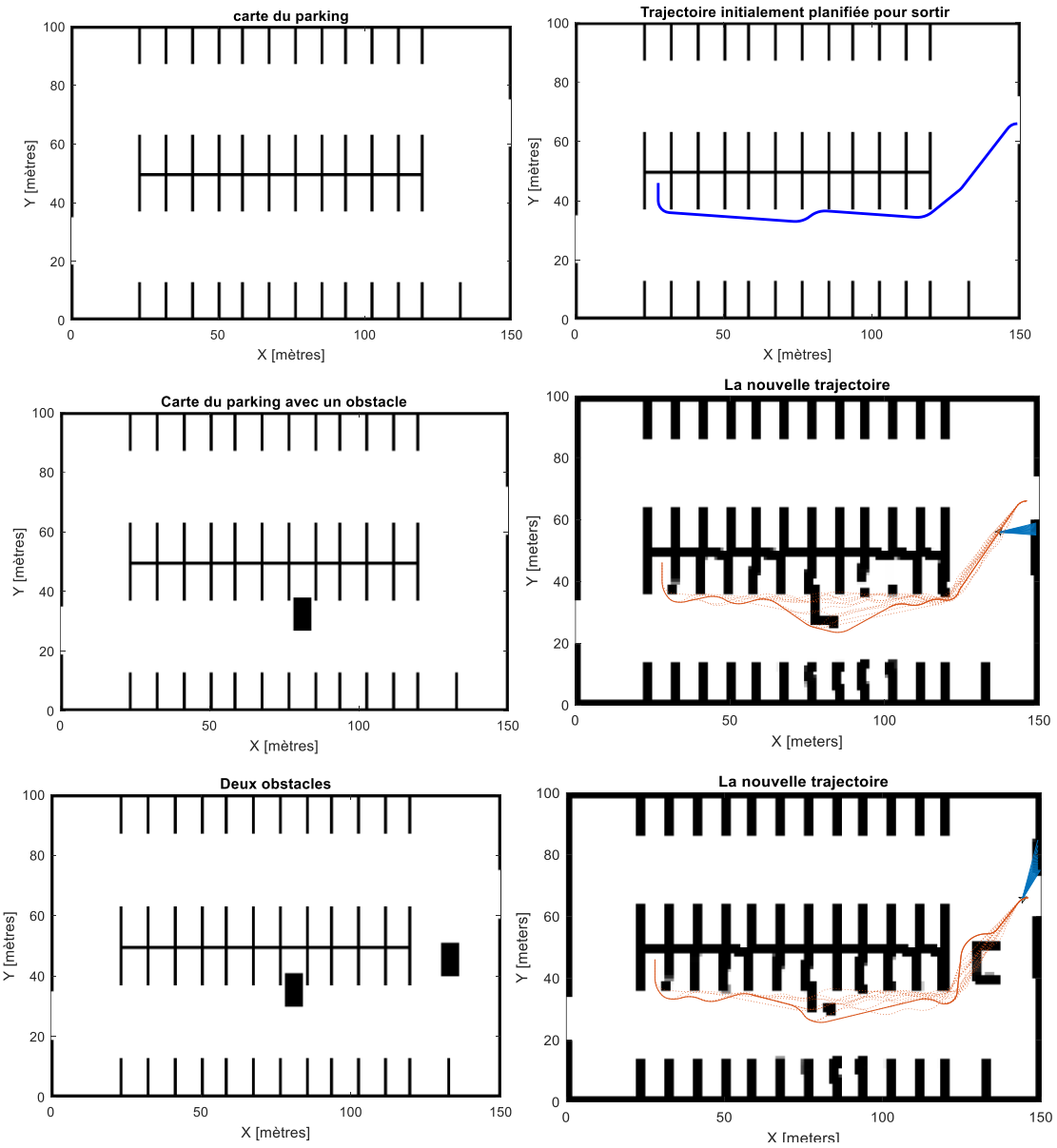
III.3.3. Replanification dynamique par AStar et D Star Lite

Dans cette partie, nous allons considérer le cas où on a un environnement connu, alors le robot planifie sa trajectoire initiale. Mais au cours de son déplacement, il peut détecter, sur son chemin, des obstacles statiques ou dynamiques imprévus grâce à ses capteurs (télémètre Laser par exemple), il replanifie alors sa trajectoire en utilisant l'une des méthodes suivantes : Replanification dynamique par Aster ou bien par l'algorithme Dstar Lite.

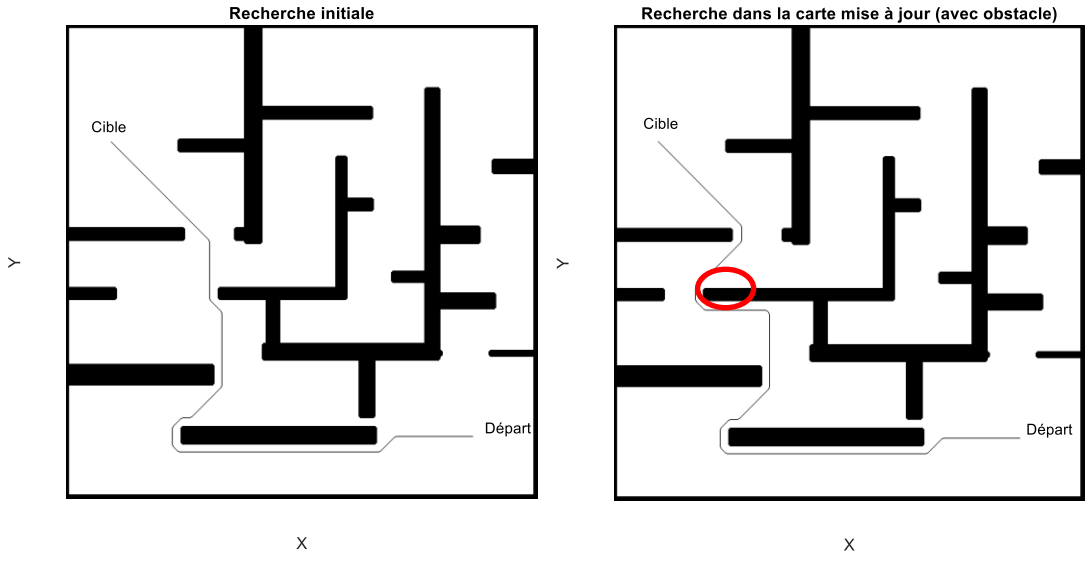
Les figures (III.15) et (III.16) montrent les résultats de la Replanification dynamique par A star et D star Lite respectivement.

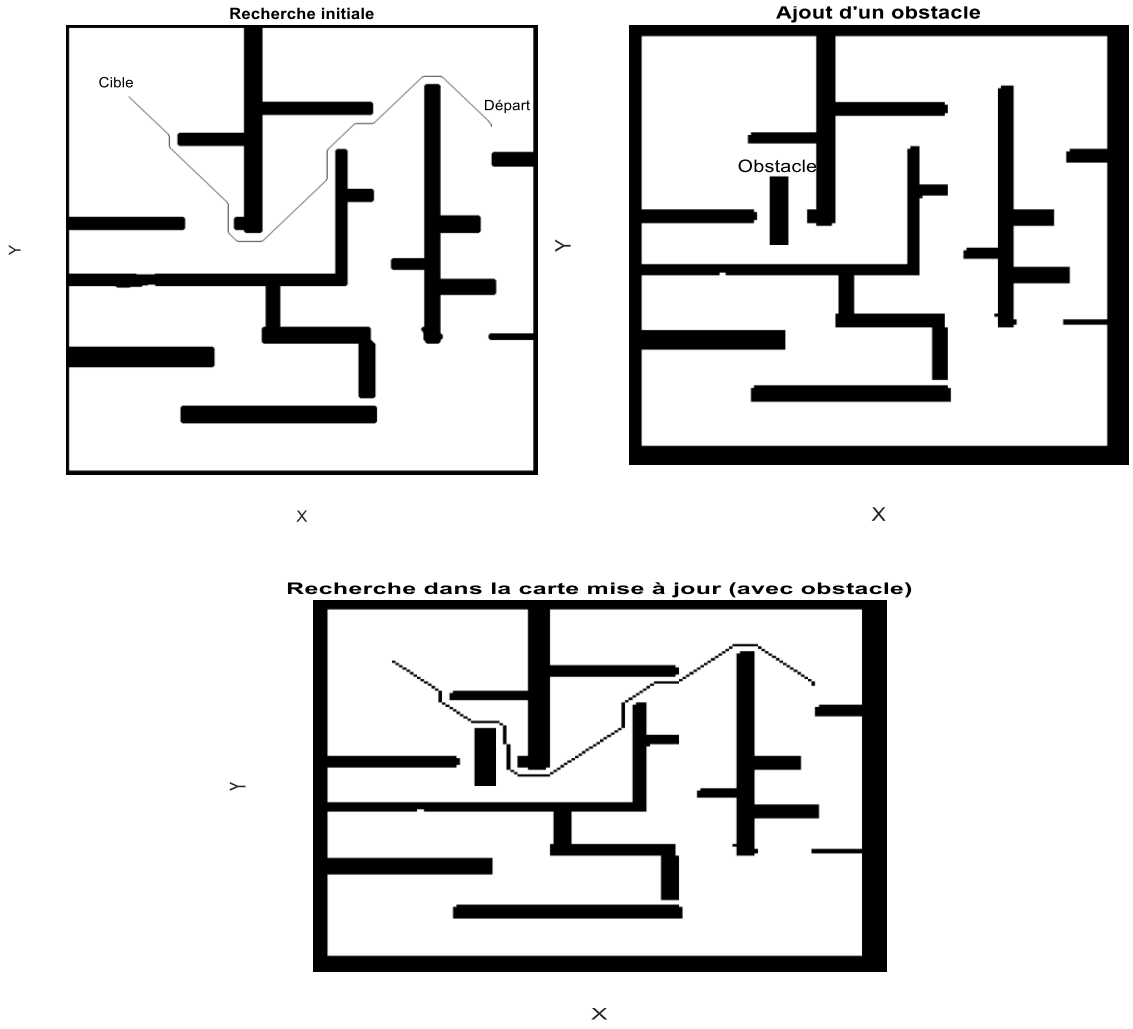
Commentaire

- Dans le cas de la replanification dynamique par AStar, dès que le robot détecte un obstacle par le biais de son capteur, il met à jour sa carte et replanifie sa trajectoire par l'algorithme Astar à partir du point de départ et retrouve un autre chemin. Il continue comme ça jusqu'à ce qu'il atteint la cible (voir Figure III.15).
- Par contre dans le cas d'algorithme D star lite, lorsque le robot détecte un obstacle, il refait la planification de la nouvelle trajectoire en considérant sa position actuelle comme point de départ ce qui rend cet algorithme plus rapide (voir la figure III.16). le temps d'exécution est 0.994278 secondes pour la recherche initiale alors que celui de la recherche après la détection de l'obstacle est 0.042787 secondes qui est très court vu que le calcul se fait à partir de la position actuelle du robot et au fait que le planificateur garde les coûts des cellules obtenus dans la recherche initiale et continue sa nouvelle recherche sans avoir besoin d'initialisé tous les coûts.
- Sur le base de ce qui précède, nous concluons que la Replanification dynamique par Dstar lite prend moins de temps d'exécution par rapport à la Replanification dynamique par Astar.



FigureIII. 15: Replanification dynamique par Astar



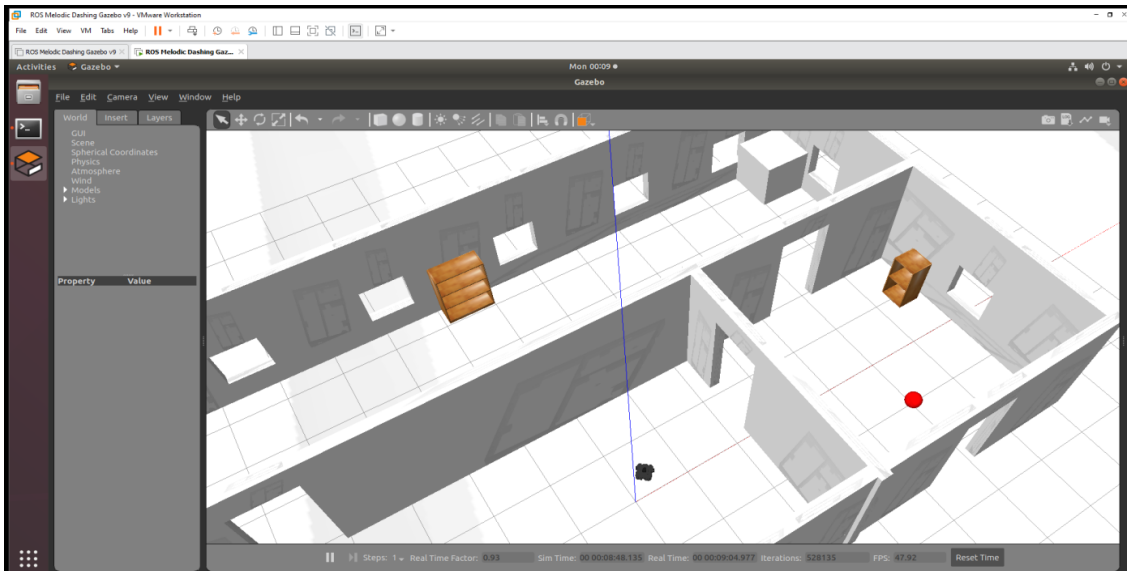


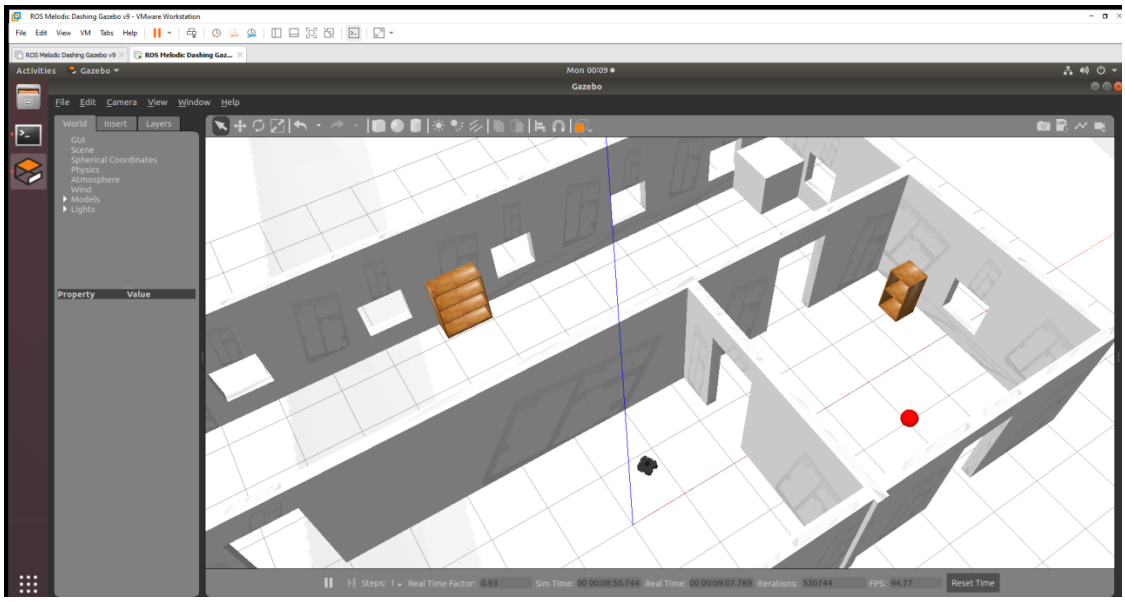
FigureIII. 16: Replanification dynamique par Dstar Lite

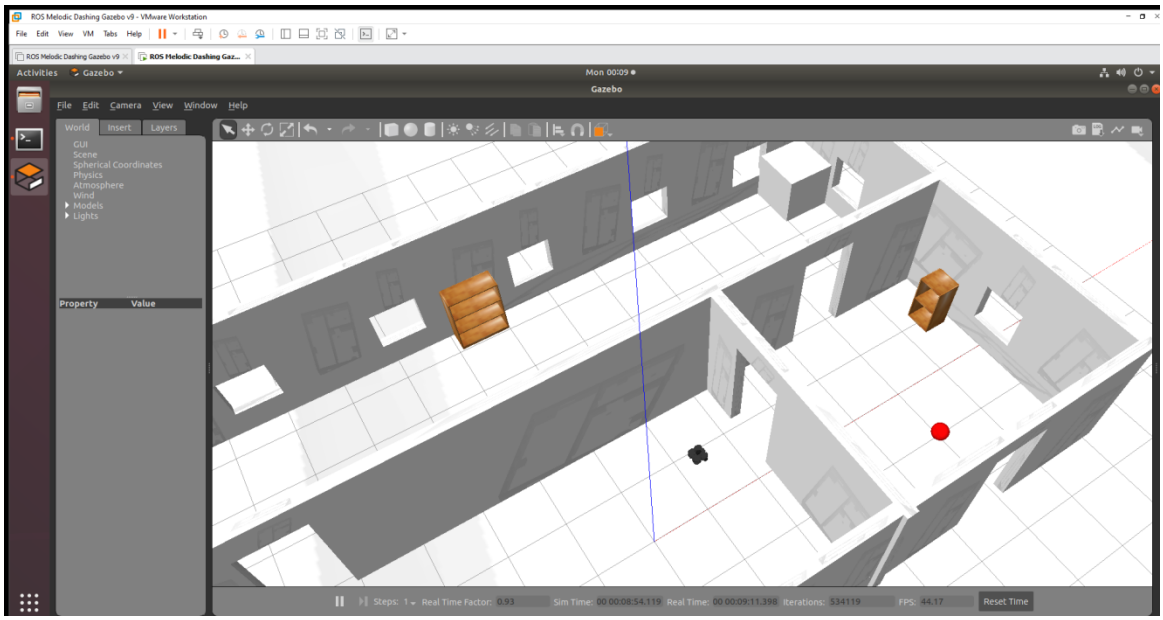
III.3.4. Résultats de planification et de commande dans Gazebo

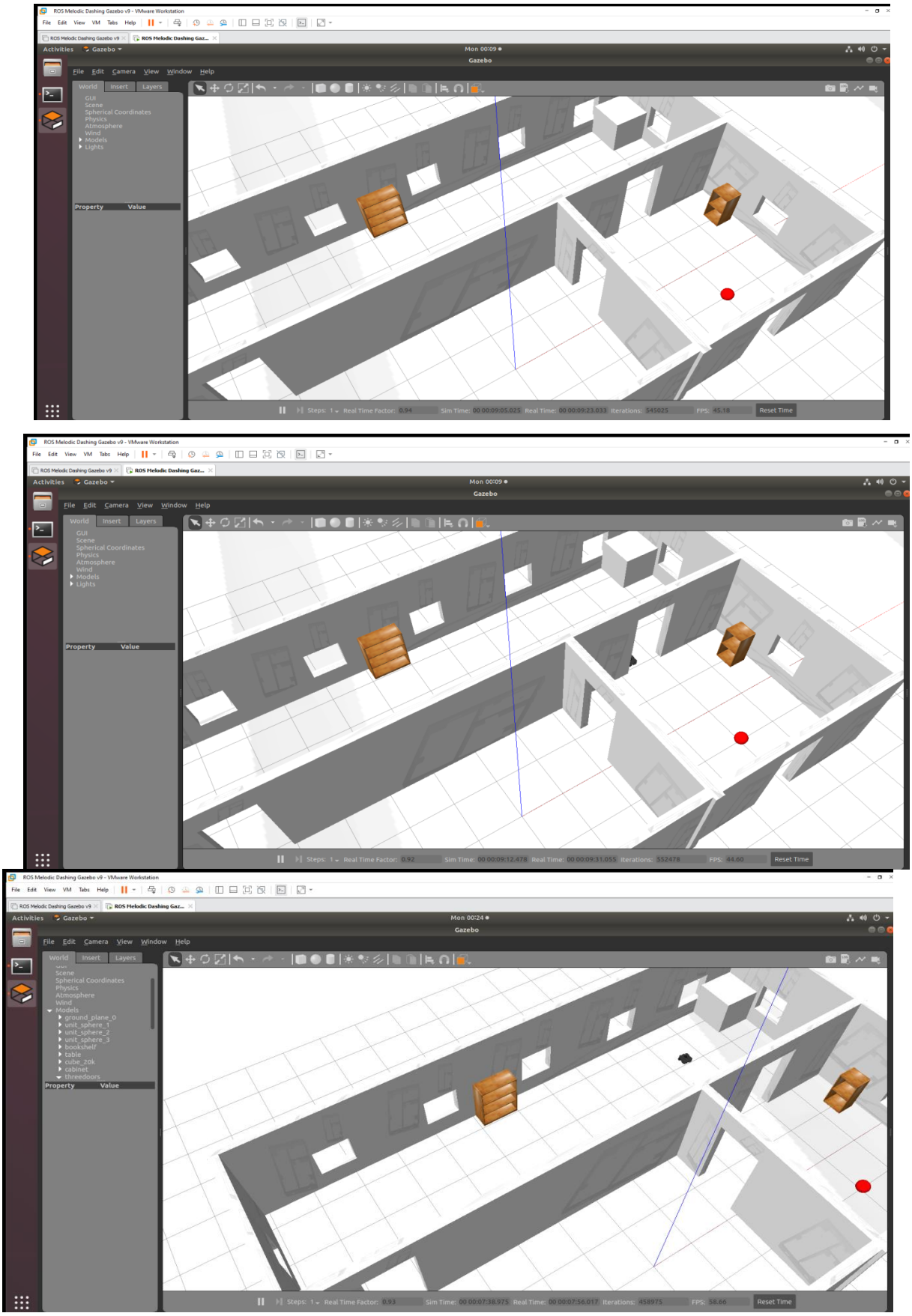
Gazebo offre la possibilité de simuler avec précision et efficacité des populations de robots dans des environnements intérieurs et extérieurs complexes. Dans notre travail, nous avons simulé un robot Turtlebot et grâce à l'interface Matlab-ROS nous avons pu faire la planification de sa trajectoire dans l'environnement office de Gazebo à l'aide de l'algorithme PRM, ensuite à l'aide de l'algorithme Astar. Nous avons également commandé le robot Turtlebot par la loi de commande PurePursuit pour le déplacer de façon autonome le long du chemin planifié.

Les figures (III.17) et (III.18) suivantes montrent les résultats de cette simulation dans Gazebo, nous considérons que le robot doit démarrer de la position [0 0] vers la destination située à la position [4.5 4] dans les deux cas.

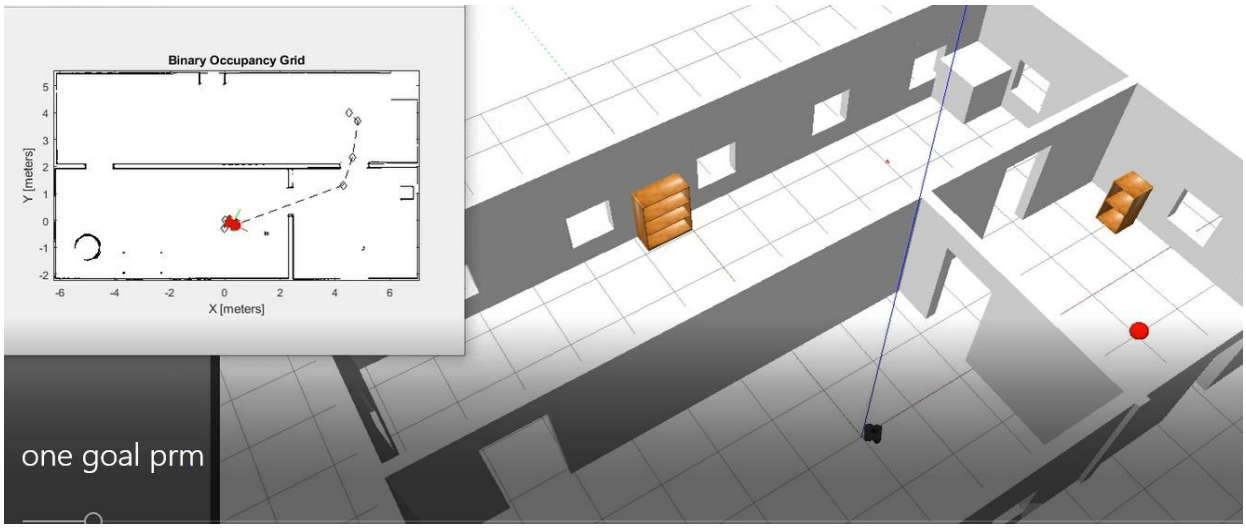

ROS Melodic Dashing Gazebo v9 - VMware Workstation
Activities
Gazebo
Mon 00:09
File Edit Camera View Window Help
World Insert Layers
GUI
Scene
Spherical Coordinates
Physics
Atmosphere
Wind
Models
Lights
Property Value
Steps: 1 Real Time Factor: 0.93 Sim Time: 00 00:08:48.135 Real Time: 00 00:09:04.977 Iterations: 528135 FPS: 47.92
Reset Time

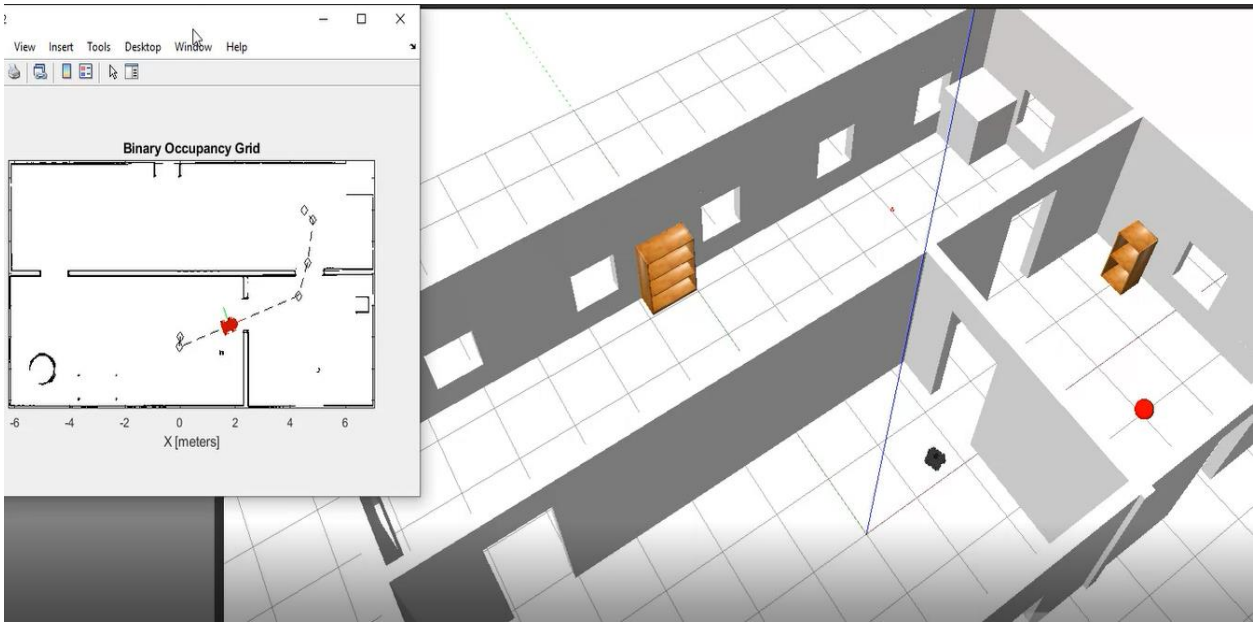

ROS Melodic Dashing Gazebo v9 - VMware Workstation
Activities
Gazebo
Mon 00:09
File Edit Camera View Window Help
World Insert Layers
GUI
Scene
Spherical Coordinates
Physics
Atmosphere
Wind
Models
Lights
Property Value
Steps: 1 Real Time Factor: 0.93 Sim Time: 00 00:08:50.744 Real Time: 00 00:09:07.789 Iterations: 530744 FPS: 44.77
Reset Time

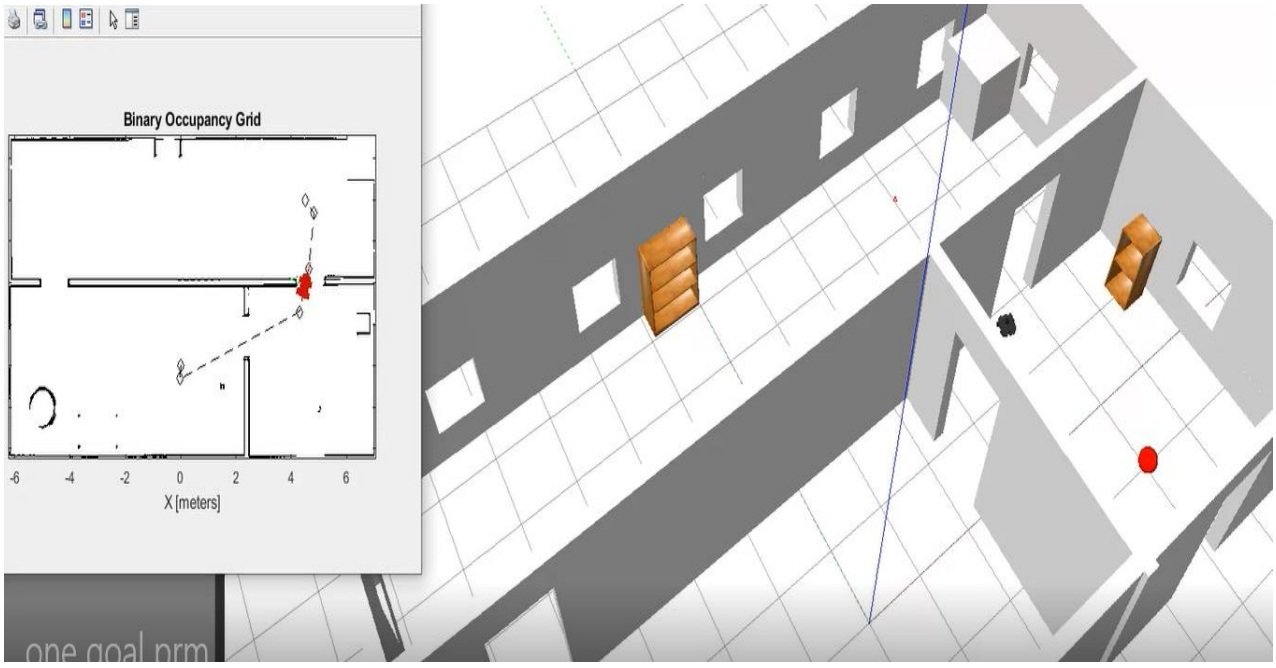

ROS Melodic Dashing Gazebo v9 - VMware Workstation
Activities
Gazebo
Mon 00:09
File Edit Camera View Window Help
World Insert Layers
GUI
Scene
Spherical Coordinates
Physics
Atmosphere
Wind
Models
Lights
Property Value
Steps: 1 Real Time Factor: 0.93 Sim Time: 00 00:08:54.119 Real Time: 00 00:09:11.398 Iterations: 534119 FPS: 44.17
Reset Time

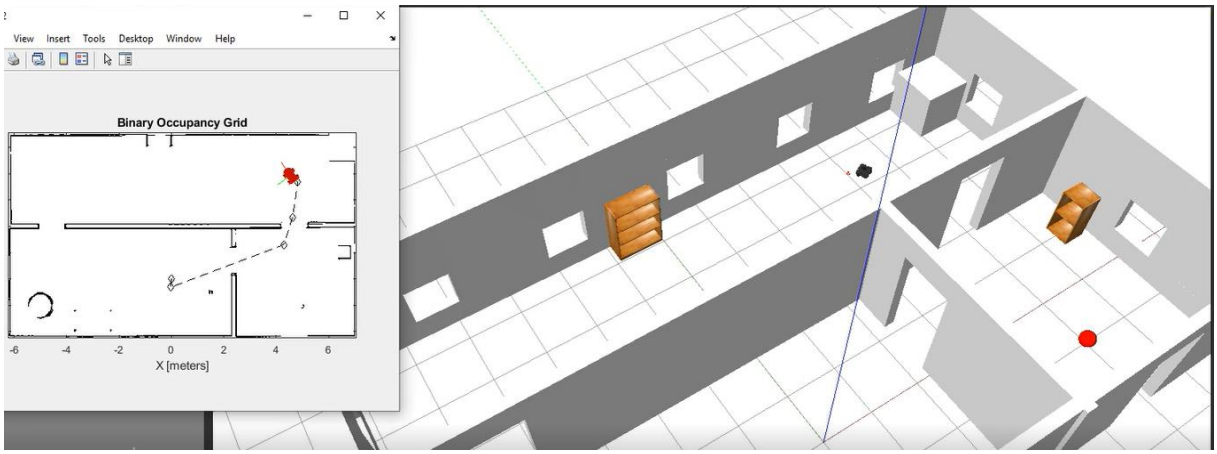


FigureIII. 17: Planification à l'aide A star et commande du robot dans Gazebo


Binary Occupancy Grid
Y [meters]
X [meters]
one goal prm


View Insert Tools Desktop Window Help
Binary Occupancy Grid
X [meters]


Binary Occupancy Grid
X [meters]



FigureIII. 18: Planification à l'aide PRM et commande du robot dans Gazebo

- Sur la base de ces figures, on peut dire que les résultats de planification et suivi de trajectoire du robot Turtlebot dans GAZEBO ont été très satisfaisants et performants.
- Et qu'il est possible de commander le robot Turtlebot réel par les mêmes algorithmes par les mêmes codes juste en changeant quelques instructions.

III.4. Conclusion

Dans ce chapitre nous avons tout d'abord commencé par donner les outils nécessaires et les environnements de développement pour la réalisation de notre travail. Ensuite, nous avons présenté et commenté les résultats d'implémentation des algorithmes de planification de trajectoire, PRM, RRT, Astar et D star Lite, dans des environnements statiques et dynamiques. Les résultats obtenus étaient très concluants. De plus, l'approche PurePursuit et la loi de commande non linéaire à base de la fonction de Lyapunov ont été utilisées pour contrôler le robot et lui permettre ainsi de suivre la trajectoire souhaitée en évitant les obstacles. Leurs performances ont été évaluées par simulation. Nous avons constaté que le chemin réel suit bien le chemin désiré. Finalement, nous avons montré les résultats de la simulation du robot Turtlebot en utilisant le simulateur GAZEBO.

Conclusion générale

Conclusion générale

Le travail effectué dans cette mémoire a porté essentiellement sur l'élaboration de différents algorithmes de commande et de planification de trajectoire robuste pour la poursuite de trajectoire d'un robot mobile non holonome de type unicycle. L'objectif est de forcer le robot mobile à atteindre la configuration d'arrivée à partir d'une configuration de départ avec un minimum d'erreurs et sans collision avec les obstacles et sans intervention humaine.

En effet ce travail s'est articulé autour de trois chapitres : le premier qui avait pour but de présenter un état de l'art sur la robotique mobile ainsi que la modélisation cinématique d'un robot mobile unicycle qui est largement utilisé, notamment dans le domaine de la recherche en raison de sa simplicité et de sa facilité de travail. Ensuite des techniques de commandes ont été développées principalement pour ce type de robots pour résoudre problème de suivi de trajectoire.

Dans le deuxième chapitre, nous avons détaillé les différentes théories de planification de trajectoire à savoir les méthodes PRM, RRT, AStar et Dstar Lite et exposé leurs algorithmes. Enfin dans le dernier chapitre. Ces méthodes de planification ont été testées dans des environnements statiques et dynamiques de formes différentes. Dans ces différents tests, ces méthodes ont prouvé leur efficacité dans le calcul de chemin court et sans collision. Nous avons également discuté et commenté les résultats de suivi de trajectoire du robot mobile moyennant des lois de commande telles que : l'approche Pure Pursuit et une loi de commande non linéaire à base de la fonction de Lyapunov. Nous avons remarqué que les deux méthodes sont très efficaces et donnent de bons résultats et que la méthode Pure Pursuit donne une plus grande précision.

Nous avons également présenté les résultats de la Co-simulation entre Matlab et le système ROS en utilisant le simulateur Gazebo permettant de créer des environnements et simuler le robot Turtlebot. Les résultats de planification et de suivi de trajectoires étaient très satisfaisants et concluants.

Finalement, nous pouvons dire que tous les objectifs tracés au début ont été accomplis et que nous avons acquis beaucoup de connaissances dans le domaine de la robotique et l'interfaçage entre Matlab et le système ROS.

Comme travail futur, et en se basant sur les méthodes étudiées et implémentées dans ce mémoire, nous proposons d'apporter des améliorations permettant de planifier des trajectoires en temps réel dans des environnements inconnus et/ou comportant des obstacles dynamiques.

[1] www.icube-avr.unista.fr/cours_rob_intro.pdf.

[2] Agnès Guillot, « la robotique de A à Z », le laboratoire de recherche dédié au Développement ,20/01/2003.Georges

[3] Boimond Jean-Louis, Cours-Robotique

[4] A. ALLOUI, A.HAJ Brahim. « Proposition d'une solution multi-agent pour la Command et la coopération multi-robot mobile ».Mémoire d'ingénieur d'état en automatique Université Biskra, juin 2007

[5] BALI, C, ABAIDI, H. « Réalisation d'un robot mobile avec évitement d'obstacle et Trajectoire programmé ». Mémoire de Master en Génie des systèmes industriels. Université Mohamed Khider de Biskra. 2012

[6] RANDRESTA, TH. « L'autonomisation des robots sur le champ de bataille. La guerre, le droit et l'éthique ». Revue internationale et stratégique, vol 4, 18-27, 2013.

[7] LHOMME-DESAGES, D. « Commande d'un robot mobile rapide à roues non Directionnelles sur sol naturel ». Thèse de doctorat en Mécanique Robotique. Université paris 6 Pierre et Marie Curie. 10 avril 2008.

[8] JORY, L. « Commande des mouvements et de l'équilibre d'un robot humanoïde à roues Omnidirectionnelles ». Thèse de Doctorat en Electronique, Electrotechnique, Automatique et Traitement du signal. Université Grenoble alpes. 2 juillet 2015.

[9] ZIANE, Med L. « Navigation flou d'un robot mobile ». Mémoire de Master en Electronique. Université Abderrahmane Mira de Bejaia. 2013/2014.

[10] Dou, M. DJOKHRAB, A. « Commande d'un robot mobile type voiture par réseaux de Neurones ». Mémoire d'ingénieur d'état en automatique. Université de Biskra. Juin 2005.

[11] Dr. SLIMANE, N. « Système de localisation pour robots mobiles ». Thèse de doctorat en Electronique. Université de Batna. 2005.

[12] MEDJOUBI, H., ABDESSEMED Y et HASSAM ABDELOUAHAB. « Leader-follower formation control using PI Controller». 1st International conference on electrical and electronics engeneering IC3E, 2018.

[13] BAHAZ RADOUANE et BAHAZ ABDELBASSET. Navigation autonome et suivi de trajectoires d'un robot mobile à entraînement différentiel. Automatique/ Automatique et systèmes. Université de Ghardaïa.2019/2020

[14] BOUFERA FATIMA. Contribution des outils de l'intelligence artificielle dans la robotique mobile. Thèse de doctorat en Informatique. Université d'Oran .2014

[15] « Observer-Based Trajectory Tracking Control for a Wheeled Mobile Robot », Asian Control Conference, 2009. ASCC 2009. 7th.

[16] Wallace, R. et al. "First Results in Robot Road-Following", repon witbio CMU-RI-TR-8&4

[17] PEREZ T. «Spacial planning : A configuration space approach. In IEEE transactions on Computers» : 1983

[18] JUAN MANUEL AHUCTZIN LARIOS « Le fil d'Ariane : Une méthode de planification générale : Application à la planification automatique de trajectoire », Thèse en vue de l'obtention du titre de doctorat Informatique de l'Institut National Polytechnique de Grebnoble, septembre 2004.

[19] BOUHLASSA LOUBNA, planification de trajectoire d'un robot basée sur les réseaux De neurones et les algorithmes génétiques, thèse de magister, USTO.

[20] A.SCHEUER, «planication de chemins à courbure continue pour robot mobile non holonome ». Thèse PhD, INPG, France, 1998.

[21] F. LAMIRAUX, J.-P. LAUMOND, « Smooth Motion Planning for Car-Like Vehicles ». IEEE Trans.On Robotics and Automation, Vil.17, No.4, pp.498-502, Aout 2001.

[22] L. E DUBINS, « On curve of minimal length with a contrain on average curvature, and with prescribed initial and terminal position and tengents ».American Journal of Mathematics, Vol.79, No.3, pp.497-516,1957.

[23] T.FRAICHARD, A.SCHEUER, « From Reeds and Shepp's to continuous curvature path ».IEEE Trans .On Robotics, Vol. 20, No. 6, pp.1025-1035, Decembre 2004

[24] NASSIM BLIN. Planification interactive de mouvement avec contact. Robotique Institut National Polytechnique De Toulouse, 2017. Français

[25] HART, P.E.; NILSSON, N.J.; RAPHAEL, B. A formal basis for the heuristic determination of minimum cost paths in graphs. IEEE Trans. Syst. Sci. Cybern. 1968, 4, 100–107.

[26] E.W.DIJKSTRA. "A Note on Two Problems in Connexion with Graphs". In: Numerische Mathematik 1.1 (Dec. 1959), pp. 269–271.2005

[27] H. CHOSET, A. MILLS-TETTEY, K. TANTISEVI ET V. LEE-SHUE JR. Prasad Narendra Atkar, «Robotic Motion Planning: A* and D* Search,» Carnegie Mellon University, Pittsburgh, 2005.

[28] KOENIG, S.; LIKHACHEV, M.; FURCY, D. Lifelong Planning A*. Artif. Intell. 2004, 155, 93–146. [CrossRef]

[29] LINA MEJBAR. (2020). Véhicule autonome pour cartographie. Rapport de projet de fin d'étude Ingénieur. Département Informatique Électronique et Automatique. Université de Lille.

[30] https://fr.m.wikipedia.org/wiki/fichier:ROS_logo.svg

[31] http://fr.mathworks.com/products/ros.html

[32] MUAHOATIEU. (2021).communication between matlab ros toolbox and ros network

[33] KOENIG, N. AND HOWARD, A. (2004). Design and use paradigms for Gazebo, an open-source multi-robot simulator. In Intelligent Robots and Systems, 2004. (IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on, volume 3, pages 2149–2154. IEEE.

[34] www.turtlebot.com

[35] www.turtlebot.com/about/