REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Ecole Nationale Polytechnique

المدرسة الوطنية المتعددة التقنيات
Ecole Nationale Polytechnique

Ecole Nationale Polytechnique
Département d'Electronique
Centre de développement des technologies avancées

C.D.T.A

Département d'Electronique

Mémoire de projet de fin d'études
pour l'obtention du diplôme d'ingénieur d'état en électronique

Contribution à la planification de mouvements d'une chaise roulante robotisée : Applicaiton sous ROS

DJERIDI Maria

Sous la direction de **Mme. Sara BOURAINE Dr.**

Présenté et soutenue publiquement le (29/06/2019)

Composition du jury :

Président	**M. Mourad HADDADI**	**Dr.**	**ENP**
Promoteur	**Mme. Sara BOURAINE**	**Dr.**	**CDTA**
Promoteur	**M. Cherif LARBES**	**Dr.**	**ENP**
Examinateur	**M. Adel BELOUCHRANI**	**Prof**	**ENP**

ENP 2019

REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Ecole Nationale Polytechnique

المدرسة الوطنية المتعددة التقنيات
Ecole Nationale Polytechnique

Ecole Nationale Polytechnique
Département d'Electronique
Centre de développement des technologies avancées

C.D.T.A

Département d'Electronique

Mémoire de projet de fin d'études
pour l'obtention du diplôme d'ingénieur d'état en électronique

Contribution à la planification de mouvements d'une chaise roulante robotisée : Applicaiton sous ROS

DJERIDI Maria

Sous la direction de **Mme. Sara BOURAINE Dr.**

Présenté et soutenue publiquement le (29/06/2019)

Composition du jury :

Président	**M. Mourad HADDADI**	**Dr.**	**ENP**
Promoteur	**Mme. Sara BOURAINE**	**Dr.**	**CDTA**
Promoteur	**M. Cherif LARBES**	**Dr.**	**ENP**
Examinateur	**M. Adel BELOUCHRANI**	**Prof.**	**ENP**

ENP 2019

ملخص

يتعامل هذا العمل مع مشكلات تخطيط الحركة لروبوت متنقل يعمل في بيئة غير منتظمة. تم تجهيز الروبوت بمستشعر ليزر UST-10LX Hokuyo مع مجال رؤية محدود. في مثل هذه الحالة ، نقترح اعتماد نهج ملاحة قادر على التغلب على هذه القيود مع ضمان الاصطدام بالعقبات. لهذا السبب ، يعتمد الحل المقترح على نهج التخطيط التفاعلي القائم على أخذ عينات مساحة التحكم (ISS).الخوارزمية المطورة تسمى NAV-ISS. علاوة على ذلك ، هناك حاجة إلى تمثيل للبيئة والذي يعتمد على طريقة شبكة الإشغال من عمليات المسح بالليزر. يتم تنفيذ الخوارزميات المقترحة بلغة Python وفقًا لنظام ROS واختبارها على الكرسي الذكي Faursa.

الكلمات المفتاحية

الحركة تخطيط ،منتظمة غير بيئة ، المستقلة الملاحة ، متنقل روبوت.

Abstract

This work deals with motion planning issues for a mobile robot operating in an unknown cluttered environment. The robot is equipped with a UST-10LX Hokuyo laser sensor with a limited field of view. In such a situation, we propose to adopt a navigation approach able to overcome these constraints while guaranteeing the non-collision with obstacles. For this reason, the proposed solution is based on reactive planning approach based on control space sampling (ISS). The developed algorithm is named NAV-ISS. Furthermore, a representation of the environment is required which is based on occupancy grid method from laser scans. The proposed algorithms are implemented in Python language under ROS system and tested on Faursa smart wheelchair.

Keywords

Mobile robot, autonomous navigation, unknown environment, motion planning.

Résumé

Ce travail traite les problèmes de planification de mouvement pour un robot mobile chaise roulante robotisée évoluant dans un environnement inconnu obstrué encombré d'obstacles. Le robot est équipé d'un capteur laser de type *UST-10LX Hokuyo* ayant un champ de vision limité. Dans une telle situation, nous proposons d'adopter une approche de navigation permettant de palier à ces contraintes tout en garantissant la non collision avec les obstacles. Pour ce faire, la solution proposée est basée sur la planification réactive par échantillonnage de l'espace de contrôles (ISS). L'algorithme développé est nommé par NAV-ISS. Pour détecter les obstacles et déterminer leur position, NAV-ISS nécessite une représentation de l'environnement. Cette dernière est basée sur la modélisation par grilles d'occupation à partir d'un scan laser. Les algorithmes proposés sont implémentés en langage Python sous le système ROS et testés sur la chaise robotisée Faursa.

Mots clés

Robots mobiles, navigation autonome, environnement inconnu, planification de mouvements

Dédicace

Je dédie ce travail à la mémoire de ma mère.

Remerciements

En préambule à ce mémoire, je souhaitais adresser mes remerciements les plus sincères aux personnes qui m'ont apporté leur aide et qui ont contribué à l'élaboration de ce travail.

Je tiens à remercier plus particulièrement **Dr. Sara Bouraine** qui, en tant qu'encadreur, s'est toujours montrée à l'écoute. Ses conseils et le temps qu'elle a bien voulu me consacrer m'ont permis de travailler dans les meilleurs conditions possibles.

Je tiens aussi à remercier mon co-encadreur **Dr. cherif LARBES**, pour sa présence, disponibilité, orientations et conseils durant tout mon cursus.

Je remercie également tous les membres de l'équipe NCRM du CDTA pour leur acceuil .

Aussi, je remercie chaleureusement ma famille et tous mes amis pour leurs soutiens et leurs encouragements.

Enfin, je tiens également à remercier Mr. Mohamed Amine Kendi et toute personne ayant participé de près ou de loin à la réalisation de ce travail.

Table des matières

Table des figures

Liste des abréviations

CDTA : Centre de Developpement des Technologies Avancées

GPS : Global Positioning System

ROS : Robot Operating System

NCRM : Équipe de Navigation et Contrôle des Robots Mobiles autonomes au sein de CDTA

CIR : Centre Instantané de Rotation

RRT : Rapidly-exploring Random Trees

ERRT : Extand Rapidly-exploring Random Trees

SRT : Sampling-Based Roadmap of Trees

PRT : Probabilistic Roadmap of Trees

PMP : Planification de mouvement partiel

ICS : Inevitable Collision States

ISS : Input Space Sampling

SSS : Space State Sampling

VO : Velocity Obstacles

PRM : Probalistic Road Map

DW : Dynamic Window

MPC : Model Predictive Control

Introduction générale

> "Les machines un jour pourront résoudre tous les problèmes, mais jamais aucune d'entre elles ne pourra en poser un !"
>
> *Albert Einstein*

Généralement, l'image que l'on se donne d'un robot est celle d'un système mécanique articulé devant effectuer des opérations prédéfinies.
Pourtant l'évolution de la robotique a permis d'étendre les secteurs d'application de ces derniers afin de pouvoir répondre aux exigences futures de l'être humain.
Différents travaux de recherches ont été élaborés pour réaliser des robots mobiles pouvant être exploités dans différents secteurs comme l'industrie manufacturière, le secteur médical (chirurgie de précision, robots infirmiers, etc.), le secteur militaire (sauvetage, surveillance, robots démineurs, drones, etc.), l'exploration spatiale, sous-marine et le transport, etc.

Prenons l'exemple d'un système d'aide à la mobilité de personnes âgées ou handicapées, à savoir, le domaine d'intérêt traité dans ce travail, afin que le robot puisse transporter une personne à partir d'un point de départ jusqu'à un point d'arrivé donné dans un environnement réel, il est souvent nécessaire de disposer d'une carte de son environnement et un planificateur de trajectoires pour lui permettre de calculer les déplacements requis pour atteindre un but précis tout en évitant les obstacles rencontrés sur le passage et en utilisant uniquement l'information issue de ses capteurs embarqués.

Ce problème a longtemps été un challenge pour les chercheurs, mais aujourd'hui l'avancement rapide vers le développement de nouvelles techniques de perception et planification de mouvement montre la capacité de ces dernières à résoudre divers problèmes.
Pour ce faire, plusieurs approches de planification de mouvements ont été proposées pour résoudre ce problème, à savoir les méthodes délibératives [5, 6], les méthodes réactives [7–9] mais aussi les approches de planification réactives [3, 10, 11].
Ainsi, afin que le robot puisse permettre à l'utilisateur d'atteindre un but dans un environnement inconnu encombré d'obstacles, il faut proposer une méthode de planification qui prend en compte les contraintes d'un tel environnement.
C'est pourquoi, dans ce travail, nous avons développé une approche de navigation NAV_ ISS basée sur la méthode de planification par échantillonnage de l'espace de contrôles (en anglais : Input Space Sampling (ISS)).

Ce mémoire est divisé en deux parties. La première composée des deux premiers chapitres ; le premier chapitre expose un état de l'art des différentes méthodes de modélisation de l'environnement ainsi que les systèmes de perception existants. Le deuxième chapitre présente un bref état de l'art des approches de navigation d'un robot mobile dans un environnement inconnu.

La deuxième partie expose l'approche de navigation proposée ainsi que son implémentation. Dans le chapitre 3, nous expliquerons en détail la méthode de modélisation de l'environnement (à partir d'un capteur laser embarqué) ainsi que le principe général de l'approche de planification de mouvements appliquée sur une chaise roulante robotisée.

Le dernier chapitre complète celui qui le précède en exposant les résultats de l'implémentation de l'approche de navigation choisie allant de la cartographie de l'environnement jusqu'à l'application finale c'est à dire l'étape de planification.

Première partie

Etat de l'art

Chapitre 1

Modélisation de l'environnement

1.1 Introduction

Durant ces dernières années, la robotique mobile a connu un essort surprenant lui permettant ainsi le développement de systèmes pouvant évoluer dans des environnements réels avec une intelligence qui rend la capacité du robot proche du raisonnement humain.Cependant, le robot a besoin d'une analyse cohérente de son environnement afin de pouvoir se mouvoir et y évoluer en toute sécurité, cela peut être réalisé grâce à la perception qui désigne la capacité du système à recueillir, traiter et modéliser des informations utiles au robot pour agir et réagir.
Afin de permettre à un robot de se déplacer de manière autonome, celui-ci doit d'une part construire une carte fiable de son environnement grace aux différents systèmes de perception existants.

Dans ce chapitre, nous allons présenter un bref état de l'art sur la perception et les différentes méthodes de cartographie afin de mieux se situer dans le contexte du travail de notre projet.

1.2 Perception

La perception est une notion en robotique mobile qui consiste globalement à recueillir les informations sensorielles dans le but d'acquérir une connaissance et une compréhension de l'environnement d'évolution du robot, qui peut être structuré, non structuré ou semi-structuré.
C'est pourquoi, la perception est préalable et indispensable à la chaine fonctionnelle d'un système de navigation (voir la figure 1.1).

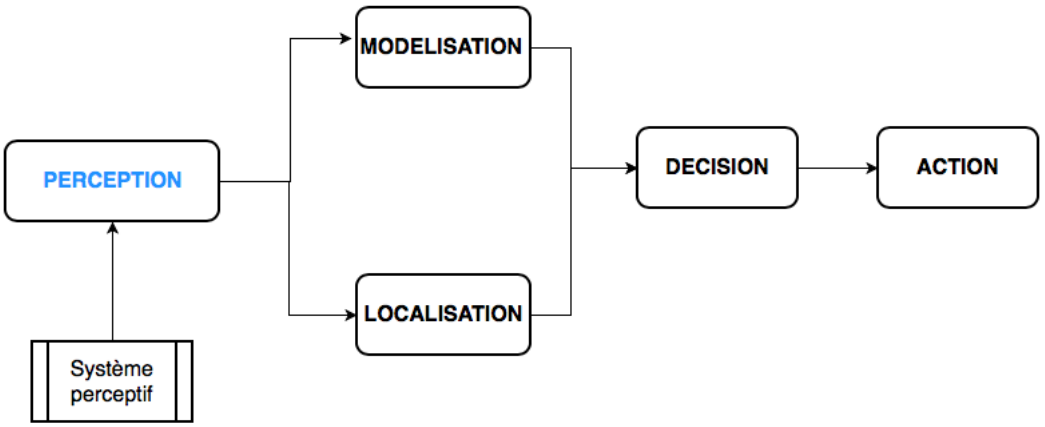

FIGURE 1.1: Chaine fonctionnelle d'un système de navigation.

1.2.1 Les systèmes de perception

Le choix d'un système de perception est souvent dépendant de l'application envisagée. C'est pourquoi, la classification des capteurs est généralement faite par rapport à deux familles :

— **Les capteurs proprioceptifs** qui fournissent des informations propres aux comportement interne du robot, i.e. déterminer son état à un instant donné.

— **Les capteurs extéroceptifs** qui fournissent des informations sur le monde extérieur au robot.

1.2.1.1 Les capteurs proprioceptifs

Ces capteurs fournissent, par intégration, des informations élémentaires sur les paramétres cinématiques ou dynamique du robot. Les informations sensorielles gérées dans ce cadre sont généralement des vitesses, des accélérations, des angles de giration, des angles d'altitude [12]. Les capteurs proprioceptifs peuvent étre regroupés en deux familles [13].

— **Les capteurs de déplacement** qui comprennent les odomètres, les accéléromètres, les radars Doppler, les mesures optiques,etc. Cette catégorie

permet de mesurer des déplacements élémentaires, des variations de vitesse ou d'accélération sur des trajectoires rectilignes ou curvilignes.

— **Les capteurs d'altitude** qui mesurent deux types de données : les angles de cap et les angles de roulis et de tangage. Ils sont principalement constitués par les gyroscopes, les gyromètres, les capteurs inertiels composites, les inclinomètres, les magnétomètres,etc. Ces capteurs sont en majorité de type inertiel.

1.2.1.2 Les capteurs extéroceptifs

Les capteurs extéroceptifs [14, 15] sont employés en robotique mobile pour collecter des informations sur l'environnement d'évolution du système mobile. Ils sont le complément indispensable aux capteurs proprioceptifs présentés précédemment. Ils sont utilisées pour conditionner et traiter les informations sensorielles. Ils sont notamment utilisés dans les domaines d'application tels que l'évitement d'obstacle, la localisation, la navigation, la modélisation d'environnement, etc. Les principaux capteurs utilisés en robotique mobile sont : les capteurs télémétriques (les ultrasons, les lasers et les infrarouges,etc.), le GPS (Global Positionning System), les caméras, etc.

1.3 Modélisation de l'environnement

La modélisation de l'environnement est la phase qui permet la construction d'une carte reflétant la structure spatiale de l'environnement à partir des différentes informations recueillies par ses capteurs et l'historique des positions réelles du robot.

Les méthodes de modélisation sont classées en deux grandes familles :

— Les méthodes de modélisation métriques.

— Les méthodes de modélisation topologiques.

1.3.1 Les méthodes métriques

Elles décrivent explicitement la position géométrique des éléments de l'environnement. Cette cartographie peut être subdivisée en deux sous familles :

— **Les méthodes de modélisation géométrique** : qui gèrent explicitement les positions « cartésiennes » des primitives cartographiques.

— **Les méthodes de modélisation par grilles d'occupation** : qui décrivent les propriétés métriques par discrétisation de l'environnement en y ajoutant des informations d'incertitude.

1.3.1.1 Méthode des grilles d'occupation

Proposées par Elfe [16, 17], ces méthodes consistent à présenter les mesures acquises par le capteur sous forme d'un ensemble de cellules. Ces dernières peuvent posséder plusieurs attributs exprimant chacun une propriété de la zone correspondante, notamment son occupation. Pour des scènes d'intérieur, ou le sol est plat, la plupart des chercheurs décrivent l'état de chaque cellule par un attribut unique qui est la probabilité d'occupation. On parle alors de grille d'occupation. Une cellule est dite libre si sa probabilité d'occupation est inférieure à un seuil choisi (prédéfini) et occupée dans le cas inverse.
Au fur et à mesure que le robot se déplace, de nouvelles mesures sont intégrées dans ces cartes affinant ainsi les connaissances des zones déjà explorées, et découvrant ainsi les autres, ainsi la grille est régulièrement mise à jour (voir la figure 1.2).

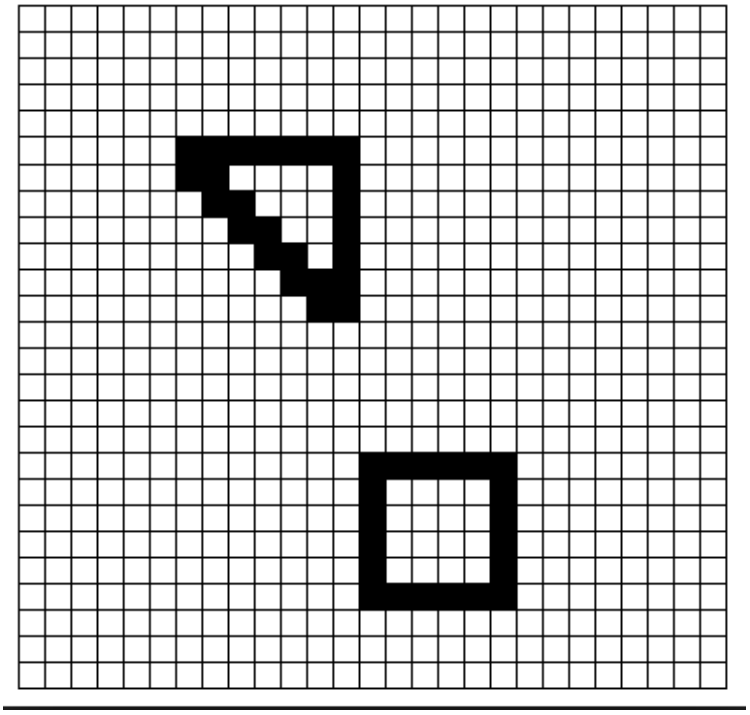

FIGURE 1.2: Exemple de grille d'occupation tel que les cellules blanches représentent les espaces libre tant dis que les cellules noir représentent ceux occupés.

En terme général, cette approche consiste en la modélisation d'une grille d'occupation qui caractérise l'environnement par un ensemble de sous-régions, appelées cellules. Les différentes informations concernant les cellules de la carte sont stockées dans une matrice représentative de la grille.

La taille de la carte est définie par la taille de la matrice. Chaque élément de la matrice porte différentes informations sur la cellule correspondante de l'espace physique. Ces informations sont exprimées à travers une valeur certaine(i.e. entre 0 et 1) de la présence d'obstacles dans la sous-régions correspondante (voir la figure 1.3).

La mise à jour de la carte se fait au cours du déplacement du robot dans son environnement grâce aux dernières mesures prises par le laser qui seront modélisées en une position dans la matrice relative à la position du robot par la suite du processus.

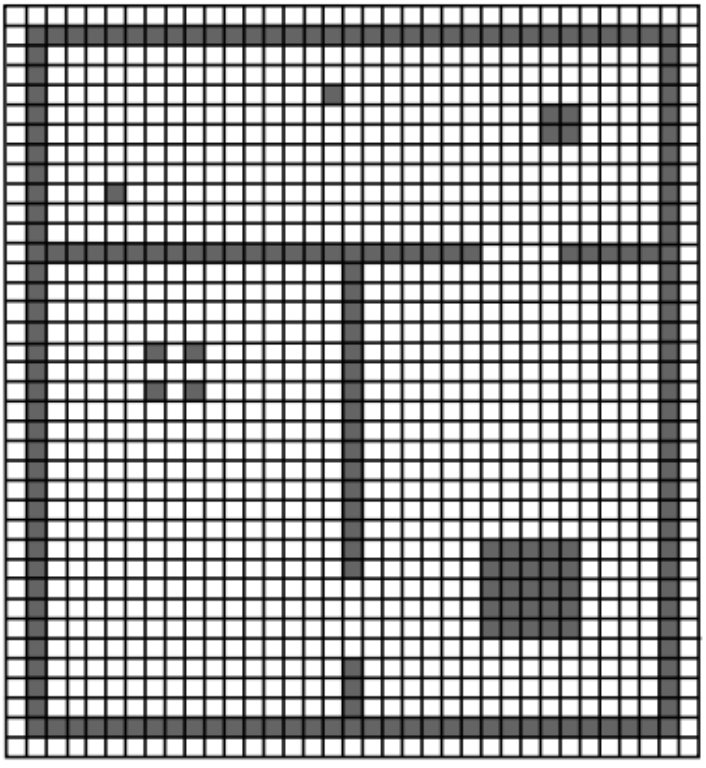

FIGURE 1.3: Exemple de grille d'occupation binaire. Les cellules blanches correspondent à des zones de l'environnement ne contenant aucun obstacle, les cellules grises correspondent à des zones occupées par des obstacles [1].

1.3.1.2 Modèle géométrique

Le principe de ce modèle est d'utiliser une liste ou un vecteur de tous les objets de l'environnement, cette liste contient des informations sur les objets tel que leurs nature, position et orientation. Il s'agit d'une présentation cartésienne de l'environnement [12] (un exemple d'une carte géométrique est illustré dans la figure 1.4)

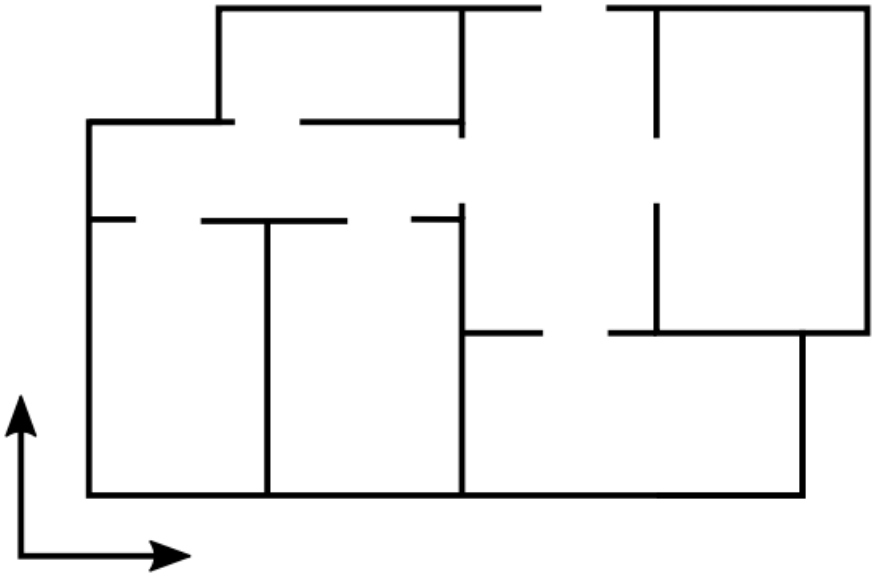

FIGURE 1.4: Exemple de carte géométrique [1].

Ainsi, ces modèles permettent de représenter les mesures sous forme de primitives géométriques telles que des points, des segments de droite (voir la figure 1.4), des rectangles, etc. Généralement, ces modèles sont obtenus soit à partir des mesures de capteurs télémétriques ou d'un capteur de vision.

Par télémétrie

Il existe différents filtrages et regroupements de points de mesure. Par exemple la méthode proposée par [18] consiste à prendre les mesures d'un capteur à ultrasons tournant une par une, et à créer un segment dès que trois mesures sont suffisamment proches et alignées. Pour qu'un nouveau point puisse appartenir à un segment existant, il faut que la distance du point au segment soit plus petite que l'incertitude sur la mesure.

Par vision

Pour obtenir une représentation géométrique des éléments de l'environnement à partir d'un capteur de vision il faut effectuer un prétraitement de l'image établie par le capteur. Ce traitement consiste à extraire les contours des objets présents dans l'image grace à un filtre de Sobel par exemple utilisé dans [19] ou encore un filtre de Deriche utilisé dans[20].

1.3.2 Les méthodes topologiques

La modélisation de l'environnement est basée sur des graphes représentant des informations de plus haut niveau comme certaines places caractéristiques de l'environnement (coins, croissement de deux couloirs, jonction en T, etc.) [21]. Les nœuds du graphe correspondent à des lieux, i.e. des positions que le robot peut atteindre. Les arêtes liant les nœuds marquent la possibilité pour le robot de passer directement d'un lieu à un autre et mémorisent en général la manière de réaliser ce passage. (voir la figure 1.2)

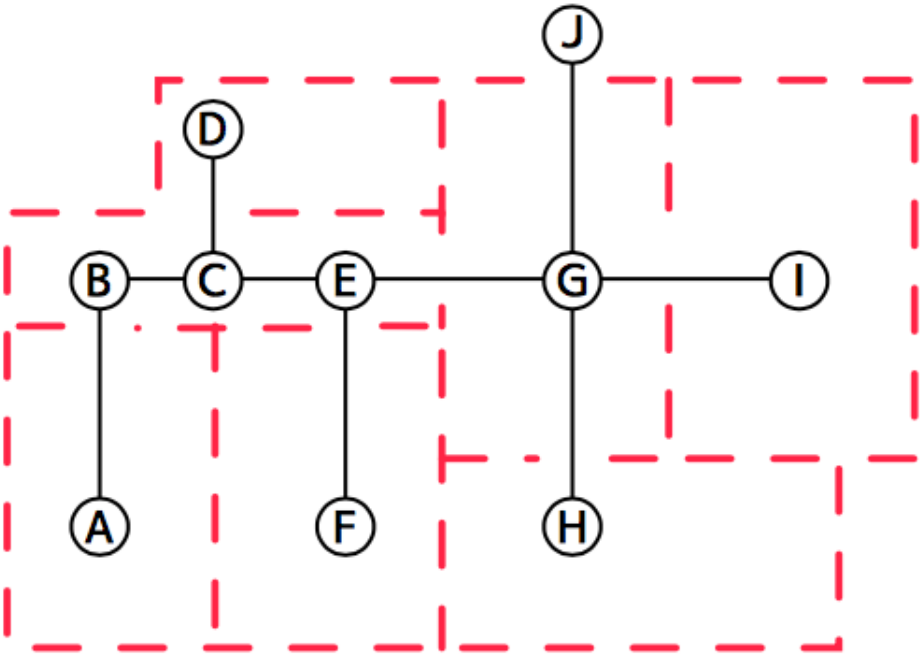

FIGURE 1.5: Exemple de carte topologique [1].

Les données mémorisées dans les arêtes du graphe sur les relations de voisinage entre les lieux proviennent, pour leur part, des données proprioceptives. Cela est une caractéristique des cartes topologiques, dans lesquelles les perceptions ne sont en général pas utilisées pour estimer les positions relatives des lieux visités, mais seulement pour reconnaitre un lieu. Ces données peuvent être des informations sur les positions relatives des nœuds, ou des informations sur les actions à effectuer pour parcourir cette arête.

1.3.3 Discussion

De manière générale, les différentes méthodes de modélisation de l'environnement se résume à travers les méthodes par cartes topologiques et les méthodes métriques. Ces dernières présentent certains avantages et inconvénients. Prénons comme exemple les méthodes par cartes topologiques, l'avantage important de celle-ci est qu'elles ne requièrent pas de modèle métrique des capteurs pour fusionner les données proprioceptives et les perceptions au sein d'une représentation unifiée de l'environnement. Cela est avantageux car le fait de ne pas fusionner les deux sources d'informations, permet de séparer les influences des erreurs correspondantes.
Cependant, cette méthode présente aussi plusieurs inconvénients, tel qu'on la déjà mentionné l'utilisation direct des perceptions sans modèle métrique empêche d'estimer ces données pour des positions non visitées. En conséquence, les cartes topologiques nécessitent en général une exploration très complète de l'environnement pour le représenter avec précision. Cela peut être un inconvénient dans le cas où les lieux présentés ont une grande densité spatiale car l'exploration complète de l'environnement demandera un temps important. De plus la reconnaissance des lieux peut être très difficile dans le cas de capteurs très bruités, ou d'environnements très dynamiques.

Quant aux méthodes de modélisation métriques, elles intègrent la notion de distance au sein de la carte. L'avantage principal de ces cartes est de permettre de présenter l'ensemble de l'environnement, et non un petit sous-ensemble de lieux comme le font les cartes topologiques. Cette présentation complète ne se limite pas aux positions physiquement explorées, mais s'étend à toutes les zones que le robot a pu percevoir

depuis les lieux qu'il a visité. Cette présentation complète permet ainsi d'estimer avec précision et de manière continue la position du robot sur l'ensemble de son environnement.
N'oublions pas que les méthodes métriques peuvent être subdivisées en deux sous familles, à savoir,les méthodes de modélisation par grilles d'occupation et les méthodes de modélisation géométrique.
L'inconvénient majeur des grilles d'occupation réside sur le fait que si la carte demande une grande résolution à travers une grande surface, la matrice devient grande et par conséquent, des problèmes apparaitront dans le cas où on utilisera un système avec mémoire ou capacité de calcul restreinte. De plus la finesse de la discrétisation étant prédéfinie, les grilles d'occupation ne sont pas capables de s'adapter automatiquement à la densité ou à la taille des obstacles. Cela peut s'avérer un problème pour certains algorithmes de navigation mais il existe des solutions tel qu'agrandir l'espace de configuration des obstacles.
Cependant, les avantages de cette représentation est que la grille d'occupation permet de représenter de grandes densités d'informations et est adaptée à des environnements de forme quelconque. De plus leurs méthodes de constructions sont plutôt économiques en ressources de calcul : leur mise à jour s'avère aisée et rapide en plus d'être facile à implémenter et extrêmement robuste selon Thrun et Brenstein [22].

Contrairement à la carte géométrique construite, elle peut être utilisée par le robot pour sa localisation dans l'environnement. Cette méthode de modélisation est largement utilisée dans les environnements d'intérieur structurés.
Cette solution est avantageuse pour sa grande résolution en plus du fait qu'elle ne nécessite pas une grande capacité de mémoire, la position des objets peut être stockée avec une grande précision. Néanmoins, cette méthode présent un inconvénient lors de la détection de primitives, car une bonne détection reste une tâche difficile pour un système autonome, même si des primitives simples peuvent être détectées, il est difficile d'en faire la distinction. Aussi, cette méthode reste compliquée quant à la manière dont les primitives dans la carte sont connectées et comment le robot se déplace par rapport à ces dernières.

Enfin, la méthode géométrique est plus difficile à implémenter que la modélisation par grille d'occupation, car cette dernière a pour avantage d'utiliser directement les valeurs des capteurs de distance afin de mettre à jour les cellules de la grille. Elle permet donc de supprimer la phase d'extraction d'objets qui est souvent couteuse en temps de calcul et soumise à une forte incertitude.

1.4 Conclusion

Dans ce chapitre, nous avons exposé les concepts de base relatives à la modélisation de l'environnement qui nécessite de s'intéresser obligatoirement aux éléments de la chaine de perception.
Aussi, cette étude nous a permis de faire un état de l'art des différentes méthodes de cartographie de l'environnement, chacune ayant des avantages et inconvénients.
Pour notre travail, notre choix s'est porté sur la modélisation par grille d'occupation, du moment que cette méthode est assez facile et robuste à implémenter, mais aussi elle peut s'adapter à différents types d'environnements (i.e. structurés ou non structurés).

Ainsi, les résultats de la modélisation de l'environnement seront utilisés dans le module de navigation dont nous allons présenter dans le chapitre suivant les différentes approches de navigation et celle qui présente le cœur de notre travail, à savoir la planification de mouvements pour des robots mobiles.

Chapitre 2

Navigation d'un robot mobile dans un environnement inconnu

2.1 Introduction

En robotique mobile, une méthode de navigation consiste à générer et exécuter une stratégie de mouvement qui permet de conduire le robot vers un but prédéfini, tout en évitant les obstacles présents dans l'environnement. Néanmoins, cette tâche n'est pas chose aisée car la navigation d'un robot mobile présente plusieurs problèmes à savoir la localisation, la modélisation de l'environnement, le contrôle du mouvement, l'évitement d'obstacle, la planification de mouvement... etc.
Afin de remédier à ces contraintes, un grand nombre de travaux de recherche sur la navigation de robot mobile a été proposé au cours des années [21, 23, 24].

Dans ce chapitre, nous allons nous intéresser sur le problème de la planification de mouvement en présentant un état de l'art sur les différentes méthodes de navigation existantes afin d'évaluer ces dernières par rapport à plusieurs critères en outre leur applicabilité dans des environnements inconnus ou le robot a une vision partielle de son environnement.

2.2 Approches de navigation

Plusieurs approches ont été proposées dans la littérature pour résoudre le problème de la navigation d'un robot mobile dans des environnements statiques ou dynamiques. Ce problème peut être traité à partir de plusieurs points de vue, mais généralement, nous distinguons deux grandes catégories d'approches :

- **les approches délibératives** : qui représentent les approches de planification de mouvement.
- **les approches réactives** : qui représentent les approches d'évitement d'obstacles.

2.2.1 Les approches délibératives

Les approches délibératives ou globales consistent à résoudre un problème de planification de mouvement, connu également comme "le problème du déménageur de piano". Cela revient à calculer un chemin complet (une séquence de configurations)

à partir d'une configuration initiale jusqu'à la configuration du but à atteindre, en se basant sur la représentation de l'environnement d'évolution (qui est généralement a priori connue ou construite pendant une première phase d'exploration) [5, 6].

2.2.1.1 Les méthodes par arbres

Cette méthode se base sur la construction d'un arbre d'exploration de l'espace, à partir de la configuration initiale du système. L'une des méthodes les plus répandues est la méthode RRT (Rapidly exploring Random Trees). C'est une technique qui consiste en la construction incrémentale de l'arbre d'exploration de manière à réduire rapidement la distance entre une configuration choisie aléatoirement et l'arbre. La construction est réalisée en démarrant d'un arbre initial (voir figure 2.1). Une configuration aléatoire $q_{aleatoire}$ est choisie ensuite à partir de l'espace de recherche, cette dernière est combiné avec la configuration la plus proche q_{proche} d'elle pour calculer une nouvelle configuration du système $q_{nouveau}$ dans la direction liant $q_{aleatoire}$ à q_{proche}. Le processus est répété jusqu'à l'obtention du chemin liant la configuration initiale à la configuration finale.

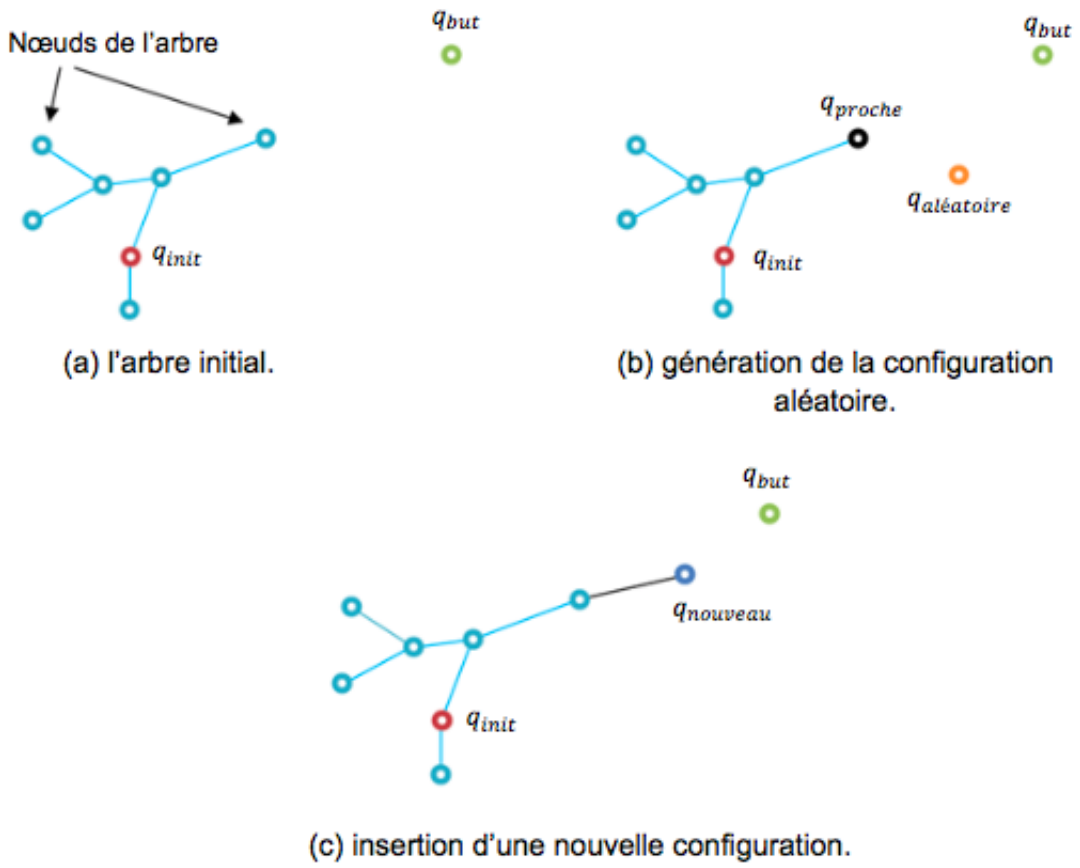

FIGURE 2.1: Principe de la diffusion de l'arbre dans la technique RRT.

2.2.1.2 Les méthodes par graphes

Le principe de base des méthodes par graphes se résume en la construction d'un graphe représentatif d'un réseau de courbes ou lignes de dimension 1.
Ces méthodes procèdent en deux étapes :

- Construction du graphe dans l'espace de recherche. Quand le système d'a aucune connaissance à priori de son environnement, il utilise ses capteurs pour collecter les différentes informations et de manière incrémentale construire ce graphe.
- Parcourir ce graphe afin de déterminer un chemin qui relie la position initiale et la position but.

Un algorithme de recherche graphique comme A* [5] ou Dijkstra [6] est utilisé pour trouver le chemin qui connecte ces deux positions. L'utilisation de ces algorithmes évite une exploration complète de l'espace de recherche.
La plupart des méthodes agissent de la sorte, ex. la décomposition cellulaire, les cartes de routes probabilistes.

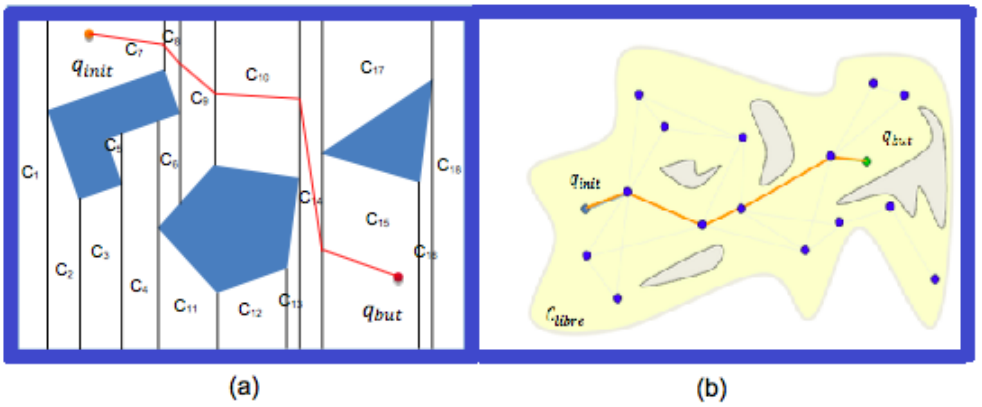

FIGURE 2.2: Calcul du chemin entre q_{init} et q_{but} représenté en rouge) par les méthodes par graphes : (a) la décomposition cellulaire, (b) les cartes de routes probabilistes.

2.2.2 Les approches réactives

Contrairement aux approches délibératives qui calculent la trajectoire complète du robot dans un environnement connu ou déjà exploré dans une première phase, les approches réactives (connues comme approches d'évitement d'obstacles) agies en utilisant les capteurs embarqués dans le système, le contrôle à appliquer sur les actionneurs est calculé à pas de temps. Ce type d'approches est mieux adaptée pour des applications en temps réel. Dans la suite on présentera quelques méthodes utilisant cette approche [7–9].

2.2.2.1 Les algorithmes Bug

Les algorithmes **Bug 1** et **Bug 2** sont parmi les plus anciens et les plus simples, ils supposent que la robot est un point dans un plan 2D (espace de navigation) et utilise un capteur tactile pour la détection d'obstacles. L'application se fait selon deux comportements : "déplacement vers le but" et "suivi d'une limite".

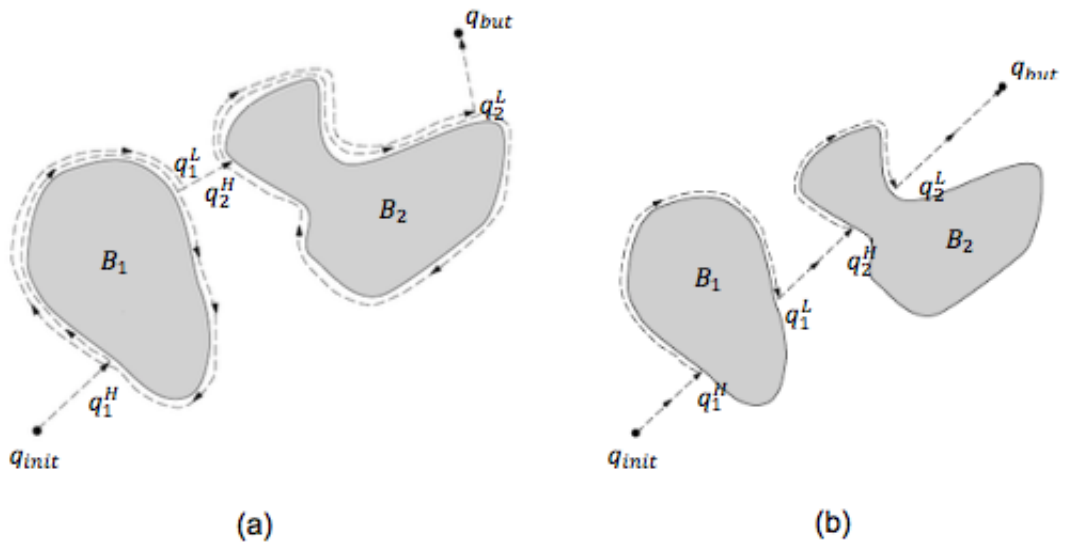

FIGURE 2.3: Calcul du chemin du robot par les algorithmes Bug : (a) Bug1, (b) Bug2. Plus de détails sont disponibles dans [20].

Dans l'algorithme **Bug 1**, lors du « déplacement vers le but », le robot se déplace tout droit vers le but (q_{but}). Dès qu'il détecte un obstacle, il exploite son comportement "suivi d'une limite". Le robot fait le tour de l'obstacle jusqu'à revenir à sa position, ceci lui permet de déterminer le point le plus proche du but pour ce déplacer vers lui. A partir de ce point il reprend son comportement « déplacement vers le but ». Le processus est répété jusqu'à atteindre q_{but} .

A l'inverse de **Bug1** qui utilise une recherche exhaustive, **Bug2** utilise une recherche dite gloutonne (greedy). Le robot se déplace en ligne droite vers le but durant le comportement "déplacement vers le but" comme pour Bug1, sauf que dans le cas de Bug2, les droites tracés par ce comportement sont toutes alignés, ceci est dû au comportement "suivi d'une limite". Durant l'exécution de ce dernier, Le robot fait le tour de l'obstacle, jusqu'à ce qu'il atteigne un nouveau point qui appartient à la droite liant la position initiale au but ou il reprend son comportement "déplacement vers le but". On répète le processus jusqu'à atteindre q_{but}.

Bien que ces algorithmes sont simples, aucune contrainte sur le robot n'est considérée (qu'elle soit géométrique, cinématique ou dynamique). Mais leur plus grand inconvénient est que, la sûreté du mouvement ne peut être garantie car le robot doit

se déplacer à proximité des obstacles. En plus ils se limitent à des espaces de configuration de dimension 2.
Un autre inconvénient de ces algorithmes est qu'ils sont destinés pour des environnements statiques.

2.2.2.2 Approche des champs de potentiel (PF)

C'est une approche qui a été proposé initialement pour le calcul du mouvement d'un bras manipulateur [25] . Le principe de cette méthode est de construire une fonction de potentiel (ou d'énergie) artificiel dans l'espace dont le gradient (la force) définit un champ de vecteurs conduisant le robot vers son but, ce dernier agit comme une force attractive alors que les obstacles agissent comme des forces répulsives. La combinaison de ces forces guide le robot vers la position q_{but} tout en évitant les obstacles initialement connus.

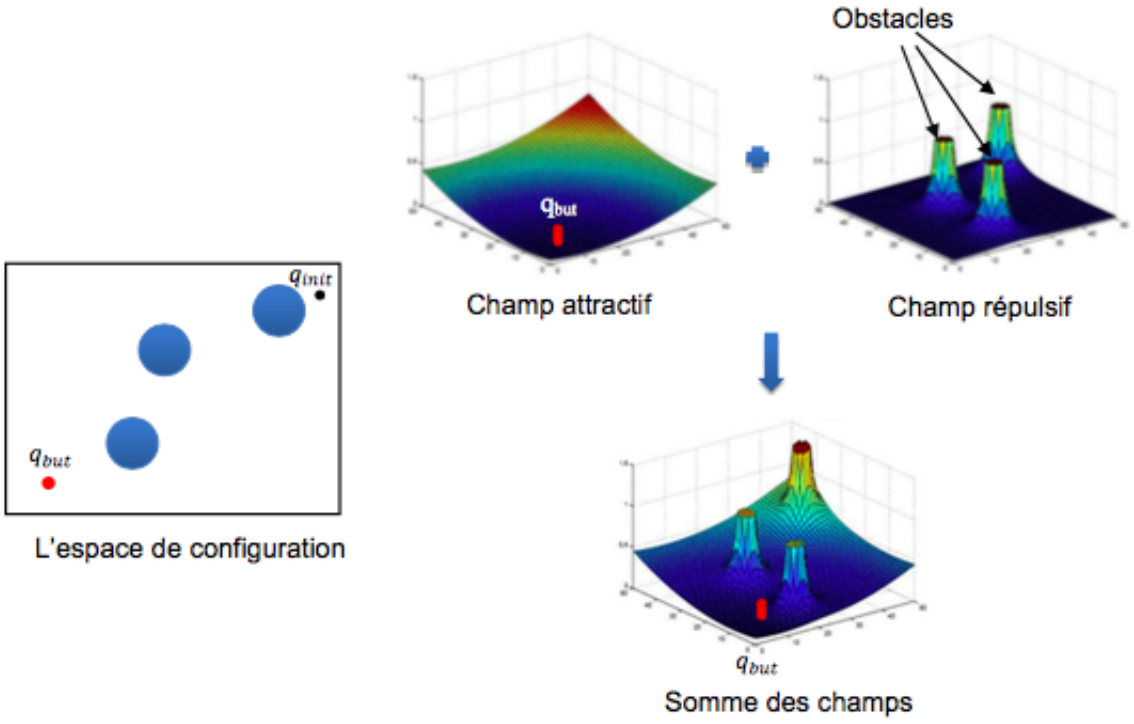

FIGURE 2.4: Champs de potentiel pour un environnement contenant trois obstacles.

Un avantage important de cette approche est le fait qu'elle s'applique à des espaces de configuration multidimensionnels, et de ce fait, elle génère une plus grande variété de chemins. En contrepartie, les contraintes mécaniques et cinématiques du robot ne

sont pas considérés, du coup, le robot aura du mal à suivre le chemin calculé.
De nombreuses améliorations pour cette approche ont été proposées dont certains travaux qui ont étendu le champ d'application vers les environnements dynamiques ou on prend en considération le mouvement des obstacles dans l'espace pour calculer le champ répulsif.

2.2.2.3 La fenêtre dynamique (DW)

Proposée par Burgardand et Thrun, cette approche opère dans l'espace des vitesses du robot, permettant la prise en considération les contraintes cinématiques et dynamiques en limitant l'espace de recherche uniquement à un espace de vitesses atteignables. A partir d'une perception locale de l'environnement à travers les capteurs, un couple vitesse linéaire v et vitesse angulaire w est générée par l'algorithme permettant de commander le robot et d'éviter les obstacles. L'application de ce couple génère un chemin circulaire ou les différentes contraintes kinodynamiques sont évaluées. DW sélectionne le couple de vitesses le plus pertinent (vitesses admissibles). La zone de recherche des vitesses est limitée par une fenêtre désignant les limitations dynamiques du robot et les vitesses admissibles qui peuvent être atteintes pendant un intervalle de temps donné (voir la figure 2.5).

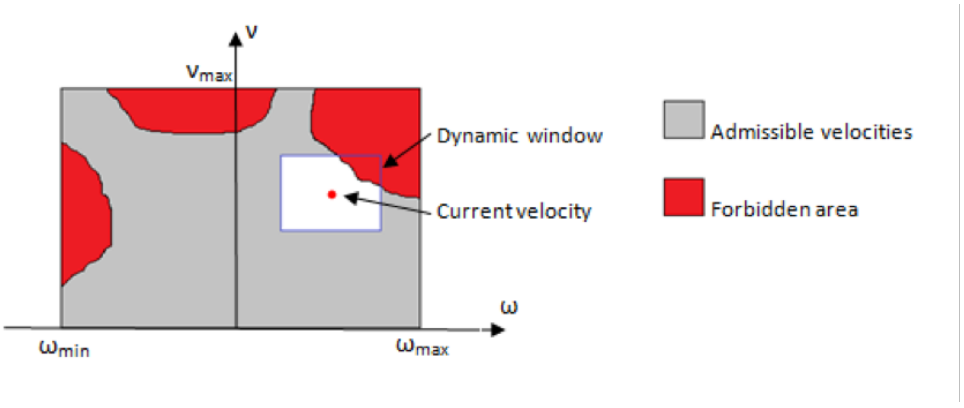

FIGURE 2.5: L'approche de la fenêtre dynamique. L'espace des vitesses est divisé en régions admissibles et interdites. DW est représentée par le rectangle bleu contenant les vitesses atteignables par le robot durant un intervalle de temps spécifique [2].

La méthode DW est adaptée pour des environnements statiques ou à la rigueur changeant (ex : cas d'une porte, un obstacle est ajouté à l'environnement). D'autres

travaux ont permis d'étendre le champ d'application de cette méthode vers les environnements dynamiques comme la TVDW (Time-Varying Dynamic Window) [7]qui calcule un ensemble de trajectoire pouvant être suivies par les obstacles dans le futur pour effectuer une vérification de collision à court terme.

2.2.2.4 Représentation des obstacles dans l'espace des vitesses (VO)

L'approche "velocity obstacles" (VO) a été proposée par Fiorini et Shiller [8] afin de prendre en compte la dynamique de l'environnement et celle du robot. De plus, le comportement futur des obstacles mobiles est considéré en supposant que l'obstacle maintient une vitesse linéaire constante. L'information de vitesse est directement utilisée pour déterminer une collision potentielle. C'est pourquoi, VO représente l'ensemble des vitesses qui mènent le robot vers une collision avec les obstacles voisins. Les obstacles statiques et dynamiques sont modélisés dans l'espace des vitesses du robot A.
Chaque point de VO représente un vecteur de vitesse dont l'origine est le point correspondant à A. Toute vitesse qui pénètre dans VO est une vitesse qui va conduire le robot vers une collision avec 'B' à un certain point dans le futur. Avec une telle représentation, une manœuvre d'évitement peut être facilement calculée en sélectionnant une vitesse à l'extérieur de VO (voir figure 2.6).

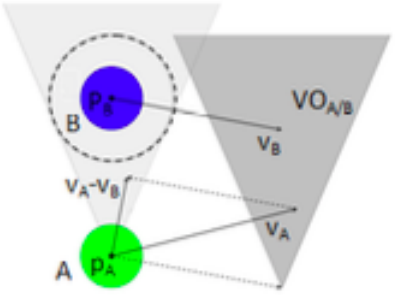

FIGURE 2.6: L'approche VO [2].

Cependant, la méthode VO connait certaines limitations, par exemple, dans un environnement fermé, chaque vitesse est interdite du moment qu'elle conduit le robot vers une collision. Cette méthode pose également l'hypothèse que le robot et l'obstacle

doivent être de forme circulaire (disques). De plus, dans le cas où l'obstacle suit une trajectoire arbitraire (avec des vitesses variables), VO ne peut plus être appliquée. C'est pour remédier à ce dernier problème que la méthode VO non linéaire (NLVO) (non linear velocity obstacles) a été développée [9].

2.2.3 Planification réactive

Pour pallier aux inconvénients de chacunes des méthodes réactives et délibératives et pour bénéficier des avantages de chacune que les méthodes de planification réactive ont été proposées [3, 10, 11].

2.2.3.1 La planification de mouvement partiel (PMP)

La méthode de planification de mouvement partiel (en anglais : Partial Motion Planning) (PMP) a été proposée [10, 11, 26] pour prendre en compte explicitement de la contrainte temps réel. Le fonctionnement de la méthode PMP se résume sur le fait qu'elle doit agir et rendre une décision pendant un temps limité (temps de décision) qui dépend de l'état actuel de l'environnement. C'est pourquoi elle est très adaptée pour faire naviguer le robot dans des environnements dynamiques. Cette méthode est classée comme méthode de planification réactive car elle agit pendant un cycle de temps limité, durant lequel elle calcule une trajectoire qui se rapproche le plus possible du but. L'algorithme PMP opère selon trois étapes, répétées périodiquement pour chaque cycle PMP :

1. Mise à jour du modèle de l'environnement à partir de l'information issue des capteurs embarqués du robot.
2. Calcul d'une trajectoire partielle sans collision vers le but. Cette trajectoire devra être exécuter pendant le cycle de temps suivant.
3. A la fin du cycle, la meilleure trajectoire partielle (i.e. celle qui optimise une fonction de cout donnée) est choisie. Elle est alors exécutée pendant le cycle suivant. (Notons que cette trajectoire n'atteint pas forcément le but.)

Les étapes de l'algorithme PMP sont traduis à travers la figure 2.8

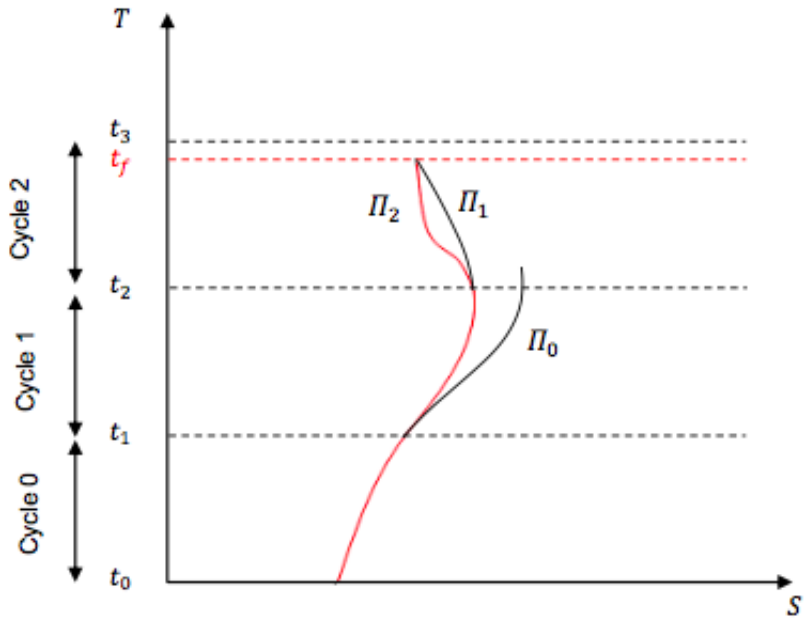

FIGURE 2.7: Planification de mouvement partiel : la trajectoire finale (i.e. ensemble de trajectoires partielles) exécutée par le robot est représentée en rouge. Durant le cycle 0, une partie de la trajectoire partielle $\prod_0$ est exécutée et la trajectoire partielle $\prod_1$ est planifiée. Durant le cycle 1, une partie de la trajectoire partielle $\prod_1$ est exécutée et la trajectoire partielle $\prod_2$ est planifiée. Enfin, durant le cycle 3, la trajectoire partielle $\prod_2$ est exécutée, et le robot atteint son but (à l'instant t_f)

L'inconvénient majeur des travaux basés sur la méthode de planification PMP est que le comportement futur des obstacles mobiles est généralement supposé connu (ex. [10, 11]) ou prédit à court-terme (ex. [27]). Dans ce cas, la sûreté de mouvement ne peut être garantie car aucune garantie sur le futur ne peut être fournie.

2.2.3.2 Les techniques par échantillonnage

Les techniques par échantillonnage sont le plus souvent utilisées dans une navigation hiérarchique, c'est à dire lorsque le but est hors de la portée des capteurs.
Ces techniques sont des planificateurs locaux permettant une planification en temps réel dans des environnements partiellement connus ou inconnus.
Le principe de ces techniques sont généralement basées sur la génération et le tri de mouvements candidats (génération de contrôles possibles).
Cette méthode présente deux types de techniques :

— Les techniques par échantillonnage de l'espace d'entrées (ISS) (en anglais : input space sampling)

— Les techniques par échantillonnage de l'espace d'états (SSS) (en anglais : state space sampling).

Échantillonnage de l'espace d'entrées (ISS).

Le principe de l'échantillonnage de l'espace d'entrées est simple ; à partir d'un état initial, le modèle prédictif du système est utilisé pour générer un ensemble de trajectoires de contrôle (leur forme dépend donc du modèle du système et de son état initial (courbures, vitesses)) (voir figure 2.9).
Ces trajectoires peuvent être triées par rapport à une fonction de coût donnée. De plus, elles peuvent être testées par rapport à la non collision avec les obstacles.

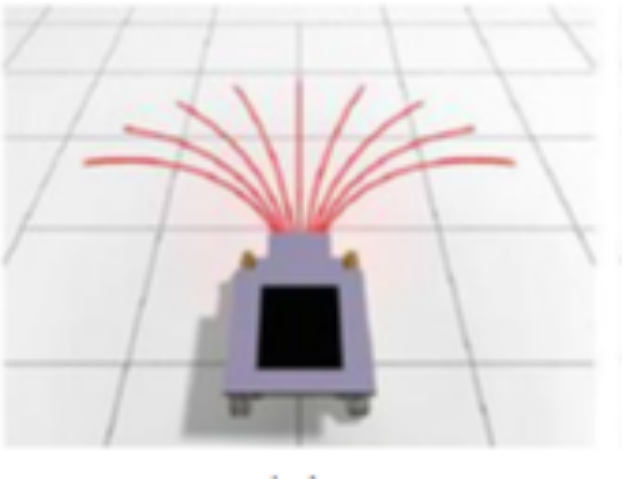

FIGURE 2.8: La technique par échantillonnage de l'espace d'entrées [2].

L'échantillonnage dans l'espace des contrôles est une méthode bien adaptée pour assurer une planification locale de plans faisables. Cependant, dans des environnements complexes, cette méthode cesse d'être efficace et un échantillonnage dans l'espace d'états est plus approprié.

Échantillonnage de l'espace d'états (SSS).

Cette technique permet la génération de controle de mouvement en spécifiant les conditions aux limites de chaque mouvement de l'ensemble des trajectoires génerées. [3, 28].

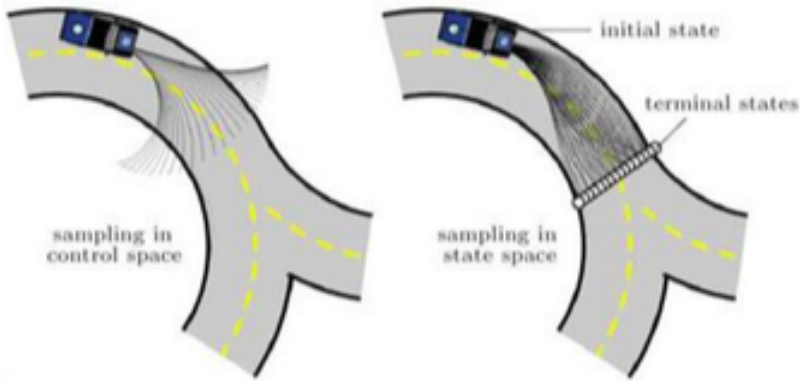

FIGURE 2.9: Illustration d'espaces de recherche générés par échantillonnage dans l'espace d'états vs. l'espace des contrôles qui sont fortement contraints (cas des réseaux routiers) (source [3]).

Cette technique est très efficace. Dans un réseau routier par exemple, il est plus bénéfique d'échantillonner uniquement à l'intérieur des voies, ainsi, le temps de calcul peut être optimisé en évitant de tester la non- collision des trajectoires qui sortent des voies. La figure 2.10 montre un exemple de trajectoires générées en échantillonnant l'espace d'états qui est comparé à un échantillonnage dans l'espace des contrôles (effectué dans les mêmes conditions de test).

2.2.3.3 Déformation de mouvement

La méthode de déformation de mouvement utilise un algorithme de planification de mouvements pour générer un chemin sans collision entre la position initiale du robot et le but en se basant sur une connaissance a priori de l'environnement. La deuxième phase de cette méthode est réalisée durant l'exécution (en temps réel) ; où le chemin du robot est déformé continuellement en fonction de la mise à jour de l'information sur l'environnement (i.e. quand un nouveau obstacle est rencontré)(un exemple est donné dans la figure 2.11)

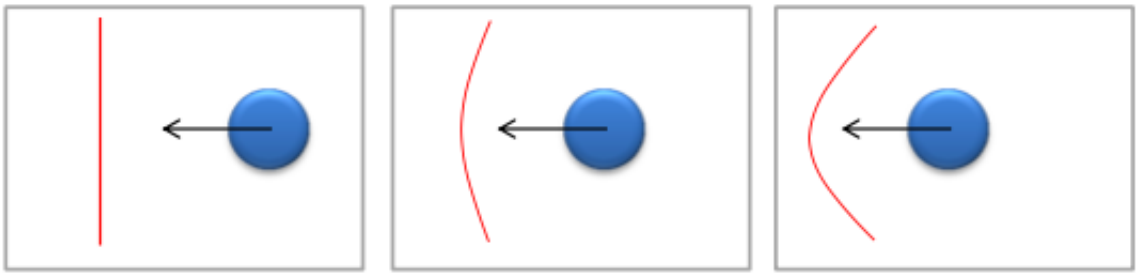

FIGURE 2.10: La déformation de chemin ; le chemin (en rouge) est continuellement déformé en réponse à l'approche d'un objet mobile [2].

Généralement la déformation résulte en de deux types de contraintes :

— Celles en provenance des obstacles (contraintes externes).

— Celles qui maintiennent la faisabilité (par rapport à la dynamique du robot) et la connectivité du chemin (contraintes internes).

Ainsi, la convergence vers le but est assurée. Les premiers travaux concernaient la déformation de chemin [29, 30], où le robot était supposé être holonome. Par la suite, la méthode a été étendue pour des robots non holonomes présentant plus de contraintes [31, 32].

2.2.3.4 La commande prédictive (MPC)

Comme les méthodes précédentes, ex. PMP et les techniques par échantillonnage, la commande prédictive (Model prédictive control) (MPC) [33–36]] est une méthode qui offre un compromis entre les méthodes réactives et les méthodes délibératives en raisonnant sur k pas de temps.

De manière générale, le principe de MPC est de raisonner sur plusieurs pas de temps mais d'effectuer qu'un seul pas. I.e. à tout instant t, étant donné le modèle de l'environnement, le mouvement du robot est planifié sur un temps horizon fini (pour un nombre 'N' de pas donné). Une séquence de contrôles optimaux est donc obtenue, mais uniquement le premier contrôle est appliqué. Le modèle de l'environnent est par la suite mis à jour et le processus est répété.

Ainsi, MPC est une méthode très efficace car elle prend en compte les contraintes kinodynamiques du robot. De plus, elle utilise l'information sensorielle pour mettre à jour régulièrement le modèle de l'environnement. Cependant, sa performance dépend de la portée du champ de vision. D'autre part, malgré que la majorité des travaux

basés sur MPC suppose que l'environnement est statique, cette méthode peut être adaptée pour traiter les obstacles mobiles.

2.2.4 Discussion

De manière générale, les différentes méthodes de navigation se résume à travers les méthodes délibératives et les méthodes réactives. Ces derniéres présentent certains avantages et inconvénients.

Parmis les méthodes de navigation présentées ci dessus, la méthode délibérative qui consiste à calculer un chemin complet et libre de collision vers le but, n'est applicable que dans des environnements statiques. C'est pourquoi, l'inconvénient majeur des méthodes délibératives est leur manque de réactivité par rapport aux changements de l'environnement pour pouvoir agir en temps réel. C'est là le point fort des méthodes réactives où le robot utilise ses capteurs embarqués pour mettre à jour l'information sur l'environnement.
Toutefois, l'aspect réactif de ces méthodes permet de réagir par rapport aux changements de l'environnement, et de générer un contrôle pour un seul pas de temps. Par conséquent, ces méthodes sont loin d'être les plus adaptées pour conduire le robot vers son but.
En raison des causes citées précédemment, une méthode alternative qui combine les deux classes de méthodes peut être la solution la plus adaptée pour un robot qui evolue dans un environnement et qui a uniquement une vision partielle de ce dernier, à savoir, les méthodes de planification réactive.
Cependant, plusieurs de ces méthodes nécessitent qu'un plan initial soit disponible et donc une connaissance a priori de l'environnement est initialement considérée (exp). Néanmoins, certaines méthodes de planification réactive ont été élaborés pour qu'elles puissent s'adapter pour des environnements initialement inconnus (exp PMP, ISS,...). Ces méthodes sont utilisées comme planificateurs locaux, ainsi elles gagnent en réactivité en méme temps elles permettent la planification de trajectoires en temps réel. Pour ce faire, grae a sa simplicité d'implémentation et d'adaptation

pour différentes situation, la solution pour laquelle nous avons opté est la méthode ISS.

2.3 Conclusion

Dans ce chapitre, nous avons présenté un état de l'art sur les méthodes de navigation existantes à savoir les méthodes délibératives et réactives, qui, soit calculent un chemin complet entre une configuration initiale du robot et une configuration but, ou calculent le mouvement à appliquer au pas de temps suivant. Néanmoins, des travaux de recherches ont abouti à une approche de planification réactives combinant les deux classes de méthodes.

Sachant que l'objectif de notre travail vise à développer un système de navigation pour un robot mobile ayant un champ de vision limité de son environnement. Dans ce cas, la méthode de planification de mouvement réactives à savoir la méthode par échantillonnage de l'espace d'entrées (ou espace de contrôles) a été choisi de manière à pallier aux contraintes de notre robot.

Deuxième partie

Implémentation et résultat

Chapitre 3

Méthodologie de conception de l'approche de navigation

3.1 Introduction

Dans ce chapitre nous allons exposer brièvement le principe général de la méthode choisie pour la modélisation de l'environnement ainsi que celle choisie pour la planification du mouvement de notre robot mobile.
Nous allons d'abord présenter le capteur laser utilisé dans l'application ensuite nous allons présenter la méthode de cartographie de l'environnement par grille d'occupation. Enfin, nous allons présenter l'approche de planification choisie à savoir la planification par échantillonnage de l'espace de contrôles (en anglais : Input Space Sampling (ISS)).

3.2 Le capteur laser UST-10LX Hokuyo

Le capteur laser UST-10LX Hokuyo (voir figure 3.1) est un petit capteur télémétrique à balayage. Léger et précis, ce capteur peut prendre en compte des distances allant jusqu'à 10m et est parfait pour des application en intérieur.
Les principaux avantages de ce laser est sa faible consommation en puissance et sa rapidité de traitement de signal.

FIGURE 3.1: Le capteur UST-10LX Hokuyo [4]

3.2.1 Caractéristiques techniques de l'UST-10LX Hokuyo

Les caractéristiques techniques du laser UST-10LX Hokuyo sont comme suit :

- Zone de fonctionnement :0.02m à 10m
- Domaine d'application : Intérieure (Indoor)
- Angle d'ouverture : 270°
- Fréquence de balayage : 40Hz

— Résolution angulaire (en degré) : 0.25 °

— Alimentation : 12 à 24 VCC

— Interface de communication : Ethernet

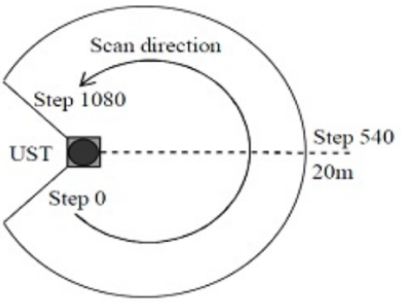

FIGURE 3.2: Angle d'ouverture et portée du capteur laser [4].

Ces caractéristiques rendent l'UST-10LX Hokuyo adéquat pour la navigation de robots mobiles ainsi que la détection d'obstacle [15].

3.3 Modélisation de l'environnement en grille d'occupation

3.3.1 Modélisation des données laser

Le laser est un capteur directive avec un angle d'ouverture $\alpha \in [\alpha_{min},\alpha_{max}]$, dont la mesure est interprétée comme étant la distance 'r' au plus proche obstacle pour chacune des directions d'angle α_i (voir la figure 3.3). Ainsi, les mesures seront modélisées comme celles d'un capteur directive qui nous procure une liste des distances r_i ne dépassant pas la portée maximum du laser (r_{max} et cela pour chaque pas d'angle.

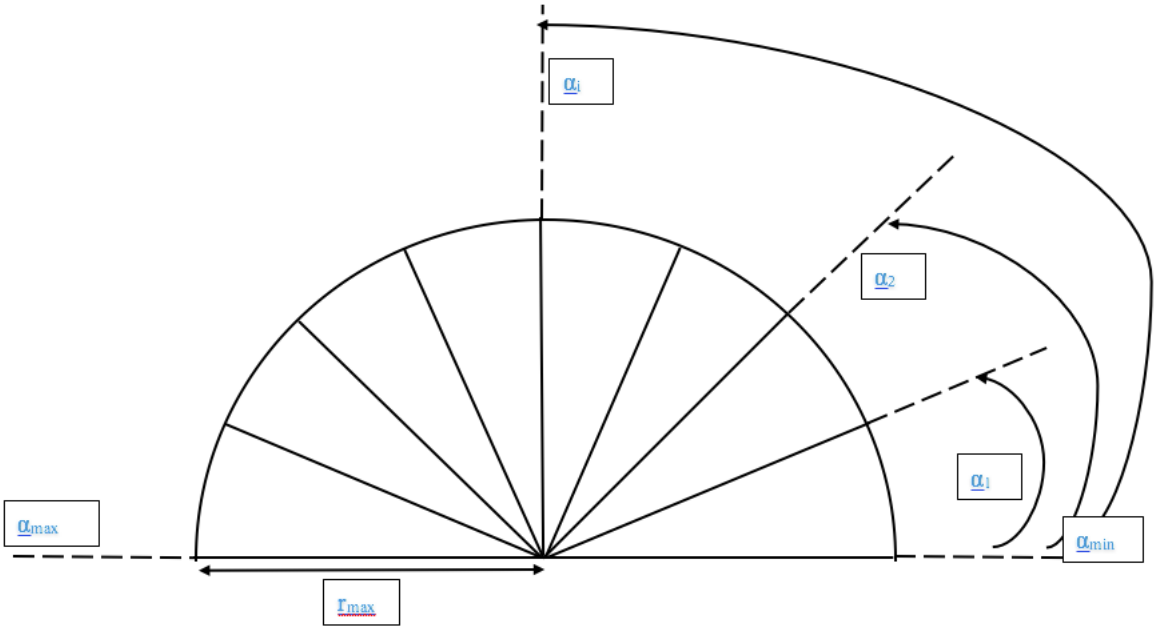

FIGURE 3.3: Angle d'ouverture d'un capteur laser.

De façon général, les données laser seront traduites en une liste de distances r entre le capteur et les obstacles détectés, l'angle correspondant à chaque angle (d'indice i $\in$ [0,N] ,avec N étant le nombre de directions) de la liste est calculé par la formule suivante :

$$\alpha_i = \alpha_{min} + \frac{i \times (\alpha_{max} - \alpha_{min})}{N - 1} \tag{3.1}$$

Chaque couple de coordonnées polaires (r_i,α_i) sera converti en coordonnées cartésiennes (x_i^C,y_i^C) dans le repère lié au capteur R^C (O,X^C,Y^C) afin de pouvoir les traiter par la suite.

3.3.2 Représentation de l'environnement

3.3.2.1 Création de la grille d'occupation

Pour construire la grille, le robot utilise le capteur laser présenté précédemment. La donnée fournie par ce dernier est une distance séparant l'obstacle détecté et le capteur en question.

Notre but est de trouver à partir de cette distance r (comme seule information disponible), la cellule correspondante à la position de l'obstacle dans la matrice représentant la grille d'occupation.

Pour cela, nous commençons par associer chaque distance r_i à l'angle respectif de détection α_i via l'équation (3.1), pour que chaque couple de coordonnées polaires (r_i,α_i), soit converti en coordonnées cartésiennes (x_i,y_i) par l'équation suivante (voir annexe A.2) :

$$\begin{pmatrix} x_i^C \\ y_i^C \end{pmatrix} = \begin{pmatrix} r_i \cos \alpha_i \\ r_i \sin \alpha_i \end{pmatrix} \tag{3.2}$$

Ensuite, chaque coordonnée de l'obstacle (x_i^C,y_i^C) sera exprimée dans le repère lié au robot R^R (O',X^R,Y^R) (voir annexe A.3). La conversion se fait par l'équation suivante :

$$\begin{pmatrix} x_i^R \\ y_i^R \end{pmatrix} = \begin{pmatrix} y_i^C + a \\ x_i^C + \frac{b}{2} \end{pmatrix} \tag{3.3}$$

Où a,b sont respectivement la longueur et la largeur de notre chaise roulante.

Enfin, nous déduisons la position de l'obstacle par rapport au repère local R^L (O,X^L,Y^L) à travers la transformation suivante :

$$\begin{pmatrix} x_i^L \\ y_i^L \end{pmatrix} = \begin{pmatrix} -y_i^R + \frac{width}{2} \\ x_i^R \end{pmatrix} \tag{3.4}$$

Ou width est la taille de notre grille suivant l'axe X_L (voir la figure 3.4) est donnée par (voir annexe A.3) :

$$\begin{pmatrix} x_i^L \\ y_i^L \end{pmatrix} = \begin{pmatrix} r_i \cos \alpha_i - \frac{b-width}{2} \\ r_i \sin \alpha_i + a \end{pmatrix} \tag{3.5}$$

Enfin, pour trouver la cellule qu'il occupe, on associe a chaque position la cellule d'indice (i,j) par :

$$\begin{pmatrix} i \\ j \end{pmatrix} = \begin{pmatrix} \frac{x_i^L}{Xl} \\ \frac{y_i^L}{Yl} \end{pmatrix} \tag{3.6}$$

Où Xl, Yl sont respectivement la résolution de la cellule selon l'axe Xl et Yl (voir figure 3.4).

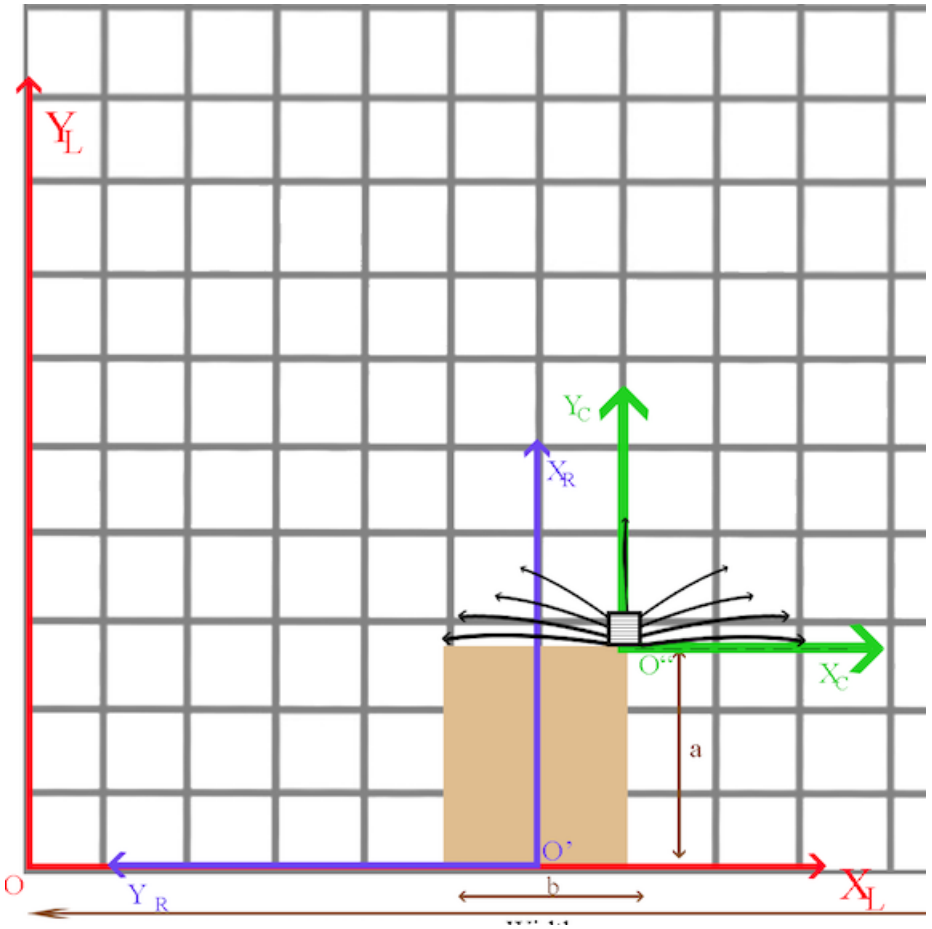

FIGURE 3.4: Mise en correspondance de la grille d'occupation avec les différents repères existants. Avec height et width sont respectivement la taille des lignes et colonnes de la grille.

3.3.3 Algorithme

La méthode de modélisation de l'environnement par grilles d'occupation peut être résumée à travers l'organigramme de la figure 3.5.

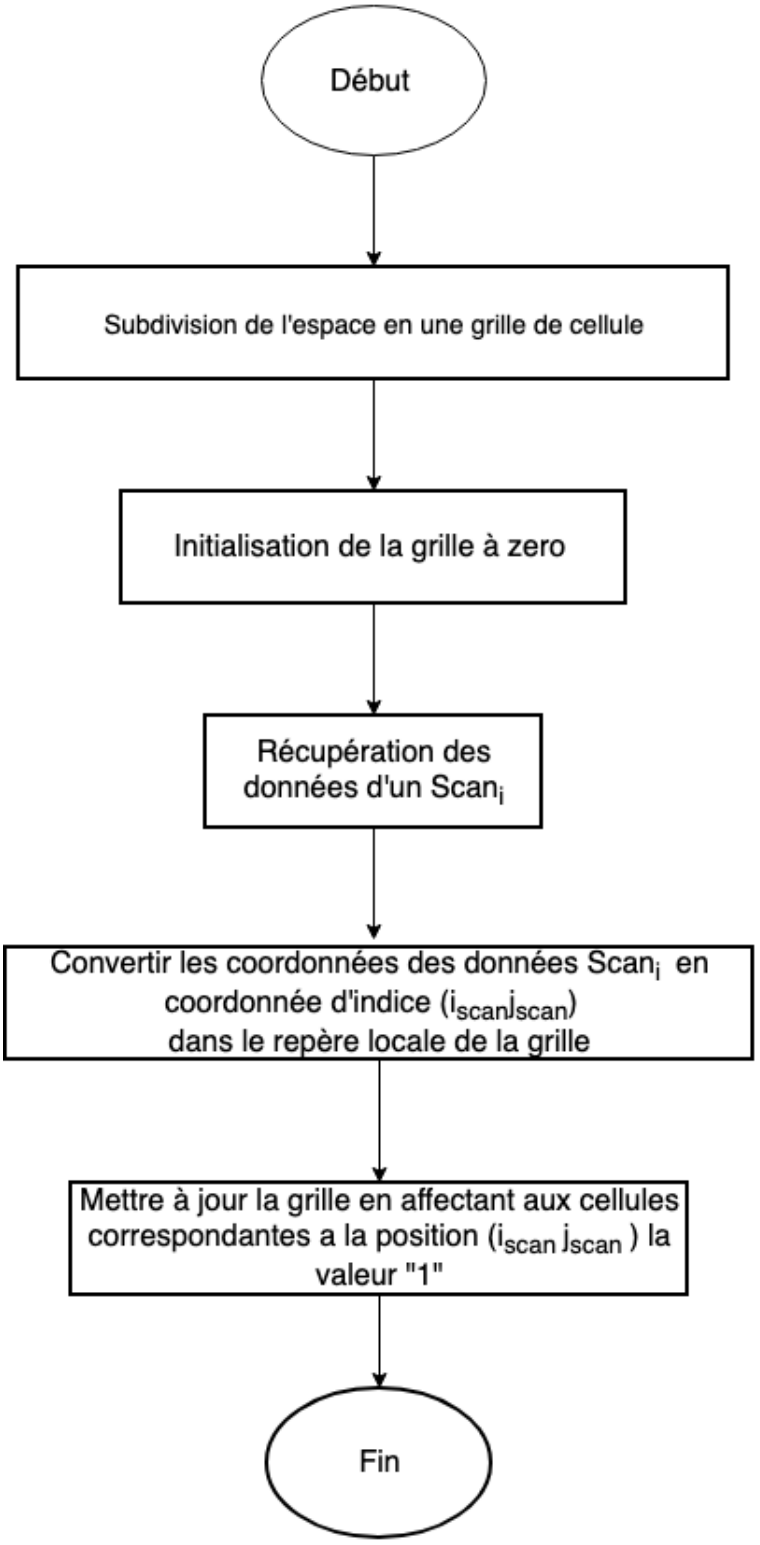

FIGURE 3.5: Organigrame de la modélisation de l'environnement par grilles d'occupation.

3.4 Planification par échantillonnage de l'espace de contrôles

3.4.1 Principe de la méthode d'échantillonnage de l'espace de contrôles : NAV_ ISS

Dans la navigation des robots mobiles, l'échantillonnage de l'espace d'entrées ou contrôles (en anglais : input space sampling (ISS)) reste une méthode de planification simple de compréhension et d'application. Pour cela, cette approche de navigation nous a mené à implémenter un planificateur "NAV_ ISS", dont le principe et l'algorithme sont décrit dans ce qui suit.

NAV_ ISS est un planificateur réactive de mouvement qui utilise la méthode d'échantillonnage ISS en temps réel et qui agit pendant un pas de temps donné δt, avec comme but, calculer un contrôle μ représenté par une vitesse linéaire constante (và et une vitesse angulaire (ω). Ce contrôle μ sera appliqué au système robotique durant le pas de temps suivant.

NAV_ ISS fonctionne comme la plupart des techniques d'échantillonnage de l'espace de contrôles. De façon général leurs principe de fonctionnement est de générer, à partir d'une configuration initiale $q_0 = (x_0, y_0, \varphi_0)$., un ensemble échantillonné de contrôles constants : $U_{ctrl} = \{\mu_1, \mu_2 ..... \mu_{Nbr}\}$ (voir la figure 3.6).

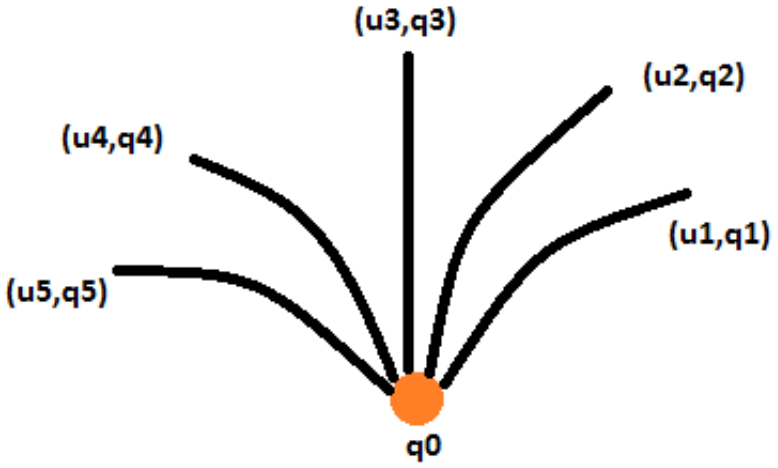

FIGURE 3.6: Exemple d'échnatillonnage de l'espace de contrôles.

Pour chaque contrôle appartenant à l'ensemble U_{ctrl}, NAV_ ISS doit tester la non collision avec les obstacles afin d'assurer la un mouvement sans collision. L'ensemble des contrôles sans collision seront définis par la suite par l'ensemble des contrôles $U*_{ctrl}$ (voir figure 3.7).

Afin de pouvoir tester la non collision avec les obstacles NAV_ ISS doit d'abord calculer la configuration correspondante pour chaque contrôle testé μ_{test}. Ce calcul se fait suivant les équations suivantes [37] :

$$\begin{aligned} x_i &= x_0 + \delta d \cos(\varphi_0 + \delta\varphi) \\ y_i &= y_0 + \delta d \sin(\varphi_0 + \delta\varphi) \\ \varphi_i &= \varphi_0 + \delta\varphi \end{aligned} \tag{3.7}$$

Ainsi, pour chaque controle μ_i correspondera une configuration $q(\mu_i)$ tel que : $q_(\mu_i) = (x_i, y_i, \varphi_i)$.

Chaque coordonnée correspondante à la configuration $q(\mu_i)$ sera convertie de manière à avoir l'indice correspondant dans la grille obtenue grâce à l'algorithme de la méthode de modélisation par grilles d'occupation (voir l'organigramme de la figure 3.5). Ces indices sont obtenus suivants l'équation (3.6).

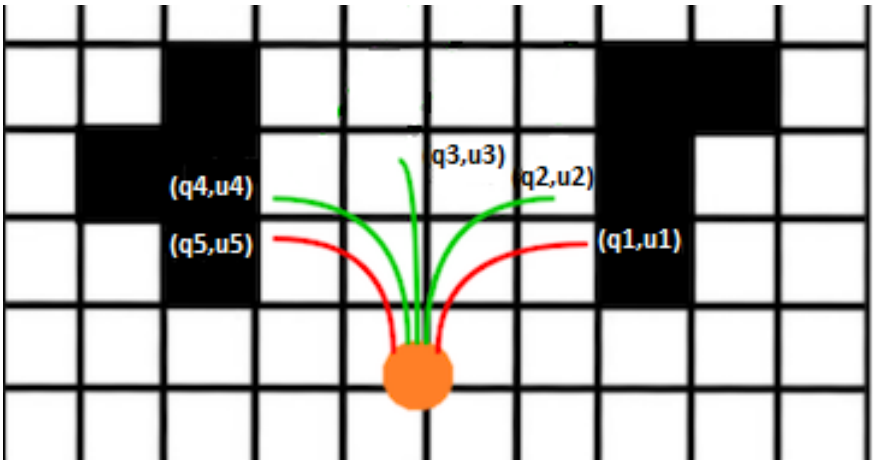

FIGURE 3.7: Exemple de géneration de 5 contrôles dans une grille d'occupation binaire. Les contrôles en vert correspondent aux contrôles sans collision dits «collision free». Tant dis que ceux en rouge représentent les contrôles non «collision free» à savoir interdits.

Ainsi, l'ensemble des contrôles jugés sans collision sont triés afin de sélectionner le contrôle le plus proche du but μ_{near}. Cette sélection se fait en sélectionnant la distance la plus petite séparant la configuration but q_{goal} avec la configuration de chaque contrôle sans collision $q(\mu test)$ (voir figure 3.8) .

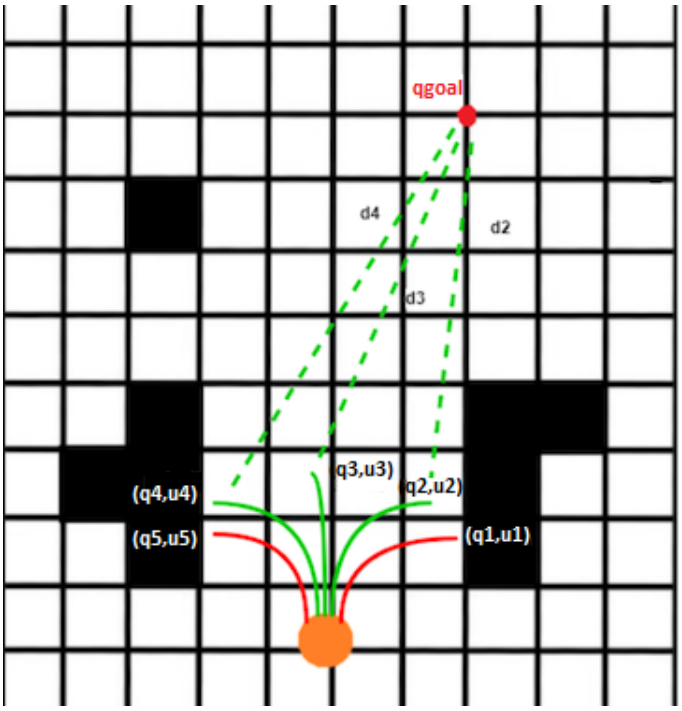

FIGURE 3.8: Principe de la selection du contrôle le plus proche du but.

3.4.2 Algorithme général

L'algorithme suivant décrit le fonctionnement de l'approche NAV_ ISS durant un pas de temps donné δt.

De façon générale, les étapes requises pour calculer un contrôle proche du but μ_{near} sont décrites dans l'algorithme 1. Cet algorithme a pour entrées une configuration initiale $q_0 = (x_0, y_0, \varphi_0)$ représentent la position initiale du robot acquise par le système odométrique mais aussi la configuration finale du but $q_{goal} = (x_{goal}, y_{goal}, \varphi_{goal})$ introduite par l'utilisateur lui-même. L'étape 1 de l'algorithme vise à échantillonner un ensemble de contrôles constants : $U_{ctrl} = \{\mu_1, \mu_2.....\mu_{Nbr}\}$ (voir la ligne 1 de l'algorithme 1) .

L'étape 2 consiste à tester la non collision avec les obstacles de chaque contrôle μ_i appartenant à U_{ctrl} (grâce à la fonction TEST_COLLISION, ligne 2 à 3 de l'algorithme 1). L'étape 3 quant à elle consiste à trier l'ensemble des contrôles jugés sans collision (ligne 4 à 5). Enfin, l'étape 4 vise à sélectionner le contrôle le plus proche du but μ_{near}. Cette sélection se fait par rapport au calcul du minimum de distance séparant la configuration but q_{goal} avec la configuration de chaque contrôle sans collision $q(\mu test)$ (grâce à la fonction REACH_GOAL, la ligne 7 à 11).

Algorithm 1: NAV_ ISS

Input: Configuration initiale q_0, Configuration du but q_{goal}, temps de l'échantillonnage δt

Output: Contrôle μ

1 Echantillonner $U \rightsquigarrow U_{ctrl} = \{\mu_1, \mu_2, ..., \mu_{Nbr}\}$ //Sélectrionner l'outil d'échantillonnage de l'espace de controle.
2 **for** $\mu_i \in U_{ctrl}$ **do**
3 Collision_ free=$TEST\_COLLISION(\mu_i, grid)$; //Tester l'ensemble des controles.
4 **if** *Collision_free is True* **then**
5 $U*_{ctrl} \leftarrow \mu_i$; //Retourner l'ensemble des controles sans collision.
6 Initialiser $d_{min} = \infty$;
7 **for** $\mu_i \in U*_{ctrl}$ **do**
8 d =REACH_GOAL(q_0, q_{goal}, d_{min}) ;
9 **if** $d < d_{min}$ **then**
10 $d_{min} \leftarrow d$;
11 $\mu_{near} \leftarrow \mu_i$
12 **return**(μ_{near}) ; // Retourner le controle à appliquer.

De manière à apporter plus de clarté sur la compréhension de l'algorithme général de l'approche NAV_ ISS, nous avons détaillé les principales fonctions utilisées dans l'algorithme 1 à travers les algorithmes suivants :

Algorithm 2: TEST_ COLLISION

Input: μ_i,matrice d'occupation $grid$, δt

Output: Boolean value $Collision\_free$

$q(\mu_{test})$ =CONFIG_FUNC$(q(\mu_0), \mu_i, t)$; //Calcul de la configuration conrrespondante au controle selon l'équation.

(i_{test}, j_{test}) ←CAMPUTE_INDICE_CELL$(q(\mu_{test}))$; //Selon l'équation (3.6).

if $grid[i_{test}][j_{test}] = 0$ **then**

    $Collision\_free = True$;

else

    $Collision\_free = False$;

return(Collision_ free).

Algorithm 3: REACH_ GOAL

Input: q_0, q_{goal}, d_{min}

Output: Distance euclidienne d

$d = \sqrt{(x_0 - x_{goal})^2 + (y_0 - y_{goal})^2}$

return(d) // Retourner la distance euclidienne séparant l'état S_i du but.

3.5 Conclusion

Dans ce chapitre, nous avons décrit en premier lieu la manière de modéliser les données du capteur laser embarqué sur notre robot mobile. Ensuite nous avons expliqué la méthodologie de conception de la carte de l'environnement à travers la méthode des grilles d'occupation. Enfin, nous avons expliqué globalement le principe de notre approche de navigation NAV_ ISS basé sur la méthode d'échantillonnage de l'espace de contrôles ISS (*Input Space Sampling*).

Le chapitre suivant présentera les résultats de l'implémentation de l'approche NAV_ISS.

Chapitre 4

Validation, test et résultat

4.1 Introduction

Après avoir développé un prototype de chaise roulante robotisée et l'approche de navigation (NAV_ISS), il est nécessaire de passer aux tests expérimentaux afin de démontrer et valider l'approche implémentée dans un environnement intérieur inconnu. C'est pourquoi ce chapitre est consacré à la présentation des résultats de l'implémentation de NAV_ISS. Nous allons d'abord présenter la chaise roulante robotisée (FAURSA) et son modèle cinématique, Ensuite, nous allons présenter les tests et les resultats expérimentaux.

4.2 Présentation du robot mobile

FAURSA (figure 4.1) est un prototype de chaise roulante robotisée construit sur la base d'un chaise roulante classique utilisée dans des environnements d'extérieurs ou d'intérieurs. Cette chaise comporte une architecture logicielle parallèle crée par l'équipe NCRM du CDTA ; permettant le développement d'applications d'intelligence artificielle.

FIGURE 4.1: La chaise roulante robotisée "FAURSA" du CDTA.

Les caractéristiques générales et techniques principales de la chaise roulante FAURSA sont les suivantes :

— Type du robot mobile : Différentiel.

— Longueur totale : 750mm, largeur totale : 730 mm, hauteur : 970mm.

— Poids total avec batteries : 16,02kg.
— Motorisation : 2 Roues motrices, chacune liée à un encodeur.
— Vitesse maximale : 2m/s).
— Navigation autonome par ordinateur embarqué.
— Un capteurs laser UST 10LX Hokuyo placé sur le pieds avant droit de la chaise roulante.

4.3 Cinématique du robot mobile

Comme nous l'avons mentionné précédemment notre robot possède une configuration différentielle il comporte deux roues fixes non orientables commandées indépendamment et deux roues folles placées à l'avant pour assurer sa stabilité. Les roues motrices ayant le même axe de rotation, nous pouvons définir le centre instantané de rotation CIR du robot comme étant un point de cet axe (voir figure 4.2).

Soit R le rayon de courbure de la trajectoire du robot, c'est à dire la distance du CIR au point O' (voir figure 4.2). Soit 2L la distance qui sépare les deux roues, et ω la vitesse angulaire du robot par rapport au CIR. Alors les vitesses des roues droite et gauche, respectivement notées V_d et V_g vérifient :

$$x = v\cos\varphi \tag{4.1}$$

$$x = v\cos\varphi \tag{4.2}$$

Ce qui permet de déterminer R et ω à partir des vitesses des roues :

$$V_d = (R + L)\omega \tag{4.3}$$

$$V_g = (R - L)\omega \tag{4.4}$$

La vitesse linéaire v du robot au point O‘ est :

$$\omega = \frac{V_d - V_g}{2L} \tag{4.5}$$

La vitesse de rotation du robot est égale à la vitesse de rotation autour du CIR :

$$R = L\frac{V_d + V_g}{V_d - V_g} \tag{4.6}$$

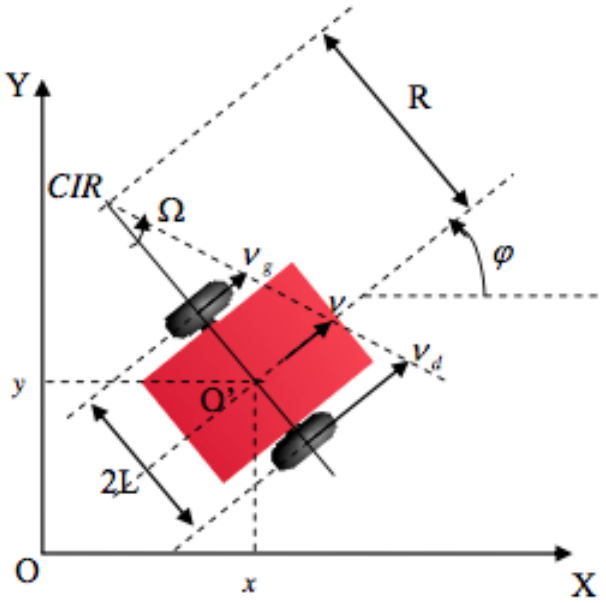

FIGURE 4.2: Représentation du CIR d'un robot mobile différentiel.

Ainsi à partir de la figure ci-dessus nous pouvons déduire les équations du modèle cinématique du robot différentiel.

$$x = v\cos\varphi \tag{4.7}$$

$$y = v\sin\varphi \tag{4.8}$$

$$\varphi = \omega \tag{4.9}$$

Ces équations relient la dérivée de la position (x, y, φ) du robot à la commande $u = (v, \omega)^T$ avec φ rotation instantanée du robot par rapport au repère (O, X, Y). De ce fait la position du robot est donnée par :

$$x(t) = \int_0^t v(\varphi)\cos\left(\varphi(\sigma)\right)d\sigma \tag{4.10}$$

$$y(t) = \int_0^t v(\varphi) \sin(\varphi(\sigma)) d\sigma \tag{4.11}$$

$$\varphi(t) = \int_0^t (\omega(\sigma)) d\sigma \tag{4.12}$$

4.4 Outils utilisés

Pendant les différentes étapes de développement de notre travail, nous avons utilisé quelques outils qui peuvent se résumer comme suit :

4.4.1 Environnement de développement

4.4.1.1 Ubuntu 16.04 LTS

Ubuntu se définie comme étant un système d'exploitation GNU/Linux [38]. Très populaire de part, son utilisation, Ubuntu a acquis alors une communauté très dynamique grâce à son interface simple, intuitive et sécurisée.

Nous avons opté pour le système d'exploitation Ubuntu, non pas pour sa stabilité et sa large communauté dynamique, mais parce qu'il est le seul système d'exploitation officiellement supporté par ROS.

4.4.1.2 Robot Operating System (ROS)

Le système d'exploitation des robots (Robot Operating System ROS), est un ensemble d'outils informatiques libre et open source permettant de développer des logiciels pour la robotique [39]. Son développement est mené par l'Open Source Robotics Foundation (OSRF) [40]. Ros est officiellement supporté par plus de 75 robots.

4.4.2 Paquets ROS utilisés

Sous la plateforme ROS, nous avons utilisé le paquet Urg_ node [41], qui est une sorte de driver pour communiquer afin de modifier les paramètres du capteur laser Hokuyo et accéder aux données fournis par ce dernier.
Au cours de nos tests, nous avons exploité aussi les outils du ROS à savoir rqt_ graph et rviz pour l'affichage (interface d'affichage).

4.5 Implémentation de l'approche NAV_ ISS

Nous avons implémenté la solution proposée dans le troisième chapitre sur la chaise roulante robotisée FAURSA, l'implémentation est faite sous le système d'exploitation pour les robots ROS (Robot Operating System) (voir l'annexe B), et en utilisant le langage de programmation Python, ainsi que la bibliothèque NumPy qui apporte les fonctionnalités mathématique au langage Python. Le schéma de communication de nos noeuds est illustré dans la figure 4.3.

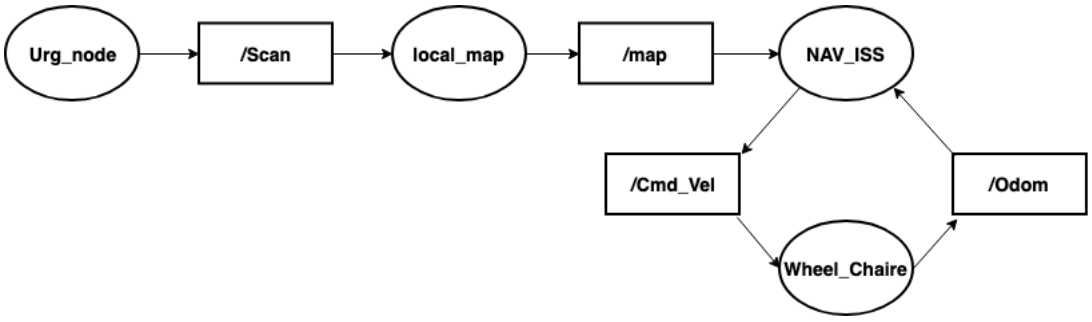

FIGURE 4.3: Schéma de communication ROS : NAV_ISS et local_map, généré par l'outil rqt_graph de ROS.

De manière générale, le noeud **Urg_node** représentant notre capteur laser publit les données de ce dernier via le topic **/Scan** qui seront récupérées par le second noeud **local_map** responsable de la construction de la carte locale liée à l'environnement de travail du robot.

A son tour, le noeud **local_map** publie la carte de l'environnement dans le topic **/map** et qui sera récupérée par le noeud **NAV_ISS**. Cette carte est régulierement mise à jour

Ce dernier, à savoir le noeud **NAV_ISS**, est chargé du traitement pour la planification de mouvements de la chaise roulante. C'est pourquoi, la carte locale publié sur topic **/map** ainsi que les données odométriques de la chaise roulante recupèrées par le biais du topic **/Odom** à travers le noeud **Wheel_ Chaire**, sont les entrées principales du noeud **NAV_ISS** afin qu'il puisse procéder au raisonnement et prise de décision.

Enfin, l'approche de planification implémenter sur le noeud **NAV_ISS** publiera des contrôles traduits à travers des vitesses linaires et angulaires, qui seront injectées au final au noeud **Wheel_ Chaire** via le topic **/Cmd_ Vel**.

4.6 Tests et validation

4.6.1 Résultat de la modélisation de l'environnment

Dans cette section, nous allons présenter quelques résultats de l'implémentation de l'algorithme de modélisation de l'environnement basé sur les grilles d'occupation. Le test a été déroulé dans la salle machine du département production et robotique du CDTA.(voir figure 4.4)

FIGURE 4.4: Environnement de test.

Afin de pouvoir visualiser les résultats de l'implémentation de l'algorithme développé sous ROS à travers le noeud **local_map**, nous utilisons l'outil Rviz qui nous permet d'afficher l'environnement de test(voir la figure 4.5).

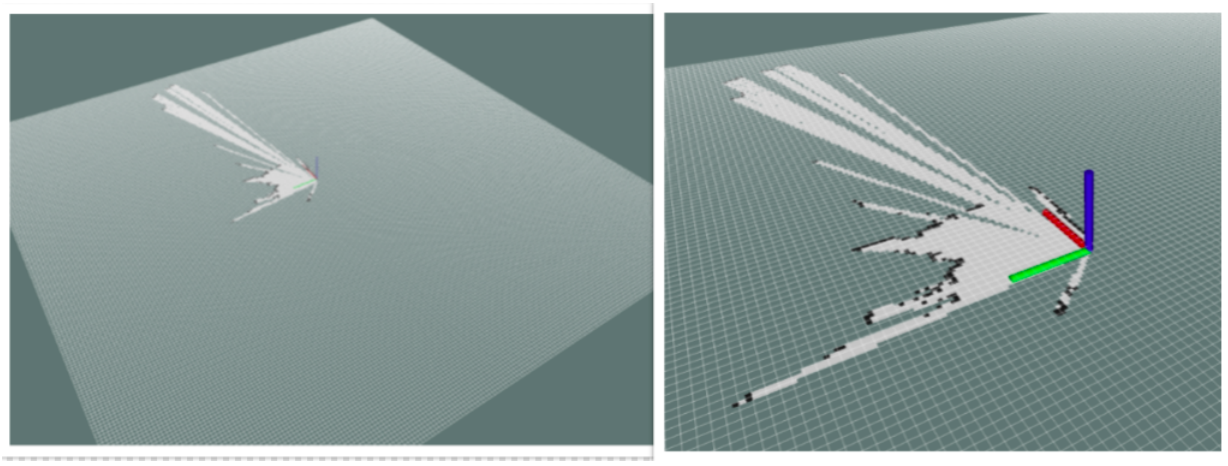

FIGURE 4.5: Représentation de la modélisation de l'environnement via l'outil Rviz.

Ainsi, comme illustré dans la figure 4.5 chaque cellule grise représente les parties inconnus de l'environnement ou dites non explorées. Quant aux cellules noirs, celles ci désignent l'espace occupée. Enfin, les espaces libres de l'environnement sont représentés par les cellules blanches.

4.6.2 Résultat des tests de l'implémentation de NAV_ISS

La position du but(goal) a atteindre dans l'environnement de test est illustrée dans la figure 4.6. La navigation vers un but (goal) se fera grace au planificateur NAV_ISS.

FIGURE 4.6: Environnement de tests de la navigation vers le but.

La figure 4.7 montre le résultat de navigation vers un but (goal). On peut constater que la chaise a bien atteint son but qui représente une position introduite par l'utilisateur.

FIGURE 4.7: Etape de la navigation vers le but.

Ainsi, suivant les étapes a,b, c et d comme illustré dans la figures 4.7 NAV_ISS permet à la chaise de naviguer dans son environnement en atteignant son but qui représente une position introduite par l'utilisateur.

4.7 Conclusion

Les tests effectués dans un milieu environnemental réel et incconu ont été une réussite ; la méthode de modélisation de l'environnement implémentée montre clairement une correspondante quasi similiaire avec l'environnement de travail.En dernier lieu le test de la navigation vers un but a tout autant été une réussite car la chaise a pu atteindre une position introduite préalablement par l'utilisateur.

Conclusion générale

La robotique mobile a connu un essor surprenant durant ces dernières années, plusieurs travaux et recherches ont été menés et développés pour étendre le secteur de la robotique de service afin d'en tirer profit et améliorer le quotidien de vie de l'être humain. Ce projet de fin d'étude, s'intègre dans le développement d'une approche de navigation dans le domaine de la robotique de service. Cette approche a été implémentée sur une chaise roulante robotisée, équipé d'un capteur laser avec un champ de vision limité.

Notre travail s'est déroulé en deux étapes principales. L'objective de la première étape était de développer un algorithme de modélisation de l'environnement basé sur la méthode de modélisation par grilles d'occupation et cela grâce aux données du télémètre laser embarqué sur la chaise roulante.

Quant à la deuxième étape, nous avons développé un algorithme de planification réactif NAV_ISS, basé sur la méthode d'échantillonnage de l'espace de contrôles.

L'implémentation de cette approche et les essais effectués sous les contraintes d'un environnement réel et inconnu où notre chaise a une vision limitée de son environnement (à cause de la limite sensorielle) nous a permis de montrer la capacité de l'algorithme développé à agir dans de telles conditions.

Perspective La méthode de planification choisie à savoir la méthode par échantillonnage de l'espace de contrôle peut être facilement adaptée pour être appliquée dans des environnements dynamiques et ainsi offrir une meilleure sûreté du mouvement.

Annexe A

Concepts fondamentales

A.1 Les transformations géométriques

En robotique, nous nous intéresse au transformations géométriques de type "translation" et "rotation", les deux types sont détaillés ci-dessus :

A.1.1 Les translations

En géométrie, une translation est une transformation géométrique qui corres- pond à l'idée intuitive de "glissement" d'un objet, sans rotation, retournement ni déformation de cet objet. En géométrie classique, la notion de translation est très fortement liée à celle de vecteur, qu'elle suit ou précède. Ainsi trouve-t-on la translation de vecteur $\overrightarrow{u}$ définie comme une transformation qui, à tout point M, associe le point M' tel que :

$$\overrightarrow{MM'} = \overrightarrow{u} \tag{A.1}$$

On dit alors que M' est le translaté de M. C'est l'image de M par cette translation. Soit disant vecteur $\overrightarrow{u}$ et un point M les deux de dimension N, la translation de vecteur transforme le point $M = (M_1 M_2 ... M_N)^t$ en $M' = (M'_1 M'_2 ... M'_N)^t$.

$$\overrightarrow{M'} = \overrightarrow{u} + \overrightarrow{M} \tag{A.2}$$

$$\begin{pmatrix} \overrightarrow{M'_1} \\ \overrightarrow{M'_2} \\ \vdots \\ \overrightarrow{M'_N} \end{pmatrix} = \begin{pmatrix} \overrightarrow{u_1} \\ \overrightarrow{u_2} \\ \vdots \\ \overrightarrow{u_N} \end{pmatrix} + \begin{pmatrix} \overrightarrow{M_1} \\ \overrightarrow{M_2} \\ \vdots \\ \overrightarrow{M_N} \end{pmatrix} \tag{A.3}$$

A.1.2 Les rotations

Dans le plan vectoriel euclidien orienté, une rotation vectorielle est simplement définie par son angleφ. Sa matrice dans une base orthonormée directe est :

$$\begin{pmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{pmatrix} \tag{A.4}$$

Autrement dit, un vecteur $\overrightarrow{U}$ de composantes $(x, y)^t$ a pour image le vecteur $\overrightarrow{V}$ de composantes $(x', y')^t$ que l'on peut calculer avec l'égalité matricielle :

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \tag{A.5}$$

A.2 Les coordonnées polaires

Les coordonnées polaires sont un système de coordonnées à deux dimensions, dans lequel chaque point du plan est entièrement déterminé par un angle φ et une distance r. Ce système est particulièrement utile dans les situations où la relation entre deux points est plus facile à exprimer en termes d'angle et de distance.
Les deux coordonnées polaires r et φ peuvent être converties en coordonnées cartésiennes x et y en utilisant les fonctions trigonométriques sinus et cosinus (voir la figure A.1) :

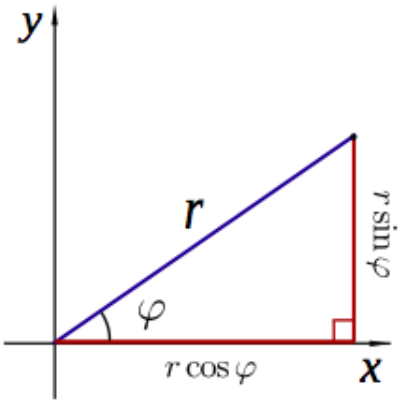

FIGURE A.1: Conversion des coordonnées polaires en coordonnées cartésiennes.

A.3 Fromules de changement de repère

A partir de la figure A.2 nous pouvons déduire les équations de passages du repère capteur $R^C(O", X^C, Y^C)$ au repère lié au robot $R^R(O', X^R, Y^R)$ mais aussi du repère robot au repère local $R^L(O, X^L, Y^L)$ (lié à l'environnement). Les formules de changement de repères sont comme suit :

Passage de $R^C \longrightarrow R^R$:

$$\begin{pmatrix} X^R \\ Y^R \end{pmatrix} = \begin{pmatrix} Y^C + x_r \\ X^C + y_r \end{pmatrix} \tag{A.6}$$

Passage de $R^R \longrightarrow R^L$:

$$\begin{pmatrix} X^L \\ Y^L \end{pmatrix} = \begin{pmatrix} -Y^R + x_L \\ X^R \end{pmatrix} \tag{A.7}$$

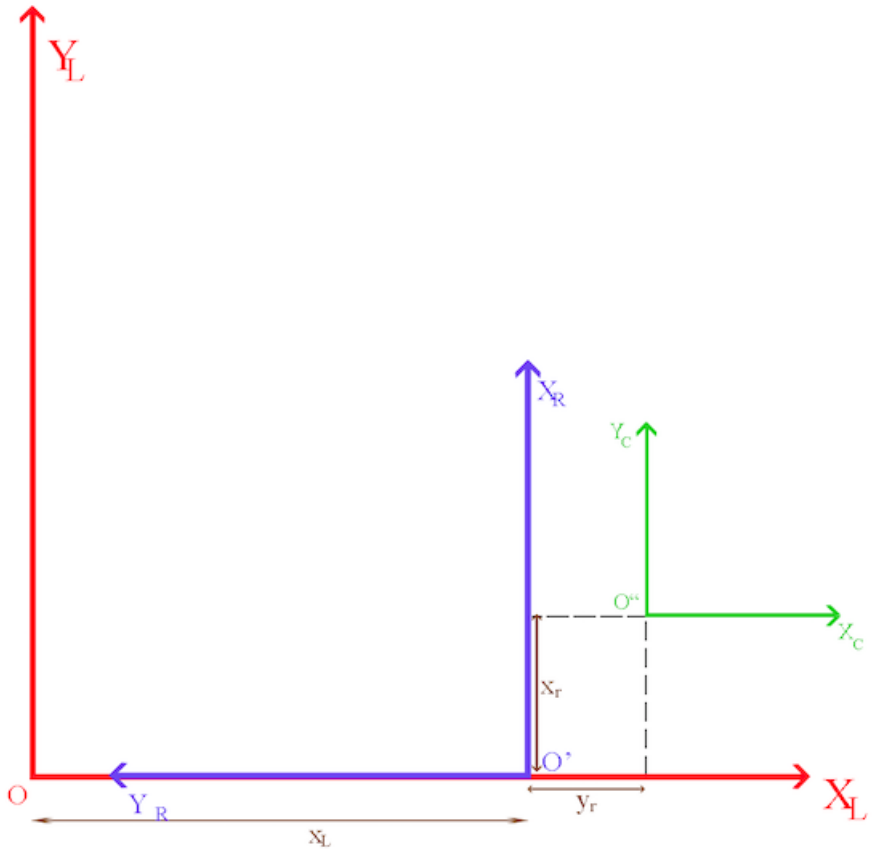

FIGURE A.2: Représentation des axes des différents repères.

Annexe B

Robot Operating System (ROS)

ROS a été développé à l'origine en 2007 sous le nom de switchyard par le Stanford Artificial Intelligence Laboratory dans le cadre du projet *Stanford AI Robot STAIR* (STanford AI Robot). La description de STAIR sur sa page d'accueil dite :

Notre plate-forme unique pour la robotique intégrera les méthodes tirées de tous les domaines de l'intelligence artificielle, y compris l'apprentissage par machine, la vision, la navigation, la planification, le raisonnement et le traitement de la parole / du langage naturel. Cela contraste nettement avec la tendance de 30 ans sur les sous- domaines fragmentés de l'IA et sera un moyen de conduire la recherche vers une véritable IA intégrée.

FIGURE B.1: Logo officiel du projet ROS.

ROS est un système d'exploitation complet pour la robotique de service. En effet, ROS est un ***méta-système d'exploitation*** , quelque chose entre le système d'exploitation et le middleware. Il fournit des services proches d'un système d'exploitation (abstraction du matériel, gestion de la concurrence, des processus...) mais aussi des fonctionnalités de haut niveau (appels asynchrones, appels syn- chrones, base centralisée de données, système de paramétrage du robot...). Une vue globales des composantes du système d'exploitation ROS sont sché- matisés dans la figure A.2.

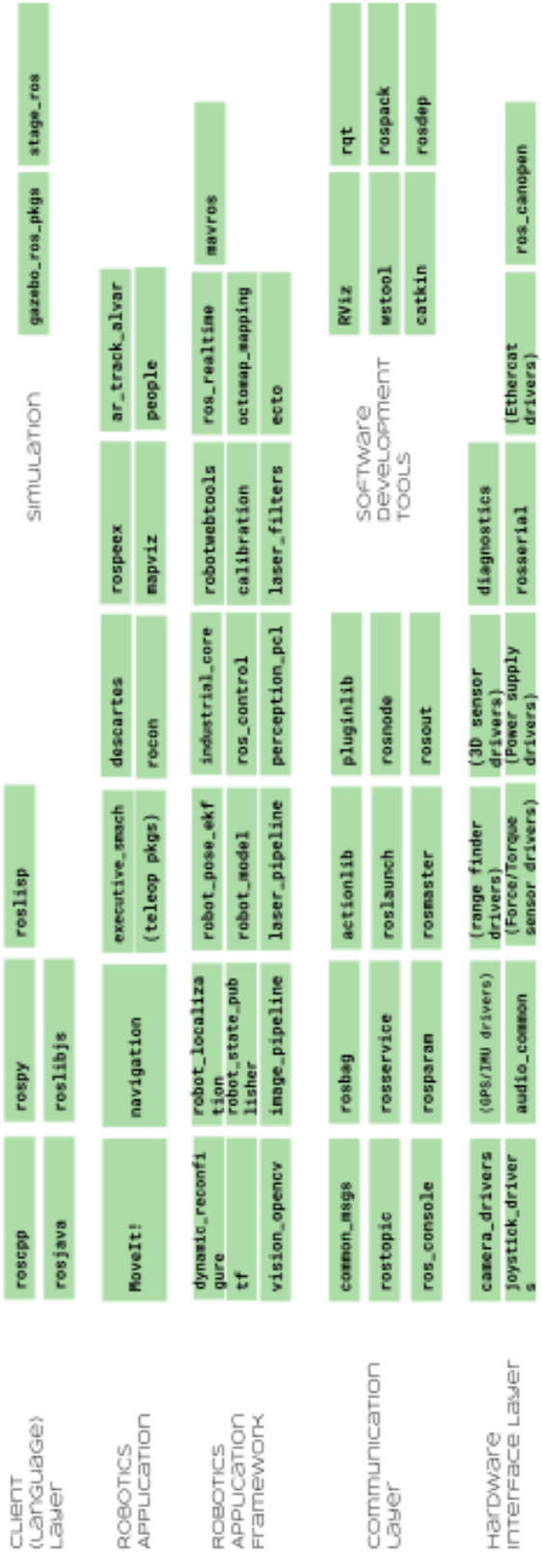

FIGURE B.2: Les composants de méta système d'exploitation ROS.

B.1 Bibliothèques du client ROS

Une bibliothèque du client ROS (ROS Client Library) est une bibliothèque qui facilite la tâche du programmeur ROS. Elle prend beaucoup de concepts ROS et les rend accessibles via le code. En général, ces bibliothèques vous permettent d'écrire des noeuds ROS, de publier (publish) et d'abonner (subscribe) aux topics, d'écrire et d'appeler des services, et d'utiliser le serveur de paramètres. Une telle bibliothèque peut être implémentée dans n'importe quel langage de programmation, bien que ROS fournit un support officiel robuste des langages de programmation C++ (roscpp , Python (rosp 1) et Lisp (roslisp), d'autres langages sont supportés par la communauté de ROS, à travers les bibliothèques de client comme rosjava, rosr, rosnodejs, roscs, rosgo, rosruby, roslua... et d'autres bibliothèques Reference23.

B.2 Philosophie du ROS

ROS a trois niveaux de concepts : le niveau du système de fichiers (*ROS File- system Level*), le niveau du graphe de calcul (*ROS Computation Graph Level*) et le niveau communautaire (*ROS Community Level*). Ces niveaux et ces concepts sont résumés ci-dessous.

B.3 Niveau du système de fichiers ROS

Les concepts de niveau de système de fichiers couvrent principalement les ressources ROS que vous rencontrez sur le disque, telles que :

Les paquets : (*Packages*) sont l'unité principale pour l'organisation de logiciels, ils sont l'élément le plus atomique dans ROS.

Un paquet peut contenir des processus d'exécution ROS (nœuds - nodes), une bibliothèque dépendant de ROS, des données, des fichiers de configuration, ou tout autre élément organisé de manière ultime.

Le manifeste du paquet :(*Manifest*, le fichier package.xml dans un paquet) fournit des méta-données sur un paquet, y compris son nom, sa version, sa description, ses informations de licence, ses dépendances et d'autres méta-informations. // La structure de fichier manifeste package.xml est définie dans REP-0127Reference24.

Les dépôts :un dépôt (*Repository*) est une collection de paquets qui partagent un système de contrôle de version (VCS) commun. Les paquets qui partagent un VCS partagent la même version et peuvent être diffusés ensemble à l'aide de l'outil d'automatisation bloom Reference25 de catkin.

Les types des messages (*msg*) : contient la description des messages, définissent les structures de données pour les messages envoyés dans ROS, stockés dans : my-package/msg/MyMessageType.msg

Les type des services (*Srv*) : contient la description de service, définissent les structures de données de demande et de réponse pour les services dans ROS, stockés dans : my-package/srv/MyServiceType.srv

Les méta-paquets :(*Metapackages*)

B.4 Niveau de traitement du ROS

Sous ROS, le Computation Graph est le réseau peer-to-peer des processus qui traitent ensemble les données. Les concepts de Computation Graph de base sont des nœuds *nocdes*, le nœud principale (*Master*), les serveurs de paramètres(*Parameter Server*) , les messages (*Messages*), les services (*Services*), les topics et les bags, qui fournissent tous des données au Graphique de différentes façons.

Ces concepts sont implémentés dans le dépôt (repository) ros-comm.

B.4.1 Les nœuds

Les nœuds (nodes) sont des processus qui effectuent le traitement. ROS est conçu pour être modulaire à une échelle fine. Un système de contrôle de robot comprend

généralement de nombreux nœuds. Par exemple, un nœud contrôle le capteur laser, un autre nœud contrôle les moteurs des roues, un nœud effectue la localisation, un nœud effectue la planification du chemin, un nœud fournit une vue graphique du système, etc. Un nœud ROS est écrit avec l'utilisation d'une bibliothèque client ROS, comme roscpp ou rospy.

Le lancement d'un nœud sous ROS se fait par la commande rosrun ou bien roslaunch.

B.4.2 ROS Master

L'un des objectifs de base de ROS est de permettre aux roboticiens de concevoir des logiciels comme une collection de petits programmes, principalement indépendants (les nœuds), qui fonctionnent tous en même temps. Pour que cela fonctionne, ces nœuds doivent pouvoir communiquer entre eux. La partie de ROS qui facilite cette communication s'appelle le*ROS Master*.

B.4.3 Le serveur de paramètres

Le serveur de paramètres (*Parameter Server*) permet de stocker des données par clé dans un emplacement central. Il fait actuellement partie du *Master*.

B.4.4 Les messages

Les nœuds communiquent entre eux en transmettant des messages. Un message est simplement une structure de données, comprenant des champs saisis.

Les types primitifs standard (les entiers int, les nombres en virgule flottante float, les boolean bool, ... etc.) sont pris en charge, de même que des tableaux de types primitifs. Les messages peuvent inclure des structures et des tableaux arbitrairement imbriqués (tout comme les structures du langage C).

B.4.5 Les sujets

Les messages sont acheminés avec la sémantique de publication / souscrire (*publish / subscribe*). Un nœud envoie un message en le publiant sur un sujet *topic.* donné. Le sujet est un nom utilisé pour identifier le contenu du message. Un nœud qui s'intéresse à un certain type de données s'abonnera (*subscribe*) au sujet approprié. Il peut y avoir plusieurs éditeurs et abonnés simultanés pour un seul sujet, et un seul nœud peut publier et/ou s'abonner à plusieurs sujets.

Logiquement, on peut penser à un sujet comme un bus de message fortement typé. Chaque bus a un nom et n'importe qui peut se connecter au bus pour envoyer ou recevoir des messages tant qu'ils sont du bon type.

B.4.6 Les sacs

Les sacs (*bags*) sont un format pour sauvegarder et lire les données du message ROS. Les sacs sont un mécanisme important pour stocker des données, telles que des données de capteurs, qui peuvent être difficiles à collecter, mais nécessaires pour développer et tester des algorithmes.

B.5 Niveau communautaire de ROS

Les concepts ROS Community Level sont des ressources ROS qui permettent aux communautés séparées d'échanger des logiciels et des connaissances. Les principaux concepts du niveau communautaire sont :

B.5.1 Les distributions

Les distributions du ROS sont des collections d'applications versionnées que vous pouvez installer. Les distributions jouent un rôle similaire aux distributions Linux : elles facilitent l'installation d'une collection de logiciels, et elles main- tiennent également des versions cohérentes sur un ensemble de logiciels.

B.5.2 Les dépôts

ROS repose sur un réseau fédéré de dépôts de code, où différentes institutions peuvent développer et publier leurs propres composants de logiciel de robot.

B.5.3 Le Wiki ROS

Le Wiki ROS est le principal forum pour documenter des informations sur ROS. Toute personne peut s'inscrire à un compte et apporter sa propre documentation, fournir des corrections ou des mises à jour, écrire des tutoriels et plus encore.

B.6 Python

Comme c'est déjà cité, le ROS supporte le langage de programmation Python officiellement a travers la bibliothèque du client ROS rospy. Python est un langage de programmation objet, multi-paradigme et multi- plateformes. Il favorise la programmation impérative structurée, fonctionnelle et orientée objet. Il est doté d'un typage dynamique fort, d'une gestion automatique de la mémoire par ramasse-miettes et d'un système de gestion d'exceptions ; il est ainsi similaire à Perl, Ruby, Scheme, Smalltalk et Tcl. Le langage Python est placé sous une licence libre proche de la licence BSD et fonctionne sur la plupart des plates_ formes informatiques, des supercalculateurs aux ordinateurs centraux Reference7, de Windows à Unix avec notamment GNU/Linux en passant par macOS, ou encore Android, iOS, et aussi avec Java ou encore .NET. Il est conçu pour optimiser la productivité des programmeurs en offrant des outils de haut niveau et une syntaxe simple à utiliser.

FIGURE B.3: Logo officiel de l'implémentation référentielle du langage de programmation Python (CPython).

B.7 La bibliotheque NumPy

NumPy est une bibliothèque logicielle open source qui offre une extension du langage de programmation Python, destinée à manipuler des matrices ou tableaux multidimensionnels ainsi que des fonctions mathématiques opérant sur ces tableaux. Plus précisément, cette bibliothèque fournit de multiples fonctions permettant notamment de créer directement un tableau depuis un fichier ou au contraire de sauvegarder un tableau dans un fichier, et manipuler des vecteurs, matrices et polynômes.

Bibliographie

[1] R. Guyonneau. Set-membership approaches for mobile robot localisation. phd thesis, université d'angers, nov. ,.

[2] Bouraine S. "contribution à la planification de mouvements en environnements dynamiques pour des robots mobiles type voiture cas robucar". pp.218, (2016).

[3] Kelly A. Howard T., Green C. and Ferguson D. "state space sampling of feasible motions for highperformance mobile robot navigation in complex environments", journal of field robotics, v. 25, no 6–7, (2008), pages 325– 345.

[4] Hokuyo ust-10lx-autonomousstuff [en ligne] https ://autonomoustuff.com/wp-content/uploads/2019/05/ust-10lx-whitelabel.pdf, (consulté le 23/04/2019).

[5] Nilsson N.J. Hart P.E. and Raphael B. "a formal basis for the heuristic determination of minimum cost paths", ieee trans. on systems science and cybernetics, v. 4, issue 2, (1968), pages 100–107..

[6] Dijkstra E. W. A note on two problems in connexion with graphs‖, numerische mathematik, v. 1, issue 1, (1959), pages 269-271.

[7] Petrovic I. Seder M. "dynamic window based approach to mobile robot motion control in the presence of moving obstacles", in : Ieee int. conf. robotics and automation, roma (it), (2007), pages 1986-1991.

[8] Fiorini P. and Shiller Z. "motion planning in dynamic environments using velocity obstacles", int. journal robotics research v. 17, issue 7, (1998), pages 760-772.

[9] Laugier C. Large F. and Shiller Z. "navigation among moving obstacles using the nlvo : Principles and applications to intelligent vehicles", autonomous robots, v. 19, (2005), pages 159-171.

[10] S.Petti and T. Fraichard. "safe motion planning in dynamic environments", proc. of the ieee-rsj int. conf. on intelligent robots and systems, (2005), pages 2210-2215.

[11] Petti S. and Fraichard T. "partial motion planning framework for reactive planning within dynamic environments", proc. of the ifac/aaai int. conf. on informatics in control, automation and robotics, (2005), pages 199-204.

[12] C Drocourt. Localisation et modélisation de l'environnement d'un robot mobile par coopération de deux capteurs omnidirectionnels. phd thésis, université de technologie de compiègne, centre de robotique, d'electrotechnique et d'automatique,. 2002.

[13] G. Frappier. Système inertiels de navigation pour robots mobiles. in séminaire ≪ les robots mobiles ≫ ec2, paris. ,. 1990.

[14] F. Zhao, Y. Liu and R. Wang. Localisation technology of indoor robot based on laser sensor. in software engineering and service science (icsess), 2016 7th ieee international conference on (2016), ieee, pp.683-686.

[15] D.E. Zlotnik and J.R. Forbes. Exteroceptive measurement filtering embedded within an so (3)-based attitude estimator. in decision and control (cdc), 2016 ieee 55th conference on (2016), ieee, pp. 296-301.

[16] A. Elfes. Using occupancy grids ofr mobile robot perception and navigation. computer 22, 6 (june 1989), 46-57.

[17] A. Elfes. "using occupancy grids for mobile robot perception and navigation", ieee computer, pp.46-57, los alamitos, usa, 1989.

[18] J.L. Crowley. "world modelling and position estimation for a mobile robot using ultrasonic ranging", proceedings of the ieee international conference on robotics and automation. 1998.

[19] N. Tomatis K.O. Arras. “improving robustness and precision in mobile robot localisation by using laser range finding and monocular vision”, 3rd european workshop on advanced mobile robots (eurobotb ”99), zurich, switzerland, september 6-8, 1999.

[20] T. Duckett and U. Nehmzow. “experiments in evidence based localization for a mobile robot”, in d. corne and j. l. shapiro, editors, proceedings of the aisb 97 workshop on spatial reasoning in animals and robots, springer, 1997.

[21] X. Li and H. Qiu. An effective laser-based approach to build topological map of unkown environment. in robotics and biomimetics (robio), 2015 ieee international conference on (2015), ieee, pp.200-205.

[22] S. Thrun. “robotic mapping”, a survey, school of computer science, carnegie mellon university, pittsburgh, february 2002.

[23] Van den Berg J. and Overmars M. ”planning time-minimal safe paths amidst unpredictably moving obstacles”, int journal robotics research v. 27, (2008), pages 1173-1174.

[24] Tsianos K. Bekris K. and Kavrak L. ”safe and distributed kinodynamic replanning for vehicular networks”, mobile networks and applications v. 14,issue 3, (2009), pages 292-308.

[25] Khatib O. ”real-time obstacle avoidance for manipulators and mobile robots”, international journal of roboticsresearch, v. 5, issue 1, (1986), pages 90-98.

[26] Petti S. and Fraichard T. ”safe navigation of a car-like robot in a dynamic environment”, in proc. of the european conf. on mobile robots, (2005).

[27] Fraichard T. Benenson R., Petti S. and Parent M. ”integrating perception and planning for autonomous navigation of urban vehicles”, in ieee-rsj int. conf. on intelligent robots and systems (iros), (2006), pages 98-104.

[28] Fraichard T. and Howard T. ”iterative motion planning and safety issue”, handbook of intelligent vehicles, springer, (2012), pages 1433-1458.

[29] Khatib O. "real-time obstacle avoidance for manipulators and mobile robots", phd thesis, laas-cnrs, (1996).

[30] Brock O. and Khatib O. "real time replanning in high-dimensional configuration spaces using sets of homotopic paths", proc. ieee intl. conf. on robotics and automation, v. 1, (2000), pages 550-555.

[31] Chatila R. Khatib M., Jaouni H. and Laumond J.-P. "dynamic path modification for car-like nonholonomic mobile robots", in proceedings ieee international conference on robotics and automation, v. 4, (1997), pages 2920–2925.

[32] Bonnafous D. Lamiraux F. and Lefebvre O. "reactive path deformation for nonholonomic mobile robots", ieee transactions on robotics and automation, v. 20, issue 6, (2004), pages 967-977.

[33] Chen H. and Allgower F. "a quasi-infinite horizon nonlinear model predictive control scheme with guaranteed stability," automatica, v.34, no.10, (1998), pages 1205–1217.

[34] Rao C. Mayne D., Rawlings J. and Scokaert P. "constrained model predictive control : Stability and optimality," automatica, v.36, no.6, (2000), pages 789-814.

[35] Seron M. M. Goodwin G. C. and De Doná J. A. "constrained control and estimation : an optimization approach", livre, springer, (2005).

[36] Kwok N. Leung C., Huang S. and Dissanayake G. "planning under uncertainty using model predictive control for information gathering," robotics and autonomous systems, v.54, (2006), pages 898-910.

[37] Bouraine S. "contribution à la localisation dynamique du robot mobile dintérieur b21r en utilisant la plateforme multi sensorielle", (november 2007).

[38] Ubuntu [en ligne] https ://ubuntu.com , (consulté le 04/02/2019).

[39] Robot operating system [en ligne]] http ://www.ros.org , (consulté le 04/02/2019).

[40] Open robotics, osrf [en ligne]] https ://www.openrobotics.org , (consulté le 13/02/2019).

[41] Ros-drivers urg-node [en ligne] https ://github.com/ros-drivers/urg$_node$, (*consultéle*23/04/2019).