الجمهـوريـــة الجزائـريـــة الديمقـراطيـــة الشـعبيـــة
République Algérienne Démocratique et Populaire
وزارة التـــعلـيـــــم العـــــالـــــي والبـحـــــث العـلمـــي
Ministère de l'Enseignement Supérieure et de la Recherche Scientifique

جامعة وهران 2 محمد بن أحمد
Université d'Oran 2 Mohamed Ben Ahmed

معهد الصيانة والأمن الصناعي
Institut de Maintenance et de Sécurité Industrielle

Département de Maintenanace en Instrumentation

MÉMOIRE

Pour l'obtention du diplôme de Master

Filière : Génie Industriel

Spécialité : Ingénieurie de la Maintenanace en Instrumentation

Thème

Étude et Réalisation d'un véhicule anti collision avec arduino

Présenté et soutenu publiquement par :

Berkani Ilhem

Devant le jury composé de :

Nom et Prénom	Grade	Etablissement	Qualité
Mme Benmansour Souhila	MCB	IMSI	**Président**
Mr Chennoufi Mohammed	MCA	IMSI	**Encadreur**
Mr Rouan Serik Mehdi	MCB	IMSI	**Examinateur**

Année 2019/2020

Remerciment

Je remercie en premier lieu, le grand Dieu pour le courage, la patience et la santé qui m'a donné pour suivre nes études.

Il est agréable d'adresser quelques expressions de remerciements et de reconnaissances à toute personne, dont l'intervention au cours de ce projet a favorisé son aboutissement.

Je tiens à exprimer mes profonds remerciements à mon professeur encadreur M. Chennoufi Mohammed pour ces conseilles qui m'a été judicieux.

J'adresse aussi mon plus vive reconnaissance à tous les enseignants de l'institut de maintenance et securité industriel (IMSI), département de maintenance en instrumentation ainsi qu'aux membres de jury qui ont accepté de juger mon travail.

Finalement, je remercie tous ceux qui ont contribué de près ou de loin à la réalisation de ce travail.

Dédicace

Mon voyage à l'université a pris fin après un travail acharné.

Et me voici, terminant ma recherche de fin d'études avec vigueur et activité,

Et je suis reconnaissant à tous ceux qui ont eu une vertu dans ma carrière,

Et aidez-moi même un peu,

Mere, frères et Sœurs, professeurs.

Je dédie ce modeste travail

Ilhem,

Abstract

Obstacle avoidance is an important task in the field of robotics, since the goal of the autonomous robot is to reach the destination without collision. Several algorithms have been proposed to avoid obstacles, with disadvantages and advantages. The present work mainly discussed different algorithms for robotic navigation with obstacle avoidance and path tracking. We also compared all the algorithms provided and mentioned their characteristics; pros and cons, so that we can select the effective final algorithm by merging the algorithms discussed.

Keywords: Autonomous control, obstacle avoidance , path planning , wheeled mobile robot.

Résumé

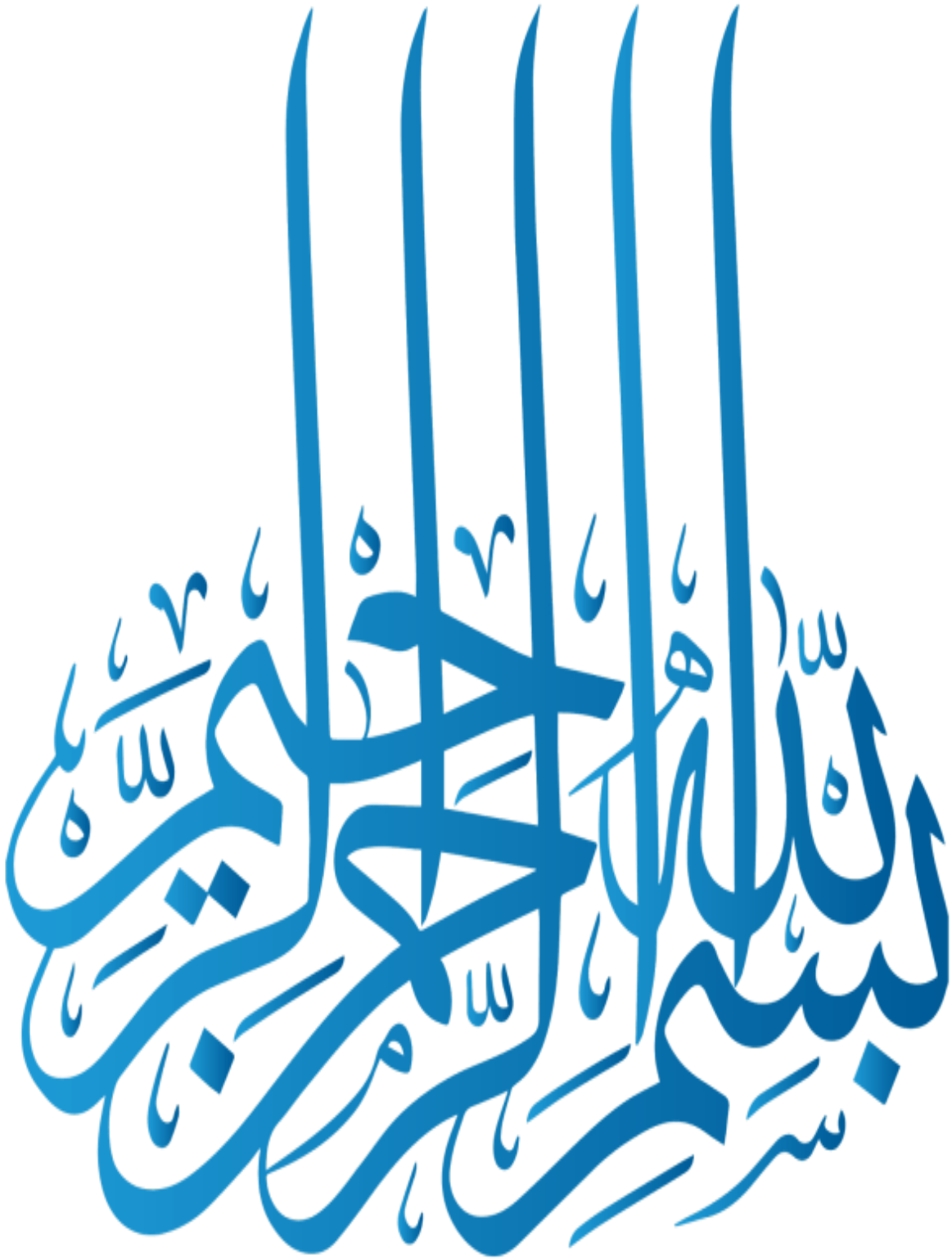
L'évitement des obstacles est une tâche importante dans le domaine de la robotique, puisque l'objectif du robot autonome est d'atteindre la destination sans collision. Plusieurs algorithmes ont été proposés pour éviter les obstacles, ayant des inconvénients et des avantages. Le present travail principalement discuté de différents algorithmes pour la navigation robotisée avec l'évitement des obstacles et le suivie de chemin. Nous avons également comparé tous les algorithmes fournis et mentionné leurs caractéristiques ; avantages et inconvénients, afin que nous puissions sélectionner l'algorithme final efficace en fusionnant les algorithmes discutés.

Mots clés: Contrôle autonome, évitement d'obstacles, planification de chemin, robot mobile à roues.

Table des matières

Table des figures

Liste des tableaux


بسم الله الرحمن الرحيم

Introduction

Ce projet a permis de développer un robot d'évitement d'obstacles qui peut se déplacer sans aucune collision en détectant les obstacles sur son parcours à l'aide d'un capteur de distance à ultrasons. Les robots guidés par cette technologie peut être utilisé à des fins diverses, par exemple pour l'étude des paysages, les véhicules, nettoyage autonome, tondeuse à gazon automatisée et robot superviseur dans les industries. Le robot développé dans le cadre de ce projet devrait permettre d'atteindre les objectifs suivants :

- Le robot aurait la capacité de détecter des obstacles sur son chemin en fonction d'une distance seuil prédéterminée.
- Après avoir détecté les obstacles, le robot changerait de trajectoire en prenant une décision autonome. - Il ne nécessiterait aucun contrôle externe pendant son fonctionnement.
- Il peut mesurer la distance entre lui et les objets environnants en temps réel.
- Il serait capable de fonctionner efficacement dans un environnement inconnu.

Première partie I

Généralité

Chapitre 1

Fondamentaux de la robotique

1.1 La Robotique

C'est une science qui traite des questions liées à la conception, à la fabrication et à l'utilisation des robots. En robotique, nous utilisons les bases de la physique, des mathématiques, du génie mécanique, du génie électronique, du génie électrique, de l'informatique et d'autres.[1]

1.2 Les types de la robotique

Les robots peuvent être regroupés en général sous le titre :

1. **Les robots manipulateurs** :Les robots ont une base fixe pour effectuer une tâche spécifique ou répétitive. Par exemple : les robots industriels et les robots sanitaires.[2]

Figure 1.1: robot manipulateur ; `https://rb.gy/tlcf50`

2. **Les robots mobiles** :Ce sont des robots capables de se déplacer dans leur environnement, les robots mobiles sur roues constituent la majorité des robots mobiles car ils sont plus faciles à construire que les autres types. Par exemple : les robots-explorateurs, les robots de service, les robots ludiques, les véhicules autonomes.[2]

Figure 1.2: robot mobile ; `https://rb.gy/lrkdzu`

3. **Les robots auto-reconfigurables** :des robots qui peuvent effectuer la tâche en question.[3]

Figure 1.3: Robot modulaire à changement de forme(auto reconfigurable) ; `https://rb.gy/scv8d4`

1.3 Base de connaissances pour la robotique

Base de connaissances typique pour la conception et le fonctionnement des systèmes robotiques :

— Modélisation et analyse des systèmes dynamiques

— Contrôle du retour d'information
— Capteurs et conditionnement du signal
— Les actionneurs (muscles) et l'électronique de puissance
— Matériel/interface informatique
— Programmation informatique [4]

1.4 Le Robot

Le terme robot vient du mot tchèque : robota qui signifie travailleur forcé ou esclave. en 1921, Karel Čapek, un dramaturge tchèque, a utilisé le terme robot pour la première fois dans son drame intitulé « Rossum's Universal Robots (RUR) »[1]

1.5 Utilisations des robots

1.5.1 Application dans les Environnements '4D'

Dangereux

Nous utilisons déjà des robots pour les situations militaires dangereuses, l'exploration de l'espace et les enquêtes sur les bombes et les explosions, et les entreprises commencent à développer des robots pour la sécurité et le travail de police ; un robot-flic fait déjà partie des forces de police de Dubaï.[5]

Figure 1.4: les traveaux dangereuse des robots ; `https://rb.gy/nbjriu` ; `https://rb.gy/fzmrai`

Sale

Il y a des tâches sales, dont la plupart ne sont pas vraiment connues de l'homme moyen, qui doivent être accomplies pour que notre monde continue de fonctionner. Les robots sont la solution idéale pour remplacer l'homme dans des tâches aussi diverses que la reconnaissance des égouts, la traite des vaches, les autopsies et l'exploration des mines. Le géant minier Rio Tinto a investi dans des robots-camions.
un programme qui est l'un des plus grands programmes de robots non militaires au monde.[5]

Figure 1.5: La saleté ; `https://rb.gy/pjaqyt`

Terne

Il y a beaucoup de tâches répétitives et fastidieuses pour lesquelles les robots sont particulièrement qualifiés. Lorsqu'ils s'en chargent, cela libère la main-d'œuvre humaine qui peut ainsi se consacrer à des activités plus créatives et plus intéressantes. Des centres d'exécution Amazon aux hôtels et même aux hôpitaux, les robots suppriment l'ennui sur de nombreux lieux de travail et permettent souvent de réaliser des économies.[5]

Figure 1.6: Les traveaux ternes ; `https://rb.gy/pjaqyt`

Difficile

Alors que la robotique développe son intelligence, sa dextérité et son autonomie, elle ouvre un nouveau chapitre dans les emplois où les robots sont mieux adaptés : des tâches "difficiles", calculées, qui requièrent une faible marge d'erreur et un niveau de détail élevé. Certaines des procédures les plus courantes effectuées par les fabricants de micropuces, les chirurgiens médicaux et les techniciens de laboratoire sont déjà partiellement ou totalement remplacées par la robotique de précision. [6, 5]

Figure 1.7: Le difficile - Un nouveau chapitre ; `https://rb.gy/pjaqyt`

1.5.2 Application pour les Tâches '4A'

Automatisation

À mesure que les machines deviennent plus intelligentes et/ou plus performantes, elles peuvent prendre en charge davantage de tâches qui étaient auparavant effectuées par les humains. Des distributeurs automatiques de billets aux kiosques d'auto-assistance dans les fast-foods, les dernières décennies ont vu l'apparition de nombreuses machines capables d'accomplir des tâches précieuses. Si les guichets automatiques et les kiosques d'entraide ont changé le travail de certains caissiers de banque et de restauration rapide, ils ne les ont pas complètement remplacés, mais les ont simplement poussés à remplir des rôles qui exigent des capacités plus nettement humaines. [7]

Figure 1.8: Automatisez les opérations de pick-and-place précises ; `https://rb.gy/d0a47m`

1.5.3 Augmentation

Les outils d'Automatisation des Processus par la Robotique (APR) peuvent aider les entreprises à améliorer l'efficacité et le rendement de leurs opérations plus rapidement et à moindre coût que les autres approches d'automatisation. Les avantages des solutions reposant sur l'APR vont au-delà de la réduction des coûts et couvrent les éléments suivants :

— Diminution des durées de cycle et amélioration des cadences de production.
— Flexibilité et possibilités d'extension.
— Amélioration de la précision.
— Amélioration du moral des employés – cela leur permet de créer davantage de valeur ajoutée.
— Permet de consacrer du temps à l'innovation et de se concentrer sur la satisfaction du client.[8]

Figure 1.9: la robotique d'entrepôt réduit les blessures des travailleurs ; `https://rb.gy/zxqcgd`

Assistance

Les roboticiens mettent actuellement au point des machines qui pourraient aider les patients dans leurs tâches de soins, comme les travaux ménagers, l'alimentation et la marche. Mais

avant qu'ils n'atteignent les bénéficiaires de soins, les robots d'assistance devront d'abord être acceptés par les prestataires de soins de santé tels que les infirmières et les aides-soignants. D'après une étude du Georgia Institute of Technology, il semble qu'ils puissent être accueillis à bras ouverts en fonction des tâches à accomplir.[9]

Figure 1.10: Vieillissement de la population : le Japon mise sur les robots plutôt que sur l'immigration ; `https://rb.gy/b5nrr0`

Autonomie

Les robots autonomes sont principalement le moteur de l'innovation et de la valeur de la chaîne d'approvisionnement en réduisant les coûts d'exploitation directs et indirects et en augmentant le potentiel de revenus. Plus précisément, les robots autonomes peuvent aider :

— Accroître l'efficacité et la productivité.
— Réduire les taux d'erreur, de reprise et de risque
— Effectuer des tâches banales et de moindre valeur afin que les humains puissent travailler en collaboration pour se concentrer sur des efforts plus stratégiques qui ne peuvent être automatisés.
— Augmenter les recettes en améliorant le taux d'exécution des commandes, la vitesse de livraison et, en fin de compte, la satisfaction des clients. Parmi les avantages potentiels secondaires des robots autonomes, on peut citer :
— Amélioration de la valeur des employés grâce à l'accent mis sur le travail stratégique plutôt que sur les tâches routinières.
— S'attacher à la sécurité personnelle en réduisant au minimum le travail dans les zones dangereuses pour les employés
— Renforcer la marque de l'entreprise en signalant les pratiques de pointe et la mise en œuvre de technologies innovantes.
— Apprentissage exponentiel par la collecte et l'analyse des données des machines.[10]

Figure 1.11: les robots autonomes collaboratifs sont d'excellents collègues ; `https://rb.gy/7zuovg`

1.6 Caractéristiques d'un robot

Un robot doit être choisi en fonction de l'application à laquelle il est destiné. Voici quelques paramètres à prendre en compte[11] :

- La charge maximale transportable.
- L'architecture de la S.M.A., le choix est guidé par la tâche à accomplir (quelle est la rigidité de la structure ?).
- Le volume de travail.
- Le positionnement absolu.
- Répétabilité
- La vitesse de déplacement.
- La masse du robot.
- Le coût du robot.
- La maintenance.

1.7 Classification des robots

Voici la classification des robots selon :

1. L'association japonaise des robots industriels (JIRA) :

   - Classe 1 :Appareil de manutention manuelle : un appareil à degrés de liberté multiples, actionné par un opérateur.

   - Classe 2 :Robot à séquence fixe : dispositif qui exécute les étapes successives d'une tâche selon une méthode prédéterminée et immuable, difficile à modifier

— Classe 3 :Robot à séquence variable : même chose que pour les classes 2, mais faciles à modifier.

— Classe 4 :Robot de lecture : un opérateur humain exécute la tâche manuellement en dirigeant le robot, qui enregistre les mouvements pour une lecture ultérieure ; le robot répète les mêmes mouvements en fonction des informations enregistrées

— Classe 5 :Robot à commande numérique : l'opérateur fournit au robot un programme de mouvement plutôt que de lui apprendre la tâche manuellement

— Classe 6 :Robot intelligent : un robot ayant les moyens de comprendre son environnement et la capacité de mener à bien une tâche malgré les changements des conditions environnantes dans lesquelles elle doit être exécutée

2. L'Institut Américain de la robotique (RIA) : considère uniquement les classes 3 à 6 de ce qui précède comme des robots. L'Association Française de Robotique (AFR) a la classification suivante :

— Type A : dispositifs de manipulation à commande manuelle pour la télérobotique
— Type B : dispositifs de manutention automatique à cycles prédéterminés
— Type C : robots programmables, servo-commandés, à trajectoire continue ou point à point
— Type D : identique à C mais avec la capacité d'acquérir des informations à partir de son environnement.[12]

1.8 Les composants du robot

Tous les robots ont trois types de composants [13] :

1. **Système de contrôle** : comme un organe de contrôle.

2. **Capteurs** : ils peuvent lire des informations sur l'environnement ou sur le robot lui-même.

3. **Les actionneurs** : ils ont un effet sur l'environnement ou sur le robot.

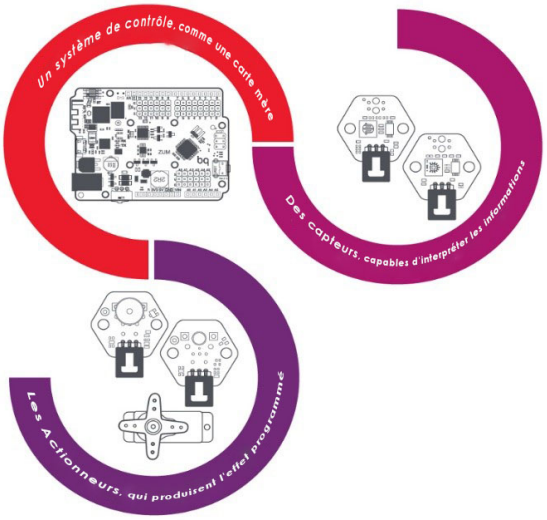


Figure 1.12: Les composants d'un robot ; [13]

Chapitre 2

Le robot mobile

2.1 Introduction

Le monde de la robotique est l'un des domaines les plus passionnants qui a connu une innovation et une évolution constantes. Il peut s'agir d'une technologie complète intégrant la technologie mécanique, la technologie de gestion automatique, l'ingénierie, l'IA, l'électronique, l'optique, la technologie de communication, la technologie des éléments de détection, etc.
Ce domaine est interdisciplinaire et fait de plus en plus partie de notre vie. C'est une orientation privilégiée de la recherche scientifique et technologique actuelle. Il ne s'agit pas d'une vision de l'avenir mais d'une réalité du présent. Les robots se développent rapidement, passant d'environnements industriels physiquement fixés à leur lieu de travail à des machines de plus en plus complexes capable d'accomplir des tâches difficiles dans notre environnement quotidien.[14]

2.2 Les Types de robot

Robot de taches (par application)

1. *Robots industriels* :Un robot industriel est défini par l'ISO comme un manipulateur polyvalent, reprogrammable, à commande automatique et programmable sur trois axes

ou plus. Le domaine de la robotique peut être plus spécifiquement défini comme l'étude, la conception et l'utilisation de systèmes robotiques pour la production (une définition de haut niveau basée sur la définition précédente du robot). Les applications typiques des robots comprennent la soudure, la peinture, l'assemblage, le "pick and place" (comme l'emballage, la palettisation et le SMT), l'inspection et le test des produits ; toutes ces opérations sont réalisées avec une grande robustesse, rapidité et précision.[15]

Figure 2.1: robot industriel ; [15]

2. *Robots médicaux* :Les robots médicaux sont des robots qui permettent aux chirurgiens d'accéder plus facilement aux zones opérées en utilisant des méthodes plus précises et moins invasives. On les trouve dans la plupart des télémanipulateurs, qui utilisent les actions du chirurgien d'une part pour contrôler l'effecteur" d'autre part.[15]

Figure 2.2: robot medicale ; [15]

3. *Robots militaires* :Les robots militaires sont des robots autonomes ou des dispositifs télécommandés conçus pour des applications militaires. Plusieurs armées étudient actuellement de tels systèmes. L'armée a toujours été à la pointe de la technologie, il n'est donc pas surprenant que les robots les plus avancés au monde soient construits en vue d'applications militaires. Si l'idée de machines autonomes portant des armes lourdes peut rendre les gens un peu nerveux, elles ont le potentiel de réduire considérablement les pertes de vies humaines en permettant aux soldats de localiser des endroits sûrs ou de pénétrer dans des sites ennemis. Beaucoup sont même conçus à des fins de soutien, plutôt que pour éliminer les menaces. Alors que de nombreux projets militaires dans le monde sont naturellement entourés de secret, certains émergent, notamment des projets de robotique.[15]

Figure 2.3: robot militaire ; [15]

Robot stationnaires

1. *Bras robotiques - Robots articulés* :un robot articulé est un robot à articulations rotatives (par exemple, un robot à pattes ou un robot industriel). Les robots articulés peuvent varier de simples structures à deux articulations à des systèmes à 10 articulations ou plus en interaction. Ils sont alimentés par divers moyens, dont des moteurs électriques.[15]

Figure 2.4: bras robotique ; [15]

2. *Robots cylindriques* :Les robots cylindriques sont des robots dont les axes forment un système de coordonnées cylindriques.[15] Il est utilisé pour :
   - Opérations d'assemblage.
   - Manipulation sur les machines-outils.
   - Soudage par points.
   - Fonctionnement des machines de coulée sous pression.

Figure 2.5: robot cylindrique ; [15]

3. *Robots sphériques* :Un robot sphérique est un robot doté de deux articulations rotatives et d'une articulation prismatique, c'est-à-dire de deux axes rotatifs et d'un axe linéaire. Les robots sphériques ont un bras qui forme un système de coordonnées sphériques.[15]

Figure 2.6: robot sphérique ; [15]

Robot à roues

1. *Les robots Multi roues* :La conception devient beaucoup plus complexe, surtout lorsqu'on utilise des roues plus puissantes, car chaque roue doit tourner à la même vitesse lorsque le robot doit avancer. Les différences de vitesse entre les roues gauche et droite des robots à direction différentielle font que le robot se déplace latéralement au lieu d'aller tout droit. La différence de vitesse entre les roues d'un même côté fait que la roue la plus lente tourne.[15]

Figure 2.7: robot a plusieurs roues ; [15]

Robots logiciel

1. *Intelligence artificielle* :C'est l'intelligence dont font preuve les machines. En informatique, le domaine de la recherche en IA est défini comme l'étude des "agents intelligents" : tout dispositif qui perçoit son environnement et prend des mesures qui maximisent ses chances de succès dans un objectif donné. Dans le langage courant, le terme "intelligence artificielle" s'applique lorsqu'une machine imite les fonctions "cognitives" que les êtres humains associent à d'autres esprits humains, telles que "l'apprentissage" et "la résolution de problèmes". À mesure que les machines deviennent de plus en plus puissantes, les structures mentales, dont on pensait qu'elles nécessitaient de l'intelligence, sont retirées de la définition.[15]
2. *Télégramme Bot* :Il s'agit d'un service de messagerie instantanée gratuit basé sur le Cloud.

Figure 2.8: l'intelligence artificielle ; [15]

Il existe des clients télégraphiques pour les systèmes mobiles (Android, iOS, Windows Phone, Ubuntu Touch) et de bureau (Windows, MacOS, Linux). Les utilisateurs peuvent envoyer des messages et échanger des photos, des vidéos, des autocollants, des fichiers audio et toutes sortes de fichiers. Le télégramme offre également la possibilité d'envoyer des messages cryptés d'un bout à l'autre. Le télégramme est soutenu par l'entrepreneur russe Pavel Durov. Son code côté client est un logiciel libre mais contient des blocs binaires, et le code source des versions récentes n'est pas toujours publié immédiatement, tandis que son code côté serveur est fermé et propriétaire.[15]

Figure 2.9: Telegramme bot ; [15]

Les Robots volants

Il n'est pas facile de regrouper les robots volants en sous-groupes. Comme les robots volants utilisent différents types de technologie, nous regrouperons ces robots en fonction de leur système de vol.
Pour l'instant, nous avons défini les types suivants (d'autres types seront ajoutés prochainement).

- Robot-ballon : ces types de robots volants utilisent une montgolfière pour flotter dans le ciel.
- Robot à voilure tournante : Robots volant comme un hélicoptère (Quadrocoptères).
- Robots à battements d'ailes : Ces robots volent comme un oiseau avec des ailes.
- Robots d'avion : Des robots qui volent comme un avion.[15]

Figure 2.10: les robots volants ; [15]

Robots de natation

Il existe différents types de robots nageurs (ou robots poissons). Nous en avons vu quelques-uns ici :

- PacX Wave Glider.
- Poisson festo.
- Le robot nageur « Swumanoid ».[15]

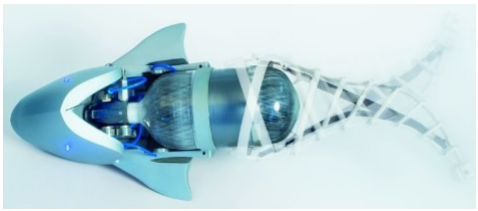


Figure 2.11: robots de natation ; [15]

Robots élastiques souples

Robots élastiques flexibles : des chercheurs de Harvard ont combiné la chimie organique, la science des matériaux flexibles et la robotique pour créer un robot flexible inspiré par des animaux comme le calmar et les vers. Le robot flexible n'a pas de squelette interne dur et n'utilise pas de capteurs.[15]

Figure 2.12: Robot elestique souples ; [15]

Robots modulaires

Les systèmes robotiques modulaires à reconfiguration automatique ou les robots modulaires à reconfiguration automatique sont des machines cinématiques autonomes à morphologie variable. Au-delà de la conduite, de la détection et du contrôle classiques, typiques des robots à morphologie fixe, les robots autoconfigurants sont également capables de modifier délibérément leur forme en réorganisant la connectivité de leurs parties pour s'adapter à de nouvelles circonstances, accomplir de nouvelles tâches ou se remettre de dommages.[15]

Figure 2.13: Robot modulaires ; [15]

Les Robots essaims

La robotique d'essaimage est une nouvelle approche de la coordination des systèmes multirobots constitués d'un grand nombre de robots physiques pour la plupart simples. On suppose qu'un comportement collectif souhaité résulte des interactions entre les robots et des interactions des robots avec l'environnement. Cette approche est apparue dans le domaine de l'intelligence artificielle des essaims, ainsi que dans les études biologiques des insectes, des fourmis et d'autres zones de la nature où se produit le comportement des essaims.[15]

Figure 2.14: Les robot d'essaims ; [15]

Les micro-robots

La microbotique (ou microrobotique) est le domaine de la robotique miniature, en particulier des robots mobiles dont les dimensions caractéristiques sont inférieures à 1 mm Le terme peut également être utilisé pour les robots capables de manipuler des composants de dimensions micrométriques. La microbotique est la branche de la robotique qui s'occupe de l'étude et de l'application de robots miniatures, tels que les robots mobiles à l'échelle micrométrique.[15]

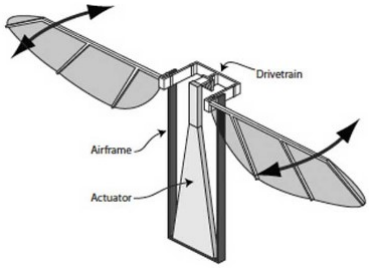


Figure 2.15: Les micro-robot ; [15]

Les nano Robots

Nano robot : la nano-robotique est le secteur technologique émergent qui crée des machines ou des robots avec des composants à l'échelle nanométrique (0,000000001 mètre - 10-9 mètres). En particulier, la nano-robotique désigne la discipline de l'ingénierie nanotechnologique qui consiste à concevoir et à construire des nanorobots, avec des dispositifs d'une taille comprise entre 0,1 et 10 micromètres, qui sont construits à l'échelle nanométrique ou avec des composants moléculaires. Les noms : - Nanobot.
- Nanoids.
- Nanites.
- Nanomachines ou nanomachines.

Ces dispositifs ont également été utilisés pour décrire des appareils qui font actuellement l'objet de recherches et de développement.[15]

Figure 2.16: Les nano-robot ; [15]

2.3 Caractéristiques du robot mobile

Un robot peut être décrit comme un système électromécanique ou une machine qui peut être commandée au moyen d'un programme informatique ou d'un équipement électronique (microprocesseur, microcontrôleur...) ; il existe de nombreuses définitions du robot, mais il présente certaines caractéristiques importantes telles que :

1. **Détection** : à l'aide de divers capteurs tels que les capteurs de lumière, les capteurs tactiles, le sonar, il doit détecter le milieu environnant.
2. **Locomotion** : Il est nécessaire de se déplacer dans l'environnement de travail ou dans le milieu de travail à l'aide de roues ou de jambes.
3. **Énergie** : pour réaliser la tâche souhaitée, il est nécessaire de consommer de l'énergie. Il peut donc être alimenté par des piles, de l'électricité ou de l'énergie solaire, selon le cas.
4. **L'intelligence** : comme nous, elle permet aux robots de recueillir des informations sur leur environnement et leur permet de commencer à prendre des décisions pour eux-mêmes.[14]

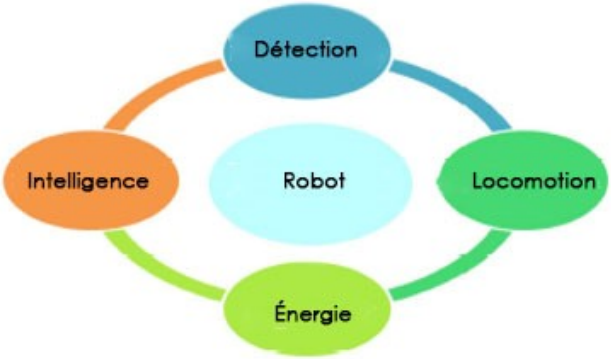


Figure 2.17: Les Caractéristiques du robot mobile ; [14]

2.4 Locomotion

2.4.1 Introduction

- Un robot mobile a besoin de mécanismes de locomotion qui lui permettent de se déplacer sans limites dans son environnement. Mais il existe une grande variété de moyens de déplacement possibles, et le choix de l'approche de la locomotion d'un robot est donc un aspect important de la conception d'un robot mobile. Dans le laboratoire, il y a des robots de recherche qui peuvent marcher, sauter, courir, glisser, patiner, nager, voler et, bien sûr, rouler.[16].

2.4.2 Questions clés pour la locomotion

Par conséquent, la locomotion et la manutention partagent les mêmes problèmes fondamentaux de stabilité, de caractéristiques de contact et de type d'environnement [16] :

1. **stabilité**
   - Le nombre et la géométrie des points de contact
   - Centre de gravité
   - Stabilité statique/dynamique
   - la pente du terrain

2. **Les caractéristiques du contact**
   - la taille et la forme du point de contact/trajectoire
   - angle de contact
   - friction

3. **Type d'établissement**
   - Structure
   - Moyen, (par exemple l'eau, l'air, un sol mou ou dur)

2.4.3 Robots mobiles à roues

La roue a été de loin le mécanisme de locomotion le plus populaire dans la robotique mobile et les véhicules construits par l'homme en général. Il peut atteindre d'excellentes performances, et le fait avec une mise en œuvre mécanique relativement simple. De plus, l'équilibre n'est généralement pas un problème de recherche dans la conception des robots à roues, car les robots à roues sont presque toujours conçus de manière à ce que toutes les roues soient toujours en contact avec le sol. Par conséquent, trois roues suffisent pour assurer un équilibre stable, même si, comme nous le verrons plus tard, les robots à deux roues peuvent également être stables. Lorsque vous utilisez plus de trois roues, vous avez besoin d'un système de suspension qui permet à toutes les roues de rester en contact avec le sol lorsque le robot rencontre un terrain inégal.[16]

La Conception de la roue

Il existe quatre principaux types de roues, comme le montre la figure 2.18 . Elles diffèrent grandement par leur cinématique, et le choix du type de roue a donc un effet important sur la cinématique globale.[16]

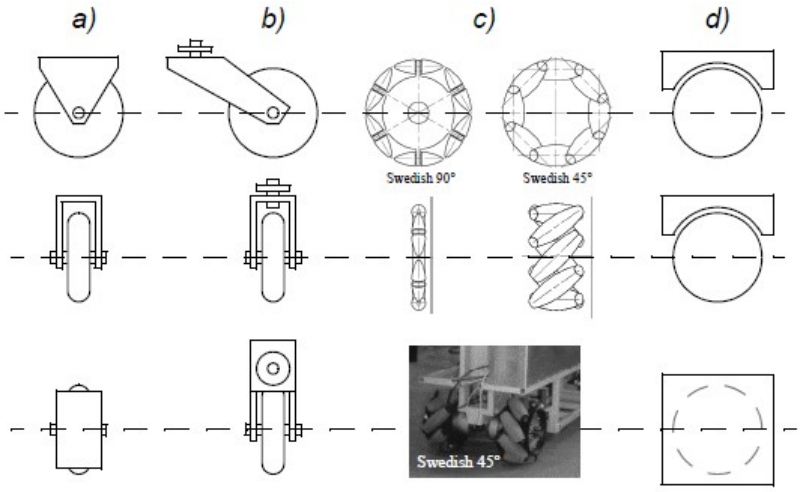


Figure 2.18: Les quatre types de roues de base. (a) Roue standard.(b) roue à pivot.(c) Roue suédoise. (d) Bille ou roue sphérique ; [16]

Nbre de roues	Description	Exemples typiques
2	Une roue directrice à l'avant, une roue motrice à l'arrière	Bicycle, motocycle
3	Deux roues libres à l'arrière, une roue motrice directrice à l'avant	Neptune(Université , Carnegie Mellon) Héro-1
4	Deux roues motrices et directrices à l'avant, deux libres les roues arrière	Voiture à traction avant
6	Deux roues motrices (différentiel) au centre une roue omnidirectionnelle à chaque coi	Terregator (Université Carnegie Mellon)

Table 2.1: Configugarion des roues

Les enjeux de la locomotion sur roues

1. Stabilité
2. Manœuvrabilité
3. Contrôlabilité[16]

Configuration des roues

Voir Table 2.1[16]

2.4.4 Autres concepts

La locomotion par roues et pattes est la plus utilisée et la plus conçue pour les robots mobiles. Mais il existe d'autres concepts, dont deux sont la locomotion sur rails et la combinaison de la locomotion par roues et par pattes.[16]

Locomotion à glissement/dérapage sur chenilles

La locomotion sur roues présente certains inconvénients, en particulier dans le cas de véhicules omnidirectionnels utilisant des roues sphériques ou suédoises, sur des terrains

accidentés et meubles, en raison d'une friction de roulement accrue due au manque d'efficacité énergétique ; en outre, les véhicules utilisant des roues sont à peine capables de combler des vides dont le diamètre est inférieur à celui des roues du véhicule. Un exemple de robot utilisant ce concept est le robot Nanokhod, qui se rendra probablement sur Mars, dans des véhicules à locomotives glissantes/dérapantes qui utilisent les rails comme réservoir. Un véhicule à chenilles est conduit en déplaçant les chenilles à différentes vitesses dans le même sens ou dans des directions opposées. L'utilisation de rails offre une surface de contact au sol beaucoup plus grande, de sorte que la traction du véhicule sur une surface libre est bien meilleure que la traction des roues. [16]

Roues de marche

Les robots à pattes sont capables de monter des escaliers et de se déplacer sur des terrains accidentés, mais ils présentent quelques inefficacités sur les surfaces planes et sont difficiles à contrôler. Les robots sur roues sont très efficaces sur les surfaces dures, même à grande vitesse, mais la plupart d'entre eux ne sont pas capables de monter les escaliers. Une idée est une solution hybride qui combine les avantages des jambes et de la locomotion sur roues. L'une des caractéristiques les plus intéressantes de la crevette est qu'elle est capable de surmonter passivement les obstacles, c'est-à-dire que le robot n'a pas de capteurs pour détecter un obstacle, la structure mécanique du robot est capable d'adapter le profil du terrain. [16]

Deuxième partie II

Étude Théorique

Chapitre 3

Capteurs et perception

3.1 Introduction

L'une des tâches les plus importantes de tout système autonome est d'acquérir des connaissances sur son environnement. Cela se fait en prenant des mesures avec différents capteurs et en extrayant ensuite des informations significatives de ces mesures. L'objectif de ce chapitre

Figure 3.1: Capteurs et perception ; [17]

est de décrire le rôle des capteurs et de la perception. La perception fournit au robot des connaissances qui lui permettent de comprendre ce qui se passe dans son environnement. Elle peut donc élaborer un plan et agir sur le monde [17].

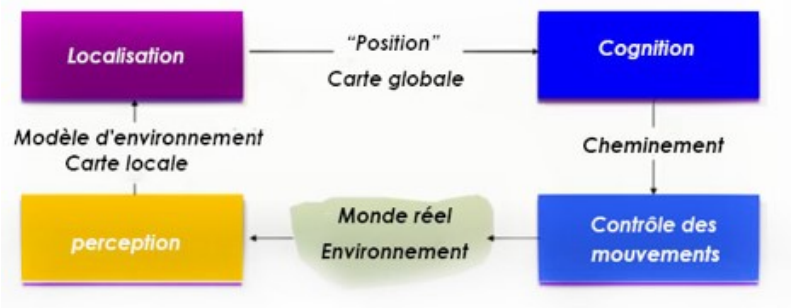


Figure 3.2: schéma generale de la perception ; [17]

3.2 Système de détection

On parle de perception lorsque le robot utilise les informations des capteurs pour extraire des informations utiles sur les robots internes ou externes. Un exemple d'état interne serait l'alimentation par batterie, les effecteurs ou la conscience si tout fonctionne dans les yeux du robot.
Un exemple d'état extérieur serait le monde des robots. Où est le robot ? Quels objets sont arrondis, etc. Un espace d'état est formé par tous les états possibles d'un système : la perception du monde par le robot et ce qu'il trouve à partir d'un ensemble de capteurs externes, et l'état interne est la perception que le robot a de lui-même, qui est obtenue à partir de la représentation proprioceptive des capteurs, où un modèle interne est créé lorsqu'un robot utilise son état interne pour se rappeler des informations sur le monde. L'espace sensoriel ou espace perceptuel du robot est l'espace de toutes les lectures sensorielles possibles basées sur tous les capteurs du robot [17].

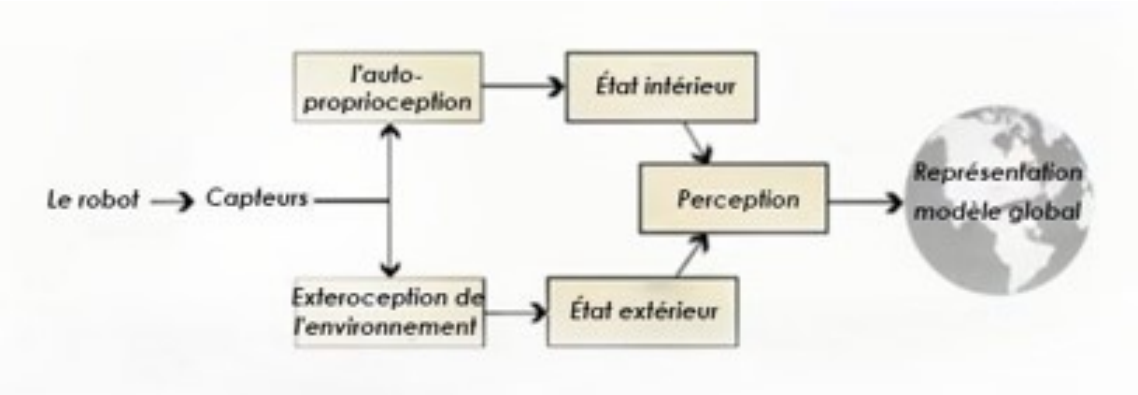


Figure 3.3: système de détection source ; [17]

3.3 Classification des capteurs

Nous classons les capteurs selon deux axes fonctionnels importants : proprioceptif/extéroceptif et passif/actif [18].

1. **Les capteurs proprioceptifs** :mesurer les valeurs au sein du système (robot), par exemple la vitesse du moteur, la charge des roues, la direction du robot, l'état de la batterie.

2. **Les capteurs extéroceptifs** :acquérir des informations sur l'environnement du robot, par exemple les distances par rapport aux objets, l'intensité de la lumière ambiante, les

caractéristiques uniques.

3. **Les Capteurs passifs** : Mesure de l'énergie environnementale ; fortement influencée par l'environnement.

4. **Les Capteurs actifs** :émettent leur propre énergie et mesurent la réaction ; meilleures performances, mais une certaine influence sur l'environnement.

3.4 Capteurs infrarouges

La lumière proche infrarouge peut être produite par une LED ou un laser. La longueur d'onde typique de la lumière infrarouge émise est comprise entre 820 et 880 nm. En conséquence, la plupart des surfaces ont une rugosité supérieure à la longueur d'onde de la lumière incidente et il se produit une réflexion diffuse, c'est-à-dire que la lumière est réfléchie de manière presque isotrope. La composante de la lumière infrarouge tombant dans l'ouverture du capteur revient presque parallèlement au faisceau émis pour les objets éloignés (figure 3.4A). Le capteur envoie une lumière modulée en amplitude à 100% à une fréquence donnée f et mesure le déphasage entre les signaux transmis et réfléchis. Si ϕ est le déphasage mesuré électroniquement et est la longueur d'onde, alors la distance 2D est $(\phi/2\pi)\lambda$, c'est-à-dire (figure 3.4B)[19].

$$D = (\frac{\lambda}{4\pi})\phi. \tag{3.1}$$

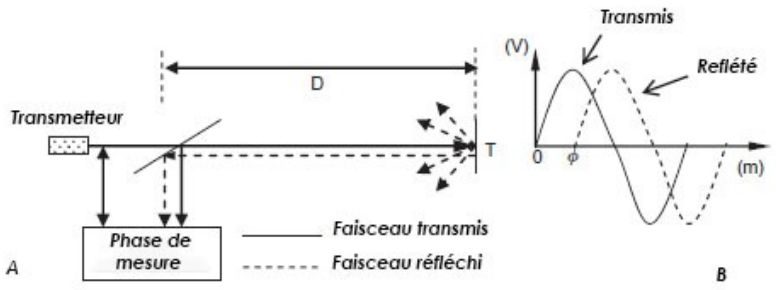


Figure 3.4: (A) La mesure de la distance par infrarouge utilise la méthode du déphasage.(B) Déphasage entre les signaux émis et reçus ; [19]

- Un véritable capteur infrarouge devant un obstacle mural est illustré à la figure 3.5

Figure 3.5: Capteur infrarouge monté sur une plate-forme robotique mobile (les capteurs infrarouges peuvent être utilisés pour éviter les obstacles, suivre les lignes et établir des cartes) ;[19]

3.5 La fusion des capteurs

La fusion de capteurs est un terme général utilisé pour tout processus qui combine les informations de plusieurs capteurs en un seul. La motivation de la fusion des capteurs provient de trois combinaisons de base de capteurs : redondants (ou concurrents), complémentaires et coordonnés. Bien que de nombreux chercheurs considèrent la fusion des capteurs comme un moyen de construire un modèle global dans un système hiérarchique ou délibératif, la fusion des capteurs peut être incorporée dans le comportement par la fission des capteurs, la fusion des capteurs orientée vers l'action et le mode de fonctionnement des capteurs.[20]

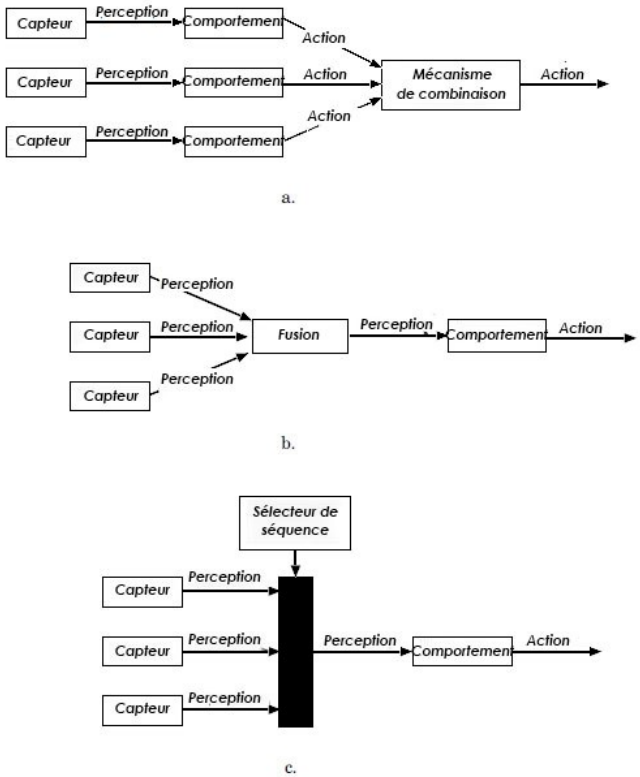


Figure 3.6: Trois types de fusion comportementale des capteurs : a.) fission des capteurs, b.) fusion des capteurs orientée vers l'action, et c.) mode des capteurs ; [20]

3.6 Les défis liés aux capteurs

Voici quelques-uns des défis à relever [17] :

1. **le champ de vision** : la disposition des images permet de couvrir la bonne zone.
2. **précision et répétabilité** : comment ça marche ? .
3. **Réponse** : dans la zone cible, dans quelle mesure cela fonctionne-t-il bien pour cette zone ?.
4. **Consommation électrique** : les piles se déchargent-elles trop vite ?.
5. **fiabilité** : peut-être un peu floue ou vulnérable.
6. **dimension** : il s'agit toujours de s'adapter au robot.
7. **calcul** :la complexité permet de traiter l'information assez rapidement.
8. **la fidélité interprétative** : croyez-vous en ce que vous dites ?.

Chapitre 4

Contrôle de robot mobile

4.1 Introduction

Voici un autre excellent graphique que j'ai utilisé pour représenter le panorama du contrôle des robots mobiles Certains des défis du contrôle des robots mobiles sont que le monde change de manière dynamique Il n'existe pas de modèle compact et les sources d'erreur sont multiples car nous avons déjà discuté du fait que la plupart des fonctions sont locales et n'impliquent pas de planification de la localisation ou de la cognition et que la trajectoire globale est plus lente et ne devrait être effectuée que lorsqu'un contrôle local est nécessaire et un contrôle plus fréquent dans le monde réel, comme l'itinérance aléatoire ou le contournement d'obstacles, ou pendant la surveillance et la surveillance globale est moins fréquente lorsque cela est nécessaire, comme dans le cas de la localisation, de la planification des trajectoires, de la cartographie, etc[17].

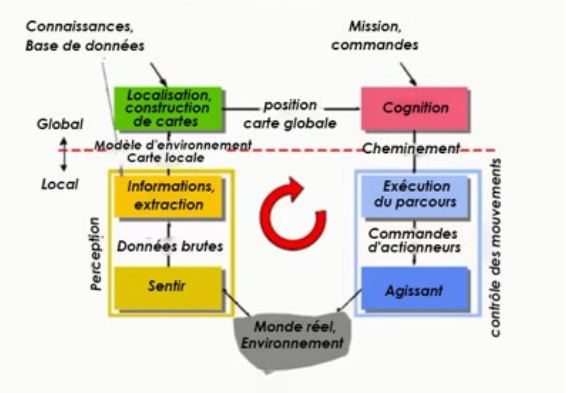


Figure 4.1: Schéma de contrôle de référence pour les systèmes de robots mobiles ; [17]

4.2 Approches de contrôle

Il existe des différences essentielles dans les architectures de contrôle et sont décrites sur la base de l'intelligence artificielle classique ou moderne[17].

1. L'IA classique : est basée sur un modèle de fonctionnement complet basé sur des horizons avec décomposition est très similaire au contrôle délibératif ou hiérarchique, en observant le graphique qui indique que pour le contrôle de l'IA classique.

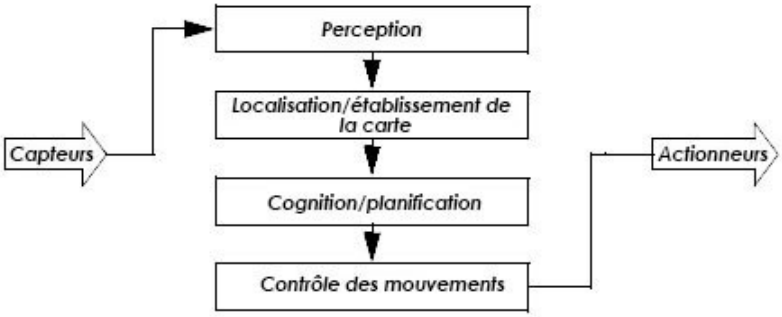


Figure 4.2: l'IA classique source ; [17]

2. L'IA moderne :est similaire à la façon dont les systèmes de contrôle linéaire sont séparés en contrôle classique et moderne : le contrôle classique est dans le domaine des fréquences et a des fonctions de transfert avec une entrée et une sortie, tandis que le contrôle moderne est dans le domaine de l'espace-temps ou de l'état et il y a plusieurs entrées et plusieurs sorties ; rappelez-vous qu'il n'y a pas une seule approche correcte ; cela dépend juste de ce qui est idéal pour votre système, vos objectifs ou la tâche du robot.[17]

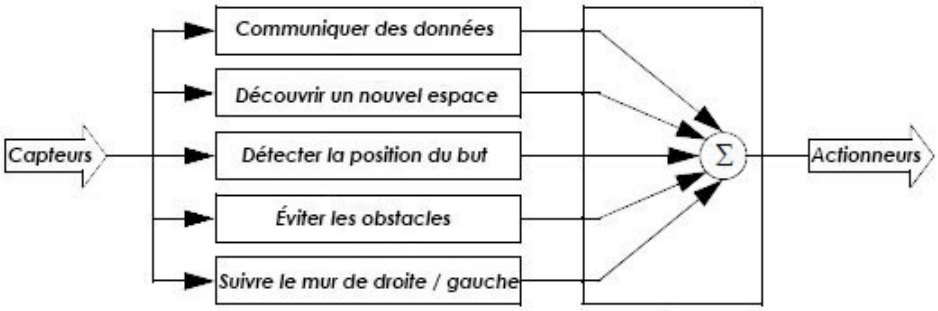


Figure 4.3: l'IA moderne ; [17]

4.3 Contrôle des retours d'information

4.3.1 Boucle ouverte /Boucle fermée

Nous avons ici deux exemples de robot suivant un mur [17] :

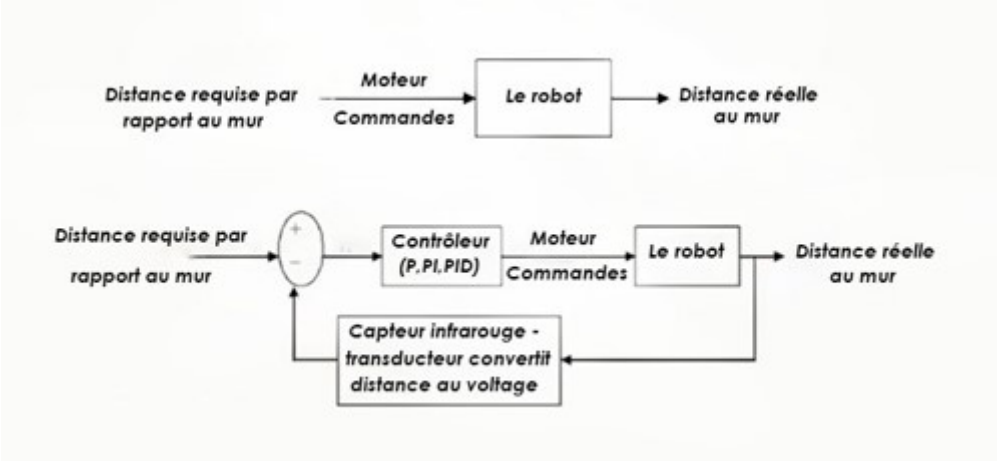


Figure 4.4: Le controle de retour d'information : en haut : Boucle ouverte et en bas : Boucle fermée ; [17]

Dans le premier modèle, les commandes sont données au moteur et le robot se déplace mais sans correction d'erreur ; dans le second, un capteur infrarouge, un sonar, un laser ou un autre est utilisé pour obtenir la distance voulue. Dans le second modèle, le contrôleur ajuste la puissance du moteur ou le rapport de rotation pour corriger la distance afin que le robot suive le mur plus précisément plutôt que par à-coups.

4.3.2 Le signal d'erreur

Dans le contrôle par rétroaction, un système atteint et maintient l'état souhaité en comparant continuellement son état actuel et l'état souhaité, puis en ajustant l'état actuel pour minimiser la différence, de sorte que l'erreur est la différence entre l'état actuel et l'état souhaité et peut avoir une erreur nulle ou non nulle, et indique s'il y a ou non une erreur et c'est la moindre information que vous pouvez avoir mais la plus utile est la direction de l'erreur ; la direction à prendre pour minimiser l'erreur ; vous pouvez également avoir la magnitude de l'erreur qui donne la distance de l'état cible où vous pouvez avoir le dépassement, c'est-à-dire lorsque le robot passe son point ou sa position cible ou change de direction avant de stabiliser le

contrôle ; c'est beaucoup plus facile si vous connaissez à la fois la magnitude et la direction de l'erreur [17].

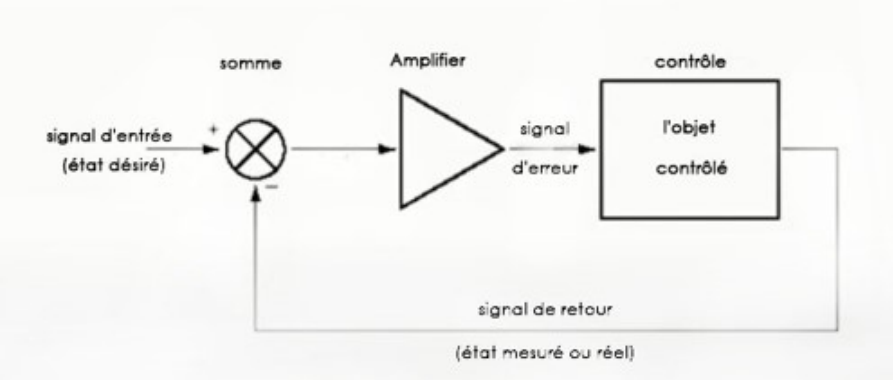


Figure 4.5: schéma illustratif sur le signal d'erreur ; [17]

4.3.3 Les contrôleurs de retour d'information

Il existe différents types de contrôleurs de rétroaction ; dans cet exemple, le robot est dans ou hors spécifications et tourne ou se déplace en ligne droite. Ce serait un mouvement très brusque, puisque le robot essaie de suivre un mur. Vous pourriez aussi avoir un contrôle proportionnel, ce qui est un peu mieux car les vitesses sont ajustées en fonction de la taille de l'air plutôt que d'un angle de rotation ou d'une puissance fixe. Les contrôleurs dérivés affectent la réponse transitoire et sont parfois appelés contrôleurs de type PID ou avant même que les contrôleurs dérivés proportionnels intégraux ne redémarrent l'erreur afin que le robot convertisse à la distance souhaitée avec une erreur minimale ; les contrôleurs intégraux affectent la réponse à l'état stationnaire du système et sont parfois aussi appelés contrôleurs de redémarrage[17].

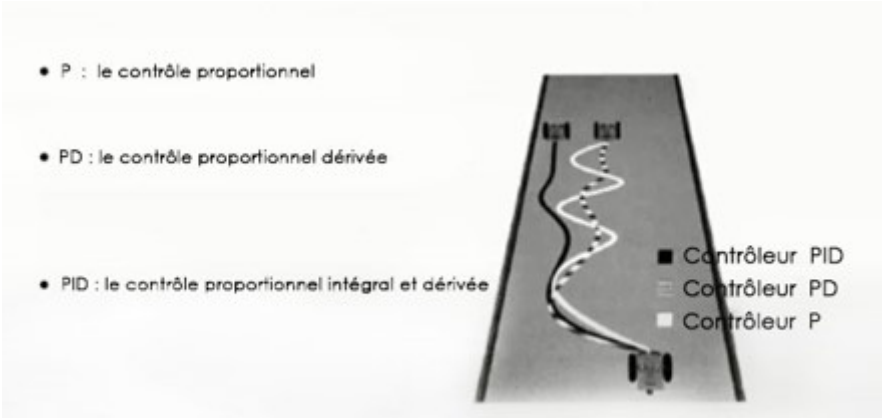


Figure 4.6: les 3 types de contrôleurs de retour d'information (feedback) ; [17]

Le contrôle proportionnel (P)

La réponse du système est proportionnelle à la quantité d'erreur ; la sortie du régulateur est proportionnelle à l'erreur de vitesse ou de vitesse d'entrée ; la sortie est donc égale à K_p fois l'erreur d'entrée :

$$Sortie = K_p * (erreur d'entrée). \tag{4.1}$$

K_p : est appelé la constante de proportionnalité obtenue ou gain.

le contrôleur génère une réponse plus forte ; plus le système s'éloigne de l'état cible, plus le robot se tourne brusquement vers le mur s'il en est vraiment éloigné ; le robot tourne lentement vers le mur s'il s'en éloigne légèrement car le gain proportionnel devient plus important introduit des surcharges et des oscillations le paramètre de la distance par rapport au mur serait utilisé pour déterminer l'angle et la distance et/ou la vitesse du mouvement du robot plus l'erreur est élevée plus l'erreur est faible plus le mouvement est faible, les paramètres qui déterminent l'ampleur de la réponse du système sont appelés gains qui déterminent le gain correct nécessite des essais et des erreurs et l'étalonnage du système le seul exemple de ceci serait d'utiliser la méthode de Ziegler Nichols[17].

Le contrôle proportionnel dérivé (PD)

La commande PD est une combinaison de la somme proportionnelle et dérivée, de sorte que la sortie est K_p fois l'erreur d'entrée plus K_d fois la dérivée de l'erreur[17].

$$Sortie = K_p * (erreur d'entrée) + K_d * \frac{d(erreur d'entrée)}{dt}. \tag{4.2}$$

K_p : est appelé la constante de proportionnalité obtenue ou gain.
K_d : est appelé la constante de dérivée.

Il est largement utilisé dans le contrôle des processus industriels car il s'agit d'une combinaison de la variation de la puissance d'entrée lorsque le système est éloigné du point de consigne et de la correction de la dynamique du système lorsqu'il s'en approche. Il est parfois appelé contrôle des taux parce qu'il est basé sur le taux de variation de l'erreur et, en le suivant, il réduirait la clarté du terme lorsque le taux de variation dépasse un certain seuil.

Le contrôle proportionnel intégral dérivé (PID)

Le contrôle PID est la somme des trois contrôles de base dont nous avons parlé ici : K_p multiplie l'erreur d'entrée plus Ki multiplie l'intégrale de l'erreur plus K_d multiplie la dérivée

de l'erreur, de sorte que la sortie est :

$$Sortie = K_{\mathrm{p}} * (erreur d'entrée) + K_{\mathrm{i}} * \int (erreur d'entrée) + K_{\mathrm{d}} * \frac{d(erreur d'entrée)}{dt}. \quad (4.3)$$

K_{p} : est appelé la constante de dérivée.
K_{i} : est appelé la constante d'intégral.
K_{d} : est appelé la constante de dérivée.

Améliorez à la fois la stabilité et l'état et la réponse transitoire de la paroi du robot selon l'algorithme comme vous le feriez dans le code ; vous choisiriez les gains pour Kp Ki et Kd par essais et erreurs car nous savons qu'un robot ne sera pas un système idéalement modélisé et vous négligeriez le terme dt car il s'agit du retard dans le code ; il suffit donc de sélectionner ce gain pour l'intégrale de l'erreur jusqu'à ce que vous atteigniez un certain seuil puis de multiplier la puissance du moteur ou le nombre de tours par le gain pour l'intégrale[17].

Les caractéristiques du système

Calcule les erreurs pour faire tourner le robot plus souvent et ensuite ajuste l'angle de rotation pour faire tourner le robot à des angles plus petits utilise une gamme de distances pour fixer la cible du robot, détermine ces deux valeurs en fonction des paramètres spécifiques du robot mais n'oubliez pas que vous tirez pour avoir d'abord un gain proportionnel avec une oscillation régulière et ensuite ajuster le gain dérivé et intégral de certaines choses dont vous aurez besoin pour fixer la cible du robot : Instable, non amorti sans compensation de l'erreur et sur-amorti avec l'erreur de compensation. Tous ces états sont des caractéristiques différentes basées sur le gain proportionnel, intégral et dérivé, et sur la reconnaissance de ce qui se passe et la compréhension de ce qui doit être changé pour corriger ces états[17].

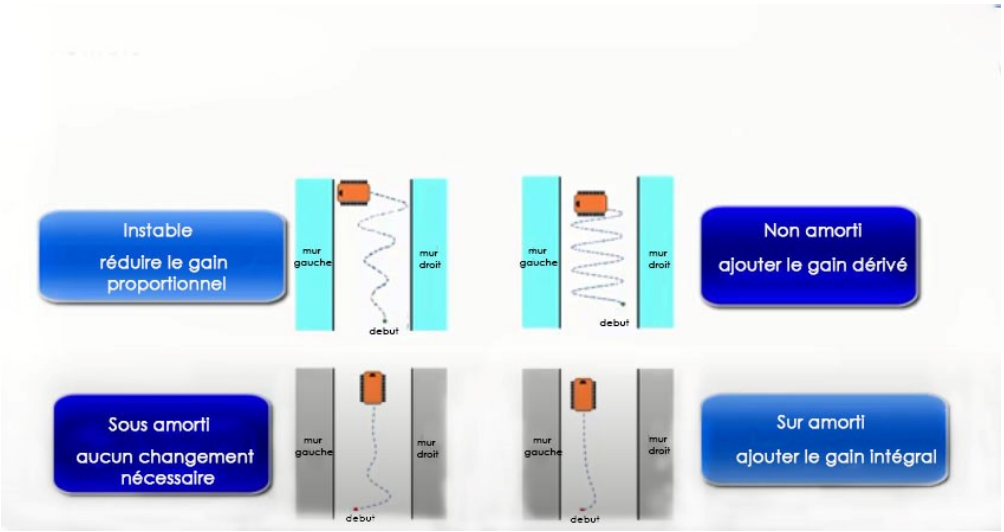


Figure 4.7: Les caractéristiques du système ; [17]

Mur suivant les angles

Il existe deux façons de franchir un coin de mur en suivant l'une d'entre elles : l'une consiste à faire de petits virages en continuant en ligne droite pour faire tourner lentement le robot à gauche ou à droite ; l'autre consiste à inverser et à tourner dès que le robot est détecté, et à continuer à inverser et à tourner jusqu'à ce que le robot ait franchi le coin ; l'inconvénient des deux méthodes est qu'elles prennent du temps et produisent des mouvements brusques ou moins élégants ; une alternative consiste à utiliser un contrôle proportionnel où une commande de rotation qui a été chronométrée pour effectuer une rotation de 90 degrés en fonction de la distance du mur est exécutée ; c'est la façon dont un être humain tournerait probablement un coin ; cela ne fonctionne de façon fiable que lorsque le robot est très prévisible, comme la traction de la batterie dans le modèle de capteur à surface de friction[17].

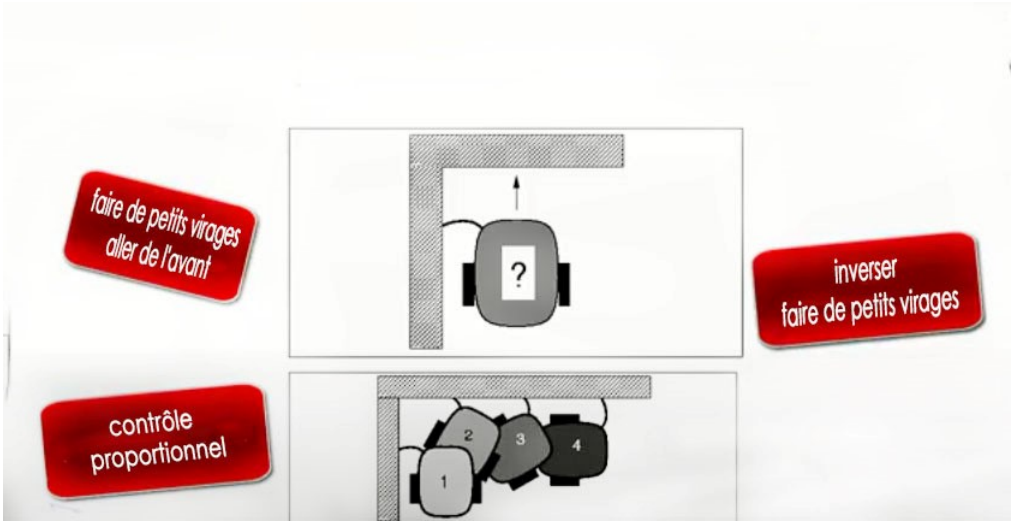


Figure 4.8: schéma d'un mur suivant les coins ; [17]

Déterminer les gains du contrôleur

1. **Empiriquement** : par exemple, par essais et erreurs, le système doit être soumis à des essais approfondis.

2. **Analytiquement** : avec les mathématiques requises au système être -bien compris et caractérisé comme un modèle.

3. **Automatiquement** : en essayant différentes valeurs au moment de l'exécution avec une sorte d'algorithme informatique[17].

Chapitre 5

Localisation et cartographie

5.1 Introduction

La navigation est l'une des compétences les plus difficiles que doit posséder un robot mobile. Pour une navigation réussie, les quatre éléments fondamentaux de la navigation sont nécessaires : la perception, le robot doit interpréter ses capteurs pour extraire des données significatives ; la position, le robot doit déterminer sa position dans l'environnement (figure 5.1) ; la cognition, le robot doit décider comment agir pour atteindre ses objectifs ; et le contrôle des mouvements, le robot doit moduler ses sorties motrices pour atteindre la trajectoire souhaitée. De ces quatre composantes (figure 5.2), c'est la localisation qui a reçu le

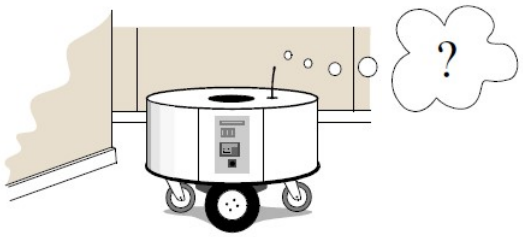


Figure 5.1: Où suis-je ? ; [18]

plus d'attention de la part des chercheurs au cours de la dernière décennie et, par conséquent, des progrès significatifs ont été réalisés sur ce front. Ce chapitre examine les méthodes de localisation qui se sont avérées efficaces ces dernières années[18].

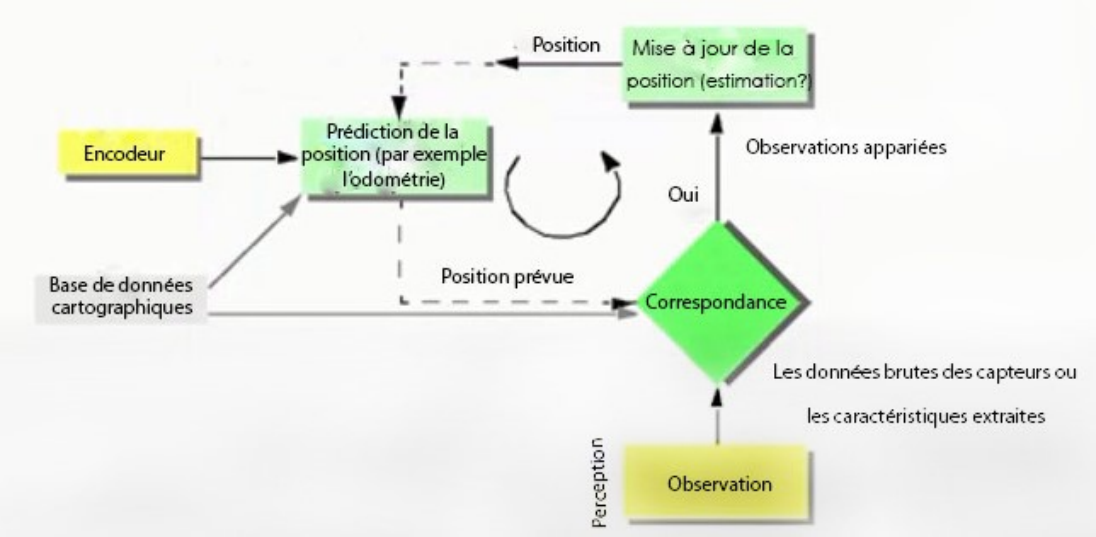


Figure 5.2: Schéma général pour la localisation d'un robot mobile ; [17]

5.2 Localisation et cartographie

Les deux questions principales en matière de navigation sont : où je suis, c'est-à-dire la localisation, et où j'ai été, c'est-à-dire la cartographie, sont étroitement liées car un robot ne peut pas créer une carte exacte s'il ne sait pas où il se trouve[18].

5.3 Les Catégories de localisation

1. **à base d'icônes** : utilisé dans une grille d'occupation de l'espace ou une grille de test qui fusionne les données des capteurs dans un modèle ou une carte du monde et la fusion est effectuée à l'aide d'un algorithme fourni par une théorie de test formelle telle que Bayesian ou Dempster Schaefer ou les architectures hybrides HIMM utilisent la grille d'occupation comme un capteur virtuel dans son adaptation pour une carte métrique des bâtiments .

2. **basé sur les caractéristiques** : qui convient à la construction de cartes topologiques et qui est basé sur le processus de décision de Markov .

3. **l'odométrie.**
4. **balise ou grille** : utilisation de balises ou de points de référence provenant de capteurs externes.

La position basée sur une carte de probabilité est également appelée position de Markov ou position de filtre commun[17].

5.4 La localisation en grille

Une autre stratégie pour estimer la position est le suivi à deux degrés : placer la grille sur le sol avec des cellules clairement identifiables ; le robot détecte le changement d'une cellule à l'autre car il est équipé d'un capteur de lumière ; la grille doit être conçue pour distinguer le changement d'une cellule à l'autre ; elle doit maximiser le contraste entre les cellules adjacentes et les cellules de la grille doivent être grandes lorsque le robot se déplace plus rapidement[17].

Figure 5.3: localisation en grille ; [17]

5.4.1 Exemple illustratif

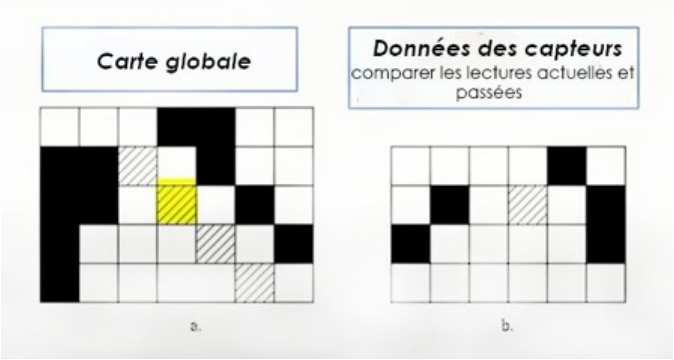


Figure 5.4: Exemple de a.) Une carte globale construite à partir de lectures précédentes et b.) Une nouvelle observation locale qui doit être adaptée à la carte globale. Les éléments ombragés en a. représentent les correspondances possibles avec l'élément ombragé en b ; [17]

5.5 Localisation métrique quantitative

Dans cette méthode, il doit y avoir une carte avec des informations métriques ; le robot explore le monde et utilise des capteurs pour extraire des caractéristiques telles que les murs et les portes des couloirs ; le robot fait ensuite correspondre les caractéristiques trouvées ; l'extrait avec un emplacement dans le monde ; le robot utilise ensuite la théorie de l'estimation de la position pour déterminer la probabilité qu'il se trouve au bon endroit[17].

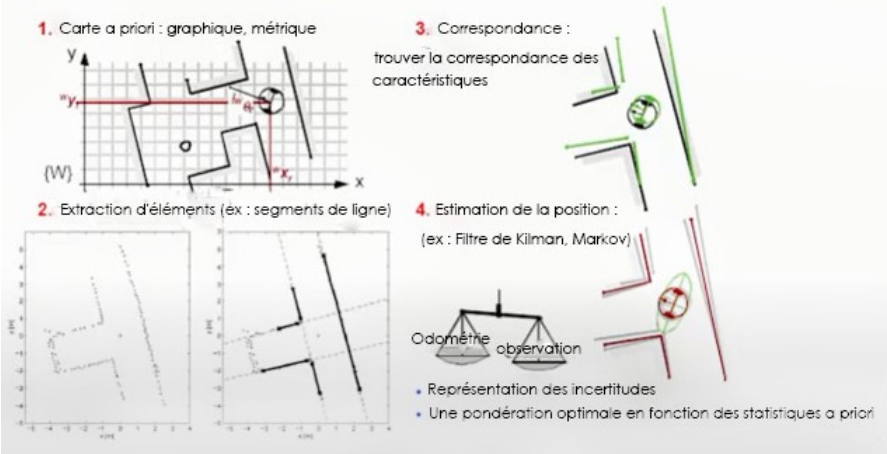


Figure 5.5: Localisation métrique quantitative ; [17]

5.6 Localisation par filtre de Kalman

représente l'état de croyance du robot à l'aide d'une fonction de densité de probabilité gaussienne unique et bien définie, utilise une représentation de la densité de probabilité de la position du robot et recherche la correspondance de position, le suivi par filtre de Kalman utilise les données brutes du capteur pour extraire des caractéristiques telles que des lignes et une estimation de la position initiale, afin qu'il n'y ait pas d'incertitude associée à chaque orientation de la ligne et que la position du robot suive le robot depuis une position connue, afin qu'elle soit précise et efficace, peut être utilisée dans les représentations du monde continu si l'incertitude du robot devient trop grande et que le modal n'est pas connu peut ne pas saisir la multitude de positions possibles du robot et peut être irrémédiablement perdu, contrairement à la position de Markov, ne considère pas indépendamment chaque position possible du robot. Résultats communs des axiomes de Markov : en supposant que l'incertitude de la position du robot est gaussienne. l'estimation par filtre de Kalman de la position d'un nouveau robot est réalisée en fusionnant la prédiction de la position du robot en magenta avec les informations obtenues à partir des mesures en vert ; l'estimation actualisée de la position du robot en rouge est obtenue ; cette estimation de la position finale correspond à la

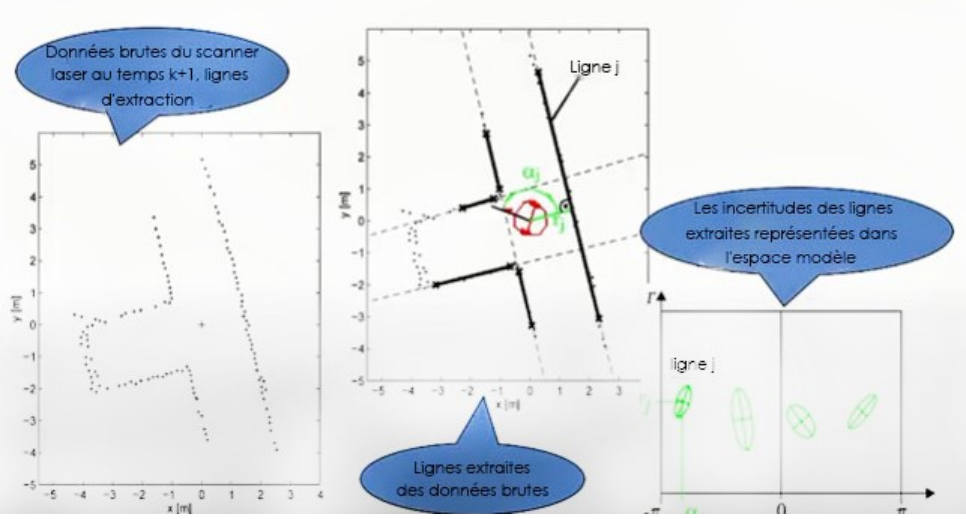


Figure 5.6: Localisation par filtre de Kalman ; [17]

somme pondérée des estimations de position de chaque paire de caractéristiques observées et prédites ; l'estimation de la position du robot est basée sur l'odométrie et les positions d'observation[17].

Figure 5.7: l'estimation par filtre de Kalman ; [17]

5.7 Localisation topologique

le robot reçoit une carte prioritaire indiquant les emplacements topologiques ou les points de repère uniques ; ceux-ci peuvent être représentés sous forme de nœuds et de bords avec des caractéristiques dans le monde ; le robot voyage autour du monde et reconnaît les emplacements distinctifs ; l'odométrie du robot et les emplacements uniques sont utilisés pour enregistrer ou localiser le robot[17].

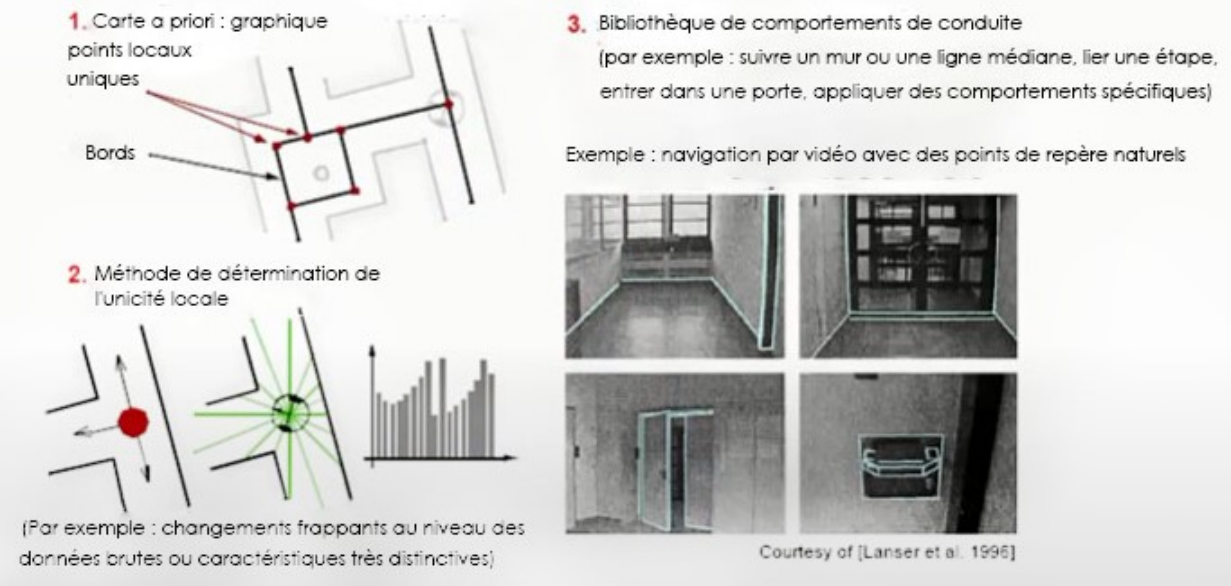


Figure 5.8: Localisation topologique ; [17]

5.8 Localisation de Markov sur une carte topologique

La position de Markov utilise une distribution de probabilité explicitement définie pour toutes les positions du robot, la position de Markov est le robot qui croit qu'un état normalement représenté par une assignation de probabilité séparée pour chaque position possible sur la carte un cas spécial d'estimation probabiliste d'état est appliqué à la position d'un robot mobile, permet la localisation à partir de n'importe quelle position inconnue et récupère d'une situation ambiguë car le robot peut suivre plusieurs solutions possibles complètes et disparates nécessite une représentation discrète de l'espace comme une grille géométrique ou un graphique topologique ; dans cette classe, il a besoin de mémoire et de puissance de calcul et peut limiter la précision et la taille de la carte ; il est identique en abstraction et en information à la carte de l'environnement et la décision implique l'attribution de nœuds et la connectivité entre eux ; il n'y a pas de limites marquées par des portes de couloir et de corridor et il sait qu'il n'y a pas d'information géométrique dans les nœuds[17].

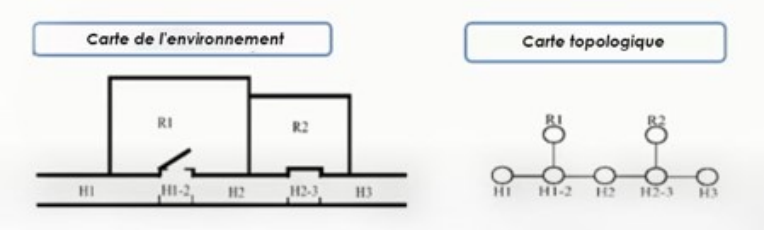


Figure 5.9: Un environnement de bureau géométrique (à gauche) et son analogue topologique (à droite) ; [17]

5.9 Les défis de la localisation

Parfois, la connaissance de la localisation absolue n'est pas suffisante, la localisation peut également être nécessaire à une échelle relative à l'homme, la cognition peut nécessiter plus que la localisation ; il peut être nécessaire de construire une carte environnementale pour planifier un itinéraire vers une cible et le processus d'estimation est indirect ; les mesures du bruit ne sont pas toujours disponibles et le cadre de référence est important même s'il est local ou relatif à l'endroit où je me trouve par rapport au cadre global, et s'il est global par rapport à l'absolu où je me trouve par rapport au cadre global, la localisation peut être spécifiée de deux façons : à la fois géométriques, comme les distances et les angles, et topologiques, comme les connexions entre les points de référence ; les défis sont plus nombreux, car les capteurs de perception et les effecteurs de contrôle de mouvement jouent un rôle clé dans la localisation ; comme nous savons que nous avons un capteur de bruit, le bruit d'aliasing de l'effecteur, et que la position automatique est basée sur une estimation[17].

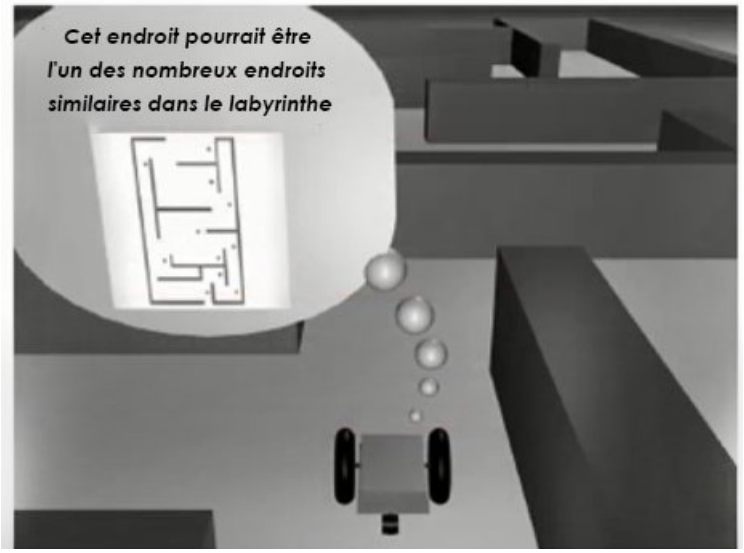


Figure 5.10: les enjeux de la localisation ; [17]

5.10 La cartographie

La cartographie et la localisation vont de pair car il est assez difficile de faire une carte précise si vous ne pouvez pas vous localiser et il est assez difficile de savoir où vous êtes dans le monde si vous n'avez pas de carte. La mesure de la qualité est basée sur la précision topologique et la précision métrique ; cela peut se faire par la construction de cartes ou de graphiques routiers qui identifient un ensemble de routes dans l'espace libre, comme par

exemple l'endroit où placer les nœuds, et si la topologie est basée sur des emplacements séparés ou sur une base métrique, les caractéristiques disparaissent ou deviennent visibles ; cela peut également se faire par la décomposition des cellules, où une distinction est faite entre les cellules libres et les cellules occupées, comme par exemple l'endroit où placer les limites des cellules, ou selon la topologie et la métrologie, où les caractéristiques disparaissent ou deviennent visibles à des intervalles discrets[17].

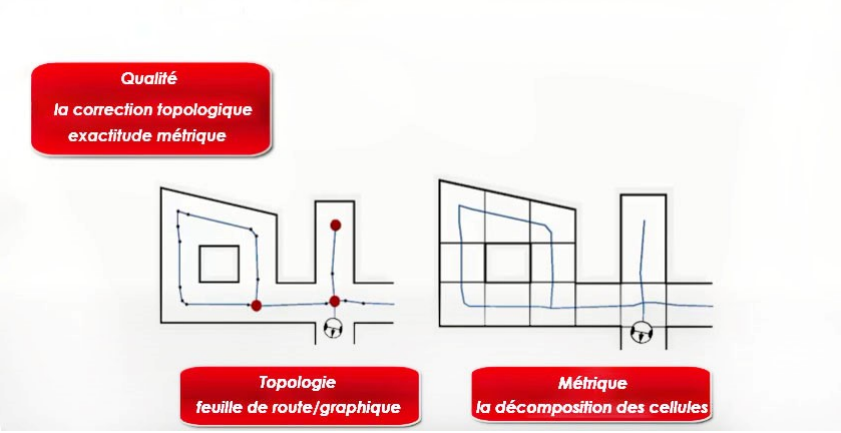


Figure 5.11: shéma sur La cartographie ; [17]

5.10.1 Construction du graphique d'exploration

est de reconnaître les lieux déjà visités dans le monde, de rechercher des ouvertures inexplorées dans le monde et, lorsqu'il a fini de fournir une topologie correcte du monde, la construction automatique de la carte implique que le robot apprenne l'environnement sur la base d'un algorithme codé et cela doit être fait efficacement malgré un monde dynamique et imprévisible ou changeant et une apparence différente due à des perceptions ou des perspectives différentes de l'espace, nécessite l'intégration de nouvelles informations provenant du modèle global existant doit contenir des informations permettant d'estimer la position du robot doit fournir des informations permettant d'effectuer des tâches de planification de trajectoire et de navigation, bien que la plupart des environnements soient un mélange de caractéristiques prévisibles et imprévisibles ; Il devrait donc y avoir une sorte d'approche de contrôle hybride[17].

5.10.2 Représentation continue

une carte à valeurs continues est une méthode précise de décomposition de l'environnement, les cartes continues sont uniquement en représentation 2d car une dimension supplémentaire

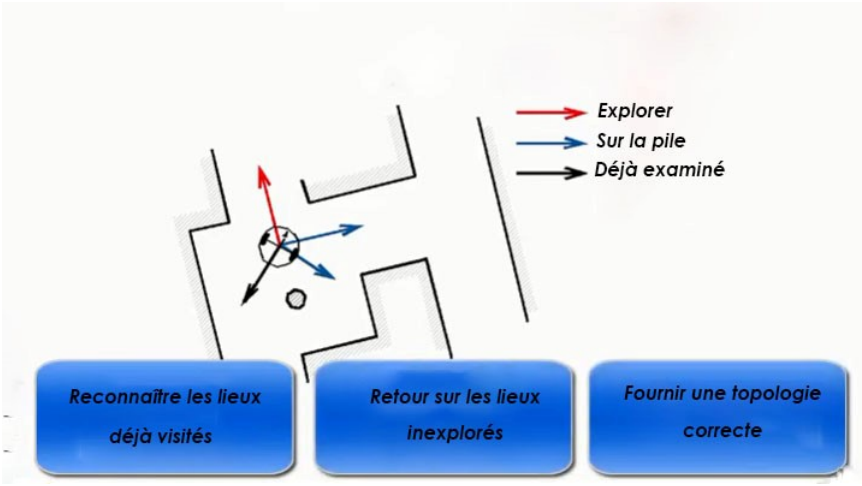


Figure 5.12: Construction du graphique d'exploration source ; [17]

peut faire exploser les calculs, la précision de la représentation continue est combinée avec l'hypothèse de la compacité d'un monde fermé, la représentation spécifiera tous les objets de l'environnement sur la carte, une carte à faible mémoire est une représentation 2d où les polygones représentent tous les obstacles, de nombreuses simulations sont effectuées exclusivement dans la mémoire des ordinateurs et les polygones ne sont pas utilisés pour décrire un environnement réel, alors que les environnements réels doivent être capturés, il existe des tendances à la sélectivité et à l'abstraction, les humains ne capturent que les objets qui peuvent être détectés par les capteurs du robot et cela représente un sous-ensemble de toutes les caractéristiques des objets dans le monde, une méthode de simplification consiste à approcher l'environnement réel[17].

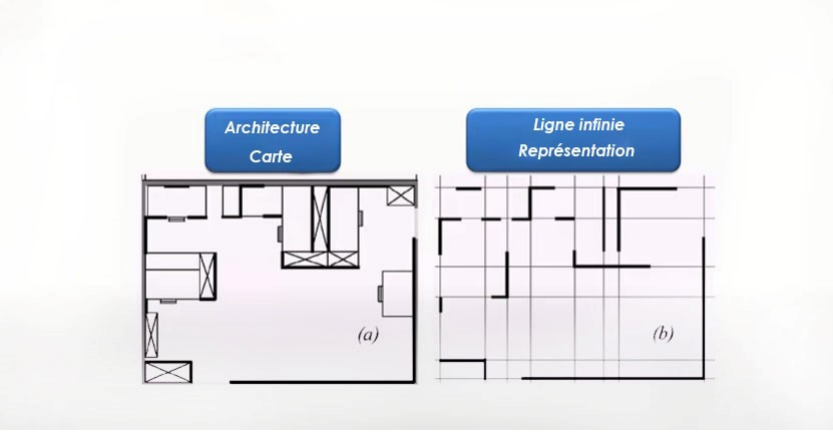


Figure 5.13: Exemple d'une représentation linéaire continue de l'EPFL. (a) Carte réelle. (b) Représentation avec un ensemble de lignes infinies ; [17]

5.10.3 Grille d'occupation

pour créer une grille d'occupation, un compteur est utilisé pour déterminer combien de fois une cellule est touchée par un capteur de distance, lorsque le compteur augmente, la cellule est considérée comme un obstacle, l'obscurité de la cellule est proportionnelle à la valeur du compteur, l'histogramme sur la carte de mouvement est un moyen de créer une grille d'occupation, il s'agit de compter le nombre de fois qu'une limite ou un obstacle est détecté et, lorsque le compte dépasse un certain seuil, la confiance dans la carte est sombre et la probabilité d'être présent augmente[17].

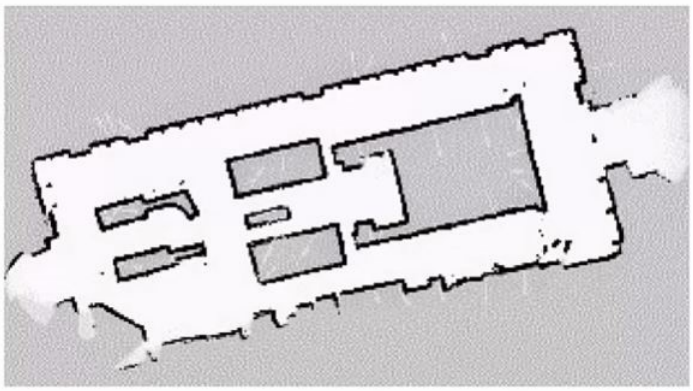


Figure 5.14: Exemple de représentation d'une carte quadrillée d'occupation ; [17]

5.10.4 Construction autonome de cartes ou SLAM

(Simultanious Localisation And Mapping) Un robot qui a réussi à se localiser possède les bons capteurs pour détecter l'environnement et le robot doit construire sa propre carte, à partir d'un point de départ arbitraire. Un robot mobile doit être capable d'explorer l'environnement de manière autonome avec ses capteurs et d'en acquérir la connaissance. L'interprétation de la scène, la construction d'une carte appropriée et la localisation par rapport à cette carte, lorsque le robot n'a pas de carte et ne sait pas où il se trouve et choisit de construire une carte à la volée pour le localiser est appelée localisation et cartographie simultanée, Slam est également appelé localisation et cartographie simultanée ou localisation et cartographie CML simultanée est l'un des problèmes les plus difficiles spécifiques aux systèmes de robots mobiles, Le slam est l'une des tâches les plus difficiles en robotique car il est basé sur l'interaction entre les mises à jour de position qui servent à localiser dans les actions de cartographie, le slam est difficile car il implique que le robot effectue deux processus parallèles et liés en cours, il y a confusion entre différentes positions qui semblent similaires et donc ambiguës ; c'est le problème de l'association de données, qui consiste à associer de façon unique des données sensorielles à la vérité absolue du terrain[17].

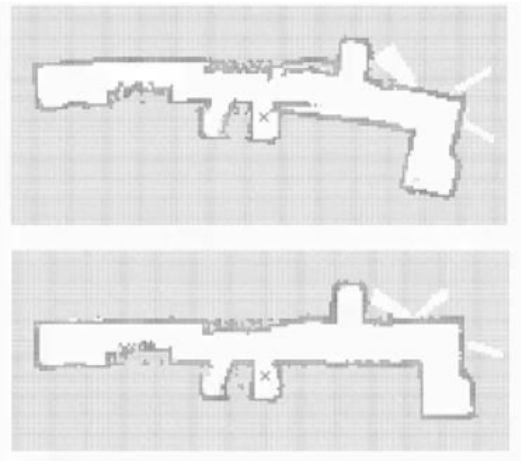


Figure 5.15: Cartographie d'une salle de 70 pieds sans localisation continue (en haut) et avec (en bas) ; [20]

5.10.5 Couverture et exploration

L'exploration tente de répondre à la question «Où suis-je allé» : Comment puis-je couvrir efficacement l'environnement inconnu ? Une méthode consiste à effectuer une recherche aléatoire ou à utiliser un champ potentiel et à couvrir la zone avec une grille après avoir scanné la zone inconnue pendant une longue période. Il existe deux méthodes de balayage de base : elles sont basées sur les frontières et le graphique généralisé de Voronoï ; lorsqu'un robot pénètre dans une nouvelle zone, il y a une frontière entre chaque zone qui a été détectée et est ouverte et la zone qui n'a pas été ouverte car les frontières sont des lignes qui forment une limite qui doit être balayée ; ce graphique montre ici un exemple de balayage basé sur les frontières[17].

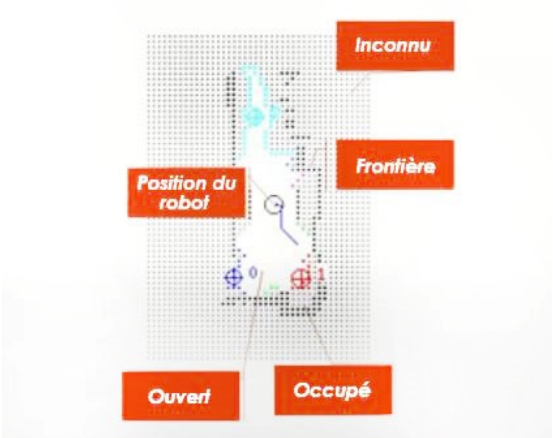


Figure 5.16: Exemple d'un robot explorant une zone en utilisant une méthode basée sur les frontières ; [17]

l'autre méthode est un diagramme de Voronoï généralisé ; en fait, lorsque nous parlons de planification d'itinéraire, le robot essaie de construire un diagramme de Voronoï généralisé réduit lorsqu'il se déplace dans l'espace pour explorer ; le robot essaie de maintenir une trajectoire qui met sa distance égale à celle de tous les objets dans ses directions, et cette trajectoire est appelée diagramme de Voronoï ; Le robot peut alors utiliser n'importe quel comportement pour suivre ce cours pour explorer la carte. Notez que l'ombrage indique une zone du graphique qui n'a peut-être pas encore été entièrement explorée, de sorte que la couleur de fond changera au fur et à mesure que le robot explorera la zone ou créera cette partie de la carte[17].

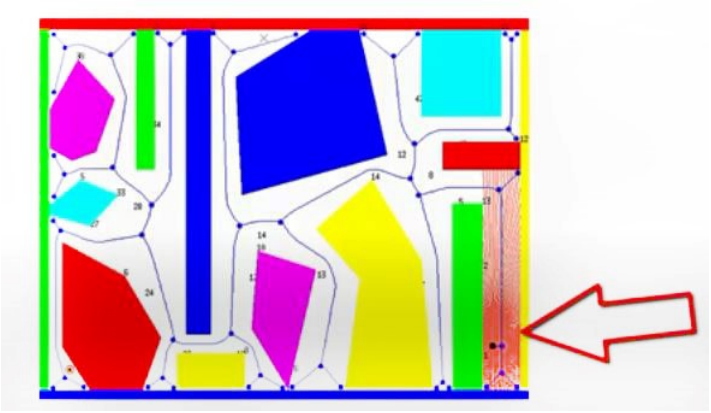


Figure 5.17: Exemple d'un robot explorant une zone intérieure à l'aide d'une méthode GVG(Generalized Voronoi Graph) graphique de voronoï généralisé ; [17]

Chapitre 6

Planification et navigation

6.1 Introduction

La planification est un aspect évident de la navigation qui répond à la question : quel est le meilleur moyen d'y parvenir ? En effet, pour les applications de robotique mobile, un robot doit être capable d'atteindre la position cible en évitant les obstacles dispersés dans l'environnement et en réduisant la longueur de la trajectoire. La planification de l'itinéraire des robots mobiles doit tenir compte de différents aspects, en raison des divers objectifs et fonctions du robot virtuel lui-même, comme le montre la figure 6.1. La plupart des approches proposées se concentrent sur la recherche du chemin le plus court entre la position initiale et la position finale[21].

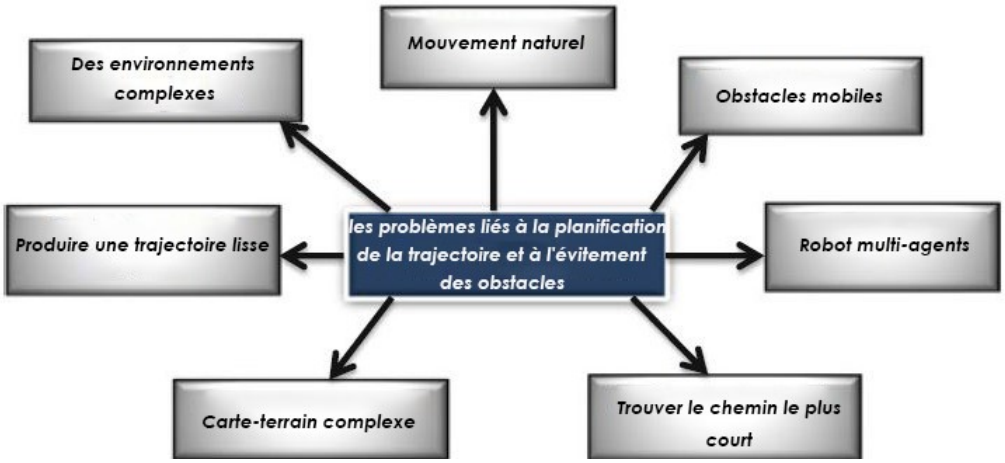


Figure 6.1: Les différents aspects de la planification des parcours ; [21]

6.2 Les catégories de planification des trajectoires

Il peut être divisé en trois catégories selon la connaissance qu'a le robot de l'environnement, la nature de l'environnement et l'approche utilisée pour résoudre le problème, comme le montre la figure 6.2[21].

Figure 6.2: Les catégories de planification des parcours ; [21]

6.3 Représentations spatiales couramment utilisées dans la planification des trajectoires

En général, les modèles de planification d'itinéraires basés sur des cartes se divisent en deux catégories selon la façon dont on regarde le monde[21].

6.3.1 Planification qualitative (de l'itinéraire) du parcours

Dans cette catégorie, il n'y a pas de carte a priori représentant le monde, mais elle est spécifiée par des itinéraires allant de la position de départ à la position de destination. Le monde est représenté comme un lien entre les points de repère, et une séquence de points de repère liés représentera l'itinéraire[21].

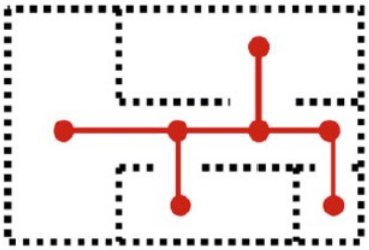


Figure 6.3: Planification qualitative du parcours ; [21]

6.3.2 Planification du cheminement métrique (tracé)

La planification d'itinéraire métrique est plus intéressante que la planification d'itinéraire qualitative car vous pouvez représenter l'environnement dans une structure claire que le robot peut utiliser pour planifier et trouver son itinéraire. Il permet également au robot de raisonner dans l'espace. En fait, les cartes métriques décomposent l'environnement en un ensemble de points différents, appelés "way points", qui sont des positions fixes identifiées par leurs coordonnées (x,y). La planification de la trajectoire consistera donc à trouver la séquence de points de passage connectés qui mènent au lieu de destination, en minimisant le coût du voyage[21].

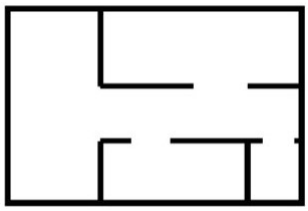


Figure 6.4: Planification de la trajectoire métrique ; [21]

6.4 Les approches utilisées pour résoudre la planification des parcours

Le but de ces algorithmes est de trouver le chemin le plus court entre deux emplacements A et B dans un environnement spécifique. Il existe plusieurs algorithmes basés sur diverses méthodes pour trouver une solution à ce problème. La complexité de l'algorithme dépend de la technologie de base et d'autres paramètres externes, tels que la précision de la carte et le nombre d'obstacles[21].

6.4.1 Les approches classiques

Planification du cheminement de la carte routière (Road-Map Path Planning)

1. **Graphique de visibilité (Visibility graph)** :

Le graphique de visibilité pour un espace de configuration du polygone C consiste en des arêtes joignant toutes les paires de sommets visibles (y compris les positions de début et de fin en tant que sommets). Les lignes droites libres (chemins) reliant ces sommets sont évidemment les distances les plus courtes entre eux. Par conséquent, la tâche du planificateur d'itinéraire est de trouver le chemin le plus court entre les positions de départ et d'arrivée le long des trajets définis par le graphique de visibilité (figure 6.5)[22].

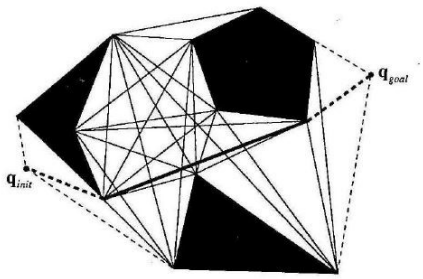


Figure 6.5: Graphique de visibilité : Les nœuds du graphe sont les points initiaux et finaux et les sommets des obstacles de l'espace de configuration (polygones). Tous les nœuds qui sont visibles les uns des autres sont reliés par des segments de ligne droite, définissant la carte routière. Cela signifie qu'il y a également des bords le long des côtés de chaque polygone ; [22]

2. **Diagramme de Voronoï (Voronoi diagram)** :

Contrairement à l'approche du graphique de visibilité, le diagramme de Voronoï est une méthode de carte routière complète qui tend à maximiser la distance entre le robot et les obstacles de la carte. Pour chaque point de l'espace libre, calculez sa distance par rapport à l'obstacle le plus proche. Tracez cette distance dans la figure 6.6 comme une hauteur sortant de la page. La hauteur augmente à mesure que vous vous éloignez d'un obstacle. En des points équidistants de deux ou plusieurs obstacles, le tracé de la distance présente des arêtes vives. Lorsque les obstacles de l'espace de configuration sont des polygones, le diagramme de Voronoï se compose de segments droits et paraboliques [22].

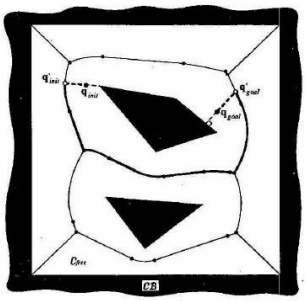


Figure 6.6: Le diagramme de Voronoï est constitué des lignes construites à partir de tous les points équidistants de deux ou plusieurs obstacles. Les configurations initiales de qinit et de qgoal de but sont cartographiées dans le diagramme de Voronoï en qinit' et qgoal' , chacun en traçant la ligne le long de laquelle sa distance à la limite des obstacles augmente le plus rapidement. La direction du mouvement sur le diagramme de Voronoï est également sélectionnée de manière à ce que la distance aux limites augmente le plus rapidement. Les points du diagramme de Voronoï représentent les transitions entre les segments de ligne (distance minimale entre deux lignes) et les segments paraboliques (distance minimale entre une ligne et un point) ; [22]

Planification du cheminement de la décomposition cellulaire (Cell Decomposition Path Planning)

1. Décomposition cellulaire exacte (Exact Cell Decomposition) :

La figure 6.7 illustre la décomposition cellulaire exacte, où la limite de la cellule est basée sur la criticité géométrique. Les cellules résultantes sont soit complètement libres, soit complètement occupées, de sorte que la planification de l'itinéraire à travers le réseau est complète, tout comme les méthodes basées sur la feuille de route ci-dessus. L'abstraction de base derrière une telle décomposition est que la position particulière du robot dans chaque cellule libre n'a pas d'importance ; ce qui compte, c'est plutôt la capacité du robot à se déplacer de chaque cellule libre vers les cellules libres adjacentes[22].
.

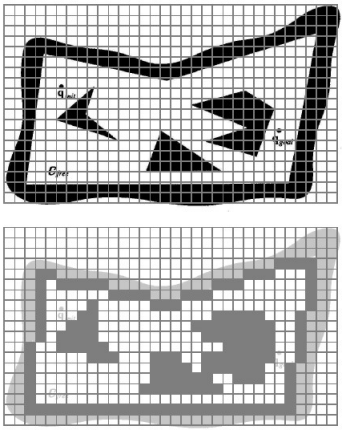


Figure 6.7: Décomposition fixe du même espace. (le passage étroit disparaît) ; [22]

2. **Décomposition cellulaire approximative (Approximate Cell Decomposition)** :

La décomposition approximative des cellules est l'une des techniques les plus populaires pour la planification de la trajectoire des robots mobiles. Cela est dû en partie à la popularité des représentations de l'environnement basées sur des grilles. Ces représentations en grille sont elles-mêmes des décompositions à taille fixe et sont donc identiques à une décomposition cellulaire approximative de l'environnement. La forme la plus populaire de cette décomposition, illustrée dans la figure 6.8, est la décomposition cellulaire à taille fixe[22].

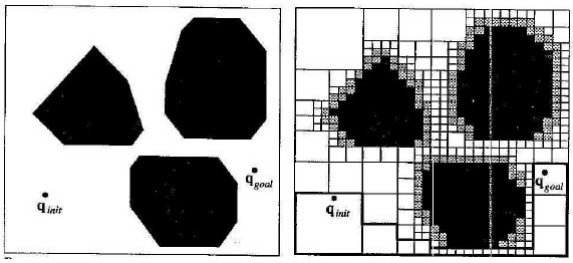


Figure 6.8: Décomposition cellulaire variable approximative : a) L'espace libre est délimité extérieurement par un rectangle et intérieurement par trois polygones. b) Le rectangle est décomposé en quatre rectangles identiques. Si l'intérieur d'un rectangle se trouve entièrement dans l'espace libre ou dans l'obstacle de l'espace de configuration, il n'est pas décomposé davantage. Sinon, il est décomposé de manière récursive en quatre rectangles jusqu'à ce qu'une résolution prédéfinie soit atteinte. Les cellules blanches se trouvent à l'extérieur des obstacles, le noir à l'intérieur et le gris font partie des deux régions ; [22]

Planification de la trajectoire potentielle sur le terrain (Potential Field Path Planning)

1. **Potentiel d'attractivité (Attractive Potential)** :

Un potentiel attractif peut par exemple être défini comme une fonction parabolique. Ce potentiel attractif est différenciable, conduisant à la force d'attraction F_{att} :

$$F_{\text{att}}(q) = -\nabla U_{\text{att}}(q) = -k_{\text{att}}. = {}_{\text{goal}}\nabla\rho_{\text{goal}}(q) = -k_{\text{att}}.(q - q_{\text{goal}}) \tag{6.1}$$

Qui converge linéairement vers 0 lorsque le robot atteint le but[22].
Avec :
- q : est le variable d'état .
- $U_{\text{att}}(q)$: Potentiel d'attractivité.
- k_{att} : est un facteur d'échelle positif.
- $\nabla U_{\text{att}}(q)$: désigne le vecteur de gradient de U à la position.
- $\rho_{\text{goal}}(q)$: indique la distance euclidienne $||q - q_{\text{goal}}||$

2. **Potentiel de répulsion (Repulsive Potential)** :

L'idée derrière le potentiel répulsif est de générer une force qui s'éloigne de tous les obstacles connus. Ce potentiel répulsif doit être très fort lorsque le robot est proche de l'objet, mais ne doit pas affecter son mouvement lorsque le robot s'éloigne de l'objet. La force répulsive F_{rep} [22] :

$$F_{\text{rep}}(q) = -\nabla U_{\text{rep}}(q) \tag{6.2}$$

$$= \begin{cases} k_{\text{rep}}(\frac{1}{\rho(q)} - \frac{1}{\rho_0})(\frac{1}{\rho^2(q)})(\frac{q-q_{\text{obstacle}}}{\rho(q)}), & \text{Si } \rho(q) \leq \rho_0. \\ 0, & \text{Si } \rho(q) \geq \rho_0. \end{cases} \tag{6.3}$$

Avec :
- $U_{\text{rep}}(q)$: Potentiel de répulsion .
- $\rho(q)$: est la distance minimale par rapport à l'objet .
- $\nabla U_{\text{rep}}(q)$: désigne le vecteur de gradient de U à la position.
- ρ_0 : la distance d'influence de l'objet.

3. **La méthode du champ potentiel étendu (The Extended Potential Field Method)** :

Khatib et Chatila [22] ont proposé une méthode de domaine étendu. Comme toutes les méthodes de champ potentiel, cette méthode utilise des forces attractives et répulsives provenant de champs potentiels artificiels. Cependant, deux sont ajoutés au champ potentiel de base : le champ rotatif potentiel et le champ de mission potentiel[22].

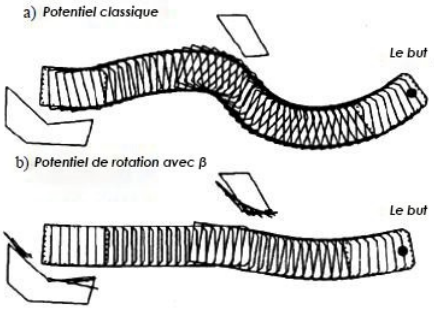


Figure 6.9: Comparaison entre un champ potentiel classique et un champ potentiel étendu ; [22]

6.4.2 Les Approches de recherche de graphiques

L'algorithme de Dijkstra

L'algorithme de Dijkstra est utilisé dans la planification de la trajectoire du robot. Le chemin le plus court est sélectionné dans le processus de barrière. Les résultats de la simulation prouvent la validité du modèle ; il peut résoudre efficacement la planification du trajet du robot dans le labyrinthe[21].

L'algorithme (A*)

L'algorithme A* est un algorithme basé sur une fonction heuristique pour une planification de trajectoire correcte. Il calcule la valeur de la fonction heuristique à chaque nœud de la zone de travail et implique la vérification d'un trop grand nombre de nœuds adjacents pour trouver la solution optimale avec une probabilité de collision nulle[23].

6.4.3 Les Approches heuristiques

L'algorithme ACO

Dans la nature, certaines espèces de fourmis sont capables de choisir le chemin le plus court entre leur nid et la source de nourriture. Cela se fait par l'utilisation de dépôts de phéromones. En fait, les fourmis déposent une traînée chimique, appelée phéromone, sur le sol au cours de leur déplacement ; cette traînée attire d'autres fourmis, qui ont tendance à suivre le chemin ayant la plus forte concentration de phéromones. Après un certain temps, la concentration de phéromones augmente le plus rapidement possible et s'évapore par des voies détournées ; ce comportement est appelé stigmergie. Si un obstacle interrompt leur chemin, les fourmis choisissent d'abord leur chemin de manière probabiliste, puis après un certain temps, la concentration de phéromones devient plus élevée sur le chemin le plus court et toutes les fourmis choisissent ce chemin. Ce processus est illustré dans la figure 6.10[21].

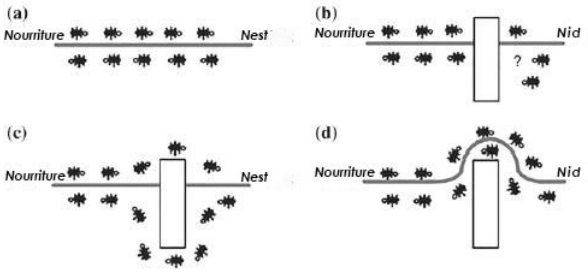


Figure 6.10: Fourmis dans une piste de phéromones entre le nid et la nourriture ; b) un obstacle interrompt la piste ; c) les fourmis trouvent deux chemins et contournent les obstacles ; d) une nouvelle piste de phéromones se forme le long du chemin le plus court ; [21]

Algorithmes génétiques

L'algorithme génétique (GA) est un métaheuriste créé par John Holland en 1975.Il simule le processus d'évolution génétique et la sélection naturelle pour résoudre des problèmes d'optimisation et de recherche. GA est un sous-ensemble d'une branche de calcul beaucoup plus importante connue sous le nom de calcul évolutionniste[21].

Recherche Tabu

S'est révélée être une approche métaheuristique très efficace pour résoudre les problèmes de combinaison. Cependant, peu de recherches basées sur des tabous ont été proposées dans la littérature pour aborder le problème de la planification routière. Par exemple, la recherche a été la première, comme indiqué dans la littérature, à présenter une solution au problème de la planification locale des itinéraires en utilisant l'approche ST, qui s'est avérée efficace. L'idée de l'algorithme est de trouver, dans chaque itération, un ensemble de mouvements tabous pour limiter les positions du robot et guider son mouvement vers la position cible[21].

Autre approches

1. **Explorer rapidement les arbres au hasard RRT (Rapidly Exploring Random Trees)** :

   Les RRT élaborent généralement un graphique en ligne au cours du processus de recherche et n'ont donc besoin que d'une carte des obstacles a priori, mais pas de la décomposition du graphique. L'algorithme commence par un arbre initial (qui peut être vide) et ajoute ensuite des nœuds, reliés par des frontières[18].

Chapitre 7

Évitement des obstacles

7.1 Introduction

La prévention des obstacles locaux consiste à modifier la trajectoire du robot en fonction des informations fournies par ses capteurs lors de son déplacement. Le mouvement du robot qui en résulte est fonction des lectures actuelles ou récentes des capteurs et de leur position par rapport à la position cible. Les algorithmes de prévention des obstacles présentés ci-dessous dépendent à des degrés divers de l'existence d'une carte du monde et de la connaissance précise qu'a le robot de sa position par rapport à cette carte. Malgré leurs différences, tous les algorithmes suivants peuvent être qualifiés d'algorithmes de prévention des obstacles car les lectures des capteurs locaux du robot jouent un rôle important dans la trajectoire future du robot. Tout d'abord, nous présentons les systèmes les plus simples pour éviter les obstacles qui sont utilisés avec succès dans la robotique mobile. L'algorithme Bug représente une telle technique car seules les dernières valeurs des capteurs du robot sont utilisées et le robot n'a besoin que d'informations approximatives sur la direction de la cible en plus des valeurs actuelles des capteurs. Des algorithmes plus sophistiqués sont présentés ci-dessous, en tenant compte de l'histoire récente des capteurs, de la cinématique des robots et même de la dynamique[18].

7.2 Les méthodes d'évitement des obstacles

7.2.1 L'algorithme de bug (Bug Algorithm)

L'algorithme de bug 1(Bug 1 Algorithm)

Avec Bug1, le robot fait d'abord un tour complet autour de l'objet, puis part du point le plus proche de la cible (figure 7.1). Cette approche est évidemment très inefficace, mais elle garantit que le robot atteint tout objectif réalisable[18].

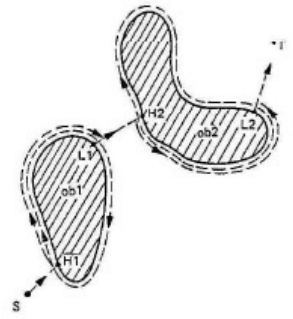


Figure 7.1: Algorithme de Bug1 avec H1, H2, points d'impact et L1, L2, points de sortie ; [18]

L'algorithme de bug 2 (Bug 2 Algorithm)

Avec Bug2, le robot commence à suivre le contour de l'objet, mais part immédiatement lorsqu'il est capable de se déplacer directement vers la cible. En général, cet algorithme amélioré de Bug aura une trajectoire totale du robot beaucoup plus courte, comme le montre la trajectoire (figure 7.2 a). Cependant, il est toujours possible de créer des situations dans lesquelles le Bug2 est arbitrairement inefficace (c'est-à-dire non optimal)[18].

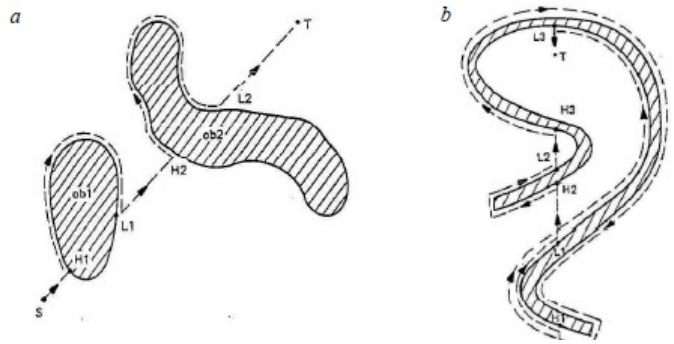


Figure 7.2: Algorithme de bug 2 avec H1, H2, points d'impact et L1, L2, points de sortie ; [18]

7.2.2 Histogramme du champ vectoriel VFH (Vector Field Histogram)

L'une des principales critiques des algorithmes de type Bug est que le comportement du robot à un moment donné est généralement fonction des dernières lectures des capteurs. Cela peut conduire à des problèmes évitables dans les cas où les lectures instantanées des capteurs du robot ne fournissent pas suffisamment d'informations pour éviter efficacement les obstacles. Les techniques VFH permettent de surmonter cette limitation en créant une carte de l'environnement local autour du robot. Cette carte locale est une petite grille d'occupation, peuplée uniquement de relevés relativement récents de la portée des capteurs. Pour éviter les obstacles, le VFH génère un histogramme polaire comme le montre la figure 7.3.Dans l'amélioration du VFH+, une des étapes de réduction tient compte d'un modèle simplifié des trajectoires possibles du robot mobile en fonction de ses limites cinématiques (par exemple le rayon de braquage d'un véhicule Ackerman). Le robot est modélisé pour se déplacer en arcs ou en lignes droites[18].

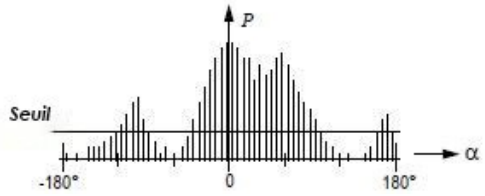


Figure 7.3: Histogramme polaire ; [18]

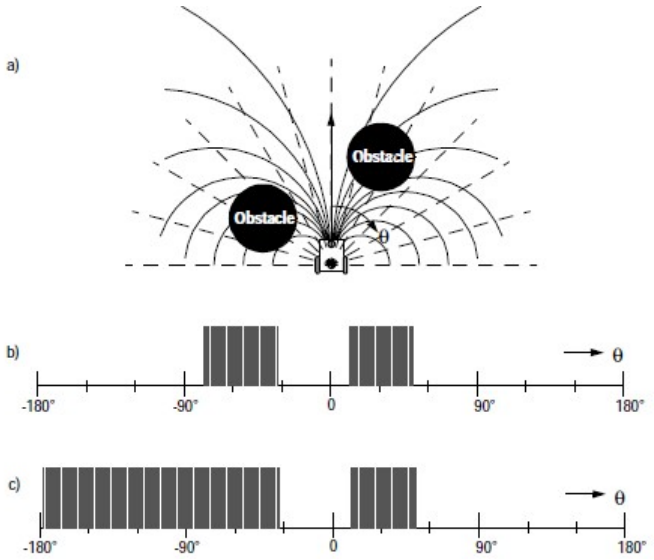


Figure 7.4: Exemple de directions bloquées et d'histogrammes polaires résultants. (a) Robot et obstacles bloquants. (b) Histogramme polaire. (c) Histogramme polaire masqué ; [18]

7.2.3 La technique du ruban à bulles (The Bubble Band Technique)

Une bulle est définie comme l'espace libre maximum autour d'un robot qui peut être garanti pendant l'exécution de sa trajectoire. La bulle est générée en utilisant un modèle simplifié du robot ainsi que les informations de portée disponibles sur la carte du robot. Même avec un modèle simplifié de la géométrie du robot, il est possible de prendre en compte la forme réelle du robot lors du calcul de la taille de la bulle Figure 7.5[18].

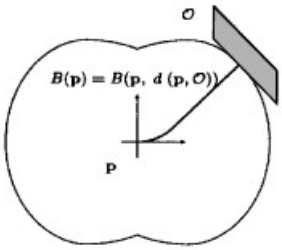


Figure 7.5: Forme des bulles autour du véhicule ; [18]

7.2.4 Techniques de vitesse de courbure (Curvature Velocity Techniques)

La méthode CVM permet de prendre en compte les contraintes cinématiques réelles et même certaines contraintes dynamiques du robot lors de l'évitement d'obstacles, ce qui est une

amélioration par rapport aux technologies plus primitives. CVM ajoute d'abord les contraintes physiques du robot et de l'environnement à l'espace de vitesse. L'obstacle part de l'objet dans la grille cartésienne, puis calcule la distance entre la position du robot et l'obstacle en fonction de la trajectoire du robot à courbure constante, et le convertit en un espace de vitesse, comme le montre la figure 7.6[18].

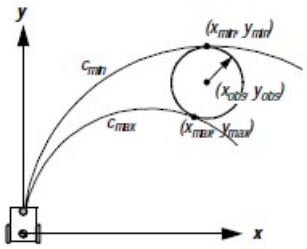


Figure 7.6: Des courbes tangentes pour un obstacle ; [18]

La méthode de la courbure des voies (The Lane Curvature Method)

LCM calcule un ensemble de couloirs souhaités, en échangeant la longueur et la largeur des couloirs avec l'obstacle le plus proche. Le couloir ayant les meilleures propriétés est choisi à l'aide d'une fonction objective. Le cap local est choisi de telle sorte que le robot passe au meilleur couloir s'il n'est pas déjà dans ce couloir[18].

7.2.5 Approches de la fenêtre dynamique (Dynamic Window Approaches)

L'approche de la fenêtre dynamique locale (The Local Dynamic Window Approach)

Dans l'approche par fenêtre dynamique locale, la cinématique du robot est prise en compte en recherchant un espace de vitesse bien choisi. L'espace de vitesse est constitué de tous les ensembles possibles de tuples (v, ω) où v est la vitesse et ω la vitesse angulaire. L'approche suppose que les robots se déplacent uniquement sur des arcs de cercle représentant chacun de ces tuples[18].

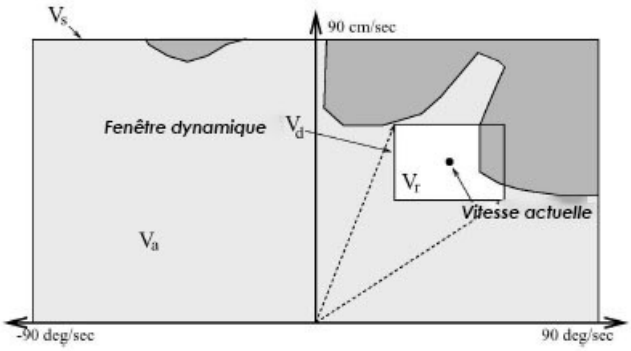


Figure 7.7: L'approche de la fenêtre dynamique. La fenêtre rectangulaire montre les vitesses possibles (v,ω) et le chevauchement avec les obstacles dans l'espace de configuration ; [16]

L'approche de la fenêtre dynamique globale (The Global Dynamic Window Approach)

L'approche de la fenêtre dynamique globale ajoute, comme son nom l'indique, une pensée globale à l'algorithme présenté ci-dessus. Cela se fait en ajoutant une fonction sans minimum appelée NF1 à la fonction cible O présentée ci-dessus. NF1 est une fonction calculée avec une technique de propagation des ondes presque identique à celle d'un feu de forêt (voir la figure 7.8 et la section sur la décomposition des cellules)[22].

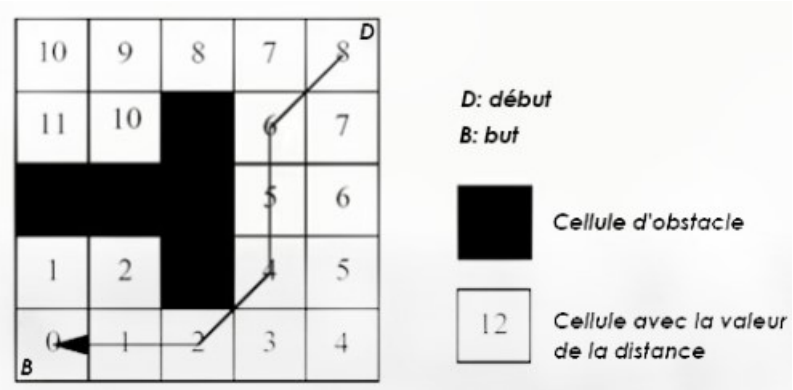


Figure 7.8: Un exemple de la transformation de la distance et du chemin résultant tel qu'il est généré par la fonction NF1. D désigne le début, B le but. Les voisins de chaque cellule i sont définis comme les quatre cellules adjacentes qui partagent un bord avec i (4-voisinage) ; [17]

L'approche Schlegel pour éviter les obstacles

Schlegel présente une approche qui prend en compte à la fois la dynamique et la forme réelle du robot. L'approche est adoptée pour les mesures de données laser brutes et la fusion de capteurs utilisant une grille cartésienne pour représenter les obstacles dans l'environnement. La performance en temps réel est obtenue grâce à l'utilisation de tableaux de recherche pré-calculés[18].

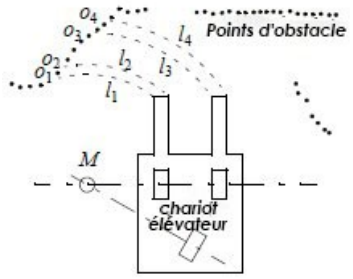


Figure 7.9: Les distances li résultant de la courbure ic lorsque le robot tourne autour de M ; [18]

7.2.6 Autres approches

Les approches décrites ci-dessus comptent parmi les systèmes de contournement d'obstacles les plus populaires. Il existe cependant de nombreuses autres techniques d'évitement d'obstacles dans le domaine de la robotique mobile. Par exemple, Tzafestas et Tzafestas donnent un aperçu des approches floues et neurofluides de l'évitement des obstacles. Inspirés par la nature, Chen et Quinn présentent une approche biologique dans laquelle ils reproduisent le réseau neural d'un cafard. Ce réseau est ensuite appliqué à un modèle de véhicule à quatre roues. Les fonctions de Liapunov forment une théorie bien connue qui peut être utilisée pour prouver la stabilité des systèmes non linéaires. Dans l'article de Vanualailai, Nakagiri et Ha, les fonctions de Liapunov sont utilisées pour mettre en œuvre une stratégie de contrôle de deux masses ponctuelles se déplaçant dans un environnement connu. Tous les obstacles sont définis comme des anti-cibles ayant une position exacte et une forme circulaire. Les anticipées sont ensuite utilisées lors de l'élaboration des lois de contrôle du système. Toutefois, à notre connaissance, ce modèle mathématique complexe n'a pas été testé sur un robot du monde réel[18].

Troisième partie III

Étude Expérimentale

Chapitre 8

Cinématique

8.1 Le Modèle de simulation et de cinématique différentielle

Dans le contexte des Robots mobiles autonomes on va faire un Modèle de simulation et de cinématique différentielle.

```
% CURSOS ONLINE (MI PLATAFORMA RECOMENDADO)
%
% Rob tica con Matlab y Arduino: Modelo y Simulaci n: http://bit.ly/2MHWYPb
% Rob tica con Matlab y Arduino: Dise o de controladores: http://bit.ly/2SXaDW3

clc
clear
close all

ts=0.1;
t=0:ts:60;

u=0.1*ones(1,length(t));
w=0.08*ones(1,length(t));


xr(1)=0;
yr(1)=0;
phi(1)=0;


for k=1:length(t)

    xrp(k)=u(k)*cos(phi(k));
    yrp(k)=u(k)*sin(phi(k));

    xr(k+1)=xr(k)+ts*xrp(k);
    yr(k+1)=yr(k)+ts*yrp(k);
    phi(k+1)=phi(k)+ts*w(k);
```

```
30
31  end
32
33  pasos=20;  fig=figure;
34  set(fig,'position',[10 60 980 600]);
35  axis square; cameratoolbar
36  axis([-3 3 -3 3 -0.5 1]); grid on
37  MobileRobot;
38  M1=MobilePlot(xr(1),yr(1),phi(1));hold on
39  M2=plot(xr(1),yr(1),'b','LineWidth',2);
40
41  for i=1:pasos:length(t)
42
43      delete (M1)
44      delete (M2)
45      M1=MobilePlot(xr(i),yr(i),phi(i)); hold on
46      M2=plot(xr(1:i),yr(1:i),'b','LineWidth',2);
47
48      pause(ts)
49
50  end
```

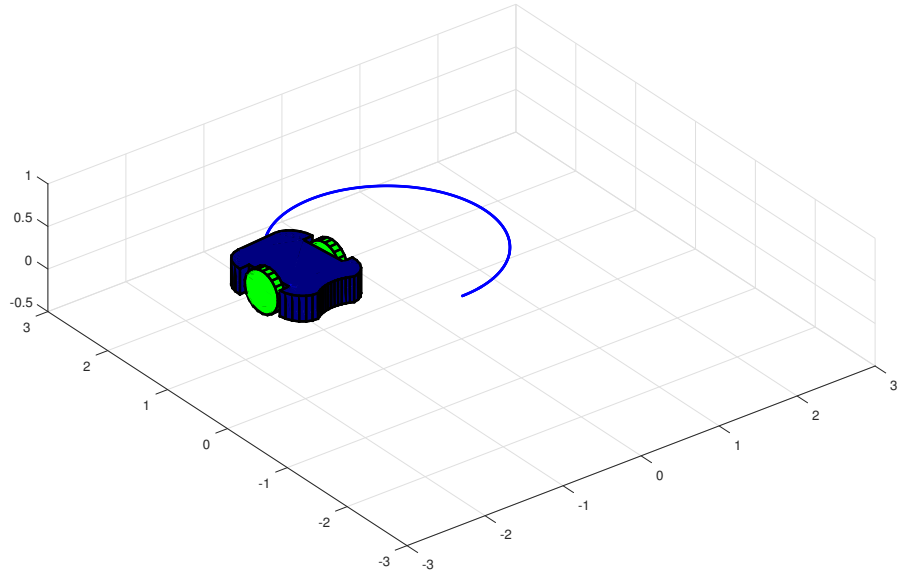


Figure 8.1: Resultat de simulation

8.2 le Modéle de Simulation Monocycle 3D

On va essayer de tester le Modèle de Simulation Monocycle 3D.

```
clc  % Clear Command Window
clear all %Remove items from workspace, freeing up system memory
close all % Remove specified figure

ts=0.1; % sample time
t=0:ts:30; % vector time

a=0.3;
% initial center position of robot
    x(1)=0;     % X[m]
    y(1)=0;      % Y[m]
    phi(1)=0;     % [rad]

% 3) Desired reference
    hxd = 2*cos(0.2*t);
    hyd = 2*sin(0.2*t);

    hxdp = -2*0.2*sin(0.2*t);
    hydp = 2*0.2*cos(0.2*t);

    K=1;

   hx(1)=x(1)+a*cos(phi(1));
   hy(1)=y(1)+a*sin(phi(1));

for k=1:length(t)

% Following trayectory Control

% Error
    hxe(k)=hxd(k)-hx(k);
    hye(k)=hyd(k)-hy(k);

    e=[hxe(k) hye(k)]';

% Jacobian Matrix

    J=[cos(phi(k)) -a*sin(phi(k));...
       sin(phi(k)) a*cos(phi(k))];

% Desired velocities
    hdp=[hxdp(k) hydp(k)]';

    v=inv(J)*(hdp+(K*e));

    u(k)=v(1);
    w(k)=v(2);


 % Set action control in robot(real) or model

    xp(k)=u(k)*cos(phi(k));
    yp(k)=u(k)*sin(phi(k));
    phip(k)=w(k);

 %  Position in k+1
    x(k+1)=x(k)+ts*xp(k);
    y(k+1)=y(k)+ts*yp(k);
    phi(k+1)=phi(k)+ts*phip(k);

    hx(k+1)=x(k+1)+a*cos(phi(k+1));
    hy(k+1)=y(k+1)+a*sin(phi(k+1));

end
```

```
66
67  %% Simulacion
68
69  fig=figure; % new figure
70  set(fig,'position',[10 60 980 600]); % position window
71  axis equal; % Set axis aspect ratios
72  axis([-3 3 -3 3 -1 1]); % Set axis limits
73  grid on; % Display axes grid lines
74
75  MobileRobot; % Parameters of robot
76  M1=MobilePlot(x(1),y(1),phi(1)); % Plot robot in initial position hx,hy and phi orientation
77  hold on; % Retain current plot when adding new plot
78  plot(hxd,hyd,'r'); % plot trayectory.
79  xlabel('x(m)'); ylabel('y(m)'); zlabel('z(x)'); % Label axis
80  camlight('rigth');
81
82  step=5;
83  %%
84  for i=1:step:length(t) % Loop emulation
85
86      delete (M1)
87      M1=MobilePlot(x(i),y(i),phi(i)); hold on
88      hold on; % Retain current plot when adding new plot
89      plot(hx(1:i),hyd(1:i),'b'); % plot trayectory.
90      pause(ts)
91  end
92
93  %% Graficas
94  figure('Name','Contr le d`erreurs')
95  plot(t,hxe); hold on;
96  plot(t,hye); hold on;
97
98
99  figure('Name','Contr le d`action')
100 subplot(211)
101 plot(t,u,'linewidth',2), grid on
102 legend('Vitesse lin aire u')
103 xlabel('Temps'), ylabel('Vitesse [m/s]')
104 subplot(212)
105 plot(t,w,'g','linewidth',2), grid on
106 legend('Vitesse angulaire w')
107 xlabel('Temps'), ylabel('Vitesse [rad/s]')
```

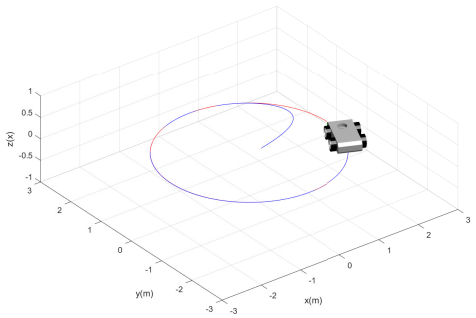


Figure 8.2: Resultat de simulation

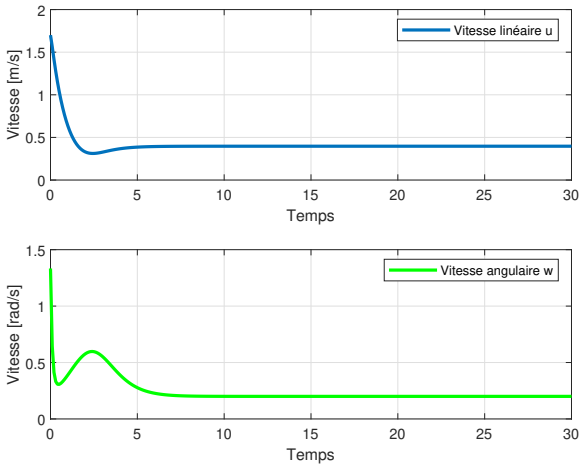


Figure 8.3: Contrôle des actions

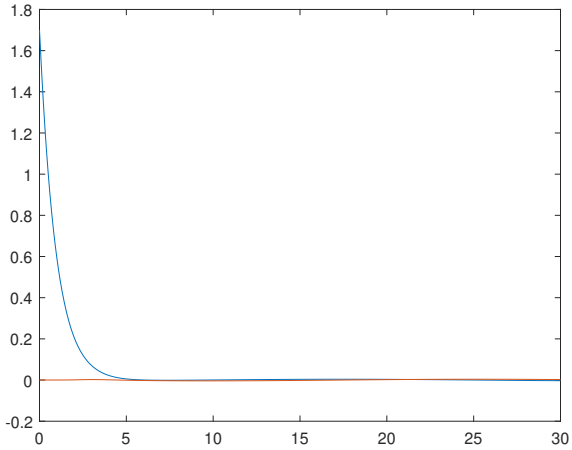


Figure 8.4: Contrôle des erreurs

Chapitre 9

Capteur et Perception

9.1 Le Modèle de Simulation de Fusion des capteurs

Dans cette section on va faire un modèle de simulation de fusion des capteurs basé sur le filtre de Kalman.

```
function fusion
% Kalman Filter sensor fusion example based on
%
% http://www.slideshare.net/antoniomorancardenas/data-fusion-with-kalman-filtering
%
% % See http://home.wlu.edu/~levys/kalman_tutorial for background
%
% Copyright (C) 2014 Simon D. Levy
%
% This code is free software: you can redistribute it and/or modify
% it under the terms of the GNU Lesser General Public License as
% published by the Free Software Foundation, either version 3 of the
% License, or (at your option) any later version.
%
% This code is distributed in the hope that it will be useful,
% but WITHOUT ANY WARRANTY without even the implied warranty of
% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
% GNU General Public License for more details.
%
% You should have received a copy of the GNU Lesser General Public License
% along with this code. If not, see <http:#www.gnu.org/licenses/>.

% Parameters ==============================================================

% Biases
BIAS1 = +1;
BIAS2 = -1;

% Measurement noise covariances
```

```
R1 = 0.64;
R2 = 0.64;

% Process noise covariance
Q = .005;

% State transition model
A = 1;

% Observation model
C1 = 1;
C2 = 1;

% Duration
N = 1000;

% Run ============================================================

% Generate the signal x as a sine wave
x = 20 + sin(5*linspace(0,1,N)*pi);

% Add some process noise with covariance Q
w = sqrt(Q) * randn(size(x));
x = x + w;

% Compute noisy sensor values
z1 = BIAS1 + x + sqrt(R1) * randn(size(x));
z2 = BIAS2 + x + sqrt(R2) * randn(size(x));

% Run a single sensor example and plot it
xhat = kalman(z1, A, C1, R1, Q);

% Plot sensed and estimated values
clf
plotsigs(1, z1, xhat, 'Capteur 1')
title('Un capteur : le nettoyage des signaux')

% Plot estimate and actual values
plotsigsrms('Un capteur', 2, x, xhat)

% Run the Kalman filter on fused sensors
xhat = kalman([z1; z2], A, [C1; C2], [R1 0; 0 R2], Q);

% Plot fusion example
plotsigsrms('Deux capteurs', 3, x, xhat)


% Helper functions ================================================

function plotsigs(pos, sig1, sig2, sig1label)
subplot(3,1,pos)
hold on
plot(sig1, 'k')
plot(sig2, 'r')
legend({sig1label, 'Estimation'})
ylim([15 25])
hold off

function plotsigsrms(label, pos, x, xhat)
plotsigs(pos, x, xhat, 'R el')
title(sprintf('%s: Erreur RMS = %f', label, sqrt(sum((x-xhat).^2)/length(x))))
```

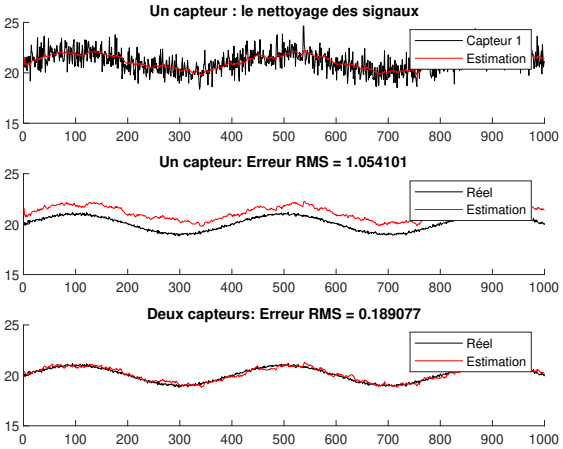


Figure 9.1: Résultat de simulation

Chapitre 10

Contrôle de Robot Mobile

10.1 Contrôle de position avec orientation différentielle du robot

Maintenant on va faire un modèle de Contrôle de position avec orientation différentielle du robot.

```
% CURSOS ONLINE (MI PLATAFORMA RECOMENDADO)
%
% Rob tica con Matlab y Arduino: Modelo y Simulaci n: http://bit.ly/2MHWYPb
% Rob tica con Matlab y Arduino: Dise o de controladores: http://bit.ly/2SXaDW3

clc
clear
close all

% 1) Tiempo
ts=0.1;
t=0:ts:30;

% 2) Condiciones iniciales
    xr(1)=-4;     %Posicion del centro de eje de las ruedas en X[m]
    yr(1)=4;    %Posicion del centro de eje de las ruedas en Y[m]
    phi(1)=0;    %orientacion inicial con respecto al eje x [rad]

    xc(1)=xr(1);
    yc(1)=yr(1);
% 3) Referencias deseadas
    xrd = 4;
    yrd = -4;
    phid=pi/2;

    K1=0.5;
    K2=0.5;
```

```
    q2=0.5;

for k=1:length(t)

% 4) Control

        %a) Errores de control
            l(k)=sqrt((xrd-xr(k))^2+(yrd-yr(k))^2);
            zeta(k)=atan2((yrd-yr(k)),(xrd-xr(k)))-phi(k);
            psi(k)=atan2((yrd-yr(k)),(xrd-xr(k)))-phid;


        %d) Ley de control
            u(k)=K1*cos(zeta(k))*l(k);     % Velocidad lineal de entrada
            w(k)=K2*zeta(k)+(K1/zeta(k))*cos(zeta(k))*sin(zeta(k))*(zeta(k)+q2*psi(k));     % Velocidad
                 angular de entrada;


 % 5) Aplicar acciones de control al robot

    xrp(k)=u(k)*cos(phi(k));
    yrp(k)=u(k)*sin(phi(k));

 % Hallar posiciones
    xr(k+1)=xr(k)+ts*xrp(k);
    yr(k+1)=yr(k)+ts*yrp(k);
    phi(k+1)=phi(k)+ts*w(k);

    xc(k+1)=xr(k+1);
    yc(k+1)=yr(k+1);

end


%% Simulacion

pasos=10;  fig=figure('Name','Simulacion');

set(fig,'position',[60 60 980 600]);
axis square; cameratoolbar
axis([-6 6 -6 6 0 1]);
grid on
MobileRobot;
M1=MobilePlot(xr(1),yr(1),phi(1));hold on
M2=MobilePlot(xr(end),yr(end),phi(end));
M3=plot(xr(1),yr(1),'b','LineWidth',2);


for i=1:pasos:length(t)

    delete (M2)
    delete (M3)
    M2=MobilePlot(xr(i),yr(i),phi(i)); hold on
    M3=plot(xr(1:i),yr(1:i),'b','LineWidth',2);

    pause(ts)

end


% %% Graficas
figure('Name','Erreurs')
subplot(311)
plot(t,l,'linewidth',2), grid on
legend('Erreur en l')
xlabel('Temps'), ylabel('Erreur [m]')
```

```
subplot(312)
plot(t,zeta,'g','linewidth',2), grid on
legend('Erreur en zeta')
xlabel('Temps'), ylabel('Erreur [rad]')
subplot(313)
plot(t,psi,'g','linewidth',2), grid on
legend('Erreur en psi')
xlabel('Temps'), ylabel('Erreur  [rad]')

figure('Name','Actions de contr le')
subplot(211)
plot(t,u,'linewidth',2), grid on
legend('Vitesse lin aire u')
xlabel('Temps'), ylabel('Vitesse [m/s]')
subplot(212)
plot(t,w,'g','linewidth',2), grid on
legend('Vitesse angulaire w')
xlabel('Temps'), ylabel('Vitesse  [rad/s]')
```

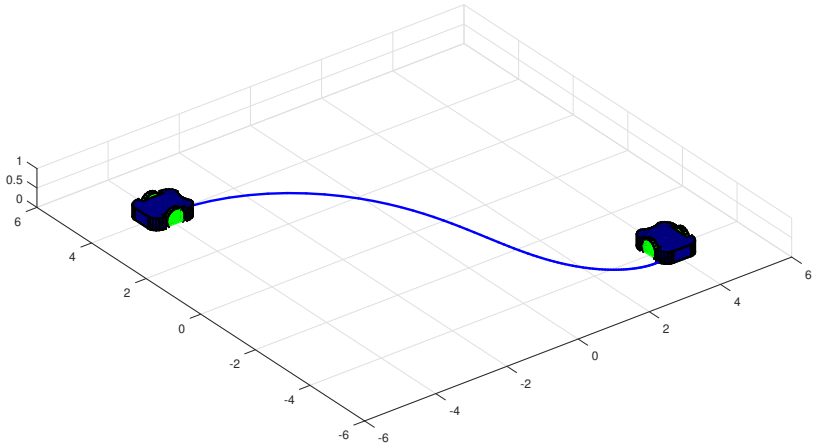


Figure 10.1: Résultat de simulacion

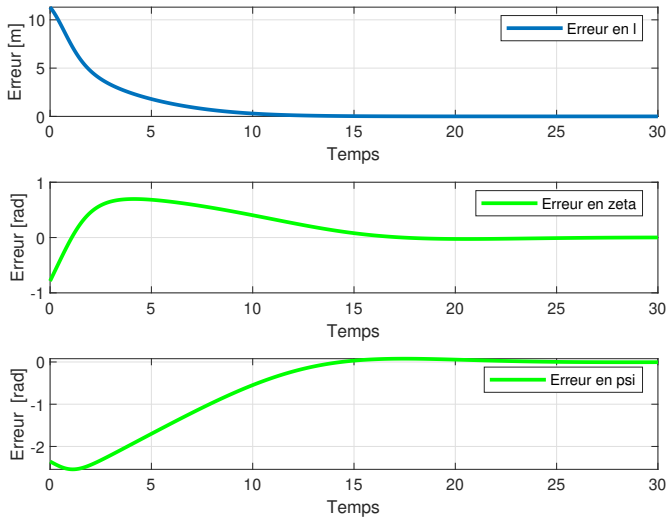


Figure 10.2: les erreurs

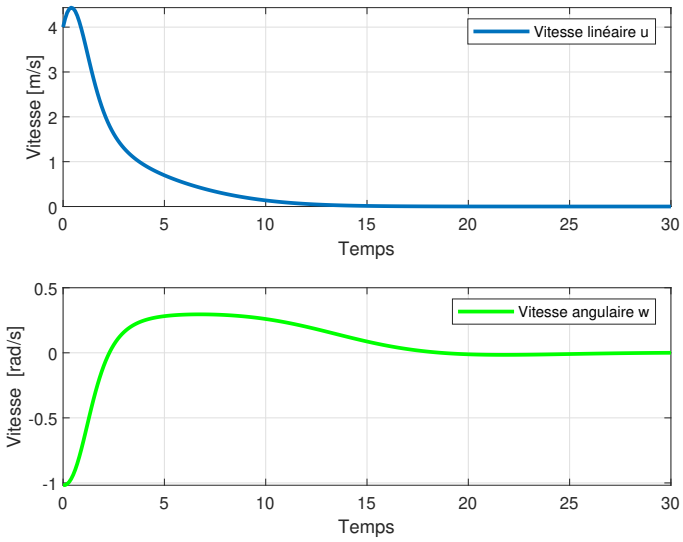


Figure 10.3: Les actions de contrôle

10.2 Commande différentielle de suivi des robots mobiles

Dans le cadre de contrôle de robot mobile on va faire la modélisation et la Commande différentielle de suivi des robots mobiles.

```
clc
clear
close all

% 1) Tiempo
ts=0.1;
t=0:ts:60;

% 2) Condiciones iniciales
    xc(1)=0;     %Posicion del centro de eje de las ruedas en X[m]
    yc(1)=0;    %Posicion del centro de eje de las ruedas en Y[m]
    phi(1)=pi;    %orientacion inicial con respecto al eje x [rad]

% 3) Referencias deseadas
    % Trayectoria de un circulo
%     xrd = 4*cos(0.2*t);
%     yrd= 4*sin(0.2*t)+1;
%
%     xrdp= -4*0.2*sin(0.2*t);  %Posici n x1
%     yrdp= 4*0.2*cos(0.2*t);   %Posici n y1

    % Trayectoria de una senoidal
%     xrd =  0.1*t;
%     yrd= 2*sin(0.1*t);
%
%    % Trayectoria de un 8
%     xrd = 4*sin(0.4*t);                          %Posici n x
%     yrd = 4*sin(0.2*t);                              %Posici n y

    % Trayectoria de un corazon
    xrd = (8*sin(0.1*t)-9*sin(3*0.1*t))/3;
    yrd= (15*cos(0.1*t)-5*cos(2*0.1*t)-2*cos(3*0.1*t)-cos(4*0.1*t))/4;

%     % Trayectoria de 5 puntas
    div=10;

%       a=1/div; b=1/div; c=1/div; d=1/div; j=3; k=2.5;
% %     xrd = cos(a*t)-cos(b*t).^j;
% %     yrd= sin(c*t)-sin(d*t).^k;



    xrdp= diff([0 xrd]);  %Posici n x1
    yrdp= diff([0 yrd]);   %Posici n y1

a=0.2;

xr(1)=xc(1)+a*cos(phi(1));
yr(1)=yc(1)+a*sin(phi(1));

for k=1:length(t)

% 4) Control

        %a) Errores de control
        xre(k) = xrd(k) - xr(k);
```

```
57          yre(k) = yrd(k) - yr(k);
58          e = [xre(k);yre(k)];
59
60          %b) Matriz Jacobiana
61          J=[cos(phi(k)) -a*sin(phi(k));...
62             sin(phi(k))  a*cos(phi(k))];
63
64          %c) Matriz de ganancia
65          K = [1 0;...
66              0 1];
67
68          %d) Ley de control
69          hdp=[xrdp(k);yrdp(k)];
70
71          v = inv(J)*(hdp+K*tanh(e));
72
73
74          u(k)=v(1);     %Velocidad lineal de entrada al robot
75          w(k)=v(2);     % Velocidad angular de entrada al robot
76
77   % 5) Aplicar acciones de control al robot
78
79      xrp(k)=u(k)*cos(phi(k))-a*w(k)*sin(phi(k));
80      yrp(k)=u(k)*sin(phi(k))+a*w(k)*cos(phi(k));
81
82   % Hallar posiciones
83      xr(k+1)=xr(k)+ts*xrp(k);
84      yr(k+1)=yr(k)+ts*yrp(k);
85      phi(k+1)=phi(k)+ts*w(k);
86
87      xc(k+1)=xr(k+1)-a*cos(phi(k+1));
88      yc(k+1)=yr(k+1)-a*sin(phi(k+1));
89
90  end
91
92
93  %% Simulacion
94
95  pasos=10;  fig=figure('Name','Simulacion');
96
97
98  set(fig,'position',[60 60 980 600]);
99  axis square; cameratoolbar
100 axis([-6 6 -6 6 0 1]);
101 grid on
102 MobileRobot;
103 M1=MobilePlot(xr(1),yr(1),phi(1));hold on
104 M2=plot(xr(1),yr(1),'b','LineWidth',2);
105 plot(xrd,yrd,'r','LineWidth',2);
106
107 for i=1:pasos:length(t)
108
109     delete (M1)
110     delete (M2)
111     M1=MobilePlot(xc(i),yc(i),phi(i)); hold on
112     M2=plot(xr(1:i),yr(1:i),'b','LineWidth',2);
113
114    pause(ts)
115
116 end
117
118
119 %% Graficas
120 figure('Name','Erreurs')
121 subplot(211)
122 plot(t,xre,'linewidth',2), grid on
```

```
legend('Erreur en x')
xlabel('Temps'), ylabel('Erreur  [m]')
subplot(212)
plot(t,yre,'g','linewidth',2), grid on
legend('Erreur en y')
xlabel('Temps'), ylabel('Erreur  [m]')

figure('Name','Actions de contr le')
subplot(211)
plot(t,u,'linewidth',2), grid on
legend('Vitesse lin aire u')
xlabel('Temps'), ylabel('Vitesse [m/s]')
subplot(212)
plot(t,w,'g','linewidth',2), grid on
legend('Vitesse angulaire  w')
xlabel('Temps'), ylabel('Vitesse  [rad/s]')
```

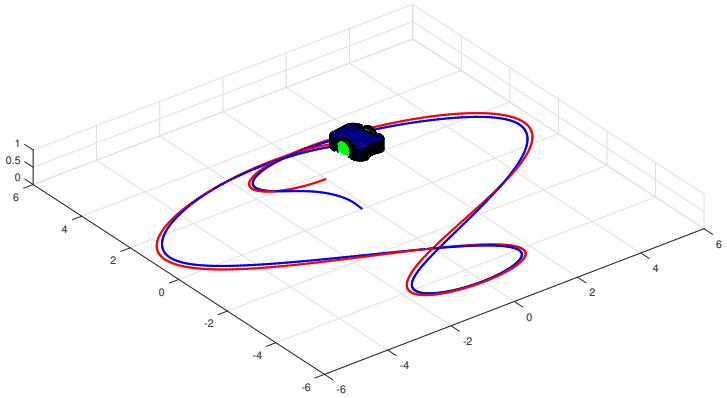


Figure 10.4: Résultat de simulation

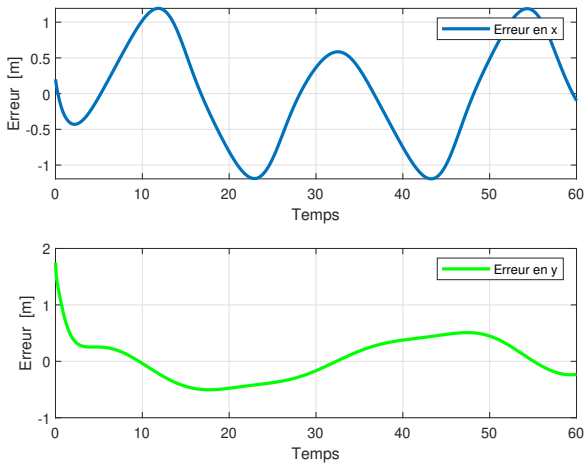


Figure 10.5: Les erreurs

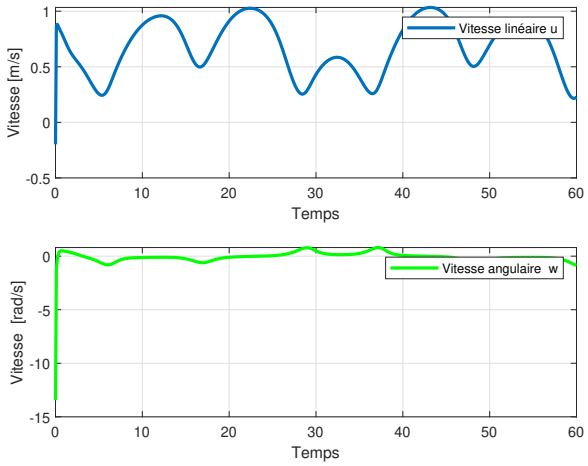


Figure 10.6: Les actions de contrôle

10.3 Robot mobile à roues

Dans cette section On va simuler un robot mobile a roues .

```
% Copyright 2016, Ehab Al Khatib, All rights reserved.


clear all
clc
close all
%%
global  b

b = 0.18;
his.dt=[];

% IO_SFL
dt=0.1;
T=360;
k=2;
bref=0.08;
x=0.2;

%%
y=0.2;
theta=0/2;
X=[x;y;theta];

Vref=0.0;        % max 0.5 m/s
Omegaref=0.0;    % max 3 rps

% Reference Trajectory
tperiod = 2*[0 5 10 15 20 ];

points =[0 0 ;1 0  ; 1 1  ; 0 1; 0 0];
x = points(:,1);
y = points(:,2);
```

```
34
35
36  t = 0:dt:tperiod(end);
37  x_des = interp1(tperiod,x,t);
38  y_des = interp1(tperiod,y,t);
39  % plot(x_des,y_des,'.',x_des(1),y_des(1),'ro',x_des(end),y_des(end),'go')
40
41
42  his.X=[];
43  his.ex=[];
44  his.ey=[];
45
46  %%  Graphics
47  f3=figure;
48  ax3=axes('parent',f3);
49  PLOT.RefTrajectory=line('parent',ax3,'XData',[],'YData',[],'LineStyle',':','color','r','LineWidth',2);
50  PLOT.Robot=line('parent',ax3,'XData',[],'YData',[],'LineStyle','-','color','g','LineWidth',2);
51  xlabel('x (m)')
52  ylabel('y (m)')
53  legend('Reference Trajectory','Acual ')
54  grid on
55
56  %  robot dimensions
57  A.R_w = 0.3/2; % robot width/2
58  A.R_l=0.5/2;   % robot length/2
59  A.a1 = [-A.R_l -A.R_w]';
60  A.b1 = [A.R_l -A.R_w/2]';
61  A.b2 = [A.R_l A.R_w/2]';
62  A.c = [-A.R_l A.R_w]';
63  A.P = [A.a1 A.b1 A.b2 A.c];
64  pl=[];
65
66  A.Rot = [ cos(X(3)) -sin(X(3)); sin(X(3)) cos(X(3))]*A.P; %rotated car
67  A.Prot_trasl = A.Rot + [ ones(1,4)*X(1); ones(1,4)*X(2)]; % add offset of car's center
68
69  A.P_robot=patch(A.P(1,:),A.P(2,:),'g');
70  A.P_robot.XData=A.Prot_trasl(1,:)';
71  A.P_robot.YData=A.Prot_trasl(2,:)';
72  axis([-0.5 1.5 -0.5 1.5])
73
74
75
76  %% Main loop
77
78
79  for i=1:length(x_des)
80
81      % Simulate WMR
82      X=Move_Robot(X,[Vref,Omegaref],dt);
83
84      %     IOSFL Controller
85      xb=X(1)+bref*cos(X(3));
86      yb=X(2)+bref*sin(X(3));
87
88      ex= x_des(i)-xb;
89      ey= y_des(i)-yb;
90
91
92      Vd_x=k*ex;
93      Vd_y=k*ey;
94
95      Vref = cos(X(3))*Vd_x + sin(X(3))*Vd_y ;
96      Omegaref = (1/bref)*(cos(X(3))*Vd_y-sin(X(3))*Vd_x);
97
98
99      % Store the data
```

```
100
101     his.X=[his.X X];
102     his.ex=[his.ex ex];
103     his.ey=[his.ey ey];
104
105     % update the figures
106
107     set(PLOT.RefTrajectory,'XData',x_des(1:i),'YData',y_des(1:i));
108     set(PLOT.Robot,'XData',his.X(1,:),'YData',his.X(2,:));
109
110     A.Rot = [ cos(X(3)) -sin(X(3)); sin(X(3)) cos(X(3))]*A.P; %rotated car
111     A.Prot_trasl = A.Rot + [ ones(1,4)*X(1); ones(1,4)*X(2)]; % add offset of car's center
112     A.P_robot.XData=A.Prot_trasl(1,:)';
113     A.P_robot.YData=A.Prot_trasl(2,:)';
114     drawnow
115
116
117 end
```

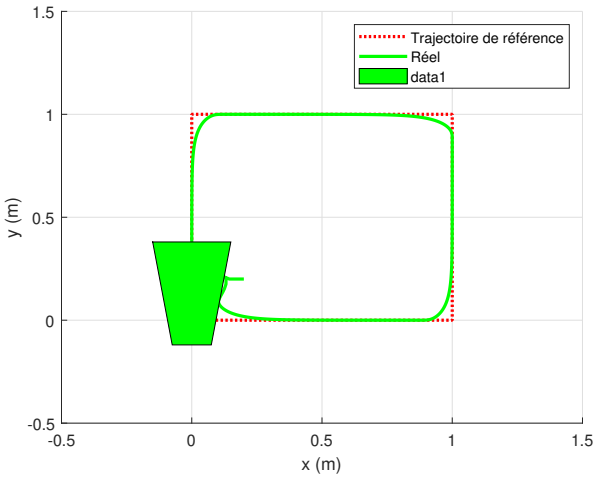


Figure 10.7: Résultat de simulation

Chapitre 11

Localisation et Cartographie

Concernant la localisation et cartographie on va simuler le filtre de Kalman étendu.

11.1 Filtre Kalman étendu

```
close all
clear

rad = pi/180.0;
global v w  X_predicted goal_reached current_pose goalx goaly;
syms x_p y_p phi_p v w

a1=0.19;
a2=0.19;
a3=0.16;
a4=0.16;

covariance=zeros(3,3);

map_obs();

goalx = [ -20 -5 ];
goaly = [  -7  20];

% ObstacleX = [-20 -20 -5  30 30 8];
% ObstacleY = [-25  0  25  -25  0 25];

x_p = 2.5;
y_p = -20;
phi_p = pi/2;

previous_pose=[x_p;y_p;phi_p];
X=previous_pose;
X_predicted=X;
```

```
previous_pose_predicted=X_predicted;
delt=1;

 %cov_p=[0 0 0;0 0 0; 0 0 0];


circle_draw([X(1) X(2)] ,1);
x=[X(1) X(1)+3*cos(X(3))];
y=[X(2) X(2)+3*sin(X(3))];
line(x,y,'color','g');

circle_draw([X_predicted(1) X_predicted(2)] ,1);
x=[X_predicted(1) X_predicted(1)+3*cos(X_predicted(3))];
y=[X_predicted(2) X_predicted(2)+3*sin(X_predicted(3))];
line(x,y,'color','r');

 %f=[x_p+v*cos(phi_p+w);y_p+v*sin(phi_p+w);phi_p+w];
hold off;
goal_reached=[0 0];

count=0;


for j=1:2

current_pose=X_predicted;
movebase(j);
while goal_reached(j)~=1

count=count+1;
map_obs();

%axis ([-50 50 -50 50]);

 v=0.65;
 if(count<=12)
 w=0.08;
 else
 w=0;
 end;
%pause;

  %v=input('enter velocity');
   %   w=input('enter angle');
 %     w=w*rad*(-1.0);
   %   r=[1 0;0 1];
r=[(a1*abs(v)+a2*abs(w))^2 0;
   0 (a3*abs(v)+a4*abs(w))^2];
ran=mvnrnd([v w],r);
v_actual=ran(1);
w_actual=ran(2);
  % ran=ran_gauss([0 0],r);

  change_pose_p=[v*delt*cos(X_predicted(3)+w*delt);v*delt*sin(X_predicted(3)+w*delt);w*delt];
  X_predicted=previous_pose_predicted+change_pose_p;
  f_jac=[ 1 0 -v*delt*sin(previous_pose_predicted(3)+delt*w);
          0 1  delt*v*cos(previous_pose_predicted(3)+delt*w);
          0 0  1];
  g=[ delt*cos(previous_pose_predicted(3) + delt*w)  -delt^2*v*sin(previous_pose_predicted(3) + delt*w);
      delt*sin(previous_pose_predicted(3) + delt*w)   delt^2*v*cos(previous_pose_predicted(3) + delt*w);
      0                                  delt];
  covariance =f_jac*covariance*f_jac'+g*r*g';



%    disp(v);
```

```
%    disp(w);
%    disp(r);
    %  disp(p_pose(3));

change_pose=[v_actual*delt*cos(X(3)+w_actual*delt);v_actual*delt*sin(X(3)+w_actual*delt);w_actual*delt];
X=previous_pose+change_pose;

%



     %    disp(f(1));
      %   disp(f(2));
     % disp(f(3));
    %  phi_p=phi_p+w*delt;
 %     disp(phi_p);

 ObstacleX=[-20 -20 -5  30 30 8];
 ObstacleY=[-25  0  25  -25  0 25];

       %disp(f);
       %disp(f_p);
 temp=X;

%

X
X_predicted
sum_1=zeros(3,1);
sum_2=zeros(3,3);
detected=0;
 for i=1:6

         q=(ObstacleX(i)-X(1))^2+(ObstacleY(i)-X(2))^2;
     if  sqrt(q)<12.0

         detected=2;
    %     disp('inside');
         disp(i);
          Q=[0.02 0 0;
              0 0.05 0;
              0 0 0.09];
     %     disp('inside if');
      %    disp(sqrt(q));
         q_predicted=(ObstacleX(i)-X_predicted(1))^2+(ObstacleY(i)-X_predicted(2))^2;
         z_predicted=[sqrt(q_predicted);
                      atan2(ObstacleY(i)-X_predicted(2),ObstacleX(i)-X_predicted(1))-X_predicted(3);
                      0];
         z_actual=[sqrt(q);atan2(ObstacleY(i)-X(2),ObstacleX(i)-X(1))-X(3);0];
         %disp(z_pre);


         H=[-(ObstacleX(i)-X_predicted(1))/sqrt(q_predicted)  -(ObstacleY(i)-X_predicted(2))/sqrt(
              q_predicted) 0;
             (ObstacleY(i)-X_predicted(2))/q_predicted   -(ObstacleX(i)-X_predicted(1))/q_predicted -1/
                  q_predicted;
             0 0 0];
         S=H*covariance*H'+Q;
         K=(covariance*H')/(S);
         %K=(covariance)*pinv(S);
         sum_1=sum_1+K*(z_actual-z_predicted);
         sum_2=sum_2+K*H;
         %X_predicted=X_predicted+K*(z_actual-z_predicted);

        % covariance=(eye(3)-K*H)*covariance;
         %X_predicted;
```

```
     end;

 end;
t_covv=covariance(1:2,1:2);
[b,d]=eig(t_covv);
dir=[cos(0:0.1:2*pi);sin(0:0.1:2*pi)];
el=b*sqrtm(d)*dir;
x=[X_predicted(1);X_predicted(2)];
el=[el el(:,1)]+repmat(x,1,size(el,2)+1);
plot(el(1,:),el(2,:),'color','r');

if detected==2

    X_predicted=X_predicted+sum_1;
    covariance=(eye(3)-sum_2)*covariance;
end;

circle_draw([temp(1) temp(2)] ,1);
x=[temp(1) temp(1)+3*cos(temp(3))];
y=[temp(2) temp(2)+3*sin(temp(3))];
line(x,y,'color','g');

circle_draw([X_predicted(1) X_predicted(2)] ,1);
x=[X_predicted(1) X_predicted(1)+3*cos(X_predicted(3))];
y=[X_predicted(2) X_predicted(2)+3*sin(X_predicted(3))];
line(x,y,'color','r');
if(detected==2)
    detected=0;
    t_covv=covariance(1:2,1:2);
[b,d]=eig(t_covv);
dir=[cos(0:0.1:2*pi);sin(0:0.1:2*pi)];
el=b*sqrtm(d)*dir;
x=[X_predicted(1);X_predicted(2)];
el=[el el(:,1)]+repmat(x,1,size(el,2)+1);
plot(el(1,:),el(2,:),'color','g');

end;
mov(count)=getframe();
% movie2avi(mov,'ekf.avi','fps',2);
pause(0.1);
% close all;




     %disp(covv);

     %disp(d);
     %disp(v);
%
%
% %      change_pose(1)=v*cos(p_pose(3)+w*rad);
% %        change_pose(2)=v*sin(p_pose(3)+w*rad);
% %          change_pose(3)=w*rad;
% %
% %          present_pose=p_pose+change_pose;
%
%
%
   previous_pose_predicted=X_predicted;
   previous_pose=X;
   movebase(j);
end
end
```

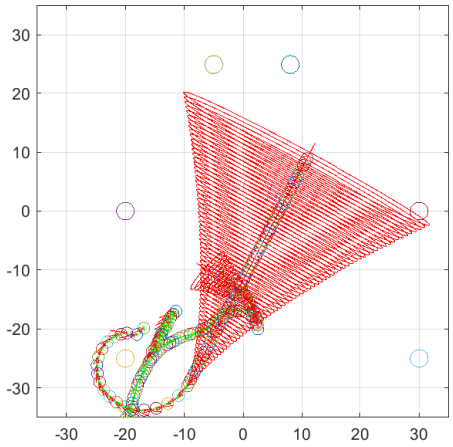


Figure 11.1: Résultat de simulation

Chapitre 12

l'évitement des obstacles

Maintenant on s'intéresse au algorithme de bug pour l'évitement des obstacles et on va faire une simple simulation .

12.1 L'algorithme de Bug

```
% Bug 1 test script
% algorithm should circle each obsticle, remember the closest location to goal
% return to that location and move towards goal from there

clear
start = [0 0]; goal = [5 3];
ss = .1;
obs = {[1 2], [1 0], [3 0]; [2 3] [4 1] [5 2]};


[path, dist_T] = computeBug1(start, goal, obs, ss)

figure
plot(path(:,1),path(:,2))

figure
plot(dist_T)
```

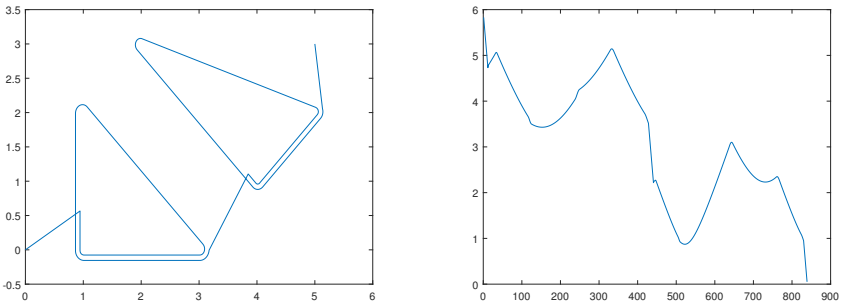


Figure 12.1: Résultat de simulation

Chapitre 13

planification du parcours

Dans ce chapitre on va simuler la planification du parcours on utilisant differents methodes.

13.1 la décomposition des cellules

Maintenant, nous sommes en train de simuler la planification du parcours on utilisant la methode de la decomposition cellulaire sous un planificateur GUI.

```
%% Version 1.0
%% NOTE
% uncomplete version
%%%%%%%%%%Bugs%%%%%%%%%%
% Reload obstacles maps
% when compute adjecent cells
%% Created by Ahmad Abbadi
% VUT Brno
% UAI, FME
%% FOR AUTHOR USE:
%version 1.x5 of this software used for scientific papers :
%1- ABBADI, A.; MATOU EK , R.; O MERA , P.; KNISPEL, L. SPATIAL GUIDANCE TO RRT PLANNER USING CELL-
     DECOMPOSITION ALGORITHM.%In 20th International Conference on Soft Computing, MENDEL 2014.%Mendel
     Journal series. 2014. Brno: VUT, 2014. s. 273-278. ISBN: 978-80-214-4984- 8. ISSN: 1803- 3814.
%2- A. Abbadi and R. Matousek,  PATH  PLANNING IMPLEMENTATION USING MATLAB,   presented at the
     International Conference of Technical Computing Bratislava 2014, Bratislava, 2014, pp. 1 5 .DOI
     :10.13140/2.1.3324.5767,%ISBN:978-80-7080-898-6

%%
% folderpath=pwd;
% addpath(strcat(folderpath,'/_code'));
% addpath(strcat(folderpath,'/_gui'));
```

```
% addpath(strcat(folderpath,'/_code/_polygon'));
% addpath(strcat(folderpath,'/_code/_polygon/_xternal'));
 addpath(genpath(pwd));
planner_GUI;
clear folderpath;
clc
```

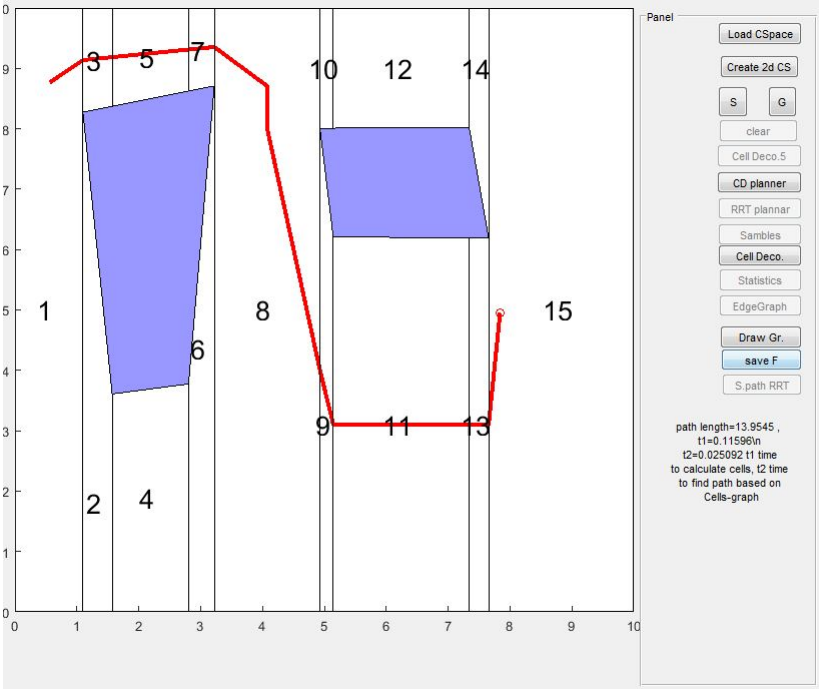


Figure 13.1: Le Planificateur GUI

13.2 le champ potentiel

Puis nous sommes allés à la simulation de la planification du parcours on utilisant la methode du champ potentiel.

```
%
% PotentialFieldScript.m
%

%% Generate some points

nrows = 400;
ncols = 600;

obstacle = false(nrows, ncols);

[x, y] = meshgrid (1:ncols, 1:nrows);   % note that x refers to the horizental axis (1:ncols), y to
     vertical (1:nrows)
% [X,Y] = meshgrid(xgv,ygv) replicates the grid vectors xgv and ygv to
%     produce the coordinates of a rectangular grid (X, Y). The grid vector
```

```
%     xgv is replicated numel(ygv) times to form the rows of X. The grid
%     vector ygv is replicated numel(xgv) times to form the columns of Y.
%% Generate some obstacle

obstacle (300:end, 100:250) = true;             % rectangular
obstacle (150:200, 400:500) = true;             % rectangular

t = ((x - 200).^2 + (y - 50).^2) < 50^2;        % a circle centered at (200, 50), radius = 50
obstacle(t) = true;

t = ((x - 400).^2 + (y - 300).^2) < 100^2;      % a circle, (400, 300), r=100
obstacle(t) = true;

%% Compute distance transform

d = bwdist(obstacle);
%  D = bwdist(BW) computes the Euclidean distance transform of the
%  binary image BW. For each pixel in BW, the distance transform assigns
%  a number that is the distance between that pixel and the nearest
%  nonzero pixel of BW. bwdist uses the Euclidean distance metric by
%  default.  BW can have any dimension.  D is the same size as BW.

% Rescale and transform distances
d2 = (d/100) + 1;

d0 = 2;
nu = 800;

repulsive = nu*((1./d2 - 1/d0).^2); % every point (x, y) on 2D space has a repulsive value. repulsive is a
     matrix.

repulsive (d2 > d0) = 0;            % d0 is like a threshold and repulsive force is 0 when distance from
     obstacle < d0


%% Display repulsive potential

figure;

% mesh(Z) and mesh(Z,C) use x = 1:n and y = 1:m.  In this case, the height,
% Z, is a single-valued function, defined over a geometrically rectangular
% grid.
m = mesh (repulsive);
m.FaceLighting = 'phong';
axis equal;

title ('Potentiel de r pulsion');

%% Compute attractive force

goal = [400, 50];

xi = 1/700;

attractive = xi * ( (x - goal(1)).^2 + (y - goal(2)).^2 );  % every point on 2D space has an attractive
     force value

figure;
m = mesh (attractive);
m.FaceLighting = 'phong';
axis equal;

title ('Potentiel d ''attractivit ');

%% Display 2D configuration space

```

```
figure;
imshow(~obstacle);
%  imshow(BW) displays the binary image BW. imshow displays pixels with the
%  value 0 (zero) as black and pixels with the value 1 as white.
hold on;
plot (goal(1), goal(2), 'r.', 'MarkerSize', 25);
hold off;

axis ([0 ncols 0 nrows]);
axis xy;
axis on;

xlabel ('x');
ylabel ('y');

title ('Espace de configuration');

%% Combine terms

f = attractive + repulsive;

figure;
m = mesh (f);
m.FaceLighting = 'phong';
axis equal;

title ('Potentiel total');

%% Plan route
start = [50, 350];

route = GradientBasedPlanner (f, start, goal, 1000);

%% Plot the energy surface

figure;
m = mesh (f);
axis equal;

%% Plot ball sliding down hill

[sx, sy, sz] = sphere(20);

scale = 20;
sx = scale*sx;
sy = scale*sy;
sz = scale*(sz+1);

hold on;
p = mesh(sx, sy, sz);
p.FaceColor = 'red';
p.EdgeColor = 'none';
p.FaceLighting = 'phong';
hold off;

for i = 1:size(route,1)
    P = round(route(i,:));
    z = f(P(2), P(1));

    p.XData = sx + P(1);
    p.YData = sy + P(2);
    p.ZData = sz + f(P(2), P(1));

    drawnow;

    drawnow;
```

```
144
145 end
146
147 %% quiver plot
148 [gx, gy] = gradient (-f);
149 skip = 20;
150
151 figure;
152
153 xidx = 1:skip:ncols;
154 yidx = 1:skip:nrows;
155
156 quiver (x(yidx,xidx), y(yidx,xidx), gx(yidx,xidx), gy(yidx,xidx), 0.4);
157
158 axis ([1 ncols 1 nrows]);
159
160 hold on;
161
162 ps = plot(start(1), start(2), 'r.', 'MarkerSize', 30);
163 pg = plot(goal(1), goal(2), 'g.', 'MarkerSize', 30);
164 p3 = plot (route(:,1), route(:,2), 'r', 'LineWidth', 2);
```

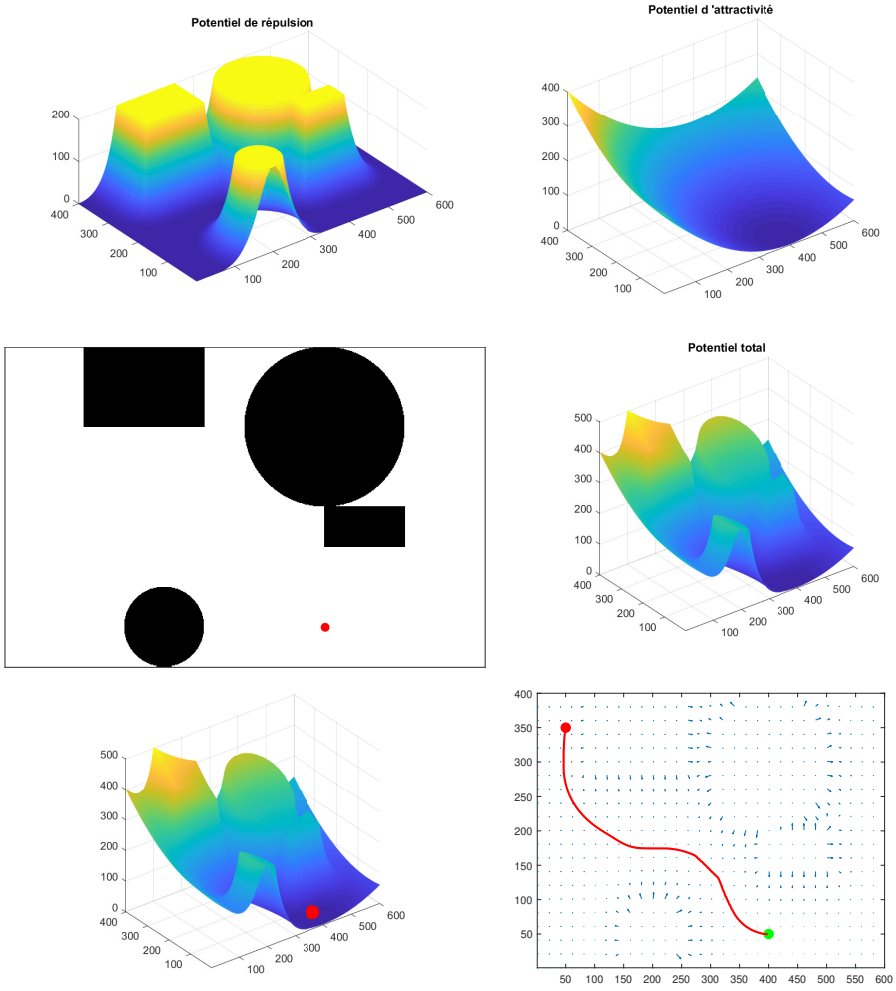


Figure 13.2: Résultat de simulation

13.3 la planification du parcours en utilisant la logique floue

Nous l'avons fait aussi la simulation de la planification du parcours on utilisant la logique floue.

```
%    Rahul Kala, IIIT Allahabad, Creative Commons Attribution–ShareAlike 4.0 International License.
% The use of this code, its parts and all the materials in the text; creation of derivatives and their publication; and sharing the code publically is permitted without permission.
% Please cite the work in all materials as: R. Kala (2014) Code for Robot Path Planning using Fuzzy Logic, Indian Institute of Information Technology Allahabad, Available at: http://rkala.in/codes.html

map=int16(im2bw(imread('map1.bmp'))); % input map read from a bmp file. for new maps write the file name here
source=[20 20]; % source position in Y, X format
goal=[480 480]; % goal position in Y, X format
robotDirection=pi/4; % initial heading direction
robotSize=[10 10]; %length and breadth
robotSpeed=10; % arbitrary units
maxRobotSpeed=10; % arbitrary units
S=10; % safety distance
distanceThreshold=30; % a threshold distace. points within this threshold can be taken as same.
maxAcceleration=10; % maximum speed change per unit time
directionScaling=60*pi/180; % fuzzy outputs to turn are restriect to –1 and 1. these are magnified here. maximum turn can be 60 degrees

%%%%% parameters end here %%%%%

fuz=readfis('fuzzyBase.fis'); % fuzzy inference system used. to read/edit use fuzzy(readfis('fuzzyBase.fis')) at the command line
distanceScaling=(size(map,1)^2+size(map,2)^2)^0.5; % all inputs are scaled by this number so that all distance inputs are between 0 and 1. maximum distance can be distanceScaling
currentPosition=source; % position of the centre of the robot
currentDirection=robotDirection; % direction of orientation of the robot
robotHalfDiagonalDistance=((robotSize(1)/2)^2+(robotSize(2)/2)^2)^0.5; % used for distance calculations
pathFound=false; % has goal been reached
prevTurn=0; % preffered turn at the previous time step, used for turning heuristic, see variable turn being set below.
prevDistanceLeftDiagonal=distanceScaling; % diagonal distance at the previous time step, used for tracking obstacles, used for turning heuristic, see variable turn being set below.
prevDistanceRightDiagonal=distanceScaling; % diagonal distance at the previous time step, used for tracking obstacles, used for turning heuristic, see variable turn being set below.
pathCost=0;
t=1;
imshow(map==1);
rectangle('position',[1 1 size(map)–1],'edgecolor','k');
pathLength=0;
if ~plotRobot(currentPosition,currentDirection,map,robotHalfDiagonalDistance)
    error('source lies on an obstacle or outside map');
end
M(t)=getframe;
t=t+1;

if ~feasiblePoint(goal,map), error('goal lies on an obstacle or outside map'); end

tic;
while ~pathFound

    % calculate distance from obstacle at front
    for i=robotSize(1)/2+1:distanceScaling
        x=int16(currentPosition+i*[sin(currentDirection) cos(currentDirection)]);
        if ~feasiblePoint(x,map), break; end
```

```
48      end
49      distanceFront=(i-robotSize(1)/2)/distanceScaling; % robotSize(1)/2 distance included in i was inside
            the robot body
50
51      % calculate distance from obstacle at front-left diagonal
52      for i=robotHalfDiagonalDistance+1:distanceScaling
53          x=int16(currentPosition+i*[sin(currentDirection-pi/4) cos(currentDirection-pi/4)]);
54          if ~feasiblePoint(x,map), break; end
55      end
56      distanceFrontLeftDiagonal=(i-robotHalfDiagonalDistance)/distanceScaling;
57
58      % calculate distance from obstacle at front-right diagonal
59      for i=robotHalfDiagonalDistance+1:distanceScaling
60          x=int16(currentPosition+i*[sin(currentDirection+pi/4) cos(currentDirection+pi/4)]);
61          if ~feasiblePoint(x,map), break; end
62      end
63      distanceFrontRightDiagonal=(i-robotHalfDiagonalDistance)/distanceScaling;
64
65      % calculate angle deviation to goal
66       slopeGoal=atan2(goal(1)-currentPosition(1),goal(2)-currentPosition(2));
67       angleGoal=slopeGoal-currentDirection;
68       while angleGoal>pi, angleGoal=angleGoal-2*pi; end % check to get the angle between -pi and pi
69       while angleGoal<-pi, angleGoal=angleGoal+2*pi; end % check to get the angle between -pi and pi
70       angleGoal=angleGoal/pi; % re-scaling the angle as per fuzzy modelling
71
72       % calculate diatnce from goal
73       distanceGoal=( sqrt(sum((currentPosition-goal).^2)) )/distanceScaling;
74       if distanceGoal*distanceScaling<distanceThreshold, pathFound=true; end
75
76       % calculate preferred turn.
77       % this indicates, if the front obstacle is far away, turn so as to more face the goal
78       % if the front obstacle is close and a new front obstacle is encountered, turn using the side of the
             goal is preferred
79       % if the front obstacle is close and the same obstacle as encountered in the previous step is found,
             same turn is made
80       if (prevTurn==0 || prevTurn==1) && distanceFront<0.1 && (distanceFrontLeftDiagonal-
             prevDistanceLeftDiagonal)*distanceScaling<maxRobotSpeed, turn=1;
81       elseif prevTurn==-1 && distanceFront<0.1 && (distanceFrontRightDiagonal-prevDistanceRightDiagonal)*
             distanceScaling<maxRobotSpeed, turn=-1;
82       else turn=(angleGoal>=0)*1+(angleGoal<0)*(-1);prevTurn=turn;
83       end
84       prevDistanceLeftDiagonal=distanceFrontRightDiagonal;
85       prevDistanceRightDiagonal=distanceFrontLeftDiagonal;
86
87       % pass all computed inputs to a fuzzy inference system
88       computedSteer=evalfis([distanceFront distanceFrontLeftDiagonal distanceFrontRightDiagonal angleGoal
             turn distanceGoal],fuz);
89       currentDirection=currentDirection+computedSteer*directionScaling;
90
91       % speed is set based on the front and diagonal distance so as not to make the robot collide, but make
             it slow and even stop before possible collission
92       % distances here include additional safety distance of S
93       distanceFrontSafety=max([distanceFront*distanceScaling-S 0]);
94       distanceFrontLeftDiagonalSafety=max([distanceFrontLeftDiagonal*distanceScaling-S 0]);
95       distanceFrontRightDiagonalSafety=max([distanceFrontRightDiagonal*distanceScaling-S 0]);
96
97       % maximum speeds admissible as per the above safety distance
98       maxSpeed1=min([sqrt(2*maxAcceleration*distanceFrontSafety) maxRobotSpeed]);
99       maxSpeed2=min([sqrt(maxAcceleration*distanceFrontLeftDiagonalSafety) maxRobotSpeed]);
100      maxSpeed3=min([sqrt(maxAcceleration*distanceFrontRightDiagonalSafety) maxRobotSpeed]);
101      maxSpeed=min([maxSpeed1 maxSpeed2 maxSpeed3]);
102
103      % setting the speed based on vehicle acceleration and speed limits. the vehicle cannot move backwards.
104      preferredSpeed=min([robotSpeed+maxAcceleration maxSpeed]);
105      robotSpeed=max([robotSpeed-maxAcceleration preferredSpeed]);
106      robotSpeed=min([robotSpeed maxRobotSpeed]);
```

```
107       robotSpeed=max([robotSpeed 0]);
108
109       if robotSpeed==0, error('robot had to stop to avoid collission'); end
110
111       % calculating new position based on steer and speed
112       newPosition=currentPosition+robotSpeed*[sin(currentDirection) cos(currentDirection)];
113       pathCost=pathCost+distanceCost(newPosition,currentPosition);
114       currentPosition=newPosition;
115       if ~feasiblePoint(int16(currentPosition),map), error('collission recorded'); end
116
117       % plotting robot
118       if ~plotRobot(currentPosition,currentDirection,map,robotHalfDiagonalDistance)
119          error('collission recorded');
120       end
121       M(t)=getframe;t=t+1;
122   end
123   fprintf('processing time=%d \nPath Length=%d \n\n', toc,pathCost);
```

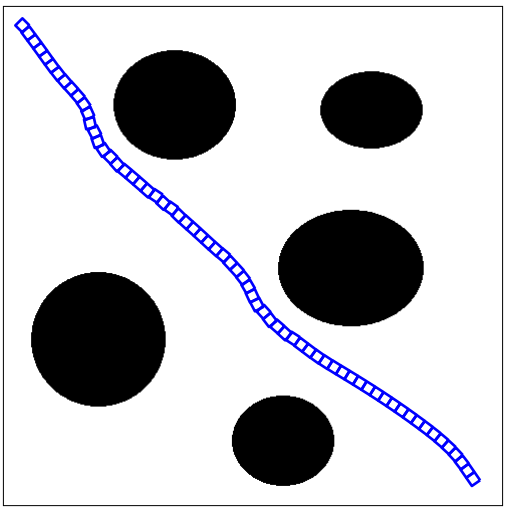


Figure 13.3: Résultat de simulation

13.4 la planification du parcours en utilisant l'algorithme de Dijkstra

Dans le même contenu, nous avons également faire la simulation de la planification du parcours on utilisant la methode de Dijkstra .

```
1   %% File Information
2   %! @author  Yash Bansod
3   %! @file    dijkstra.m
4   %! @date    15th September 2017
5   %! @breif   This demostrates the dijkstra path planning algorithm
6
7   %% Initialize the maps, color maps, start points and destination points
8   input_map = false(10);              % Create an Input Map
9   input_map (3:9, 5:7) = 1;           % Add an obstacle
```

```
start_coords = [6, 1];              % Save the location of start coordinate
dest_coords = [8, 9];               % Save location of destination coordinate
cmap = [1   1   1;                  % Create a color map
    0   0   0;
    1   0   0;
    0   0   1;
    0   1   0;
    1   1   0;
    0.5 0.5 0.5];
colormap(cmap);                     % Sets the colormap for the current figure
[nrows, ncols] = size(input_map);   % Save the size of the input_map

map = zeros(nrows,ncols);           % Create map to save the states of each grid cell
map(~input_map) = 1;                % Mark free cells on map
map(input_map) = 2;                 % Mark obstacle cells on map
start_node = sub2ind(size(map), start_coords(1), start_coords(2));      % Generate linear indices of start
    node
dest_node = sub2ind(size(map), dest_coords(1), dest_coords(2));         % Generate linear indices of dest
    node
map(start_node) = 5;                % Mark start node on map
map(dest_node) = 6;                 % Mark destination node on map

% Initialize distance from start array to inifinity
distanceFromStart = Inf(nrows,ncols);

% Create a map that holds the index of its parent for each grid cell
parent = zeros(nrows, ncols);

distanceFromStart(start_node) = 0;  % distance of start node is zero

image(1.5, 1.5, map);
grid on;                        % Display grid lines
axis image;                     % Set axis limits
drawnow;                        % Update figure

drawMapEveryTime = true;            % To see how nodes expand on the grid

%% Process the map to update the parent information and distance from start
while true                              % Create an infinite loop
    map(start_node) = 5;                % Mark start node on map
    map(dest_node) = 6;                 % Mark destination node on map

    if (drawMapEveryTime)
        image(1.5, 1.5, map);
        grid on;                        % Display grid lines
        axis image;                     % Set axis limits
        drawnow;                        % Update figure
    end

    % Find the node with the minimum distance
    [min_dist, current] = min(distanceFromStart(:));
    % Compute row, column coordinates of current node from linear index
    [i, j] = ind2sub(size(distanceFromStart), current);

    % Create an exit condition for the infinite loop to end
    if ((current == dest_node) || isinf(min_dist)) break
    end

    % Update distance value of element right of current element
    if (i+1 <= nrows && distanceFromStart(i+1, j) > min_dist + 1)
        if (parent(i+1, j) == 0 && input_map(i+1,j)~=1 && parent(current)~= sub2ind(size(map), i+1, j))
            distanceFromStart(i+1, j) = min_dist + 1;
            map(sub2ind(size(map), i+1, j)) = 4;    % Mark the neighbour of current as processing
            parent(i+1, j)= current;
        end
    end
```

```
74
75      % Update distance value of element left of current element
76      if (i-1 >= 1 && distanceFromStart(i-1, j) > min_dist + 1)
77          if (parent(i-1, j) == 0 && input_map(i-1,j)~=1 && parent(current)~= sub2ind(size(map), i-1, j))
78              distanceFromStart(i-1, j) = min_dist + 1;
79              map(sub2ind(size(map), i-1, j)) = 4;    % Mark the neighbour of current as processing
80              parent(i-1, j)= current;
81          end
82      end
83
84      % Update distance value of element top of current element
85      if (j-1 >= 1 && distanceFromStart(i, j-1) > min_dist + 1)
86          if (parent(i, j-1) == 0 && input_map(i,j-1)~=1 && parent(current)~= sub2ind(size(map), i, j-1))
87              distanceFromStart(i, j-1) = min_dist + 1;
88              map(sub2ind(size(map), i, j-1)) = 4;    % Mark the neighbour of current as processing
89              parent(i, j-1)= current;
90          end
91      end
92
93      % Update distance value of element bottom of current element
94      if (j+1 <= ncols && distanceFromStart(i, j+1) > min_dist + 1)
95          if (parent(i, j+1) == 0 && input_map(i,j+1)~=1 && parent(current)~= sub2ind(size(map), i, j+1))
96              distanceFromStart(i, j+1) = min_dist + 1;
97              map(sub2ind(size(map), i, j+1)) = 4;    % Mark the neighbour of current as processing
98              parent(i, j+1)= current;
99          end
100     end
101
102     distanceFromStart(current) = -log(0);   % change the distance of current from start as infinity
103     map(current) = 3;                       % mark the current point as processed
104 end
105
106 %% Construct route from start to dest by following the parent links
107 if (isinf(distanceFromStart(dest_node))) route = [];    % if distance to destination node is infinity
108 else route = [dest_node];                               % else backtrace the route from destination node
109     while (parent(route(1)) ~= 0)                       % check front of route for start node
110         route = [parent(route(1)), route];              % add parent of current node to front of route
111     end
112
113     for k = 2:length(route) - 1         % To visualize the map and the path
114         map(route(k)) = 7;
115         pause(0.001);                   % Pause the code for a while
116         image(1.5, 1.5, map);
117         grid on;                        % Display grid lines
118         axis image;                     % Set axis lengths
119     end
120 end
121
122 % Add legends to the colormapped image
123 hold on;
124 for K = 1:7
125     hidden_h(K) = surf(zeros(2, 2), 'edgecolor', 'none', ...
126         'facecolor', cmap(K, :));
127 end
128 hold off
129
130 uistack(hidden_h, 'bottom');
131 legend(hidden_h, {'Espace libre', 'obstacles', 'n uds  ferm s', ...
132     'n uds ouverts', 'n ud de d part', 'n ud de but', 'le plus court chemin'} )
```

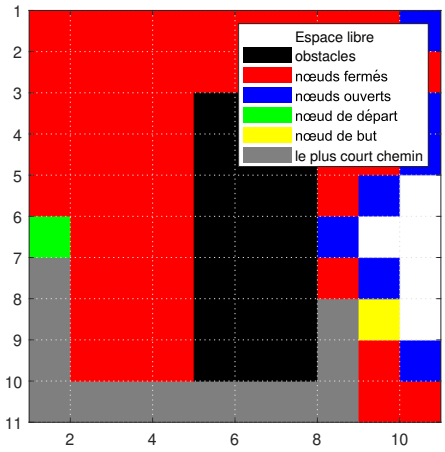


Figure 13.4: Résultat de simulation

13.5 la planification du parcours en utilisant l'algorithme de A*

Nous avons également abordé a la simulation de la planification du parcours on utilisant la methode de A*.

```
%%
%   Author: Yash Bansod
%
% GitHub: <https://github.com/YashBansod>

%% Clear the environment and the command line
clear;
close all;
clc;

%% Initialize the maps, color maps, start points and destination points
map_size = [20, 20];
input_map = false(map_size);              % Create an Input Map

input_map (6:7, 8:9) = 1;           % Add an obstacle
input_map (6:9, 12) = 1;            % Add another obstacle
input_map (12, 5:10) = 1;           % Add another obstacle
input_map (13:14, 13:16) = 1;       % Add another obstacle
input_map (16:18, 6:8) = 1;         % Add another obstacle
input_map (4:6, 17:18) = 1;         % Add another obstacle

start_coords = [3, 6];              % Save the location of start coordinate
dest_coords = [18, 15];             % Save location of destination coordinate

drawMapEveryTime = true;            % To see how nodes expand on the grid

cmap = [1   1   1;                  % Create a color map
    0   0   0;
    1   0   0;
    0   0   1;
```

```
    0   1   0;
    1   1   0;
    0.5 0.5 0.5];

colormap(cmap);                     % Sets the colormap for the current figure
[nrows, ncols] = size(input_map);   % Save the size of the input_map

map = zeros(nrows,ncols);           % Create map to save the states of each grid cell
map(~input_map) = 1;                % Mark free cells on map
map(input_map) = 2;                 % Mark obstacle cells on map

start_node = sub2ind(size(map), start_coords(1), start_coords(2));      % Generate linear indices of start
    node
dest_node = sub2ind(size(map), dest_coords(1), dest_coords(2));         % Generate linear indices of dest
    node

map(start_node) = 5;                % Mark start node on map
map(dest_node) = 6;                 % Mark destination node on map

distanceFromStart = Inf(nrows,ncols);   % Initialize distance from start array to inifinity
distanceFromEnd = Inf(nrows,ncols);     % Initialize distance from end array to inifinity

parent = zeros(nrows, ncols);           % Create a map for holding parent's index for each grid cell

distanceFromStart(start_node) = 0;  % distance of start node from start is zero
distanceFromEnd(dest_node) = 0;     % distance of end node from end is zero

% Update the values of all grid pixels for distance from end
[X, Y] = meshgrid(1:ncols, 1:nrows);
xd = dest_coords(1);
yd = dest_coords(2);
distanceFromEnd = abs(X - yd) + abs(Y - xd);    % Manhattan Distance

image([0.5, map_size(1)-0.5], [0.5, map_size(2)-0.5], map);
ax = gca;
ax.YTick = 0:1:map_size(1);
ax.XTick = 0:1:map_size(2);
grid on;                        % Display grid lines
% drawnow limitrate nocallbacks;  % Update figure
drawnow;

%% Process the map to update the parent information and distance from start
while true                              % Create an infinite loop
    map(start_node) = 5;                % Mark start node on map
    map(dest_node) = 6;                 % Mark destination node on map

    if (drawMapEveryTime)
        image([0.5, map_size(1)-0.5], [0.5, map_size(2)-0.5], map);
        ax = gca;
        ax.YTick = 0:1:map_size(1);
        ax.XTick = 0:1:map_size(2);
        grid on;                        % Display grid lines
        % drawnow limitrate nocallbacks;  % Update figure
        drawnow;
    end

    % Find the node with the minimum heuristic distance
    heuristicDist = distanceFromStart + distanceFromEnd;
    [min_dist, current] = min(heuristicDist(:));

    % Compute row, column coordinates of current node from linear index
    [i, j] = ind2sub(size(heuristicDist), current);

    % Create an exit condition for the infinite loop to end
    if ((current == dest_node) || isinf(min_dist))
        break
```

```
    end

    % Update distance value of element right of current element
    if (i+1 <= nrows && distanceFromStart(i+1, j) > distanceFromStart(i,j) + 1)
        if (parent(i+1, j) == 0 && input_map(i+1,j)~=1 && parent(current)~= sub2ind(size(map), i+1, j))
            distanceFromStart(i+1, j) = distanceFromStart(i,j) + 1;
            map(sub2ind(size(map), i+1, j)) = 4;    % Mark the neighbour of current as processing
            parent(i+1, j)= current;
        end
    end

    % Update distance value of element left of current element
    if (i-1 >= 1 && distanceFromStart(i-1, j) > distanceFromStart(i,j) + 1)
        if (parent(i-1, j) == 0 && input_map(i-1,j)~=1 && parent(current)~= sub2ind(size(map), i-1, j))
            distanceFromStart(i-1, j) = distanceFromStart(i,j) + 1;
            map(sub2ind(size(map), i-1, j)) = 4;    % Mark the neighbour of current as processing
            parent(i-1, j)= current;
        end
    end

    % Update distance value of element top of current element
    if (j-1 >= 1 && distanceFromStart(i, j-1) > distanceFromStart(i,j) + 1)
        if (parent(i, j-1) == 0 && input_map(i,j-1)~=1 && parent(current)~= sub2ind(size(map), i, j-1))
            distanceFromStart(i, j-1) = distanceFromStart(i,j) + 1;
            map(sub2ind(size(map), i, j-1)) = 4;    % Mark the neighbour of current as processing
            parent(i, j-1)= current;
        end
    end

    % Update distance value of element bottom of current element
    if (j+1 <= ncols && distanceFromStart(i, j+1) > distanceFromStart(i,j) + 1)
        if (parent(i, j+1) == 0 && input_map(i,j+1)~=1 && parent(current)~= sub2ind(size(map), i, j+1))
            distanceFromStart(i, j+1) = distanceFromStart(i,j) + 1;
            map(sub2ind(size(map), i, j+1)) = 4;    % Mark the neighbour of current as processing
            parent(i, j+1)= current;
        end
    end

    distanceFromStart(current) = -log(0);   % change the distance of current from start as infinity
    map(current) = 3;                       % mark the current point as processed
end

%% Construct route from start to dest by following the parent links
if (isinf(distanceFromStart(dest_node)))
    route = [];    % if distance to destination node is infinity
else
    route = [dest_node];                              % else backtrace the route from destination node
    while (parent(route(1)) ~= 0)                      % check front of route for start node
        route = [parent(route(1)), route];             % add parent of current node to front of route
    end

    for k = 2:length(route) - 1         % To visualize the map and the path
        map(route(k)) = 7;
        pause(0.001);                   % Pause the code for a while
        image([0.5, map_size(1)-0.5], [0.5, map_size(2)-0.5], map);
        ax = gca;
        ax.YTick = 0:1:map_size(1);
        ax.XTick = 0:1:map_size(2);
        grid on;                        % Display grid lines
        % drawnow limitrate nocallbacks;  % Update figure
        drawnow;
    end
end

% Add legends to the colormapped image
hold on;
```

```
for K = 1:7
    hidden_h(K) = surf(zeros(2, 2), 'edgecolor', 'none', ...
        'facecolor', cmap(K, :));
end
hold off

uistack(hidden_h, 'bottom');
legend(hidden_h, {'Espace libre', 'obstacles', 'n uds ferm s', ...
    'n uds ouverts', 'n ud de d part', 'n ud de but', 'le plus court chemin'} )
```

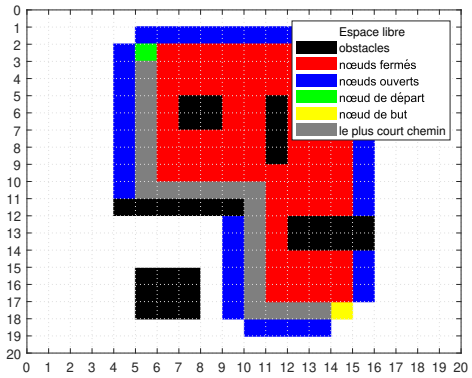


Figure 13.5: Résultat de simulation

Quatrième partie IV

Étude Pratique

Chapitre 14

La Réalisation

14.1 Le Matériel

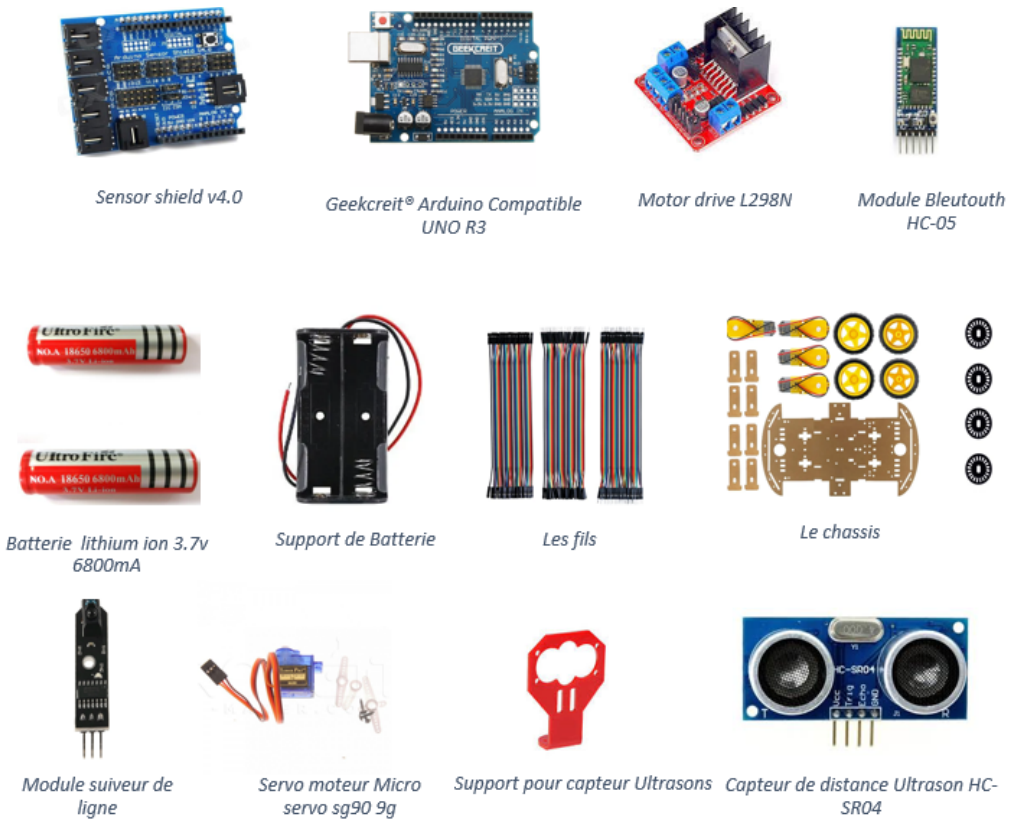


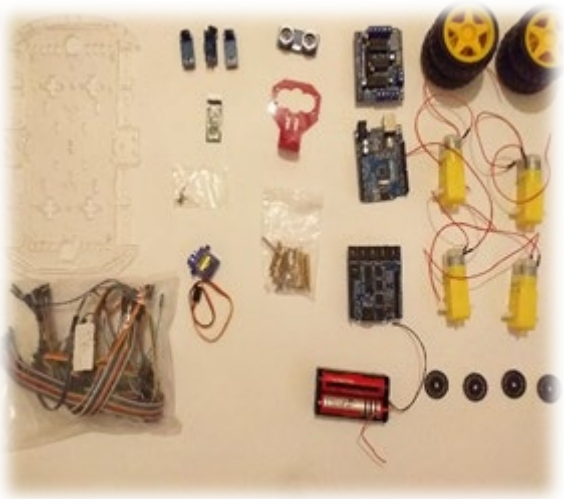
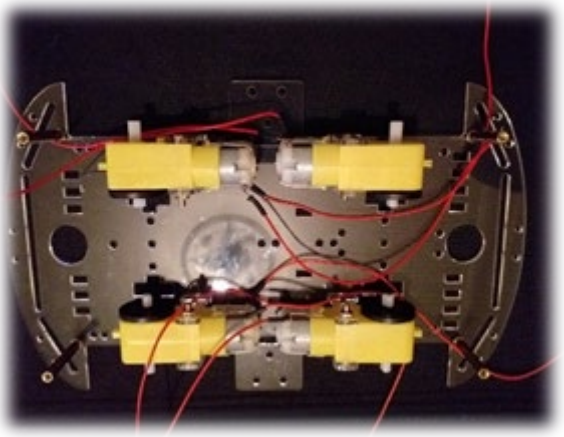
Figure 14.1: Le Matériel utilisé

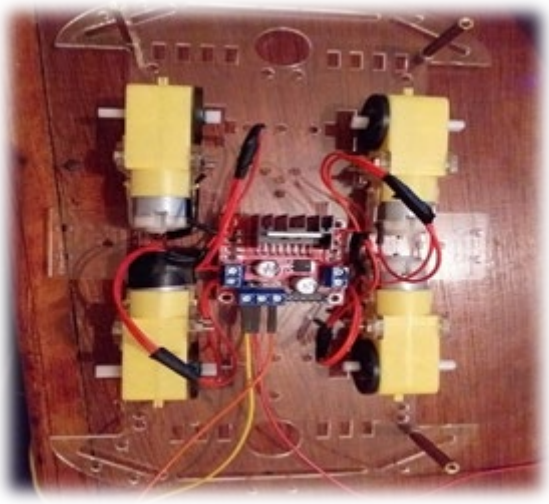
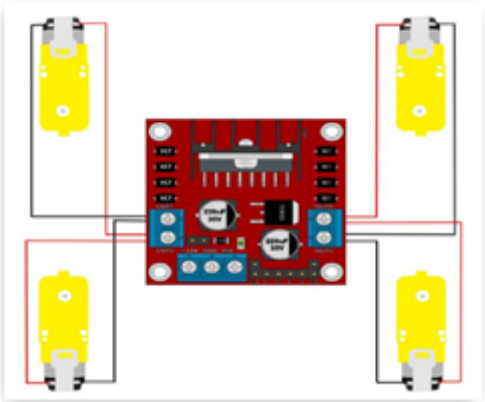
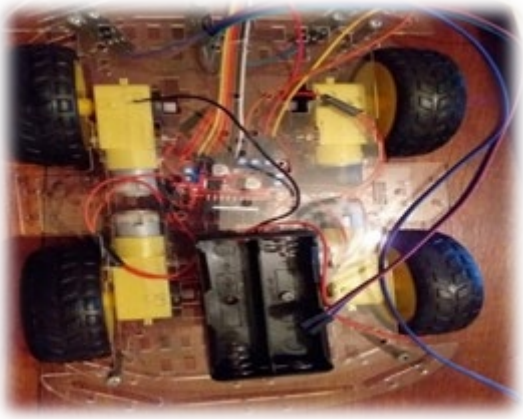
14.2 Branchements des différentes pièces du robot

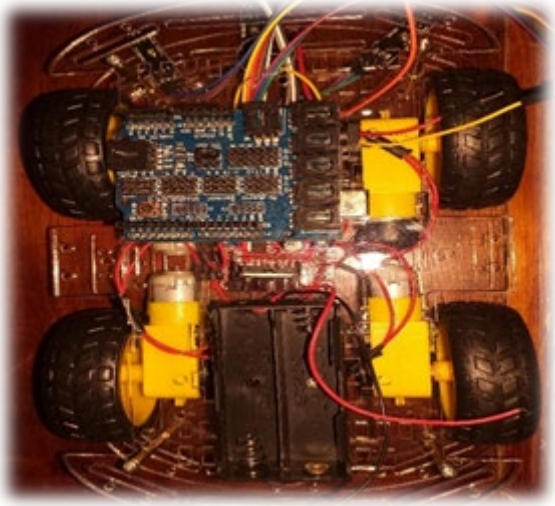
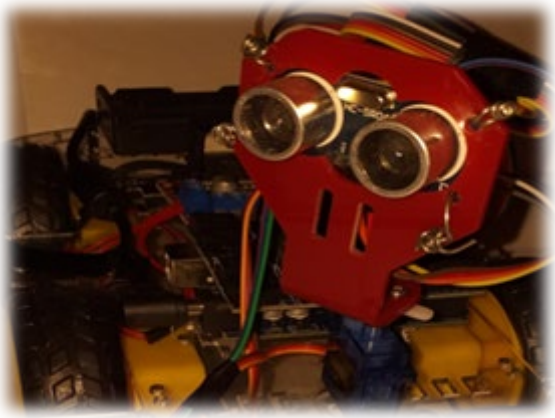
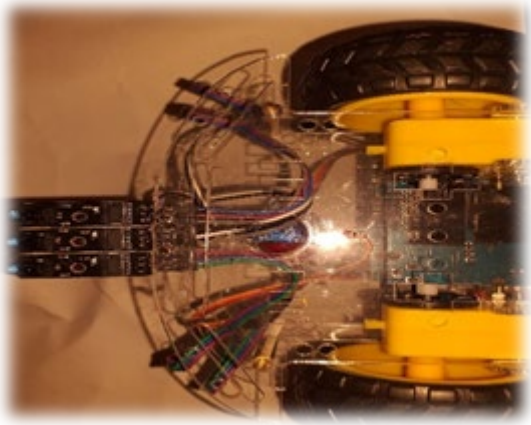
On va maintenant présenter le branchement des différentes pièces du robot (l'ensemble du matériel éxposé dans la section 14.1).

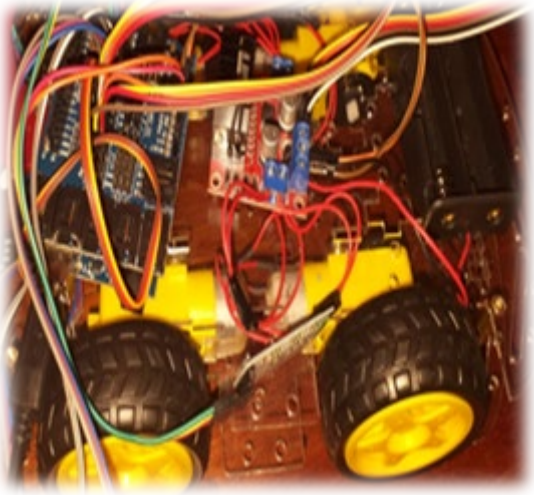
```
Capteur ultrason
  * Vcc = 5V sur l Arduino
  * Trig = pin 12 du Arduino
  * Echo = pin 13 du Arduino
  * Gnd = Gnd sur l Arduino

Bluetooth
  * Vcc = 5V sur l Arduino
  * RX = pin 1 (TX) de l Arduino
  * TX = pin 0 (RX) de l Arduino
  * Gnd = Gnd de l Arduino

Capteur de couleur gauche
  * Vcc = 5V sur l Arduino
  * Out = pin 7 de l Arduino
  * Gnd = Gnd sur l Arduino

Capteur de couleur du milieu
  * Vcc = 5V sur l Arduino
  * Out = pin 4 de l Arduino
  * Gnd = Gnd sur l Arduino

Capteur de couleur droit
  * Vcc = 5V sur l Arduino
  * Out = pin 3 de l Arduino
  * Gnd = Gnd sur l Arduino

Drive de moteur L298N
  * ENA = 5V sur l Arduino
  * ENB = 5V sur l Arduino
  * IN1 = pin 5 de l Arduino
  * IN2 = pin 6 de l Arduino
  * IN3 = pin 10 de l Arduino
  * IN4 = pin 11 de l Arduino

  * Connecteur a vis MOTORA = Placez la carte de fa on a lire le nom des connecteurs a l endroit et
      branchez les deux fils rouge des moteurs gauche du cote droit du connecteur et les deux fils noirs
      du cote gauche.
  * Connecteur a vis MOTORB = Placez la carte de fa on a lire le nom des connecteurs a l endroit et
      branchez les deux fils rouge des moteurs droit du cote droit du connecteur et les deux fils noirs du
       cote gauche.
  * Connecteur a vis VMS = fil denude rouge du socle pour batteries
  * Connecteur a vis Gnd = fil denude noir du socle pour batteries
  * On ne connecte rien sur les pins 5V de la drive de moteur

Socle pour batteries
  * Connecteur = se branche dans le connecteur correspondant sur l Arduino
```

14.3 Installation Matérielle

Etape 1 : *Pour commencer, nous allons présenter le matériel utilisé pour la réalisation du robot.*	
Etape 2 : *Tout d'abord, fixez les quatre supports plastiques (ce sont des blocs Circulaire en noire) à chaque moteur à l'aide de deux longs boulons et de deux écrous.*	
Etape 3 : *Maintenant, chaque moteur peut être fixé au châssis en utilisant deux boulons courts dans le bas de chaque support plastique. Voici une vue de la partie inférieure du châssis afin que vous puissiez voir où les boulons doivent être placés.*	

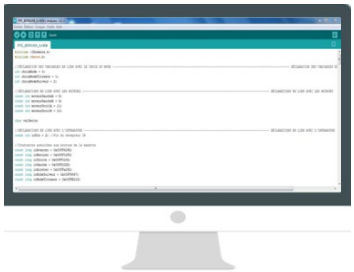
Etape 4 : *installer la plaque d'entraînement du moteur (motor drive L298N) : fixer la carte de commande du moteur (la carte rouge) au châssis.* *Remarque : il est recommandé d'attendre que tous les fils soient fixés au pont en H avant de procéder à cette opération.*	
Etape 4 : *Connectez le câble moteur, diagramme de circuit.*	
Etape 5 : *câblage de l'alimentation (boîte à batterie) sur la partie suppérieur du chassis.* *Fixation des roues : faites attacher les roues sur les arbres du moteur et fixez les quatre arbres aux endroits indiqués sur la photo ci-dessous Ce robot commence à prendre forme !*	

Etape 6 : *Attachez l'Arduino avec le sheild v4.0 : Maintenant, mettez de côté la partie précédente du châssis et attrapez l'autre qui se trouvera au-dessus. Et mieux fixer le vôtre avec quelques boulons et écrous.*	
Etape 7 : *installer le servo moteur et le capteur d'ultrason.*	
Etape 8 : *Installer les 3 modules suiveurs de ligne.*	

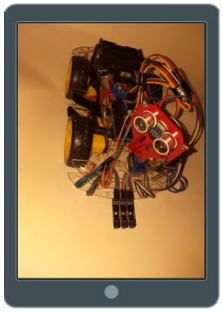
Etape 9 : *Installer le module bleutooth.et monter le moteur drive L298N a la partie supérieure pour faciliter le travaile.*	
Etape 10 : *Forme finale après avoir terminé l'assemblage.*	

14.4 Schéma de fonctionnement

Le programme est prêt à etre televersé au robot.



Le programme est televersé a la carte arduino et donc au robot



L'application « BlueTooth Serial Controller » permet le command du robot

Rq : le programme est le code arduino ecrit dans le chapitre 15.

14.5 Résultats et Interprétations

1 Le test du mode avancé :

- L'appui sur mode avancé permet le robot de parcouru un chemin d'une point de départ vers la point final et il continu sur le chemin si nous ne changeons le mode sélectionné.

2 Le test du mode reculé :

- L'appui sur mode reculé permet le robot de parcouru un chemin d'une point de départ vers la point final et il continu sur le chemin si nous ne changeons le mode sélectionné. (Dans notre exemple le point de départ est la point final de l'exemple précèdent)

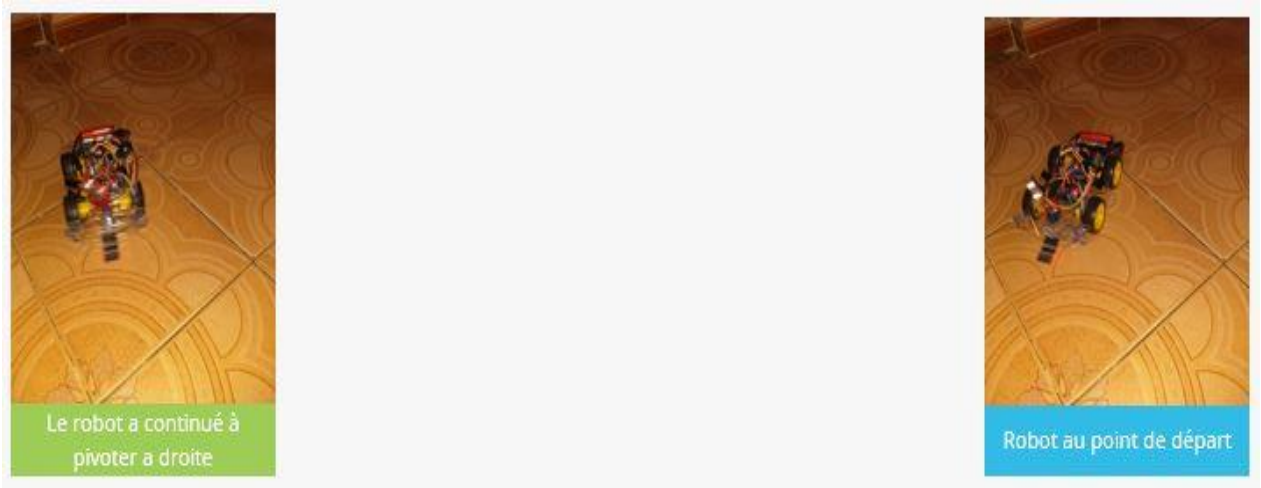
3 Le test du mode droite :

- La sélectionne du mode droite permet le robot de tourné à droit (Just les 2 roues droites qui tourne pour assurer ce mode).

4 Le test du mode gauche :

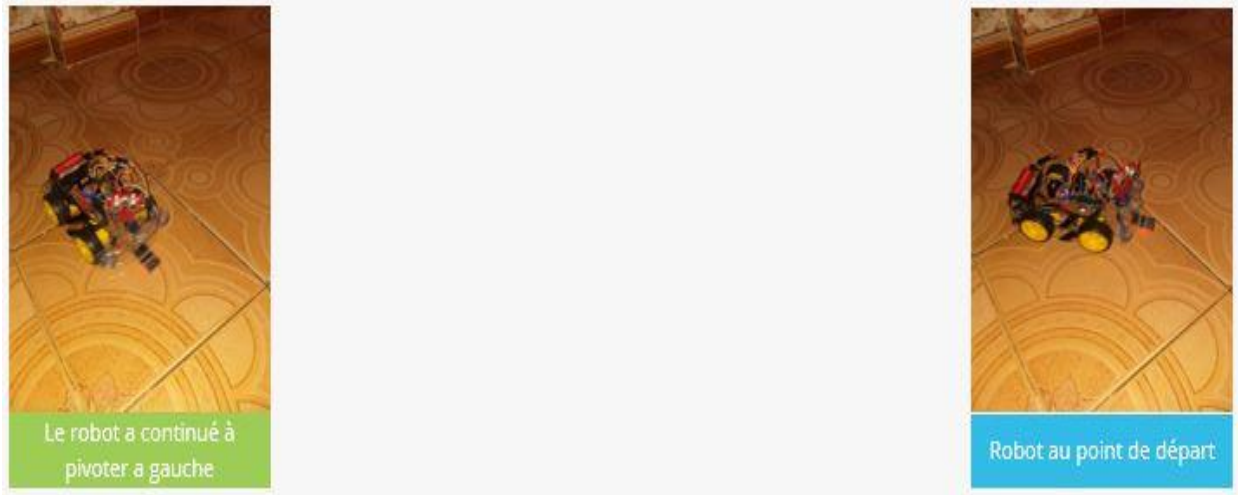
- La sélectionne du mode gauche permet le robot de tourné à gauche (Just les 2 roues gauches qui tourne pour assurer ce mode).

5 Le test du mode pivoté à droite :



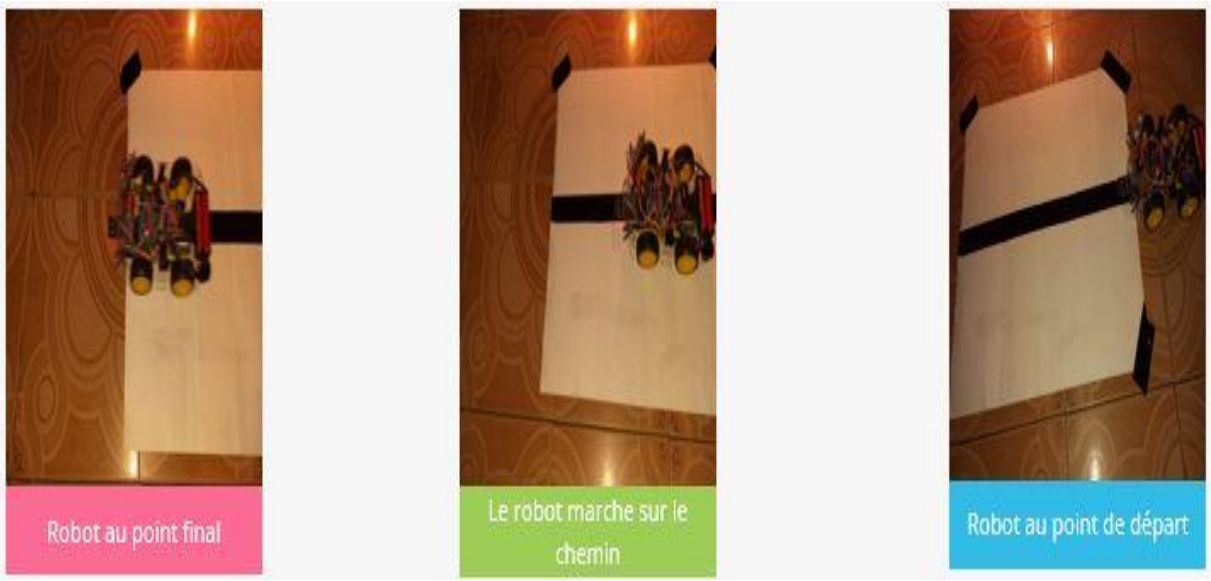
- Le mode pivoté a droite permet le robot de faire une tourné e à droite (les 4 roues fonctionnent pour assurer ce mode).

6 Le test du mode pivoté à gauche :



- Le mode pivoté a gauche permet le robot de faire une tourné e à gauche (les 4 roues fonctionnent pour assurer ce mode).

7 Le test du mode suiveur de chemin :

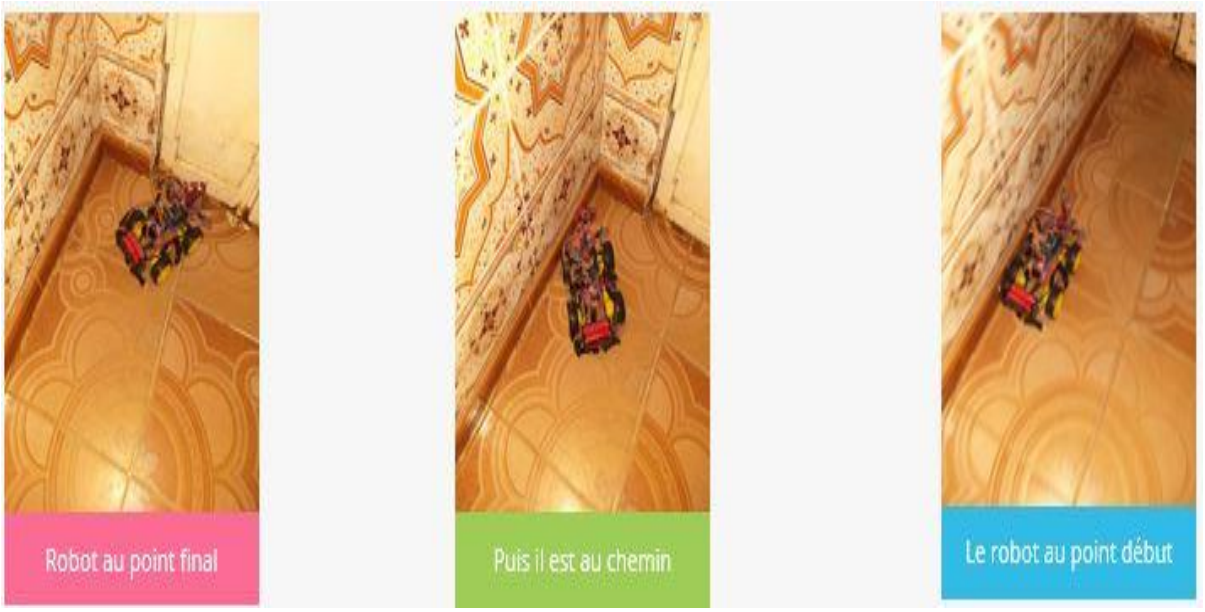


- Le mode suiveur de ligne ou de chemin permet de suivre une ligne noire désiré.

8 Le test du mode ultrason (détecteur d'obstacle) :

- Le mode ultrason permet le robot de détecter et éviter un obstacle statique, lorsque ce mode n'est pas activé : Le robot entre en collision avec des objets.

9 Le test du mode ultrason (détecteur d'obstacle) :



- Le mode ultrason est activé : le robot va éviter les obstacles et donc on aboutira au but qu'il y a un évitement de collision.

Chapitre 15

La Simulation

15.1 le code Arduino

```
#include <IRremote.h>
#include <Servo.h>

%% D CLARATION DES VARIABLES EN LIEN AVEC LE CHOIX DU MODE
int choixMode = 0;
int choixModeUltrason = 1;
int choixModeSuiveur = 2;

%%DECLARATIONS EN LIEN AVEC LES MOTEURS
const int moteurGaucheA = 5;
const int moteurGaucheR = 6;
const int moteurDroitA = 11;
const int moteurDroitR = 10;

char valSerie;

//DECLARATIONS EN LIEN AVEC L'INFRAROUGE
const int irPin = 2; //Pin du r cepteur IR

//Constantes associ es aux boutons de la manette
const long irAvancer = 0x00FF629D;
const long irReculer = 0x00FF02FD;
const long irDroite = 0x00FFC23D;
const long irGauche = 0x00FF22DD;
const long irArreter = 0x00FFa25D;
const long irModeSuiveur = 0x00FF9867;
const long irModeUltrason = 0x00FFE21D;

IRrecv irrecv(irPin); //Objet recevant les signaux    d coder
decode_results irResultat; //Contient le r sultat du d codage

//DECLARATIONS EN LIEN AVEC LE CAPTEUR DE DISTANCE ET LE SERVOMOTEUR
const int echoPin = 13;
const int triggerPin = 12;

```

```
36  int distanceAvant;
37  int distanceDroite;
38  int distanceGauche;
39  float intervalle;
40
41  Servo myservo; //Creation de l'objet servo
42  const int delay_time = 250; //Temps accorde au servo pour la mesure de la distance de chaque  c  t
43
44  //DECLARATIONS EN LIEN AVEC LES CAPTEURS DE LUMINOSIT
45  const int capteurGauche = 7;
46  const int capteurCentre = 4;
47  const int capteurDroit = 3;
48  bool etatCapteurGauche;
49  bool etatCapteurCentre;
50  bool etatCapteurDroit;
51
52  //SETUP
53  void setup()
54  {
55    //Initialisation de la communication serie
56    Serial.begin(9600);
57
58    //Initialisation des broches des moteurs
59    pinMode(moteurGaucheA, OUTPUT);
60    pinMode(moteurGaucheR, OUTPUT);
61    pinMode(moteurDroitA, OUTPUT);
62    pinMode(moteurDroitR, OUTPUT);
63    digitalWrite(moteurGaucheA, LOW);
64    digitalWrite(moteurGaucheR, LOW);
65    digitalWrite(moteurDroitA, LOW);
66    digitalWrite(moteurDroitR, LOW);
67
68    //Initialisation du decodage infrarouge
69    irrecv.enableIRIn();
70
71    //Initialisation du capteur de distance et du servomoteur
72    pinMode(echoPin, INPUT);
73    pinMode(triggerPin, OUTPUT);
74    myservo.attach(9);
75    myservo.write(90);
76
77    //Initialisation des capteurs de couleur
78    pinMode(capteurGauche, INPUT);
79    pinMode(capteurCentre, INPUT);
80    pinMode(capteurDroit, INPUT);
81  }
82
83  //FONCTIONS POUR LES MOTEURS
84  void avancer()
85  {
86    digitalWrite(moteurGaucheA, HIGH);
87    digitalWrite(moteurGaucheR, LOW);
88    digitalWrite(moteurDroitA, HIGH);
89    digitalWrite(moteurDroitR, LOW);
90  }
91
92  void gauche()
93  {
94    digitalWrite(moteurGaucheA, LOW);
95    digitalWrite(moteurGaucheR, LOW);
96    digitalWrite(moteurDroitA, HIGH);
97    digitalWrite(moteurDroitR, LOW);
98  }
99  void droite()
100 {
101   digitalWrite(moteurGaucheA, HIGH);
```

```
102   digitalWrite(moteurGaucheR, LOW);
103   digitalWrite(moteurDroitA, LOW);
104   digitalWrite(moteurDroitR, LOW);
105 }
106 void reculer()
107 {
108   digitalWrite(moteurGaucheA, LOW);
109   digitalWrite(moteurGaucheR, HIGH);
110   digitalWrite(moteurDroitA, LOW);
111   digitalWrite(moteurDroitR, HIGH);
112 }
113 void arreter()
114 {
115   digitalWrite(moteurGaucheA, LOW);
116   digitalWrite(moteurGaucheR, LOW);
117   digitalWrite(moteurDroitA, LOW);
118   digitalWrite(moteurDroitR, LOW);
119 }
120 void pivoterGauche()
121 {
122   digitalWrite(moteurGaucheA, LOW);
123   digitalWrite(moteurGaucheR, HIGH);
124   digitalWrite(moteurDroitA, HIGH);
125   digitalWrite(moteurDroitR, LOW);
126 }
127 void pivoterDroite()
128 {
129   digitalWrite(moteurGaucheA, HIGH);
130   digitalWrite(moteurGaucheR, LOW);
131   digitalWrite(moteurDroitA, LOW);
132   digitalWrite(moteurDroitR, HIGH);
133 }
134
135 //FONCTIONS POUR LE MODE ULTRASON
136 void mesurerDistanceAvant()
137 {
138   myservo.write(90);
139   delay(delay_time);
140   digitalWrite(triggerPin, LOW);
141   delayMicroseconds(2);
142   digitalWrite(triggerPin, HIGH); //Envoie d'une onde sonore
143   delayMicroseconds(10);
144   digitalWrite(triggerPin, LOW);
145   intervalle = pulseIn(echoPin, HIGH); //Reception de l'echo
146   intervalle = intervalle/5.8/10; //Conversion de la difference de temps entre l'envoie de l'onde sonore et
        la reception de l' cho  en distance (cm)
147   Serial.println("Distance avant:");
148   Serial.println(intervalle);
149   distanceAvant = intervalle; //Arrondissement de la distance
150 }
151
152 void mesurerDistanceGauche()
153 {
154   myservo.write(180);
155   delay(delay_time);
156   digitalWrite(triggerPin, LOW);
157   delayMicroseconds(2);
158   digitalWrite(triggerPin, HIGH); //Envoie d'une onde sonore
159   delayMicroseconds(10);
160   digitalWrite(triggerPin, LOW);
161   intervalle = pulseIn(echoPin, HIGH); //Reception de l'echo
162   intervalle = intervalle/5.8/10; //Conversion de la difference de temps entre l'envoie de l'onde sonore et
        la reception de l' cho  en distance (cm)
163   Serial.println("Distance gauche:");
164   Serial.println(intervalle);
165   distanceGauche = intervalle;
```

```
166 }
167
168 void mesurerDistanceDroite()
169 {
170   myservo.write(0);
171   delay(delay_time);
172   digitalWrite(triggerPin, LOW);
173   delayMicroseconds(2);
174   digitalWrite(triggerPin, HIGH); //Envoie d'une onde sonore
175   delayMicroseconds(10);
176   digitalWrite(triggerPin, LOW);
177   intervalle = pulseIn(echoPin, HIGH); //Reception de l'echo
178   intervalle = intervalle/5.8/10; //Conversion de la difference de temps entre l'envoie de l'onde sonore et
        la reception de l' cho  en distance (cm)
179   Serial.println("Distance droite:");
180   Serial.println(intervalle);
181   distanceDroite = intervalle;
182 }
183
184 void modeUltrason()
185 {
186   mesurerDistanceAvant();
187
188   if(distanceAvant < 25) //Si la distance avant est de moins de 25cm
189   {
190     arreter();
191     mesurerDistanceGauche();
192     delay(delay_time);
193     mesurerDistanceDroite();
194     delay(delay_time);
195
196     if(distanceGauche < 15 && distanceDroite < 15) //Si la distance    gauche et la distance    droite sont
          de moins de 15cm
197     {
198       reculer();
199       delay(500);
200       pivoterGauche();
201       delay(250);
202     }
203     else if(distanceGauche > distanceDroite) //Si la distance gauche est plus grande que la distance droite
204     {
205       pivoterGauche();
206       delay(250);
207     }
208     else if(distanceGauche <= distanceDroite) //Si la distance gauche est plus petite ou  gale     la
          distance droite
209     {
210       pivoterDroite();
211       delay(250);
212     }
213   }
214   else //Si la distance avant est de plus de 25cm
215   {
216     avancer();
217   }
218 }
219
220 void commandeSerie()
221 {
222   if(Serial.available())
223   {
224     valSerie = Serial.read();
225
226     choixMode = 0;
227
228     if(valSerie == 'w')
```

```
229     avancer();
230     else if(valSerie == 'a')
231     gauche();
232     else if(valSerie == 's')
233     reculer();
234     else if(valSerie == 'd')
235     droite();
236     else if(valSerie == 'q')
237     arreter();
238     else if(valSerie == 'z')
239     pivoterGauche();
240     else if(valSerie == 'x')
241     pivoterDroite();
242     else if(valSerie == '1')
243     choixMode = choixModeUltrason;
244     else if(valSerie == '2')
245     choixMode = choixModeSuiveur;
246   }
247 }
248
249 void modeSuiveur()
250 {
251   etatCapteurGauche = digitalRead(capteurGauche);
252   etatCapteurCentre = digitalRead(capteurCentre);
253   etatCapteurDroit = digitalRead(capteurDroit);
254
255   if(etatCapteurCentre) //Si le capteur du centre d tecte du noir
256   {
257     if ((etatCapteurGauche) && (!etatCapteurDroit)) //S'il y a du noir    gauche et du blanc    droite,
          tourner    gauche
258     gauche();
259     else if ((!etatCapteurGauche) && (etatCapteurDroit)) //S'il y a du blanc    gauche et du noir    droite
          , tourner    droite
260     droite();
261     else //Si les conditions plus haut ne s'appliquent pas, continuer tout droit
262     avancer();
263   }
264   else //Si le capteur du centre d tecte du blanc
265   {
266     if ((etatCapteurGauche) && (!etatCapteurDroit)) //S'il y a du noir    gauche et du blanc    droite,
          tourner    gauche
267     gauche();
268     else if ((!etatCapteurGauche) && (etatCapteurDroit)) //S'il y a du blanc    gauche et du noir    droite
          , tourner    droite
269     droite();
270     else //Si les conditions plus hautdne s'appliquent pas, reculer
271     reculer();
272   }
273 }
274
275 void lireInfrarouge()
276 {
277   if (irrecv.decode(&irResultat)) //Reception d'un signal IR
278   {
279     irrecv.resume(); //Accepter les nouveaux resultats IR
280
281     if((irResultat.value < 0xFFF000) && (irResultat.value > 0xFF0000)) //Filtrage des mauvaises d tections
           IR
282     {
283       choixMode = 0;
284
285       if (irResultat.value == irAvancer) //Avancer
286       avancer();
287       else if (irResultat.value == irReculer) //Reculer
288       reculer();
289       else if (irResultat.value == irDroite) //Tourner a droite
```

```
290       droite();
291       else if (irResultat.value == irGauche) //Tourner a gauche
292       gauche();
293       else if (irResultat.value == irArreter) //Arreter
294       arreter();
295       else if(irResultat.value == irModeSuiveur)
296       choixMode = choixModeSuiveur;
297       else if(irResultat.value == irModeUltrason)
298       choixMode = choixModeUltrason;
299
300       irResultat.value = 0;
301     }
302   }
303 }
304
305 //BOUCLE
306 void loop()
307 {
308   commandeSerie();
309   lireInfrarouge();
310
311   if(choixMode == choixModeUltrason)
312   {
313     modeUltrason();
314   }
315   else if(choixMode == choixModeSuiveur)
316   {
317     modeSuiveur();
318   }
319 }
```

Conclusions

Ce projet a développé un robot d'évitement d'obstacles pour détecter et éviter les obstacles sur son chemin. Le robot est construit sur la plate-forme Arduino pour le traitement des données et son homologue logiciel a permis de communiquer avec le robot pour lui envoyer des paramètres de guidage des mouvements. Pour la détection des obstacles, un capteur de distance à ultrasons a été utilisé, ce qui a permis d'élargir le champ de détection. Le robot est entièrement autonome et après le chargement initial du code, il ne nécessite aucune intervention de l'utilisateur pendant son fonctionnement. Lorsqu'il est placé dans un environnement inconnu avec des obstacles, L'ajout d'un dispositif Bluetooth peut également permettre de passer d'un mode de contrôle à un autre à distance. il se déplace en évitant tous les obstacles avec une précision considérable. Le travail effectué dans le cadre de ce projet peut servir de base à d'autres améliorations visant à accroître la précision et l'adaptabilité de la détection d'obstacles dans divers environnements. À l'avenir, ce projet vise à tester la faisabilité de l'intégration de différents types de capteurs pour compléter les inconvénients de chacun.et de travailler avec 2 robots mobiles (donc l'obstacle devient dynamique).

Bibliographie

[1] D.K.Pratihar, "Lecture 01 : Introduction to robots and robotics." https://rb.gy/tzaodu, 2018.

[2] L. Zineb, "la robotique mobile les différents méthodes de navigation pour un robot mobile," 2013.

[3] A. Pepe and C. Melchiorri, "Introduction to Mobile Robotics Course of Industrial Robotics M,"

[4] "Introduction to Robotics,section Knowledgebase for Robotics," *BJU International*, vol. 108, no. 6 B, pp. 1018–1023, 2011.

[5] B. Marr, "The 4 ds of robotization : Dull, dirty, dangerous and dear." https://rb.gy/gvgngv, 2017.

[6] sbah, "Microsoft Robotics Studio Developer," 2012.

[7] P. Smedley, "Robot workers : Whom will automation replace ?." https://rb.gy/dznsgg„ 2019.

[8] A. Russo, "Automatisation des processus par la robotique." https://rb.gy/9e3b6j„ 2020.

[9] G. I. of Technology, "How would you like your assistant—human or robotic ?." https://rb.gy/9tjwf9„ 2013.

[10] J. Fitzgerald, "Using autonomous robots to drive supply chain innovation." https://rb.gy/ycys8c„ 2020.

[11] J.-L. Boimond, "ROBOTIQUE," pp. 1–85.

[12] R. Hughes, *Introduction To Robotics Analysis,Control,Applications*, vol. 53. 2008.

[13] J. Caro, "Qu'est-ce qu'un robot ? apprendre à connaître les capteurs et les actionneurs." https://rb.gy/rtipfs„ 2016.

[14] H. A.-F. BERKANI, "Design & Construction of mobile robot with a manipulator arm,a thesis submitted in partial fulfillment for the academic degree requirements of : "master in electronics of embedded systems"," no. July, 2019.

[15] C. F. P. (CFP), "Type of robots." `https://rb.gy/dnz68b`„ 2016.

[16] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, *Introduction to Autonomous Mobile robots Second Edition*. MIT press, 2011.

[17] Dr.C.A.Berry, "Ece425 : Mobile robotics course." `https://rb.gy/sv8b4g`„ 2018.

[18] R. Siegwart and I. Nourbakhsh, "Sensors for Mobile Robots," pp. 79–158.

[19] S. G. Tzafestas, *Introduction to Mobile Robot Control*.

[20] Robin.R.Murphy, *Introduction to AI Robotics*.

[21] A. Koubaa, H. Bennaceur, I. Chaari, S. Trigui, A. Ammar, M.-F. Sriti, M. Alajlan, O. Cheikhrouhou, and Y. Javed, *Robot Path Planning and Cooperation Foundations, Algorithms and Experimentations*, vol. XXII. 2018.

[22] "Planning and Navigation : Where am I going ? How do I get there ?," pp. 231–274.

[23] A. K. Guruji, H. Agarwal, and D. K. Parsediya, "Time-Efficient A * Algorithm for Robot Path Planning," vol. 23, pp. 144–149, 2016.