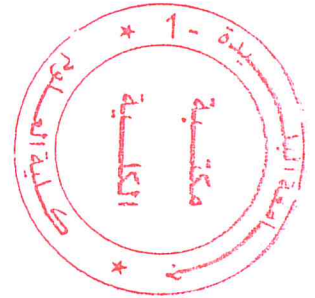




Ministère de l'Enseignement Supérieur
Et de la Recherche scientifique
Université SAAD DAHLEB de BLIDA



Faculté des sciences
Département d'Informatique



Mémoire de fin d'étude
Présenté en vue d'obtenir le diplôme de Master

Thème

**Planification de chemins optimaux par l'approche RRT
pour des robots mobiles**

Organisme d'accueil : Centre de Développement des Technologies Avancées (CDTA)



Présenté par :

-Yahiaoui Djelloul

Encadreur :

-M. Hentout Abdelfetah

Promoteur :

-M. Kameche Abdallah Hicham

Année universitaire : 2017/2018

Remerciements

Merci à Dieu de sa grâce, source de notre force et courage tout au long de nos études universitaires.

Je tiens à exprimer mes vifs remerciements et mes profondes gratitude à **Mr HENTOUT**, de m'avoir encadré et dirigé dans mon mémoire de fin d'étude.

Je remercie également **Mr KAMECHE** pour son aide précieuse et son encouragement qu'il n'a cessé de me communiquer.

Je remercie les membres du jury pour avoir accepté d'évaluer notre travail.

Un grand merci également à ma famille pour leur soutien aussi bien moral que financier et pour leurs sacrifices.

je ne pourrais terminer sans remercier tous ceux qui ont participé d'une manière ou d'une autre dans l'élaboration de ce projet de fin d'études.

Dédicaces

Que ce travail témoigne de mes respects :

A mes parent mon père **Ahmed** et ma chère mère qui m'ont toujours poussé et motivé dans mes études. Grâce à leurs tendres encouragements et leurs grands sacrifices, Je prie le bon Dieu de les bénir, de veiller sur eux, en espérant qu'ils seront toujours fiers de moi

A mes frères **Mohamed, Samir et Youcef.**

A mes sœurs **Karima, Amel, Fouzia et Fethia.**

A ma chère **Hind.**

A mon beau-frère **Khalil.**

Ils vont trouver ici l'expression de mes sentiments de respect et de reconnaissance pour le soutien qu'ils n'ont cessé de me porter

A tous mes Amis **Oussama, Ghano, Salim, Islem, Adlen, Anis, Ali, Asaad, Hicham, Toufik, Djilali, Mehdi, Rede, Rafik, Ilyes, Sami, Minato, Djimy.**

Ils vont trouver ici le témoignage d'une fidélité et d'une amitié infinie

A tous ceux qui m'ont aidé et soutenu pendant tout mon cursus universitaire.

ملخص

إن تخطيط المسارات الممكنة لروبوت متنقل مهمة جدًا في مجال الروبوتات المتنقلة. ليتمكن من العثور على مسار يربط بين نقطة البداية ونقطة النهاية في البيئة مع تجنب العقبات. تحقيقاً لهذه الغاية، توجد العديد من الأنظمة والتكنولوجيات لمعالجة مشكلة تخطيط المسار ولكن هناك مشكلة أساسية أخرى وهي تحسين هذا المسار.

الهدف من هذا المشروع هو التخطيط لمسار خالٍ من التصادم باستخدام طريقة (RRT) للقواعد المتنقلة التفاضلية في البيئات الداخلية التي تشوشها العوائق الثابتة. بيئة تطور الروبوتات المتنقلة تستخدم نظام رؤية الذي بدوره يتكون من كمرتين مايكروسوفت، Kinect V1.

المسار المثالي الذي تم إنشاؤه بدون أي عقبات يجب عليه ربط موضع البداية بموضع النهاية المفروض، لهذا الغرض، معيار الأداء الذي يجب تحسينه سيضم ثلاث معاملات مع عوامل ترجيح مختلفة:

- طول المسار الذي تم إنشاؤه.
- الانحراف الكلي.
- خطأ التصادم مع العقبات.

Résumé

La planification des chemins faisables pour un robot mobile est une tâche très importante dans la robotique mobile. Elle permet de trouver un chemin qui relie une configuration initiale à une configuration finale dans l'environnement tout en évitant les obstacles. À cet effet, de nombreux systèmes et technologies existent pour répondre au problème de la planification de chemin mais il y a un autre problème fondamental qui est l'optimisation de ce chemin.

Le but de ce projet est de planifier un chemin optimal sans collision en utilisant la méthode **Rapidly-Exploring Random Trees (RRT)** pour des bases mobiles dans des environnements d'intérieur encombré d'obstacles statiques. L'environnement d'évolution des robots mobile est modélisé en exploitant un système de vision constitué de deux caméras Microsoft, **Kinect V1**.

Le chemin optimal sans collision généré doit connecter une position initiale à une position finale imposée. À cet effet, le critère de performance à optimiser regroupe trois paramètres avec différents facteurs de pondération :

- La longueur de chemin généré.
- La déviation totale.
- L'erreur de collision avec les obstacles.

Mots clé : RRT, A*, planification, robot mobile, optimisation, navigation.

Abstract

Planning paths for a mobile robot is a very important task in mobile robotics. It can find a path that connects an initial configuration to a final configuration in the environment while avoiding obstacles. To this end, many systems and technologies exist to solve the problem of path planning but there is another fundamental problem which is, the optimization of this path.

The purpose of this project is to plan an optimal collision-free path by using the Rapidly-Exploring Random Trees (RRT) method for differential mobile bases in indoor environments cluttered with static obstacles. The evolution environment of mobile robots is modeled using a vision system consisting of two Microsoft cameras, Kinect V1.

The generated optimal path with no collision must connect an initial position to an imposed end position. For this purpose, the performance criterion that should be optimized will include three parameters with different weighting factors:

- The path length generated.
- The total deviation.
- The collision error with the obstacles.

Keywords : RRT, A*, Planification, path planning, mobile robotics, optimization, navigation

Table des matières

Remerciements	I
ملخص	III
Résumé	IV
Abstract	V
Introduction générale	9
1. Introduction.....	12
2. Définitions	12
3. Différents aspects de la planification.....	13
4. Différents types de planification.....	14
5. Conclusion	20
1. Introduction.....	22
2. Principe de l'algorithme RRT	22
2.1. Algorithme de construction d'arbre	22
2.2. Algorithme d'extension de l'arbre	22
2.3. Ajouter un nouveau nœud	24
2.4. Vérification de la taille du pas.....	25
2.5. Vérification de l'intersection.....	26
2.6. Critère d'arrêt de construction du RRT	26
2.7. Sélection du chemin réalisable	26
3. Conclusion	27
1. Introduction.....	29
2. Architecture du système	29
3. Description de l'architecture.....	30
3.1. Processus de planification	30
3.2. Principe de l'algorithme A*	30
3.3. Principe de RRT-A*.....	32
3.6. RobuterULM sous Gazebo:	33
4. Conclusion	34
1. Introduction.....	36

2. Langages de développement.....	36
3. Modélisation de l'environnement.....	39
4. Planification de trajectoires	43
5. Conclusion	49
Conclusion générale et perspectives	50
Bibliographie	52
Annexe A.....	55
Annexe B	60
Annexe C	62

Liste des Figures

Figure 1 : Plan solution pour le problème “faire du café” [4].	14
Figure 2 : Planification de mouvements fins [5].	14
Figure 3 : Classification des approches de planification de chemins [9].	15
Figure 4 : Fonctionnement de la méthode de passage de coins : (a) image des obstacles, (b) forme binaire des of obstacles, (c) chemin planifié (chemin rouge est optimal) [10].	16
Figure 5 : Représentation d’un exemple d’espace de travail (road block obstacle) [12]	17
Figure 6 : Conduite sur route avec Lattice planner [14].	18
Figure 7 : Construction incrémentale de l’algorithme RRT [17].	19
Figure 8 : Phase d’extension d’un RRT [17].	23
Figure 9 : Ajouter un nouveau sommet au graphe.	25
Figure 10 : Vérification de la distance entre qnearest et grand.	25
Figure 11 : Vérification de l’intersection entre qnear et grand.	26
Figure 12 : Sélection du chemin réalisable.	27
Figure 13 : Schéma synoptique du planificateur développé.	30
Figure 14 : Algorithme de planification proposé.	32
Figure 15: visualisation de RobuterULM sous Gazebo.	33
Figure 16 : Banc d’essai robotique expérimental.	36
Figure 17 : Environnement d’évolution du robot mobile.	40
Figure 18 : Images acquises par les deux caméras Kinect.	41
Figure 19 : Carte binaire de l’environnement global.	41
Figure 20 : Carte binaire sûre de l’environnement global	42
Figure 21 : Approximation par des polygones de l’environnement 1 d’évolution du robot mobile.	42
Figure 22: Approximation par des polygones de l’environnement 2 d’évolution du robot mobile.	43
Figure 23 : Planification RRT-A* avec une taille de pas de 8 pixels.	44
Figure 24 : Planification RRT avec une taille de pas de 35 pixels	44
Figure 25 : Planification RRT avec une taille de pas de 60 pixels	45
Figure 26 : Comparaison entre RRT-A* et RRT.	47
Figure 27: Le niveau du système de fichiers ROS	57
Figure 28: Structure de package typique	58
Figure 29: ROS calcul Niveau graphique	58
Figure 30: Le test de kinect dans le simulateur Rviz	62
Figure 31: L'architecture du système robotique expérimental.	63

Liste des Tableaux

Tableau 1 : Comparaison entre les déferant méthodes de planification	20
Tableau 2 : Résumé des principaux résultats obtenus	45
Tableau 3 : Résumé des principaux résultats obtenus	46
Tableau 4 : Comparaison entre RRT-A* et RRT	47
Tableau 5 : Comparaison entre RRT-A* et RRT pour différents espaces de travail avec différentes dimensions et nombre d'obstacles	48
Tableau 6: comparaison RRT et RRT-A*	48

Introduction générale

Un robot est une machine équipée de capacités de perception, de décision et d'action qui lui permettent d'agir de façon autonome dans son environnement. Les robots sont largement utilisés pour aider ou éventuellement remplacer l'être humain dans plusieurs domaines : industriel, militaire, chirurgical, ainsi que dans la vie quotidienne où ils peuvent réaliser des tâches de nettoyage et d'entretien ou encore fournir de l'assistance à une personne handicapée.

Un robot doté de capacité de perception et d'information sur son environnement, doit pouvoir se mouvoir de façon autonome (localisation, planification et navigation), tout en évitant les obstacles. Une des tâches que doit accomplir le robot est la planification de sa trajectoire dans son environnement d'évolution.

La planification automatique de trajectoires en robotique sert à trouver une suite de mouvements valides/faisables qui permettent à un robot de passer d'un état initial à un état final désiré. D'une manière générale, un mouvement valide/faisable est un déplacement qui ne produit pas de collision et qui respecte les contraintes cinématiques du robot [1]. Si la planification de trajectoires semble relativement facile à faire pour un opérateur humain, leur détermination automatique est en fait un problème très complexe.

Depuis une vingtaine d'années, de nombreuses techniques de planification automatique de trajectoires ont été proposées ; néanmoins, aucune d'entre elles ne s'est véritablement imposée comme une méthode générale pouvant résoudre de façon satisfaisante ce problème (trouver une trajectoire, minimiser la fonction coût, etc.), notamment au niveau de l'industrie. En première approximation, on peut distinguer deux types de méthodes : celles qui cherchent à construire une représentation de l'espace libre (méthodes globales) et celle qui se basent sur des informations locales pour construire de façon incrémentale une trajectoire (méthodes locales) [1].

L'avantage du premier type d'approches (méthodes globales) est la complétude ; c'est-à-dire une solution sera trouvée s'il en existe une. L'inconvénient est le temps de calcul qui peut croître de façon exponentielle avec le nombre de degrés de liberté, ce qui limite en pratique leurs utilisations. Quant aux méthodes locales, elles permettent de résoudre les problèmes fondamentaux de la planification dans un temps raisonnable.

Dans ce cadre, nous nous intéressons à présenter une méthode de planification de trajectoires, qui se base sur l'algorithme RRT (Rapidly-exploring Random Trees). Notre méthode de planification de trajectoires devra proposer une solution sûre et efficace au problème fondamental de la planification et même au problème d'optimisation. Dans ce travail, le chemin est défini à partir de la modélisation de l'environnement du robot par RRT. Ce chemin est équivalent à une fonction coût définie comme étant une somme de deux énergies, l'une est liée à la pénalité de collision générée par chaque obstacle, et l'autre est la pénalité de la longueur totale de la trajectoire.

Pour assurer une meilleure présentation du travail effectué et garantir la clarté de ce document, outre cette introduction générale, ce manuscrit se compose de quatre chapitres et une conclusion générale ; chacun met en évidence une contribution particulière du travail effectué :

- **Chapitre I** : ce chapitre présente les approches de planification de chemin existants avec leurs différentes méthodes. Il décrit aussi leurs principaux objectifs ainsi que leurs avantages et inconvénients.
- **Chapitre II** : ce chapitre présente l'approche de planification de chemin Rapidly-exploring Random Tree (RRT) dont on s'est basé pour la mise en œuvre de notre solution, ce chapitre explique aussi son fonctionnement ainsi que ces différentes APIs.
- **Chapitre III** : ce chapitre est dédié à l'étude conceptuelle ainsi que la réalisation de l'approche développée.
- **Chapitre IV** : ce chapitre est consacré aux tests et validation.

Chapitre 1

Approches de planification de chemin

1. Introduction

La planification de chemins est l'un des problèmes fondamentaux de la robotique mobile. Comme mentionné par Latombe [2], la capacité de planifier efficacement ses mouvements est « éminemment nécessaire puisque, par définition, un robot accomplit des tâches en se déplaçant dans le monde réel ». En particulier dans le contexte de robots mobiles autonomes, les techniques de planification de chemins doivent simultanément résoudre deux tâches complémentaires. D'une part, leur tâche consiste à minimiser la longueur du chemin de la position de départ à l'emplacement cible ; d'autre part, ils devraient maximiser la distance aux obstacles afin de minimiser le risque de collision.

Il existe différentes approches qui ont été introduites dans le but d'implémenter la planification de chemins pour un robot mobile [3]. Elles sont en fonction de l'environnement, du type de capteurs, des capacités du robot, etc. Dans ce chapitre, nous allons présenter les principaux algorithmes de planification de chemins et discuter de leurs avantages et inconvénients.

2. Définitions

Dans cette section, nous allons donner les définitions des principaux termes utilisés dans ce travail :

- *Planification* : elle sert à donner les moyens au robot de trouver une suite d'actions à appliquer sur le monde, pour le faire passer de l'état initial à un état qui satisfait le but à atteindre [4].
- *Trajectoire ou plan* : c'est un ensemble structuré d'actions qui mènent au but. La détermination de cette trajectoire se fait tout en respectant un certain nombre de contraintes et de critères [4]:
 - *Contraintes relatives au robot* : elles concernent la géométrie, la cinématique, la dynamique et même l'architecture (type) du robot. En effet, le robot peut correspondre à un système articulé d'objets rigides (bras manipulateur), un robot mobile, une main à plusieurs doigts, ou encore à plusieurs systèmes de robots à coordonner tels que des bras manipulateurs ou des robots mobiles.
 - *Contraintes de l'environnement* : elles concernant essentiellement la non-collision avec les obstacles, la prise en compte d'interactions de contact avec le robot, etc.

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

3. Différents aspects de la planification

La planification présente plusieurs aspects importants ; ils sont énumérés dans ce qui suit [4]:

- **Résolution automatique du problème** : la solution est l'assemblage d'un ensemble d'actions élémentaires élaboré lors de la planification et qui permet au robot d'atteindre l'objectif.
- **Prévision** : une trajectoire ou un plan est une prévision d'une séquence d'actes à accomplir. Lors de l'exécution, la surveillance du monde, permet de vérifier que les actions sont bien exécutées et que le plan mène vers la solution.
- **Création d'une nouvelle connaissance** : une fois le plan (trajectoire) est élaboré, il peut être conservé pour le réutiliser lorsqu'un problème similaire se posera (domaine d'apprentissage). Le problème « *faire du café* », un plan solution pourrait être [4]:
 - **Action 1** : prendre le café moulu.
 - **Action 2** : le mettre dans la cafetière.
 - **Action 3** : remplir d'eau dans la cafetière.
 - **Action 4** : mettre l'interrupteur de la cafetière sur la position "Marche".
 - **Action 5** : attendre que toute l'eau soit passée dans la cafetière.
 - **Action 6** : arrêter la cafetière.

Si on utilise les deux mains, **Action 3** s'exécute en même temps que les **Actions 1** et **2**. Le plan est schématisé de la façon suivante (Figure 1) :

- **Résolution automatique du problème** : dans l'exemple, le problème "faire du café" est résolu par le plan composé de 6 actions élémentaires de 1 à 6.
- **Prévision** : cet exemple, permet de prévoir la suite d'actions à effectuer pour préparer du café ; si les actions 1 à 3 sont réalisées, il faut vérifier que la cafetière est à l'état « Marche ».
- **Création d'une nouvelle connaissance** : le plan de solution de cet exemple conservé, peut être réutilisé pour résoudre un problème similaire.

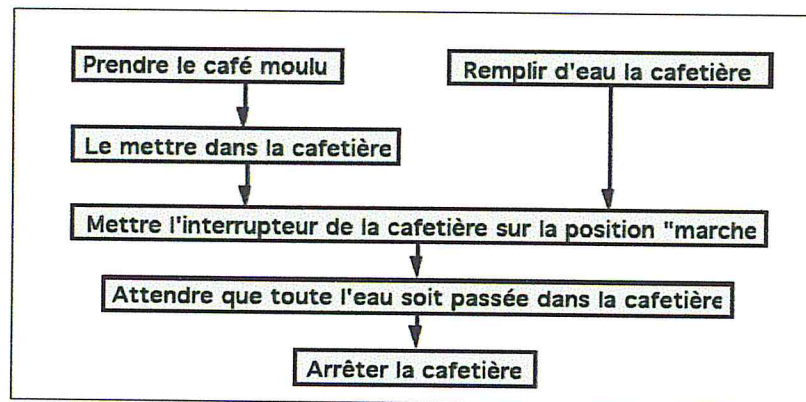


Figure 1 : Plan solution pour le problème “faire du café” [4].

4. Différents types de planification

Cette section décrit les deux types de planification existantes dans la littérature [4]:

4.1. Planification de mouvements fins

La saisie délicate ou l’insertion précise de pièces mécaniques, demandent souvent une stratégie particulière constituée d’une série de petits mouvements représentant des essais, des erreurs et des corrections de faibles amplitudes (Figure 2).

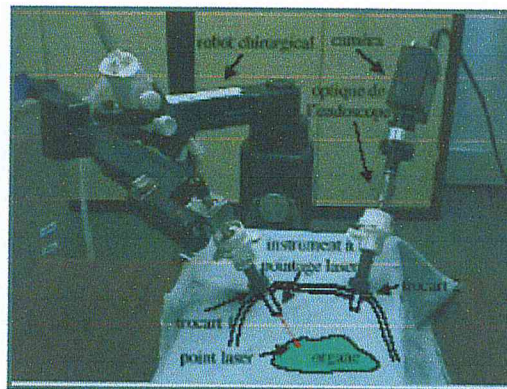


Figure 2 : Planification de mouvements fins [5].

4.2. Planification du chemins (trajectoires)

Dans l’univers géométrique où évolue le robot, il est indispensable de pouvoir déterminer à l’avance le parcours du robot ou la trajectoire d’un bras manipulateur afin d’éviter les collisions.

Ce type de planification, est utilisé par plusieurs systèmes tels que *Handey* du MIT et *NSS* du LAAS.

Le problème de base de planification de mouvements est de sélectionner un chemin ou une trajectoire [6] [7], d'un état initial donné à un état final cible. Les méthodes existantes de planification peuvent être classées en deux approches :

- **Planification en ligne** : le chemin vers la cible est calculé de façon incrémentale pendant le mouvement. Ces méthodes utilisent des informations incomplètes sur l'environnement et construisent le chemin étape par étape. Ainsi, ce type permet une détection rapide de collisions et une planification rapide de chemins. Ces approches sont requises dans des applications où des obstacles sont détectés pendant le mouvement [8].
- **Planification hors ligne** : cela consiste à générer le chemin complet de la position initiale à la position cible avant que le mouvement ne commence. Ces approches utilisent des informations complètes sur l'espace de travail ; aussi, un critère d'optimisation peut être utilisé pour rechercher le chemin optimal sans collision. Elles sont très utiles pour des tâches répétitives dans des environnements statiques où l'optimalité est un critère essentiel (applications industrielles, etc.) [9].

Le diagramme suivant donné par la figure 3 représente la classification des méthodes existantes de planification de chemins.

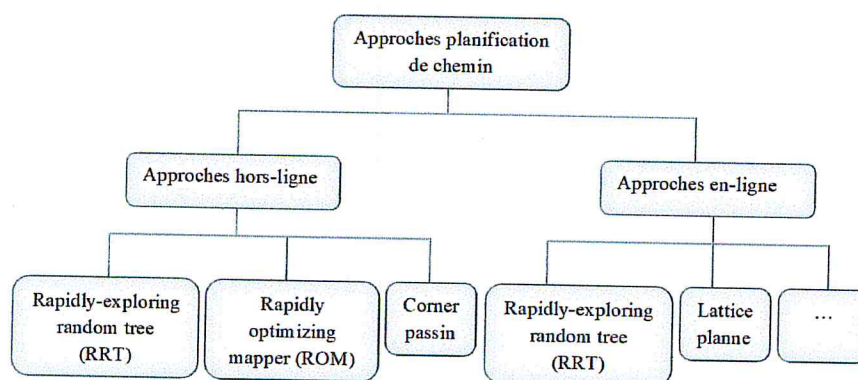


Figure 3 : Classification des approches de planification de chemins [9].

Dans la suite, nous allons nous focaliser uniquement sur les approches de planification de chemins hors ligne et en ligne pour les robots mobiles.

4.2.1. Méthode de passage de coins

La méthode de dépassement de coins est basée sur l'image binaire créée à partir d'obstacles (Figure 4). D'abord, une ligne est tracée du point de départ vers la cible ; si un obstacle existe dans le chemin, deux chemins du point de départ seraient considérés au coin le plus extérieur de l'objet. Le même processus pour chaque chemin devrait être répété jusqu'à atteindre la destination sans aucune collision avec des obstacles. La distance de chaque chemin serait calculée ; le chemin ayant la distance la plus courte sera considéré comme optimal. Cette méthode nécessite beaucoup de calcul. En effet, à chaque étape, les coins les plus à l'extérieur de l'objet doivent être calculés mais, cela donne le chemin optimal [10].

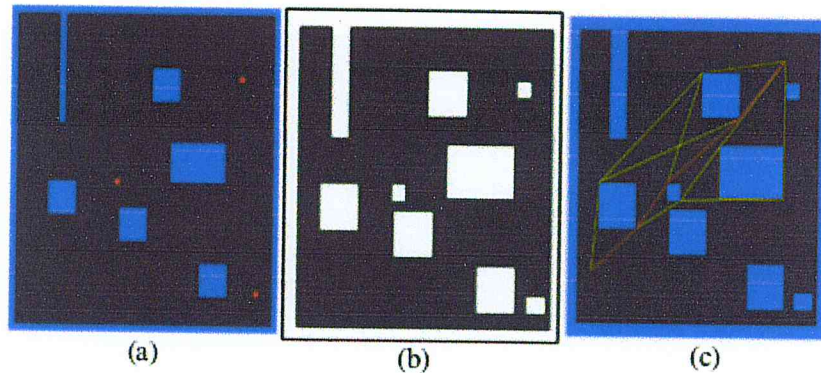


Figure 4 : Fonctionnement de la méthode de passage de coins : (a) image des obstacles, (b) forme binaire des obstacles, (c) chemin planifié (chemin rouge est optimal) [10].

4.2.2. Rapidly-Optimizing Mapper (ROM)

L'optimisation rapide du mappeur (ROM) est une solution hors-ligne pour un robot point holonome évoluant dans un environnement statique. ROM est capable de planifier un chemin sans collision depuis le point de départ jusqu'à la configuration finale en utilisant une série de méthodes et de paramètres différents pour analyser l'espace de travail et calculer le plus court chemin sans collision. La stratégie de ROM se concentre sur un nombre moindre d'obstacles au lieu de considérer tous les obstacles de l'environnement afin d'obtenir une meilleure performance et de construire un chemin optimal [11]. Le planificateur ROM comprend quatre phases distinctes¹:

¹ <https://www.omicsgroup.org/journals/a-robotic-path-planner-contender-2168-9695-1000131.php?aid=60230&view=mobile>

4.2.3. Lattice planner

Lattice planner se compose d'un ensemble d'états reliés par des arêtes. Pour construire l'ensemble des états, chaque dimension du domaine de planification est discrétisée en cellules de taille finie² [13]. Typiquement, les dimensions métriques sont divisées en petites grilles ; il est représenté comme une série d'états connectés dans un graphe basé sur un *lattice* construit en utilisant des primitives de mouvement et des algorithmes de recherche efficaces.

Un graphe basé sur un treillis est construit en utilisant des primitives de mouvements courts comme des arêtes qui se retrouvent au centre des cellules (Figure 6). Les primitives de mouvements sont des mouvements courts, réalisables sur le plan cinématique, qui constituent la base de mouvements pouvant être effectués par la plateforme robotique [14].

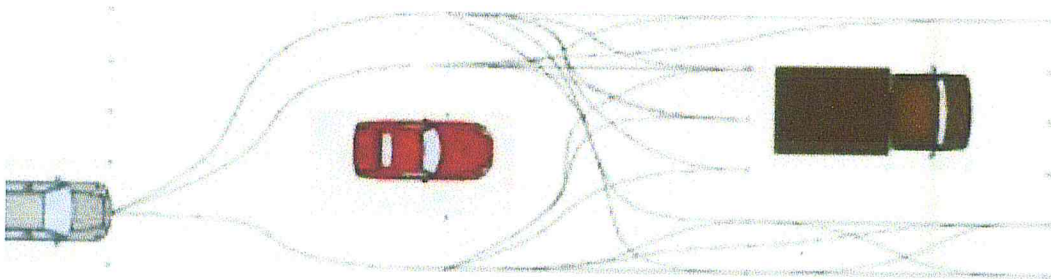


Figure 6 : Conduite sur route avec *Lattice planner* [14].

4.2.4. Rapidly-exploring Random Tree (RRT)

RRT est un algorithme conçu pour rechercher efficacement des espaces de grande dimension non convexes [15]. RRT sont construits de manière incrémentale qui réduit rapidement la distance attendue d'un point choisi au hasard à l'arbre. Les RRT sont particulièrement adaptés aux problèmes de planification de chemins qui impliquent des obstacles et des contraintes différentielles (non-holonomie ou kino-dynamiques). Les RRT peuvent être considérés comme une technique permettant de générer des chemins en boucle ouverte pour des systèmes non linéaires soumis à des contraintes d'état [16].

RRT construit une arborescence en utilisant un échantillonnage aléatoire dans l'espace de recherche (Figure 7). L'arbre commence à partir d'un état initial et se développe pour trouver un

² <http://sbpl.net/node/53>.

chemin vers le but. L'arbre se développe progressivement à mesure que l'itération se poursuit. Lors de chaque itération, un état aléatoire est sélectionné dans l'espace de configuration [4].

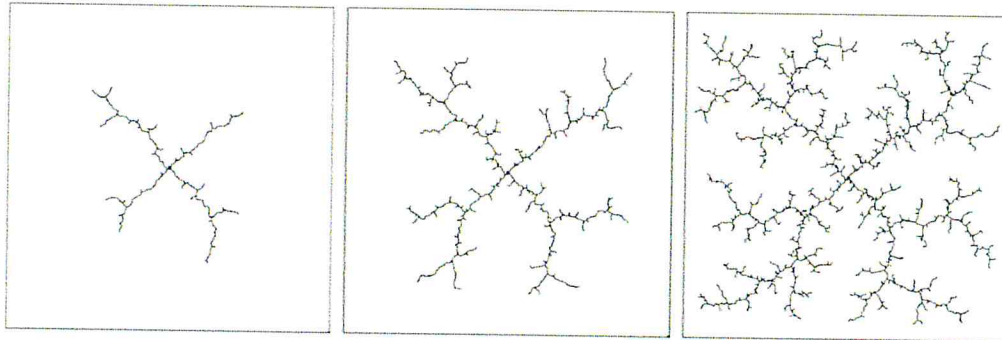


Figure 7 : Construction incrémentale de l'algorithme RRT [17].

4.3. Avantages et inconvénients de chaque méthode

Les principaux avantages et inconvénients des approches étudiées dans ce chapitre peuvent être décrits comme suit :

Méthode	Avantages	Inconvénients
Rapidly-exploring random trees	• Recherche rapide de l'espace libre. • Des techniques de décision avancées sont appliquées pour la vérification des collisions. • Optimalité dans le chemin est garantie dans les nouvelles implémentations telles que RRT*. • Faisabilité en temps réel.	• Chaque nœud de l'arbre doit être vérifié pour les collisions pendant que l'arbre est en expansion.
Rapidly optimizing mapper (ROM)	• Optimalité. • Il n'utilise pas beaucoup d'obstacles.	• Très proche des obstacles.
Corner passing method	• Algorithme facile. • Optimalité dans le chemin garanti.	• Exige beaucoup de calcul. • Complexité de calcul.

		• Méthode hors ligne pure.
Lattice planner	• Basse puissance de calcul nécessaire. • La douceur et l'optimalité du trajet sont garanties.	• Temps d'inefficacité avec le calcul d'un chemin. • Peut conduire à un échantillonnage ou des oscillations exhaustives.

Tableau 1 : Comparaison entre les différentes méthodes de planification

D'après le tableau 1 de comparaison nous remarquons que RRT est bien meilleur que les autres approches car il a beaucoup d'avantages et peu d'inconvénients, pour cette raison nous nous sommes basés dans notre travail sur cette approche.

5. Conclusion

Dans ce chapitre, nous avons décrit différents algorithmes de planification de chemins réalisables pour un robot mobile autonome. En plus, nous avons discuté de leurs avantages et inconvénients. Cela nous a permis de sélectionner le meilleur algorithme de planification. Nous avons ainsi opté pour l'algorithme (*Rapidly-exploring random trees*) pour la mise en œuvre de notre planificateur de chemins.

Dans le chapitre suivant, nous allons décrire en détails l'algorithme RRT ainsi que ses étapes de calcul. Nous allons aussi proposer une amélioration de cet algorithme afin de générer un chemin optimal reliant les deux configurations, initiale et finale.

Chapitre 2

Planification de chemins en utilisant l'algorithme RRT

1. Introduction

L'algorithme RRT été proposé à la fois comme un algorithme d'échantillonnage et une structure de données conçue pour permettre des recherches rapides dans des espaces de grande dimension dans la planification de mouvements [18]. L'algorithme RRT a une structure de données qui le rend progressivement construits vers des régions inexplorées de l'espace de travail à partir d'une configuration initiale.

2. Principe de l'algorithme RRT

Le principe des algorithmes de construction RRT de base est décrit dans cette section :

2.1. Algorithme de construction d'arbre

Les arbres RRT se construisent incrémentalement depuis la position de départ q_{init} et crée un chemin entre cette position et celle d'arrivée q_{goal} . La construction incrémentale de cet arbre est donnée par l'algorithme suivant :

```

BUILD () { //Construire un RRT qui crée éventuellement un chemin entre  $q_{init}$  et  $q_{goal}$ .
Input : Configuration initiale  $q_{init}$ , configuration de l'objectif  $q_{goal}$ , nombre de nœuds  $I_{max}$ ,
distance incrémental  $\epsilon$ ;
Output: Path  $q_{init} \dots q_{goal} \subseteq$  RRT tree  $\tau$  {
     $\tau$ . INIT( $q_{init}$ );
    for  $i=1$  to  $I_{max}$  {       $q_{rand} \leftarrow$  BASED_RANDOM_CONFIG();
                           if (ETENDRE ( $\tau$ ,  $q_{rand}$ ,  $\epsilon$ ) =Solved) { // Return found solution
                           return  $\tau$ .Path( $q_{init}$ ,  $q_{goal}$ );
                           }
    }
    Return No Solution; // No solution found within  $I_{max}$  nodes max
}

```

Algorithme 1 : Build RRT [19]

2.2. Algorithme d'extension de l'arbre

À chaque étape, une configuration q_{rand} aléatoire est choisie ; pour cette configuration, la configuration la plus proche appartenant déjà à l'arbre q_{near} est trouvée. Pour cela, une définition

de distance est nécessaire (dans la planification de mouvement, la distance euclidienne est généralement choisie comme mesure de distance). Lorsque la configuration la plus proche est trouvée, un planificateur local tente de rejoindre q_{near} avec q_{rand} avec une distance limite. Si q_{rand} a été atteint, il est ajouté à l'arbre et connecté avec un bord à q_{near} . Dans le cas contraire, la configuration q_{new} obtenue à la fin de la recherche locale est ajoutée à l'arbre de la même manière tant qu'il n'y a pas eu de collision avec un obstacle lors de la recherche. Cette opération est appelée l'étape [17].

Le résultat de la fonction **Etendre** est illustrée par Figure 8. Ce processus est répété jusqu'à ce que certains critères soient satisfaits, comme une limite sur la taille de l'arbre, par exemple.

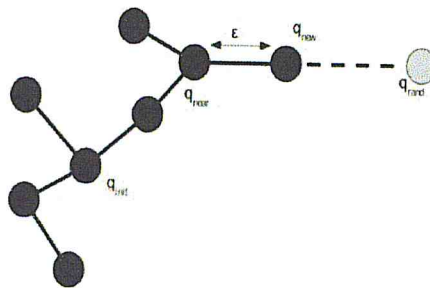


Figure 8 : Phase d'extension d'un RRT [17].

Une fois que le RRT ait été construit, de multiples requêtes peuvent être émises. Pour chaque requête, les configurations les plus proches appartenant à l'arborescence de la configuration initiale et de la configuration cible sont trouvées. Ensuite, les configurations initiales et finales sont jointes à l'arbre aux configurations les plus proches en utilisant le planificateur local et un chemin est récupéré en remontant dans l'arborescence vers la configuration initiale.

L'algorithme suivant décrit le fonctionnement de la fonction **Etendre**.

ETENDRE() { //Étendre τ vers q_{rand} avec une distance ε , après effectuer une vérification.

Input : nœud aléatoire q_{rand} , nœud cible q_{goal} , arbre RRT τ , distance incrémentale ε ;

Output : Status{

$q_{near} \leftarrow \text{NEAREST_NEIGHBOUR}(q_{rand}, \tau)$;

$q_{new} \leftarrow \text{NEW_CONFIG}(q_{near}, q_{rand}, \varepsilon)$;

if ($q_{new} \neq \text{NULL}$) {

$\tau.\text{ADD_NODE}(q_{new})$;

$\tau.\text{ADD_EDGE}(q_{near}, q_{new})$;

if ($\text{CHECK_FINISHED}(q_{new}, q_{goal})$) {

 // q_{new} connects to q_{goal} , so we have a complete path

$\tau.\text{ADD_EDGE}(q_{new}, q_{goal})$;

return Solved;

 }

else {

if ($q_{new} = q_{rand}$) {

q_{rand} reached;

 }

else {

 extended towards q_{rand} ;

 }

 }

 }

}

Algorithme 2 : RRT Etendre [19]

2.3. Ajouter un nouveau nœud

Au début, l'arbre (ou le graphe) est vide. Nous avons seulement les positions de départ et d'arrivée. Le but est de créer un arbre aléatoire entre ces deux positions. Tout d'abord, un nœud aléatoire est créé et ajouté au graphe. Une itération simple est effectuée et tente à étendre le RRT en ajoutant un nouveau sommet qui est biaisé par un état sélectionné de manière aléatoire.

L'algorithme sélectionne le sommet le plus proche, déjà dans le RRT, à l'état donné en calculant la distance euclidienne [20] (figure 9).

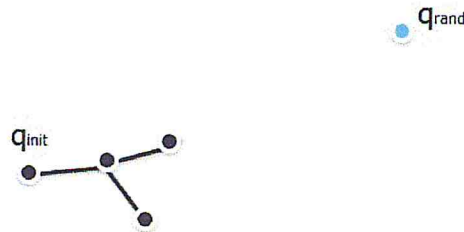


Figure 9 : Ajouter un nouveau sommet au graphe.³

2.4. Vérification de la taille du pas

Un autre processus important consiste à vérifier la distance entre le nouveau sommet (q_{rand}) et le sommet le plus proche dans RRT ($q_{nearest}$) ; ceci doit être effectué parce que la taille de l'arrête reliant q_{rand} à $q_{nearest}$ peut être plus grande que la taille du pas sélectionné. Si la distance n'est pas supérieure à la taille du pas, il faut passer à l'étape suivante. Dans le cas contraire, il faut créer un nouveau sommet (q_{new}) dont la distance du sommet $q_{nearest}$ est inférieure au pas. Ensuite, aller à l'étape suivante [20].

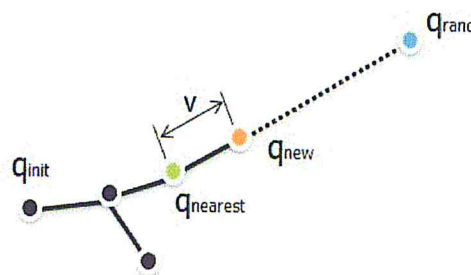
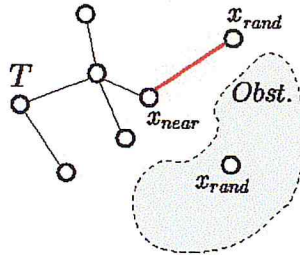


Figure 10 : Vérification de la distance entre $q_{nearest}$ et q_{rand} ³

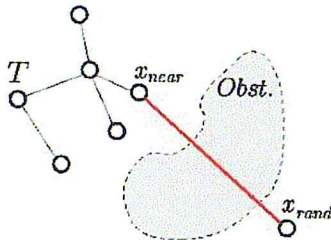
³ <http://tanergungor.blogspot.com/2015/05/robot-navigation-rapidly-exploring.html>

2.5. Vérification de l'intersection

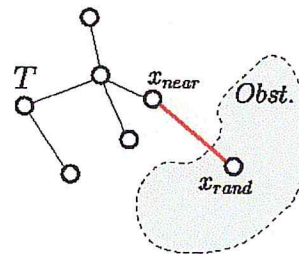
Lors de la création de l'arbre, les obstacles doivent être évités en vérifiant si un état donné se trouve à l'intérieur de l'obstacle. De plus, chaque bord du RRT correspondra à un chemin qui doit se trouver entièrement dans la région libre [20] (Figure 11).



(a) En dehors de l'obstacle, il n'y a pas d'intersection



(b) En dehors de l'obstacle, mais il y a une intersection



(c) à l'intérieur de l'obstacle

Figure 11 : Vérification de l'intersection entre q_{near} et q_{rand} ⁴.

2.6. Critère d'arrêt de construction du RRT

Il existe deux possibilités d'arrêt ; soit le RRT a atteint le nombre maximal de générations que doit effectuer l'algorithme, où il atteint la configuration finale (destination).

2.7. Sélection du chemin réalisable

Après avoir atteint la configuration de destination, il faut sélectionner le chemin la reliant à la configuration de départ. Pour cela, il suffit de connecter les nœuds à partir du point final avec leur nœud parent jusqu'au point initial (Figure 12).

⁴ <http://tanergungor.blogspot.com/2015/05/robot-navigation-rapidly-exploring.html>

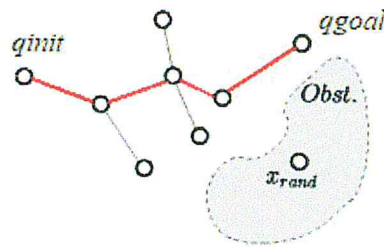


Figure 12 : Sélection du chemin réalisable⁵.

3. Conclusion

Dans ce chapitre, nous avons décrit les étapes de création d'un graphe en utilisant l'algorithme RRT de base (Rapidly-exploring random trees).

Dans le chapitre suivant, nous allons présenter une nouvelle technique de planification de trajectoires, basée sur l'algorithme RRT et l'algorithme A*.

⁵ <http://tanergungor.blogspot.com/2015/05/robot-navigation-rapidly-exploring.html>

Chapitre 3

Conception et réalisation

1. Introduction

Dans les chapitres précédents, nous avons parlé de la planification et nous avons présenté l'algorithme de planification RRT.

Dans ce chapitre, nous allons présenter la phase de conception qui représente une étape majeure dans la réalisation de ce travail. En effet, cette phase doit décrire le fonctionnement futur du système ainsi que le langage utilisé, afin d'en faciliter la réalisation et l'implémentation. En outre, nous décrivons la réalisation de ce travail avec ces détails techniques et des outils de développement adoptés. Enfin, nous allons montrer les interfaces principales de l'application développée.

2. Architecture du système

La méthode de planification de trajectoires, basée sur l'algorithme RRT, propose une solution aux problèmes fondamentaux de la planification et même au problème d'optimisation. Dans ce travail, le chemin est défini à partir de la modélisation de l'environnement du robot par l'algorithme RRT.

Les entrées du système sont les valeurs de la position du robot représentant le point initial. Le point sélectionné par l'utilisateur représentera le point final. Aussi, les données de production consistent en les nœuds de l'arbre générés à partir du point initial jusqu'au point final.

Après que le chemin ait trouvé, le système envoie au robot la consigne de commencer son mouvement vers sa cible :

- **Point de départ (initial)** : à l'aide de la fonction « odometry » qui permet de retourner les coordonnées de la position actuelle du robot.
- **Point de destination (dist)** : c'est le point où le robot est censé arriver après qu'il reçoive les coordonnées entrées par l'utilisateur.

Un schéma synoptique du planificateur basé sur l'algorithme RRT proposé est donné par la figure 13 suivante :

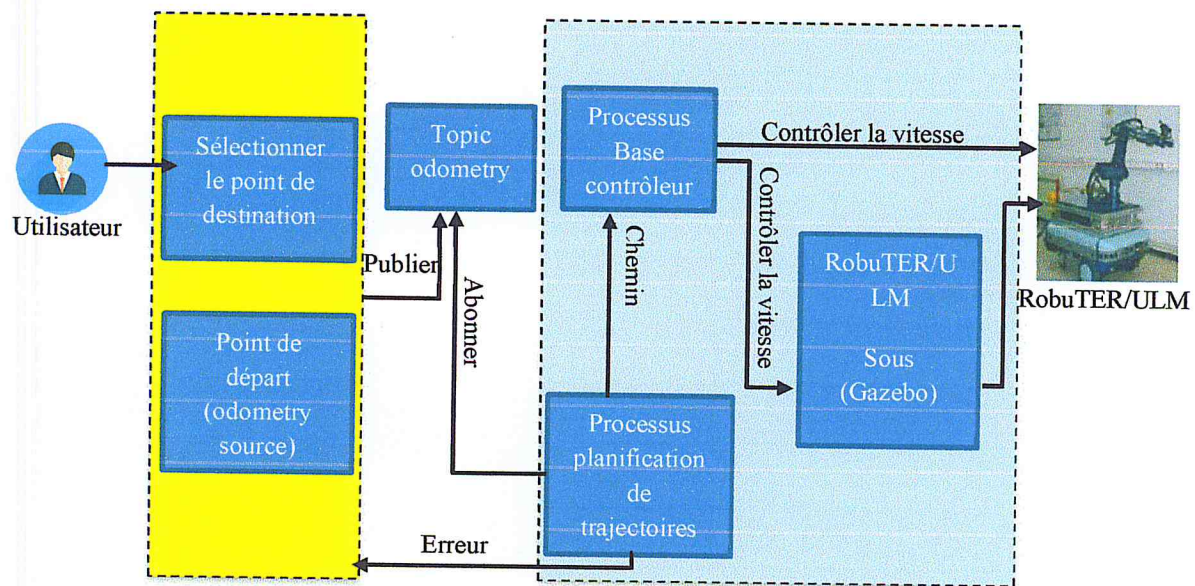


Figure 13 : Architecture du planificateur développé.

3. Description de l'architecture

3.1. Processus de planification

La méthode de planification de trajectoires proposée est basée sur l'algorithme RRT. Elle propose une solution au problème fondamental de la planification et même au problème d'optimisation. Dans ce travail, le chemin est défini à partir de la modélisation de l'environnement du robot par l'algorithme RRT fusionné avec l'algorithme A*.

Les entrées du système sont les valeurs de la position du robot, le point de départ et le point de destination ; les données de production sont les nœuds de l'arbre généré à partir du point initial jusqu'à la destination.

3.2. Principe de l'algorithme A*

L'algorithme de recherche A* est un algorithme de recherche de chemin dans un graphe entre un nœud initial et un nœud final tous deux connus. De par sa simplicité, il est souvent présenté comme exemple typique d'algorithme utilisé en intelligence artificielle. L'algorithme A* a été créé pour que la première solution trouvée soit l'une des meilleures ; c'est pourquoi il est célèbre dans des applications comme les jeux vidéo privilégiant la vitesse de calcul sur l'exactitude des

résultats. Cet algorithme a été proposé pour la première fois par Hart, Nilsson et Raphael. Il s'agit d'une extension de l'algorithme de Dijkstra⁶.

À chaque itération, A* va tenter de prendre le chemin le plus court pour aller d'une position initiale à une position finale. Pour déterminer si un nœud est susceptible de faire partie du chemin solution, il faut pouvoir quantifier sa qualité. Le plus souvent, on utilise la distance en « *vol d'oiseau* » entre la position du nœud et la destination ; mais d'autres fonctions peuvent être utilisées.

On travaille à partir d'une position de départ, d'une position d'arrivée, d'un terrain, et de deux listes de nœuds (ouvertes et fermées) qui sont vides à l'origine.

1. À partir du point de départ, on regarde quels sont les nœuds voisins correspondant aux libertés de mouvement ; par exemple, on peut aller à droite, à gauche, sauter ou ramper, reculer ou avancer (6 possibilités = 6 nœuds). On place le nœud de départ dans la liste fermée.
2. Pour chaque nœud voisin :
 - Si c'est un obstacle, on passe ; c'est-à-dire, on ne traite pas ce nœud, il ne sera ajouté nulle part.
 - S'il est déjà dans la liste fermée, on passe (on a déjà étudié ce trajet).
 - S'il est déjà dans la liste ouverte, on vérifie sa qualité. Si la qualité actuelle est meilleure, c'est que le chemin est plus court en passant par le trajet actuel ; donc on met à jour sa qualité dans la liste ouverte, et on met à jour son lien parent (avec le nœud courant qui correspond à un meilleur chemin).
 - Sinon, on ajoute ce nœud à la liste ouverte avec comme parent le nœud courant.
3. On parcourt la liste ouverte à la recherche du nœud ayant la meilleure qualité. Si la liste ouverte est vide, il n'y a pas de solution, fin de l'algorithme.
4. On prend le meilleur, et on le retire de la liste ouverte pour le mettre dans la liste fermée.
5. Nœud courant = nœud qu'on vient d'insérer.
6. Si nœud courant = destination alors, on a trouvé un bon chemin, sinon on réitère en 2.

⁶ <https://www.benicourt.com/blender/2015/03/pathfinding-algorithmes-en-a/>

La liste ouverte contient toutes les pistes de trajet qui sont encore possibles. La liste fermée contient toutes les solutions étudiées, bonnes et mauvaises. Le “bon chemin” trouvé se retrouve en remontant le long des parents des nœuds.

3.3. Principe de RRT-A*

Notre solution a pour objectif de construire un chemin optimal basé sur l’algorithme RRT. À cet effet, nous avons proposé des modifications comme nous allons le voir dans cette section :

- Générer plusieurs points aléatoires au lieu d’en générer un seul.
- Fusionner RRT avec l’algorithme A*.

La figure 14 suivante représente un schéma qui récapitule les étapes de notre algorithme de planification de chemins optimaux sans collision.

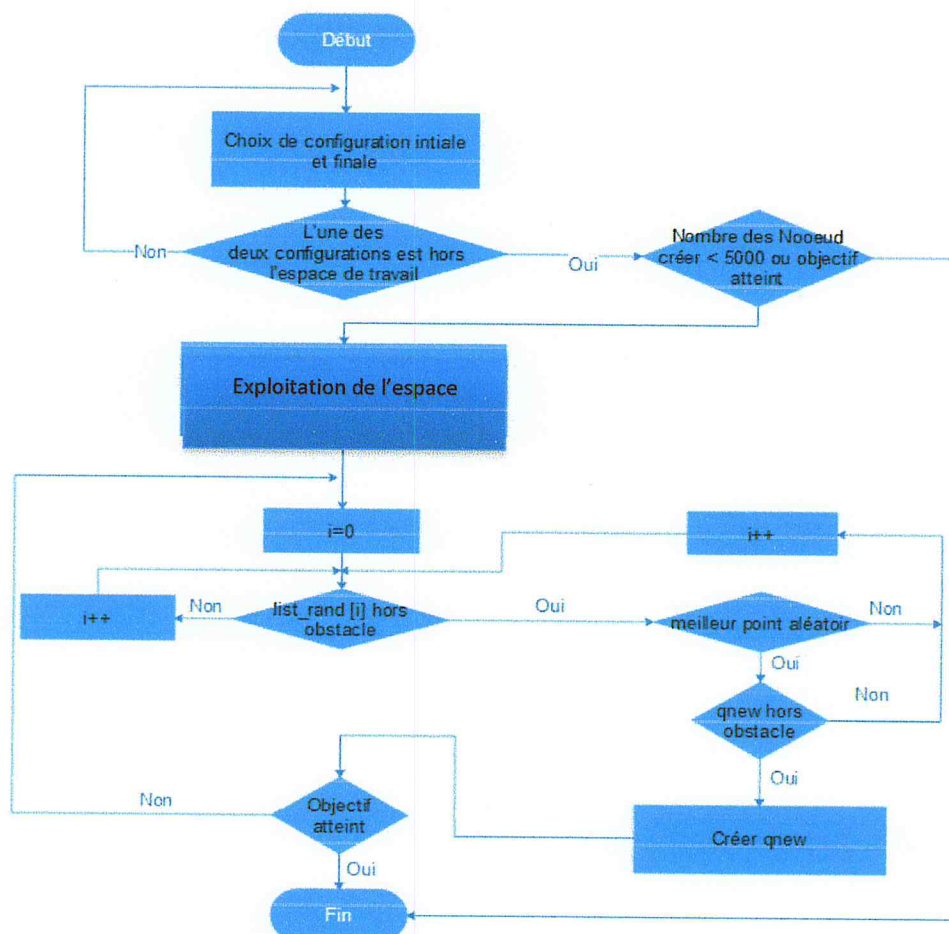


Figure 14 : Algorithme de planification proposé.

Après avoir implémenté cette solution, nous avons rencontré un problème de blocage de l'algorithme dans des environnements qui contiennent des obstacles qui se présentent sous la forme « U ». Cela nous a amené à envisager une autre solution ; nous avons appliqué d'autres modifications pour retourner sur trace et exploiter d'autre chemin (*Back-tracking*).

3.4. Back-tracking

Le retour sur trace (ou *back-tracking*) est une famille d'algorithmes qui consistent à revenir en arrière sur des décisions prises afin de sortir d'un blocage. La méthode des essais et erreurs constitue un exemple simple de *back-tracking*. Le terme est surtout utilisé en programmation, où il désigne une stratégie pour trouver des solutions à des problèmes de satisfaction de contraintes.

3.5. Processus de la base contrôleur

Ce processus contrôle tous ce qui est en relation avec la base mobile du robot concernant les vitesses des roues et leur orientation.

3.6. RobuterULM sous Gazebo:

C'est la partie de visualisation et simulation des comportements des Robot

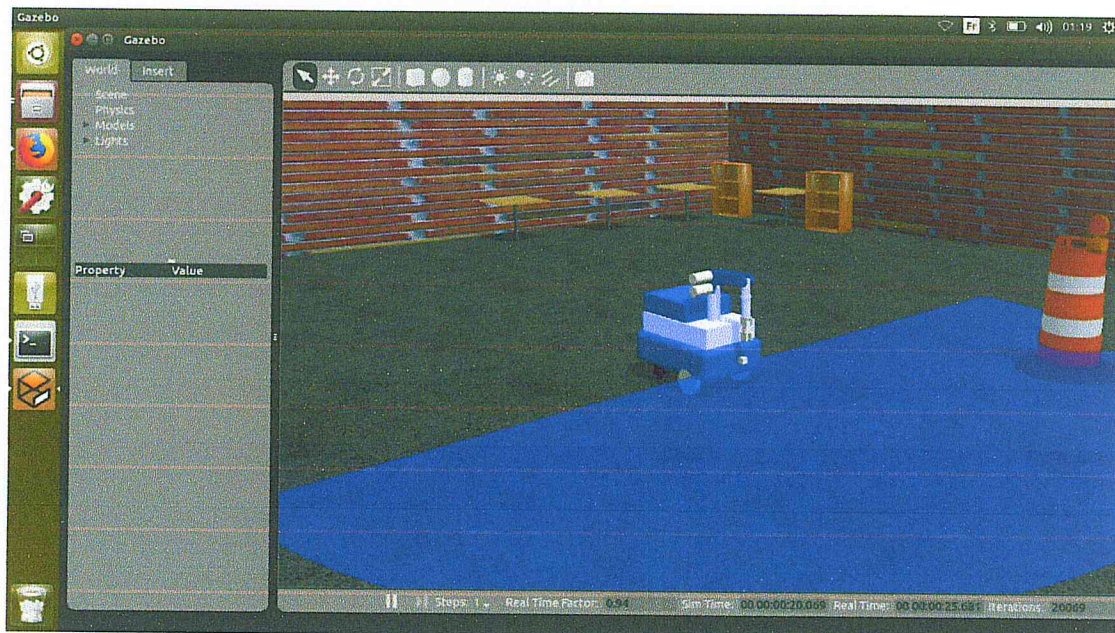


Figure 15: visualisation de RobuterULM sous Gazebo

4. Conclusion

Dans ce chapitre, nous avons présenté l'étude conceptuelle ainsi que la réalisation de l'approche développée. Il s'agit d'une approche basée sur RRT et A*. Ensuite, nous avons abordé les outils que nous avons utilisé pour la mise en œuvre de notre projet.

Dans le chapitre suivant, nous présenterons les tests mises en œuvre au cours de notre travail.

Chapitre 4

Test et Validation

1. Introduction

Les techniques et stratégies décrites dans les chapitres précédents sont appliquées sur le robot mobile différentiel RobuTER décrit dans l'annexe C. Il est disponible au niveau de la Division de productique de robotique (DPR) du CDTA où nous avons réalisé nos implémentations.

Comme le montre Figure 16, deux caméras Kinect sont fixées sur le toit de la salle machine ; chacune Kinect est placée à une hauteur de 3500mm. Ces caméras sont placées de manière à visualiser à la fois l'environnement (sol, obstacles, etc.) et le robot mobile RobuTER. Les caméras Kinect sont connectées à un PC hôte afin d'acquérir et traiter les images de la scène.

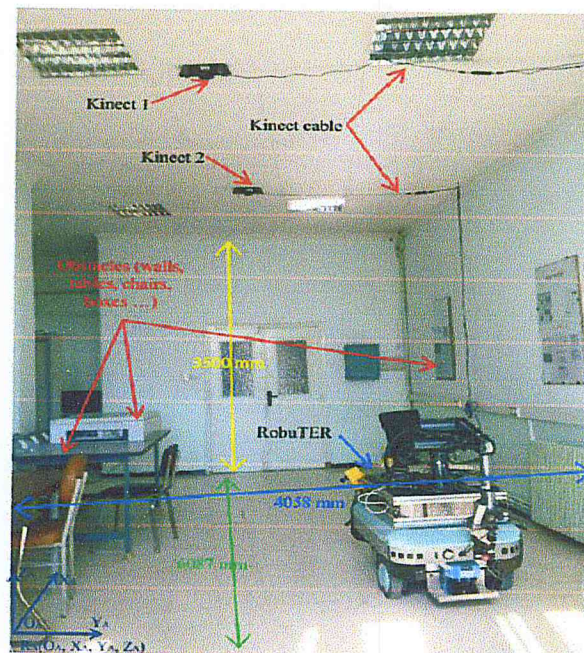


Figure 16 : Banc d'essai robotique expérimental.

Afin de comparer les résultats de notre travail avec ceux réalisés précédemment, nous avons choisi le même environnement avec les mêmes conditions que dans [21].

2. Langages de développement

Notre projet vise à développer un algorithme de planification de trajectoires optimales basé sur l'algorithme RRT. Pour cela, nous avons choisis le langage *Python* comme langage de développement. Nous avons aussi utilisé le simulateur *Gazebo* accompagné de *Rviz* pour la simulation.

2.1. Language Python



Python est un langage de programmation (au même titre que le C/C++, fortran, java, etc.), développé en 1989. Ses principales caractéristiques sont les suivantes :

- **Open-source** : son utilisation est gratuite et les fichiers sources sont disponibles et modifiables,
- **Simple et très lisible**,
- **Doté d'une bibliothèque de base très fournie**,
- **Importante quantité de bibliothèques disponibles** : pour le calcul scientifique, les statistiques, les bases de données, la visualisation, etc.,
- **Grande portabilité** : indépendant vis à vis du système d'exploitation (Linux, Windows, MacOS),
- **Orienté objet**,
- **Typage dynamique** : le typage (association à une variable de son type et allocation zone mémoire en conséquence) est fait automatiquement lors de l'exécution du programme. Ceci permet une grande flexibilité et rapidité de programmation, mais qui se paye par une surconsommation de mémoire et une perte de performance,
- **Présente un support pour l'intégration d'autres langages**.

2.2. Ubuntu



Ubuntu est un système d'exploitation GNU/Linux basé sur la distribution Linux Debian. Il est développé, commercialisé et maintenu pour les ordinateurs individuels par la société *Canonical*.

Ubuntu est la distribution la plus consultée sur Internet d'après le site Alexa, et le système d'exploitation le plus utilisé sur les systèmes Cloud ainsi que sur les serveurs informatiques. Il se divise en deux branches :

- La branche principale stable dite LTS avec mise à niveau tous les six mois et mise à jour majeure tous les 2 ans. La dernière version 18.04 ; nom de code « Bionic Beaver », est sortie le 26 avril 2018.
- La branche secondaire instable avec mise à jour majeure tous les six mois. La dernière version en date est la 18.04 ; nom de code « Bionic Beaver » est sortie le 26 avril 2018.

2.3. Wing IDE



WING est également l'une des meilleures IDE alternatives pour les programmeurs *Python*. Il est compatible avec plusieurs versions de *Python*, y compris les versions récemment publiées.

Wing offre la version 101 qui est libre d'utilisation pour Wing IDE ; ce qui signifie qu'on peut l'utiliser sans licence. C'est l'un des outils les plus basiques mais très utilisés pour développer des programmes *Python*⁷.

2.4. Gazebo



Gazebo est un simulateur multi-robots. Comme *Stage*, il est capable de simuler un ensemble de robots, capteurs et objets, mais le fait dans un monde tridimensionnel. Il génère des informations réalistes pour les capteurs et simule les interactions physiques entre les objets (en intégrant un moteur physique)⁸. Les principaux composants utilisés dans *Gazebo* sont :

⁷ <https://techlistes.com/2018/04/les-6-meilleurs-ide-pour-python/>

⁸ <http://playerstage.sourceforge.net/gazebo/gazebo.html/>

- Un descripteur d'environnement (appelé World File).
- Un descripteur de modèle de robots (appelé Model File).

Ces composants sont associés à des variables d'environnement (définies par défaut dans le fichier `/usr/share/gazebo/setup.sh`) :

- `GAZEBO_MODEL_PATH` : pour les répertoires contenant les modèles.
- `GAZEBO_RESOURCE_PATH` : pour les ressources (telles que descripteurs d'environnement).
- `GAZEBO_MASTER_URI` : pour le nœud master par similitude à ROS.
- `GAZEBO_PLUGIN_PATH` : pour les plugins et librairies.
- `GAZEBO_MODEL_DATABASE_URI` : pour les bases de données en ligne utiles à *Gazebo*.
- `OGRE_RESOURCE_PATH` : pour les fonctions d'affichage utilisant *Ogre3D*.

3. Modélisation de l'environnement

Pour la modélisation de l'environnement, nous avons considéré les hypothèses suivantes :

- Le robot est un point matériel.
- Le robot se déplace dans un espace à deux dimensions d'intérieur.
- Les obstacles sont considérés comme des polygones.

Aussi, nous avons supposé que l'espace de travail pour un robot soit comme indiqué dans la figure 17 où les parties ombrées représentent les obstacles.

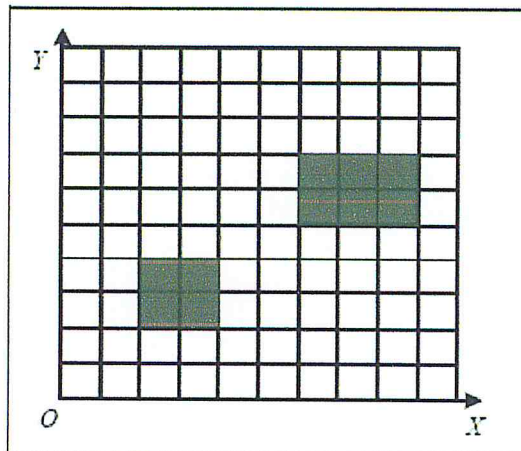
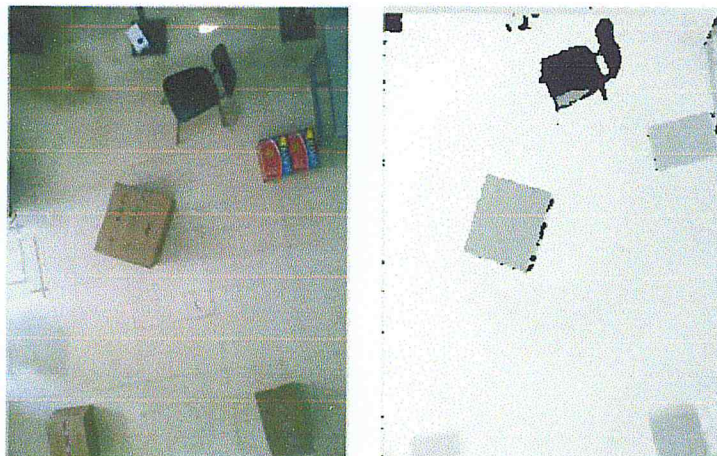
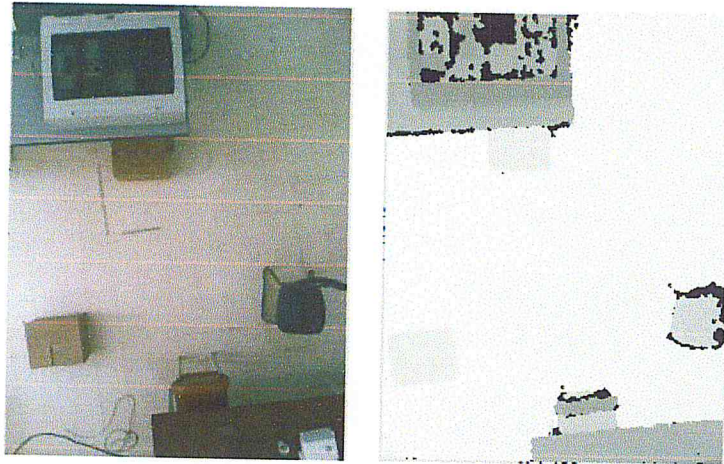


Figure 17 : Environnement d'évolution du robot mobile.

Les images acquises à partir de la première caméra Kinect (images RGB et profondeur) sont données par la figure 17(a). Aussi, les images fournies par la seconde caméra sont représentées par la figure 17(b) :



(a) Images RGB et de profondeur acquise par la première Kinect.



(b) Images RGB et de profondeur acquise par la seconde Kinect.

Figure 18 : Images acquises par les deux caméras Kinect.

Le prétraitement des images précédentes génère un modèle de carte binaire exacte pour l'environnement avec une résolution suffisante de 960x640 comme le montre la figure 18.

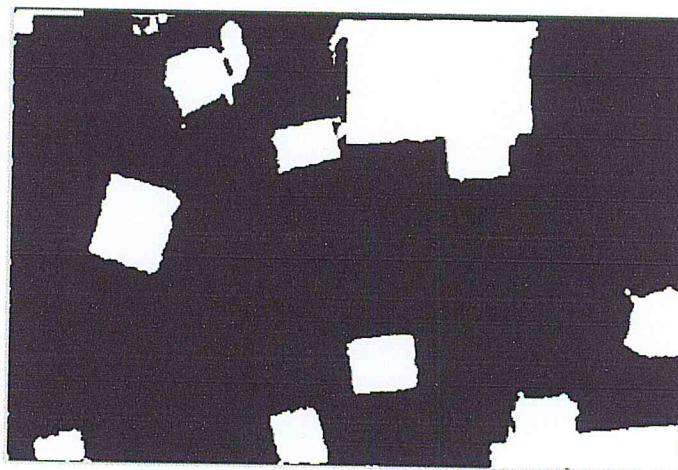


Figure 19 : Carte binaire de l'environnement global.

Afin d'offrir plus de sécurité au robot lors de l'exécution de la trajectoire générée par l'algorithme RRT-A*, les obstacles seront enveloppés et une carte binaire de sécurité est ainsi obtenue telle que montré par la figure 19 :

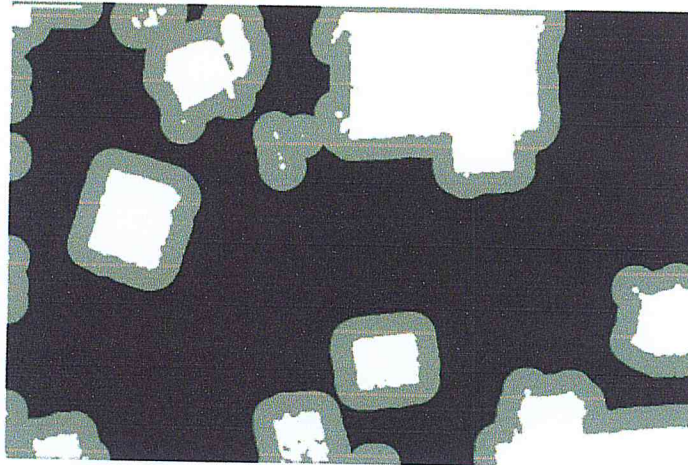


Figure 20 : Carte binaire sûre de l'environnement global

Pour la suite de ce travail, et comme nous avons considéré des polygones comme obstacles, cet environnement a été approximé par celui illustré dans la figure 20 et nous avons ajouté un autre environnement (figure 21) pour assurer l'efficacité des tests :

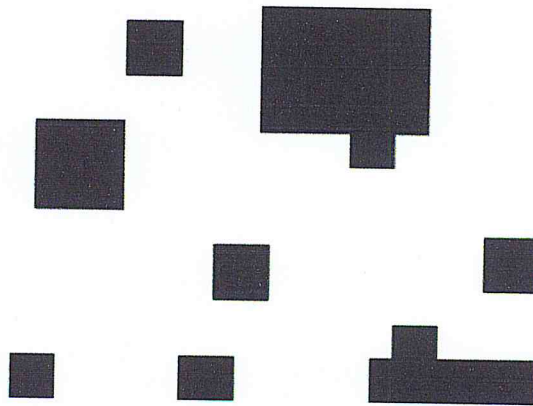


Figure 21 : Approximation par des polygones de l'environnement 1 d'évolution du robot mobile.

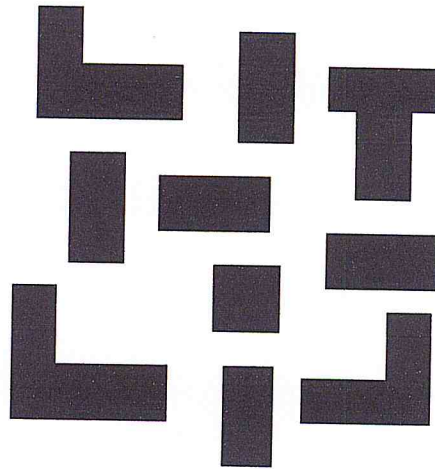


Figure 22: Approximation par des polygones de l'environnement 2 d'évolution du robot mobile [22].

4. Planification de trajectoires

4.1. Environnement 1 :

La carte binaire obtenue donne une description presque continue de l'environnement, celle est bien adaptée pour être utilisée par notre algorithme RRT-A*.

Le robot mobile RobuTER, avec des dimensions de $1200mm \times 680mm$, doit se mouvoir de la position initiale $Source(x, y)_{init} = (74mm, 1953mm)$ vers la position finale $Target(x, y)_{fin} = (5501mm, 1308mm)$.

RRT-A* choisit le premier chemin possible et crée ses nœuds aléatoirement. Par conséquent, l'algorithme devrait être exécuté plusieurs fois avec différentes tailles de pas. Enfin, nous allons choisir le meilleur pas selon concerne le temps d'exécution, le lissage et l'optimalité de la trajectoire finale générée.

Nous avons sélectionné trois tailles de pas : 8 pixels, 35 pixels et 60 pixels qui correspondent respectivement à 50.72mm, 221.9mm et 380.4mm.

4.1.1. Pas=08 pixels

Le chemin obtenu après l'exécution de l'algorithme avec une taille de pas de 8 pixels est illustré par la Figure 22. Le chemin RRT comprend 194 segments. Le chemin est généré dans environ 1.15 seconde ; sa longueur totale est $l = 7290.49mm$.

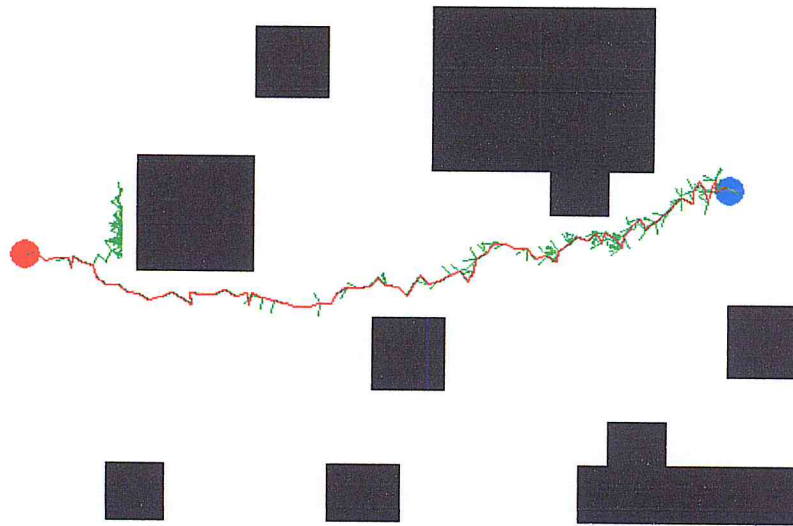


Figure 23 : Planification RRT-A* avec une taille de pas de 8 pixels.

4.1.2. Pas=35 pixels

Après l'exécution avec un pas de 35 pixels, nous avons obtenu le chemin montré dans la Figure 23. Ce chemin comprend 33 segments. Il est généré en 0.65 secondes ; sa longueur totale $l = 7120.8mm$.

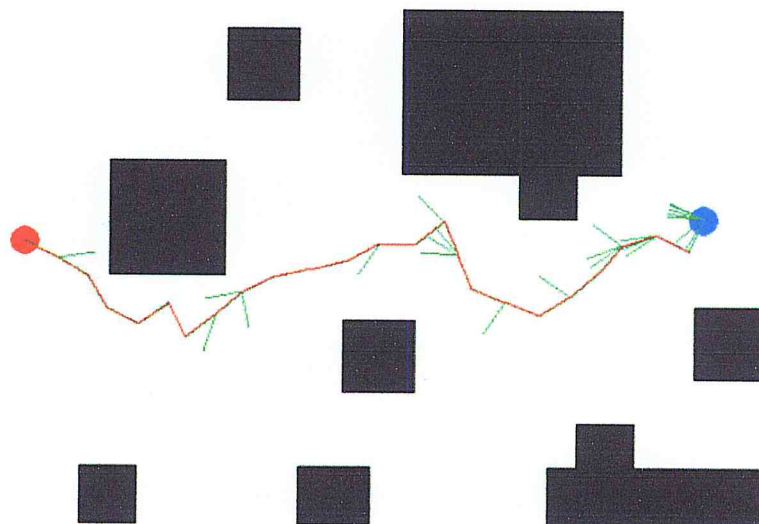


Figure 24 : Planification RRT avec une taille de pas de 35 pixels

4.1.3. Pas=60 pixels

Le chemin résultant est donné par la figure 24. En utilisant une taille de pas de 60 pixels, le chemin RRT se compose de 18 segments. Il est généré en moins de 0.45 secondes ; sa longueur totale $l = 6590.3mm$.

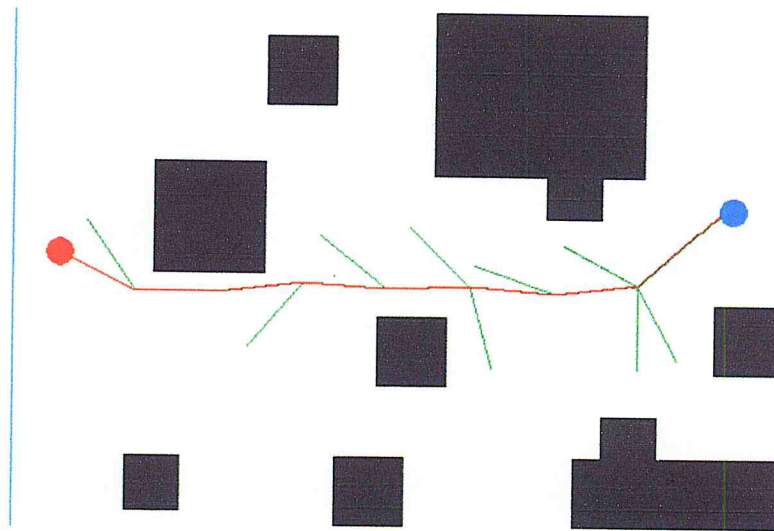


Figure 25 : Planification RRT avec une taille de pas de 60 pixels

Le tableau suivant résume les principaux résultats obtenus après l'exécution des scénarios de validations précédents. Chaque scénario a été exécuté une dizaine de fois :

Pas	8 pixels	35 pixels	60 pixels
Longueur du chemin (mm)	7290.49	7120.81	5067.3
Nombre des Nœuds	120	68	25
Temps d'exécution (seconde)	1.15	0.75	0.35
Nombre de segment	94	41	14

Tableau 2 : Résumé des principaux résultats obtenus.

Nous pouvons clairement voir que l'exécution du RRT avec une taille de pas de 60 pixels donnent le chemin le plus court en comparant avec 8 et 35 pixels. De plus, il est beaucoup plus rapide et donne moins de nombre de nœuds. Par conséquent, nous avons opté pour cette taille de pas dans ce qui suit.

4.2. Environnement 2 :

La carte binaire qu'on a ramenée donne une description presque continue de l'environnement, celle-ci est bien adaptée pour être utilisée par notre algorithme RRT-A*. Nous avons utilisé les distances euclidienne, Chebyshev et City Block.

Le tableau suivant résume les principaux résultats obtenus après l'exécution :

Pas	Euclidienne	Chebyshev	City Block
Longueur du chemin (mm)	225.56	230	308.8
Nombre des Nœuds	21	25	31
Temps d'exécution (seconde)	0.37	0.44	0.53
Nombre de segment	12	15	24

Tableau 3 : Résumé des principaux résultats obtenus.

Nous pouvons clairement voir que l'exécution du RRT avec la distance Euclidienne donnent le chemin le plus court en comparant Chebysheve et City Block. De plus, elle est beaucoup plus rapide et donne moins de nombre de nœuds. Par conséquent, nous avons opté pour cette distance pour la comparaison.

4.3. Comparaison

Nous avons déjà mentionné que le même travail a été effectué en utilisant un autre algorithme dans les mêmes environnements (environnement 1 et 2) et dans les mêmes conditions [22] [23]. Dans cette partie, nous ferons une comparais on entre les résultats des deux algorithmes et on commence par l'environnement 1 (Figure 26).

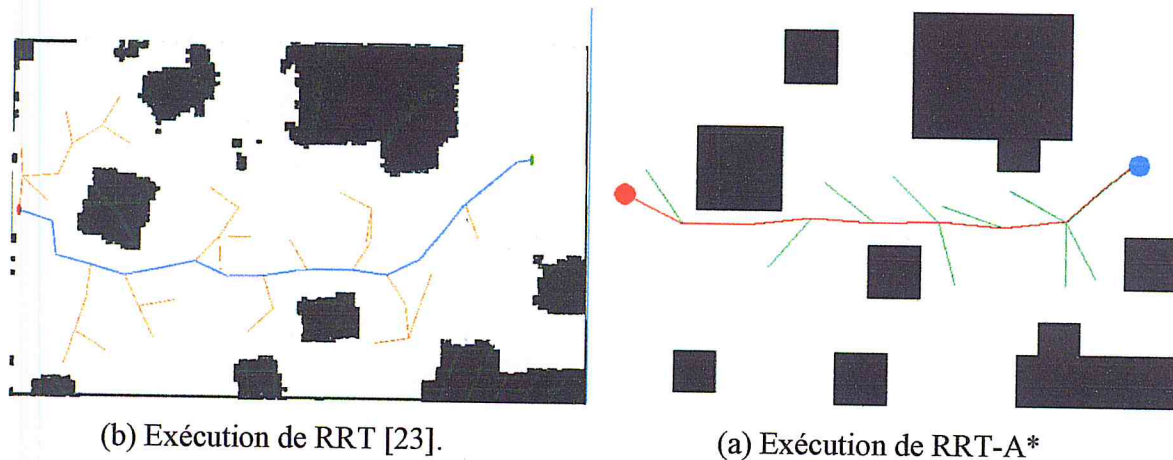


Figure 26 : Comparaison entre RRT-A* et RRT.

Le tableau 3 montre les résultats de l'exécution des algorithmes RRT et RRT-A* sur le même environnement :

Paramètres	RRT-A*	RRT
Temps d'exécution (seconde)	0.35	2
Longueur du chemin (mm)	5067.3	6590
Nombre des segments	9	18
Nombre des nœuds	21	-
Paramètre de blocage	500	-

Tableau 4 : Comparaison entre RRT-A* et RRT

À partir du tableau 3, nous avons noté tout d'abord que RRT génère un chemin plus long du point initial au but en comparaison avec l'algorithme RRT-A*. Deuxièmement, pour générer le chemin reliant les deux positions initiale et finale, RRT-A* est trois fois plus rapide que l'algorithme RRT et avec un minimum des nœuds aléatoires.

Une autre comparaison est réalisée concernant les temps de calcul pour différents espaces de travail avec différentes dimensions et nombre d'obstacles. Pour chaque cas, nous avons considéré différents nombres de caméras Kinect avec différentes positions *Source-Target*. Le tableau 4 résume la moyenne d'une dizaine d'exécutions des algorithmes RRT-A* et RRT.

Résolution	Nombre d'obstacles	Génération de temps (s) de chemin possible	
		RRT	RRT-A*
640x480 (01 Kinect)	05	0.80	0.25
	10	1.04	0.87
	15	1.25	0.99
640x960 (02 Kinect)	10	1.54	1.01
	15	1.78	1.23
	20	2.11	1.62
640x1280 (04 Kinect)	15	3.03	2.12
	20	3.2	2.24
	25	3.51	2.73

Tableau 5 : Comparaison entre RRT-A* et RRT pour différents espaces de travail avec différentes dimensions et nombre d'obstacles.

D'après le tableau 4, il est clair que RRT-A* est un algorithme meilleur et plus efficace que celui proposé dans [23]. Le temps d'exécution de RRT-A* dans les différents environnements est environ trois fois plus rapide que RRT.

Maintenant on passe vers l'environnement 2

Paramètres	RRT-A*	RRT
Temps d'exécution (seconde)	0.37	2,44
Longueur du chemin (mm)	225.6	255
Nombre des segments	12	18
Nombre des nœuds	25	-
Pas (pixel)	60	-
Nombre de points aléatoire	50	1
Paramètre de blocage	500	-

Tableau 6: comparaison RRT [22] et RRT-A*

D'après le tableau 6, il est clair que RRT-A* est un algorithme meilleur et plus efficace que celui proposé dans [22].

5. Conclusion

Ce chapitre a décrit l'implémentation des techniques décrites dans les chapitres deux et trois. En outre, une description générale de la mise en œuvre a été donnée. Aussi, les principaux résultats de planification de chemins ont été montrés avec leurs étapes de développement. À la fin, un résumé comparatif entre RRT-A* et RRT a été réalisé.

En utilisant RRT-A* et en particulier avec la technique de perception rapide utilisée dans ce travail, nous pouvons passer de la planification hors ligne à la planification en ligne.

Conclusion générale et perspectives

La planification des chemins faisables pour un robot mobile est une tâche très importante dans la robotique mobile. Elle permet de trouver un chemin qui relie une configuration initiale à une configuration finale dans l'environnement tout en évitant les obstacles. À cet effet, de nombreux systèmes et technologies existent pour répondre à cette problématique.

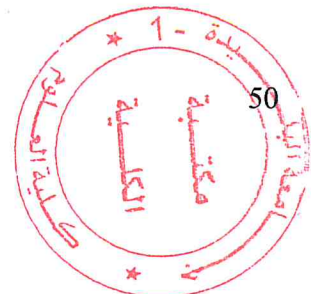
Dans ce travail, nous avons commencé par la présentation des différentes méthodes de planification, leurs avantages et leurs inconvénients dans le cadre de planification des trajectoires, notamment dans le domaine de la robotique mobile. Ensuite, nous avons choisi une approche de planification RRT d'après une comparaison entre les méthodes de planification existantes dans la littérature. Après cela, nous avons présenté l'approche RRT en détail. Puis, nous avons décrit le matériel qui nous a été confié, en particulier une caméra Kinect version 1. Ensuite, nous avons détaillé l'approche adoptée pour la planification de trajectoires ainsi que son adaptation à notre contexte.

L'objectif de ce travail était de mettre en œuvre un système de planification de chemin optimale dans un environnement pouvant contenir des obstacles quelconques (murs, objets, etc.). Nous avons par la suite simulé le système de planification de trajectoires en exploitant une technique qui utilise un outil de simulation que nous avons développé en Python.

Enfin, nous avons effectué plusieurs scénarios pour la validation de l'algorithme proposé. Les principaux résultats obtenus ont été comparés avec un autre travail similaire qui a été effectué précédemment, après avoir choisi le même environnement avec les mêmes conditions (environnement, configurations initiale et finale).

Durant l'implémentation de la solution proposée, nous avons rencontré plusieurs problèmes. En premier lieu, Le RobuTER/ULM du CDTA le quel nous étions censé travailler dessus, est tombé en panne. Cela nous a obligé à changer l'outil de travail et nous nous sommes orienté vers le simulateur Gazebo.

Bien que notre application ne soit qu'à sa première version, elle est néanmoins totalement fonctionnelle et offre un outil de planification de chemins optimaux pour les robots mobiles en simulation.



Concernant les travaux futurs, nous envisageons d'appliquer la solution proposée sur un matériel réel où évoluent une équipe de robots mobiles. Nous envisageons également d'ajouter la fonction PCHIP (*Piecewise Cubic Hermite Interpolating Polynomial*) qui permet de lisser le chemin final généré cette méthode est appelée le *smoothing*.

Bibliographie

- [1] B. BAYLE, «Robotique mobile», Strasbourg.
- [2] J.-C. Latombe, Robot Motion Planning, New York: Springer US, 1991.
- [3] N. Buniyamin , W. Wan Ngah , N. Sariff et Z. Mohamad , «A Simple Local Path Planning Algorithm for Autonomous Mobile Robots», *INTERNATIONAL JOURNAL OF SYSTEMS APPLICATIONS, ENGINEERING & DEVELOPMENT* , p. 154, 2011.
- [4] L. Perez, «Spatial Planning: A Configuration Space Approach», *IEEE Transactions on Computers* , pp. 108 - 120, 1983 .
- [5] C. D. J. G. M. d. M. L. S. a. G. M. A. Krupa, «Towards semi-autonomy in laparoscopic», chez *the Seventh International Symposium on Experimental*, Hawaii, 2000.
- [6] Z. Shiller, Off-Line and On-Line Trajectory Planning, Ariel University: Springer International Publishing Switzerland, 2015.
- [7] P. G. Zavrangas et S. G. Tzafestas, Industrial Robot Navigation and Obstacle Avoidance Employing Fuzzy Logic, Hollande: Kluwer Academic Publishers, 2000.
- [8] R. Menasri, A. Nakib, B. Daachi, H. Oulhadj et P. Siarry, «A trajectory planning of redundant manipulators based on bilevel optimization», *Applied Mathematics and Computation*, pp. 934-947, 2015.
- [9] S. Dubowsky et Z. Shiller, On computing the global time-optimal motions of robotic manipulators in the presence of obstacles, IEEE, 1991.
- [10] B. M. D. H. Mohsen AhmadiMousavi, «Two New Methods for Path Planning of Autonomous Mobile Robot», *Research Journal of Recent Sciences*, vol. 3, n° 15, pp. 110-115, 2014.
- [11] P. Kamkarian , «An Analysis for a Novel Path Planning Method», *Advances in Robotics & Automation*, 2015.

- [12] L. Kavraki, P. Svestka, J. Latombe et M. Overmars, «Probabilistic roadmaps for path planning in high-dimensional configuration spaces,» *IEEE Transactions on Robotics and Automation* , pp. 566 - 580, 1996.
- [13] J. Butzke, K. Sapkota, K. Prasad, B. MacAllister et M. Likhachev, «State lattice with controllers: Augmenting lattice-based path planning with controller-based motion primitives,» chez *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Chicago, 2014.
- [14] M. Q. W.-H. C. L. D. Christos Katrakazas, «Méthodes de planification de mouvement en temps réel pour la conduite autonome sur route: orientations de recherche de pointe et futures,» *Transportation Research Part C Emerging Technologies*, pp. 417-449, 2015.
- [15] L. KNISPEL et R. MATOUSEK, «A Performance Comparison of Rapidly-exploring Random Tree and,» *Recent Advances in Applied and Theoretical Mathematics*, pp. 154-162.
- [16] N. K. A. S. Dave Ferguson, «Replanning with RRTs,» chez *IEEE International Conference on Robotics and Automation*, Orlando, Florida, 2006.
- [17] M. V. D. B. Vidal Alcazar, «Adapting a Rapidly-Exploring Random Tree for Automated Planning,» *Proceedings, The Fourth International Symposium on Combinatorial Search* , pp. 2-9, 2011.
- [18] J. Manuel et A. Larios, «Le fil d'ariane : une méthode de planification générale. Application à la planification automatique de trajectoires,» Institut National Polytechnique de Grenoble, 1994.
- [19] S. M. LaValle et J. J. Kuffner, «Randomized Kinodynamic Planning,» *The International Journal of Robotics Research*, pp. 378-400, 2011.
- [20] A. K. Z. H. Iram Noreen, «A Comparison of RRT, RRT* and RRT*-Smart Path Planning Algorithms,» *IJCSNS International Journal of Computer Science and Network Security*, vol. 16, n° 110, pp. 20-27, 2016.
- [21] A. Bakdi et al., «Optimal path planning and execution for mobile robots using genetic Robotics and,» p. 95–109, 2017.
- [22] M. A. O. Roudabe Seif, «Mobile Robot Path Planning by RRT* in Dynamic Environments,» *I.J. Intelligent Systems and Applications*, pp. 24-30, 2015.

- [23] A. Bakdi, A. Hentout, H. Boutami, A. Maoudj, O. Hachour et B. Bouzouia, «Optimal path planning and execution for mobile robots using genetic algorithm and adaptive fuzzy-logic control,» *Robotics and Autonomous Systems*, pp. 95-109, 2017.

Annexe -A-

1. Introduction au système d'exploitation Robot ROS

Robot Operating System (ROS) est une plate-forme de développement d'applications de robot qui propose diverses fonctionnalités telles que le passage de messages, l'informatique distribuée, la réutilisation de code, etc.

Le projet ROS a été lancé en 2007 sous le nom de Switchyard par Morgan Quigley dans le cadre du projet de robot Stanford STAIR. Le développement principal de ROS s'est passé à Willow Garage.

Voici quelques-unes des raisons pour lesquelles les gens choisissent ROS sur d'autres plates-formes robotiques telles que Player, YARP, Orocos, MRPT, et ainsi de suite :

- Capacités haut de gamme : ROS est livré avec des fonctionnalités prêtes à l'emploi, par exemple, SLAM (Simultaneous Localisation and Mapping) et AMCL (Adaptive Monte Carlo Localisation) dans ROS peuvent être utilisés pour effectuer la navigation autonome dans les robots mobiles et le mouvement MoveIt pour le mouvement planification de manipulateurs de robot.
- Des tonnes d'outils : ROS est rempli de tonnes d'outils pour le débogage, la visualisation et la simulation. Les outils tels que rqt_gui, RViz et Gazebo sont quelques-uns des puissants outils open source pour le débogage, la visualisation et la simulation. Le framework logiciel qui a ces nombreux outils est très rare.
- Soutenez les capteurs et les actionneurs haut de gamme : ROS est livré avec des pilotes de périphériques et des boîtiers d'interface de divers capteurs et actionneurs en robotique. Les capteurs haut de gamme incluent Velodyne-LIDAR, les scanners laser, Kinect, etc. et les actionneurs tels que les servomoteurs Dynamixel. Nous pouvons interfacer ces composants avec ROS sans aucun problème.
- Fonctionnalité inter-plateforme : le middleware ROS de transmission de messages permet la communication entre différents nœuds. Ces nœuds peuvent être programmés dans n'importe quelle langue disposant de bibliothèques client ROS. Nous pouvons écrire des nœuds haute performance en C++ ou C et d'autres nœuds en Python ou Java. Ce type de flexibilité n'est pas disponible dans d'autres cadres.

- **Modularité** : L'un des problèmes pouvant survenir dans la plupart des applications robotiques autonomes est que, si l'un des threads du code principal tombe en panne, l'application robot entière peut s'arrêter. Dans ROS, la situation est différente, nous écrivons différents nœuds pour chaque processus et si un nœud se bloque, le système peut encore fonctionner. En outre, ROS fournit des méthodes robustes pour reprendre le fonctionnement même si des capteurs ou des moteurs
- **Gestion des ressources simultanées** : La gestion d'une ressource matérielle par plus de deux processus est toujours un casse-tête. Imaginez, nous voulons traiter une image d'une caméra pour la détection de visage et la détection de mouvement, nous pouvons écrire le code comme une seule entité qui peut faire les deux, ou nous pouvons écrire un seul code thread pour la concurrence. Si nous voulons ajouter plus de deux fonctionnalités dans les threads, le comportement de l'application deviendra complexe et sera difficile à déboguer. Mais dans ROS, nous pouvons accéder aux appareils en utilisant les sujets ROS des pilotes ROS. N'importe quel nombre de nœuds ROS peut s'abonner au message d'image à partir du pilote de caméra ROS et chaque nœud peut effectuer différentes fonctionnalités. Cela réduit la complexité du calcul et augmente également la capacité de débogage de tout le système.
- **Communauté active** : Lorsque nous choisissons une bibliothèque ou un Framework logiciel, en particulier d'une communauté open source, l'un des principaux facteurs à vérifier avant de l'utiliser est son support logiciel et sa communauté de développeurs. Il n'y a aucune garantie de support d'un outil open source. Certains outils fournissent un bon support et d'autres non. Dans ROS, la communauté de support est active. La communauté ROS a une croissance constante dans les développeurs du monde entier.

2. Comprendre le niveau du système de fichiers ROS

Similaire à un système d'exploitation, les fichiers ROS sont également organisés sur le disque dur d'une manière particulière. Dans ce niveau, nous pouvons voir comment ces fichiers sont organisés sur le disque. Le graphique suivant montre comment les fichiers ROS et le dossier sont organisés sur le disque :

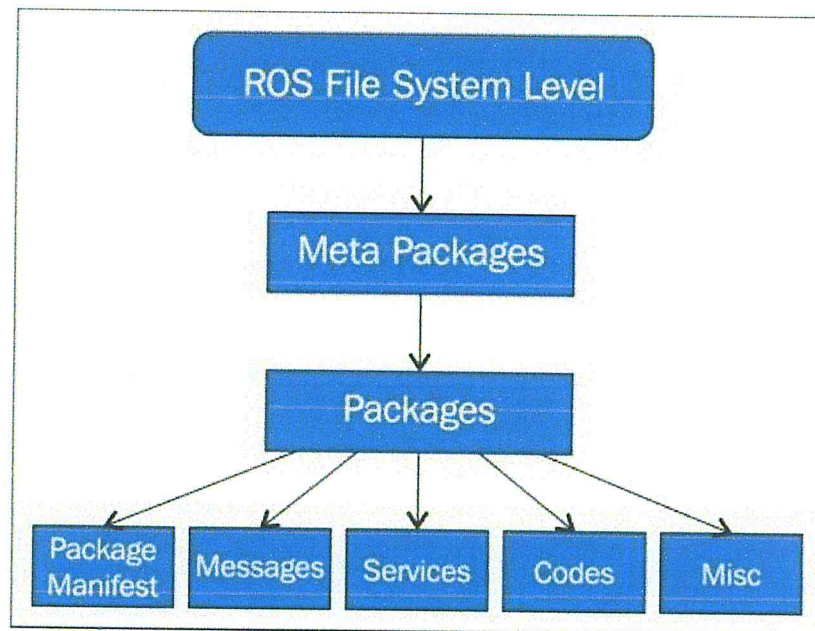


Figure 27: Le niveau du système de fichiers ROS

Similaire à un système d'exploitation, un programme ROS est divisé en dossiers, et ceux-ci

- **Package** : Les paquets forment le niveau atomique de ROS. Un package a la structure et le contenu minimum pour créer un programme dans ROS. Il peut avoir un processus d'exécution ROS (nœuds), des fichiers de configuration ; etc.
- **Manifest** : les manifestes fournissent des informations sur un paquet, des informations de licence, des dépendances, des drapeaux de compilateur, etc. Les manifestes sont gérés avec un fichier appelé `manifest.xml`
- **Stack** : Lorsque vous rassemblez plusieurs paquets avec certaines fonctionnalités, vous obtiendrez une pile. Dans ROS, il existe beaucoup de ces piles avec des utilisations différentes, par exemple, la navigation
- **Stack Manifests** : Les manifestes de pile (`stack.xml`) fournissent des données sur une pile, y compris ses informations de licence et ses dépendances sur d'autres piles.
- **Types de message (msg)** : un message est l'information qu'un processus envoie à d'autres processus. ROS a beaucoup de types standard de messages. Les descriptions de messages sont stockées dans `my_package / msg / MyMessageType.msg`.

- Types de service (srv): Les descriptions de service, stockées dans mon package / srv / MyServiceType.srv, définissent les structures de données de demande et de réponse pour les services dans ROS.

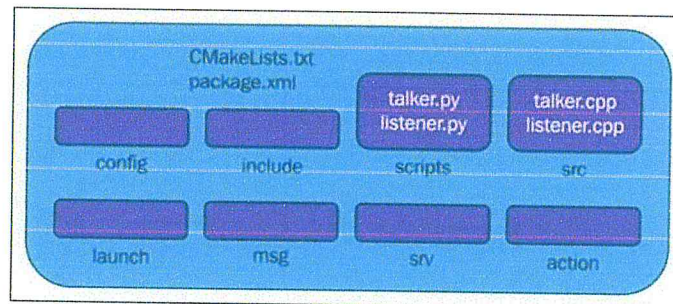


Figure 28: Structure de package typique

3. Comprendre le niveau de calcul graphique ROS

ROS crée un réseau où tous les processus sont connectés. Tout nœud du système peut accéder à ce réseau, interagir avec d'autres nœuds, voir les informations qu'ils envoient et transmettre des données au réseau.

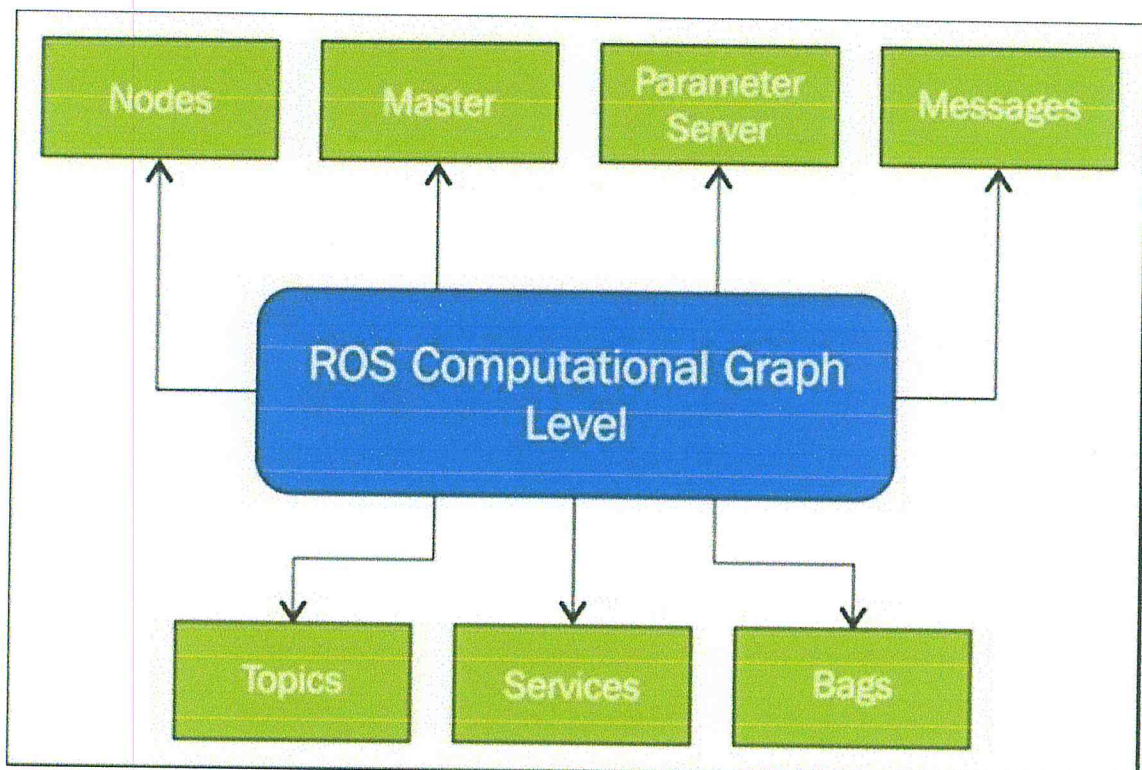


Figure 29: ROS calcul Niveau graphique

- **Nœuds** : Les nœuds ROS sont un processus qui effectue des calculs à l'aide de bibliothèques clientes ROS telles que roscpp et rospy. Un nœud peut communiquer avec d'autres nœuds à l'aide des rubriques, des services et des paramètres ROS.
- **Topic** : Chanel entre deux ou plusieurs nœuds, les nœuds communiquent en publiant et / ou en s'abonnant aux sujets appropriés
- **Services** : ROS utilise un langage de description de service simplifié pour décrire les types de service ROS. Cela se base directement sur le format de message ROS pour activer la communication de requête / réponse entre les nœuds. Les descriptions de service sont stockées dans le fichier .srv du sous-répertoire srv / d'un package.
- **Paramètres** : Le serveur de paramètres nous permet d'avoir des données stockées à l'aide de clés dans un emplacement central. Avec ce paramètre, il est possible de configurer un nœud pendant son exécution ou de modifier le fonctionnement des nœuds.

Annexe -B-

1. Installation de ROS Indigo :

Dans cette section, vous verrez les étapes à suivre pour installer ROS Electric sur votre ordinateur.

Nous supposons que le dépôt Ubuntu a été installé avec succès.

1.1. Configurez vos dépôts Ubuntu

Tout d'abord, vous devez vérifier que votre Ubuntu accepte les “universal”, “multiversal”, “repositories”

1.2. Installation de sources.list

Configurez votre ordinateur pour accepter les logiciels de packages.ros.org.

```
$sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'
```

1.3. Installation de la clé

Il est important d'ajouter la clé car avec cela nous pouvons être sûrs que nous téléchargeons le code depuis le bon endroit et qu'aucun corps ne l'a modifié.

```
$sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --recv-key 421C365BD9FF1F717815A3895523BAEEB01FA116
```

1.4. Installation

Avant de faire quelque chose, il est nécessaire de mettre à jour tous les programmes utilisés par ROS. Nous le faisons pour éviter les problèmes d'incompatibilité. Tapez la commande suivante dans un shell et attendez :

```
$sudo apt-get update
```

Il y a beaucoup de bibliothèques et d'outils différents dans ROS; celui installé et utilisé dans ce mot est desktop- full duplex

```
$sudo apt-get install ros-indigo-desktop-full
```

1.5. Initialisation rosdep

rosdep vous permet d'installer facilement les dépendances système pour la source que vous voulez compiler et est nécessaire pour exécuter certains composants de base dans ROS.

```
$sudo rosdep init
$rosdep update
```

1.6. installation de l'environnement

```
$echo "source /opt/ros/indigo/setup.bash" >> ~/.bashrc
$source ~/.bashrc
```

1.7. Obtenir rosinstall

```
$sudo apt-get install python-rosinstall
```

2. Étapes pour installer la Kinect sur ROS

Dans la suite, nous allons jeter un coup d'oeil sur le processus d'installation et d'exécution de la Kinect.

Tout d'abord, nous allons installer le pilote OpenNI et Kinect.

2.1. Installation de dépendances

```
$sudo apt-get install g++ python libusb-1.0-0-dev freeglut3-dev
$ sudo apt-get install doxygen graphviz mono-complete
$ sudo apt-get install openjdk-7-jdk
```

2.2.Instalation de OpenNI

```
$ git clone https://github.com/OpenNI/OpenNI.git
$ cd OpenNI
$ git checkout Unstable-1.5.4.0
$ cd Platform/Linux/CreateRedist
$ sudo chmod +x RedistMaker
$ ./RedistMaker
$ cd ../Redist/OpenNI-Bin-Dev-Linux-[xxx] $ sudo ./install.sh
```


2.3. Installation de driver kinect

```
$ git clone git://github.com/ph4m/SensorKinect.git
$ cd SensorKinect/Platform/Linux/CreateRedist
$ sudo chmod +x RedistMaker
$ ./RedistMaker
$ cd ../Redist/Sensor-Bin-Linux-x64-v*
$ sudo ./install.sh
```

Maintenant, nous installons `openni_launch` qui inclut des fichiers de lancement pour ouvrir un périphérique OpenNI et charger tous les modules pour convertir les flux de profondeur brute / RGB / IR en images de profondeur, images de disparité et nuages de points (enregistrés).

```
$ sudo apt-get install ros-indigo-openni-camera ros-indigo-openni-launch
```

2.4. Pour exécuter Kinect sur ROS

```
$ roslaunch openni_launch openni.launch
```

2.5. Pour visualiser les données Kinect

```
$ rosrn rviz rviz
```

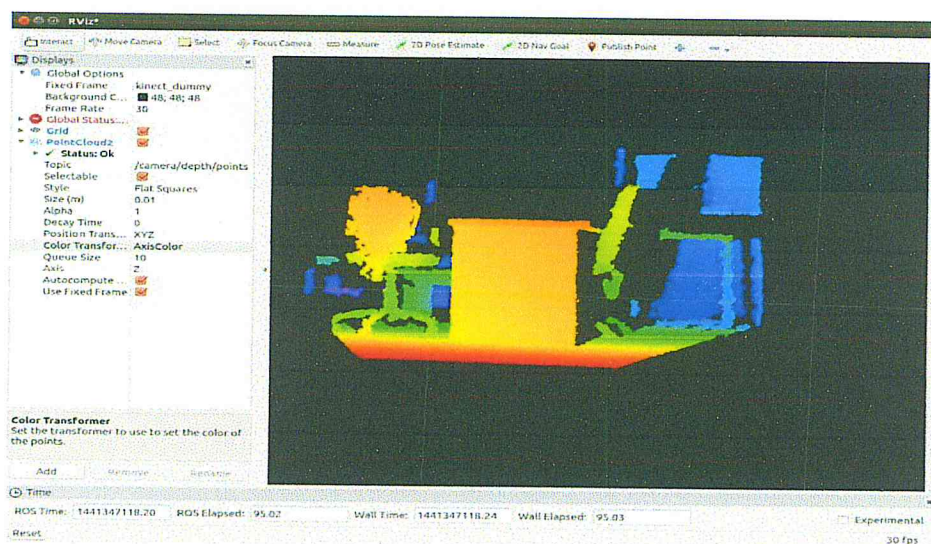


Figure 30: Le test de kinect dans le simulateur Rviz

Annex -C-

1. Description générale de RobuTER

RobuTER est une plateforme mobile robotisée disponible au Centre de Développement des Technologies Avancées (CDTA) d'Alger. Il s'agit d'une plate-forme mobile robotique non-holonyme rectangulaire, développée par la société française Robosoft.

La base du robot se compose d'une plate-forme avec deux roues et une capacité de charge de 15 kg. Les roues ont un diamètre de 250 mm et un couple nominal de 22 Nm par roue. Ils sont entraînés par des moteurs électriques à courant continu et permettent d'atteindre une vitesse nominale de 2,6 m / s. La direction de RobuTER est donnée par la vitesse différentielle des deux roues. Les deux roues sont placées à l'avant de la plate-forme pour fournir la stabilité.

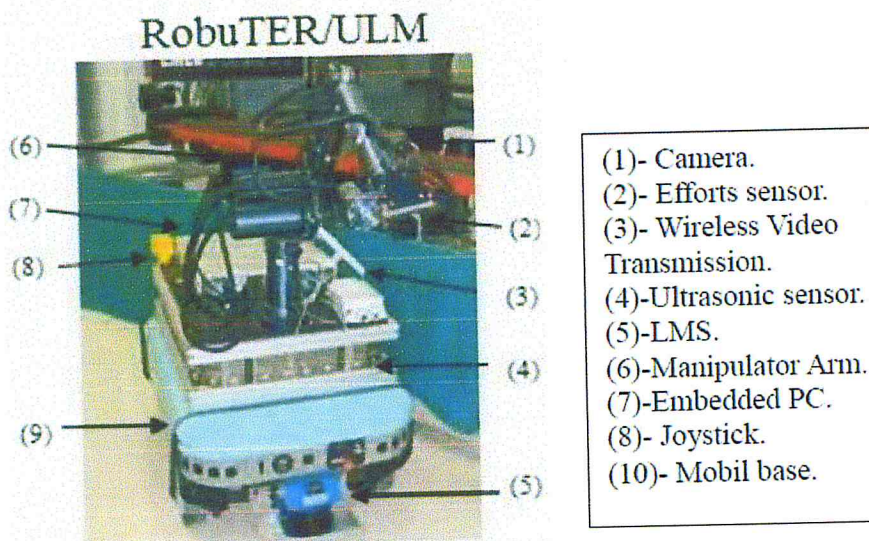


Figure 31: L'architecture du système robotique expérimental.

2. Architecture logicielle

Le PC embarqué de RobuTER fonctionne avec le système d'exploitation Linux RedHat 9.0, qui a l'avantage d'être libre, ouvert, sous licence GPL - General Public License, et très bien documenté avec tous les outils nécessaires au développement du RobuTER lui-même.

Le développement d'applications pour RobuTER est basé sur Robosoft Development Toolchain. Ce développement est basé sur l'environnement de CAO SynDEX.