REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

École Nationale Polytechnique

المدرسة الوطنية المتعددة التقنيات
Ecole Nationale Polytechnique

مخبر التحكم في العمليات التصنيعية
Laboratoire de Commande des Processus

Département d'Automatique

Laboratoire de Commande des Processus

Mémoire de projet de fin d'études en vue de
l'obtention du diplôme d'Ingénieur d'état en Automatique

Optimisation par Reinforcement Learning et implémentation d'une technique V-SLAM(2D) sur un robot mobile

Réalisé par :

Oussama DEROUICHE
El Hacene CHABANE

Présenté et Soutenu publiquement le 24 Juin 2018 devant le jury composé de :

Président	R. ILLOUL	Maître de Conférences	ENP Alger
Encadreur	M. TADJINE	Professeur	ENP Alger
Promoteur	Z. MOUSSAOUI	Ingénieur agrégé	Airbus Groupe-ECE
Examinateur	B. HEMICI	Maître de Conférences	ENP Alger

ENP 2018

REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

École Nationale Polytechnique

المدرسة الوطنية المتعددة التقنيات
Ecole Nationale Polytechnique

مخبر التحكم في العمليات التصنيعية
Laboratoire de Commande des Processus

Département d'Automatique

Laboratoire de Commande des Processus

Mémoire de projet de fin d'études en vue de
l'obtention du diplôme d'Ingénieur d'état en Automatique

Optimisation par Reinforcement Learning et implémentation d'une technique V-SLAM(2D) sur un robot mobile

Réalisé par :

Oussama DEROUICHE
El Hacene CHABANE

Présenté et Soutenu publiquement le 24 Juin 2018 devant le jury composé de :

Président	R. ILLOUL	Maître de Conférences	ENP Alger
Encadreur	M. TADJINE	Professeur	ENP Alger
Promoteur	Z. MOUSSAOUI	Ingénieur agrégé	Airbus Groupe-ECE
Examinateur	B. HEMICI	Maître de Conférences	ENP Alger

ENP 2018

ملخص

العمل المقدم في هذه المذكرة يتناول مشكلة التنقل الذاتي للروبوت المتحرك من خلال تقنيات تقوية غامضة مع تقنيات الرؤية. الهدف من هذا العمل هو تطوير بنية تحكم فعالة للتنقل التفاعلي لروبوت متنقل مستقل في بيئة غير معروفة. تعتمد الأساليب المستخدمة لمعالجة هذه المشكلة على أنظمة الاستدلال الضبابية والتعلم بالتقوية. لكل بنية مقترحة، يتم عرض نتائج المحاكاة لإظهار كفاءة أداء مختلف تطبيقات الملاحة المستقلة.

كلمات مفتاحية : روبوت متحرك، ملاحة، تقنيات الرؤيةّ، منطق ضبابي، التعلم بالتقوية.

Abstract

The work presented in this dissertation treats the autonomous navigation problem of a mobile robot using reinforcement learning and V-SLAM. The objective of this work is the development of an efficient control architecture for reactive navigation of an autonomous mobile robot in an unknown environment. The techniques employed to resolve this problem are based on the fuzzy inference systems and the reinforcement learning. For each proposed technique, simulation results are presented to show the performance of various autonomous navigation applications.

Key words : mobile robot, navigation, V-SLAM, fuzzy logic, reinforcement learning.

Résumé

Le travail présenté dans ce mémoire traite le problème de navigation autonome d'un robot mobile par apprentissage par renforcement et V-SLAM. L'objectif de ce travail porte sur le développement d'une architecture de commande efficace pour une navigation réactive d'un robot mobile autonome dans un environnement inconnu. Les techniques employées pour aborder ce problème sont basées sur les systèmes d'inférence flous et l'apprentissage par renforcement. Pour chaque technique proposée, des résultats de simulation sont présentés pour montrer les performances des diverses applications de navigation autonome.

Mots clés : robot mobile, navigation, V-SLAM, logique floue, apprentissage par renforcement.

Dédicace

أما بعد، أهدي هذا العمل المتواضع :

إلى أمي التي أكرمتني بالحنان والمحبة

و إلى أبي الذي لم يبخل علي يوماً بشيء

وإلى إخوتي عبد الرحيم و عبد الهادي

و إلى أخواتي أسماء و خديجة

و إلى جميع أصدقائي

أهدي هذه المذكرة المتواضعة، راجيا من المولى عز وجل القبول والنجاح

سبْحانَک اللّهُمّ وبحَمْدکَ أشْهدُ أنْ لا إله إلا أنْت أسْتغْفِرکَ وَأتَوبُ إليْک

أسامة درويش

Dédicace

Je dédie ce travail à mes très chers parents,

dont le sacrifice, la tendresse, l'amour, la patience, le soutien,

l'aide et l'encouragement sont l'essentiel de ma réussite.

Sans eux je ne serai pas à ce stade aujourd'hui.

A mes frères et mes sœurs pour leur soutien continue durant mon parcours.

A ma grande famille.

Et à tous mes amis.

El Hacene Chabane

Remerciements

Avant tous, nous remercions الله , le tout puissant de nous avoir donnés le courage, la patience et la force durant toutes ces années d'étude. Nous remercions aussi nos parents, qui nous ont soutenus tout au long de nos études.

Les travaux présentés dans cette thèse ont été effectués au Laboratoire de Commande des Processus (LCP) de l'Ecole Nationale Polytechnique d'Alger en partenariat avec le laboratoire ECEborg de l'Ecole Centrale d'Electronique de Paris.

Nous remercions notre encadreur Monsieur **Mohamed Tadjine** et notre promoteur Monsieur **Zeryab Moussaoui** pour leurs conseils et leurs soutiens tout au long de la réalisation de ce travail.

Nos sincères remerciements vont aussi à toute l'équipe de ECEborg, et spécialement pour **Raphael Casimir**, **Arthur Meuzeret** et **Charles Barbier**.

Nous tenons à remercier vivement Monsieur **Rachid Illoul**, Maître de Conférences à l'Ecole Nationale Polytechnique d'Alger, pour nous avoir faits l'honneur d'accepter d'examiner ce travail et de présider le jury.

Que Monsieur **Boualem Hemici**, Maître de Conférences à l'ENP, trouve ici l'expression de mes sincères remerciements pour l'intérêt qu'il a porté à ce travail en acceptant de l'examiner.

Que tous ceux qui ont contribué de près ou de loin à la réalisation de ce modeste travail trouvent ici l'expression de notre sincère gratitude.

Table des matières

Liste des tableaux

Table des figures

Liste des abréviations

DP	Dynamic Programming (Programmation Dynamique)
FIS	Fuzzy Inference System (Système d'Inférence Floue)
FLC	Fuzzy logic Controller (Régulateur Flou)
FLC20	Fuzzy logic Controller with 20 rules (Régulateur Flou avec 20 règles)
FLC49	Fuzzy logic Controller with 49 rules (Régulateur Flou avec 49 règles)
MC	Monte-Carlo
MDP	Markov Decision Process (Processus de Décision Markovien)
PEE	Politique d'Exploration/Exploitation
Q-FLC	Q- Fuzzy logic Controller (Régulateur Q-learning Flou)
RGB-D	RedGreenBlue-Depth (Rouge Vert Bleu - Profondeur)
RL	Reinforcement Learning (Apprentissage par Renforcement)
RL-FLC	Reinforcement Learning based Fuzzy logic Controller (régulateur renforcement-flou)
SARSA	State-Action-Reward-State-Action
TD	Temporal Difference (Différence Temporelle)
TS(0)	Takagi-Sugeno d'ordre zéro
V-SLAM	Visual Simultaneous Localization And Mapping

Introduction Générale

Il est évident que les machines intelligentes jouent un rôle important dans les activités de tous les jours. Les robots font partie de ces machines et aident les êtres humains dans des activités difficiles et répétitives. La robotique comporte deux grands pôles d'intérêt : la robotique de manipulation (robotique industrielle) et la robotique mobile. La robotique mobile est un domaine de recherche qui se situe au carrefour de l'intelligence artificielle, de l'automatique, de l'informatique et de la vision par ordinateur ; cette interdisciplinarité est à l'origine d'une certaine complexité. Des applications dans des domaines aussi variés que l'exploration spatiale, la robotique de service ou plus récemment les loisirs et le secteur médical, démontrent aujourd'hui l'intérêt économique et social de ces recherches [LEF06].

Son enjeu actuel consiste à développer des systèmes de navigation intelligents. Il consiste à donner à une machine la capacité de se mouvoir dans des environnements variés et sans intervention humaine pour accomplir un but désiré d'une manière autonome. La tâche de navigation vise à donner au robot la possibilité d'obtenir les informations dont il a besoin pour raisonner et à le doter de capacité de locomotion adaptée à son environnement. Au début, le système de navigation est disposé d'un modèle de l'environnement dans lequel sont représentés les obstacles. Cependant, lorsque l'environnement devient plus complexe (i.e. partiellement connu ou dynamique), il apparaît indispensable que le robot mobile soit doté de capacités décisionnelles aptes à le faire réagir automatiquement sans collision avec les objets imprévus [CHE14].

D'autre part, les stratégies de commande réactives offrent des solutions intéressantes ; qui utilisent directement l'information issue des capteurs du robot pour atteindre le but avec évitement d'obstacles en se basant sur des techniques intelligentes. Plusieurs approches ont été proposées pour résoudre le problème de navigation dont les techniques issues de l'intelligence artificielle : la logique floue, les réseaux de neurones artificiels, l'apprentissage automatique et les systèmes hybrides [CHE14].

Cependant, lorsque l'environnement est complexe et dynamique, il est difficile de construire des règles convenables parce que le nombre de situations à considérer est très élevé. D'autre part, le thème de l'apprentissage et l'adaptation est devenu très important dans les dernières années. Parmi les techniques d'apprentissage, celles qui font appel à des techniques d'auto-apprentissage. L'apprentissage supervisé nécessite la disponibilité d'un nombre convenable de paires situation-action. Malgré sa convergence rapide, il est difficile d'obtenir un nombre suffisant de données pour l'apprentissage. D'autre part, l'apprentissage par renforcement semble être une solution prometteuse pour résoudre ce problème puisqu'il ne requiert pas la disponibilité de paires situation-action pour l'apprentissage [SUT98]. Donc, au lieu de programmer un robot pour qu'il effectue une mission, on peut le laisser seul apprendre sa propre stratégie. La technique consiste à déterminer quelles sont les meilleures actions à réaliser. Si ces actions permettent au robot d'atteindre son objectif, leurs liens d'activation sont renforcés. L'idée fondamentale de l'apprentissage par renforcement est d'améliorer un comportement après chaque interaction avec l'environnement, de manière à maximiser une récompense numérique reçue [CHE14, WAT92].

La commande floue pour la navigation est souvent utilisée pour les robots mobiles, mais la construction d'un régulateur flou performant pour un mouvement souhaité n'est pas toujours facile. L'inconvénient majeur est le manque d'une méthodologie systématique pour la conception, due au nombre important de paramètres à régler (les paramètres des fonctions d'appartenances, les paramètres de la partie conclusion et les règles d'inférence). On trouve plusieurs méthodes de réglage des régulateurs flous par l'intégration des propriétés des systèmes flous avec d'autres approches de l'intelligence artificielle et de l'automatique avancée, telles que : l'apprentissage par renforcement et les colonies de fourmis [BOU09]. Ces méthodes dites hybrides combinent les propriétés de chaque approche afin d'apprendre les paramètres des systèmes d'inférence floue. Elles sont capables de générer une solution optimale ou quasi-optimale [CHE14].

Les travaux présentés dans ce mémoire se situent dans le cadre des objectifs suivants :

Étude et application des techniques avancées pour la navigation autonome d'un robot mobile dans un environnement inconnu et l'implémentation d'une approche V-SLAM sur un robot mobile, afin de permettre au robot de se mouvoir d'une position initiale à une autre finale en évitant les obstacles. Pour ce faire, nous utilisons l'approche comportementale à base de la logique floue, les algorithmes d'apprentissage par renforcement et ceux de V-SLAM.

Organisation du mémoire

Ce mémoire est organisé de la manière suivante :

Le premier chapitre est consacré à l'étude des notions de la robotique mobile et de la vision artificielle . Un aperçu général sur les deux domaines sera abordé pour examiner en bref la typologie des robots mobiles, les capteurs utilisées pour la vision et les techniques de SLAM visuel utilisées.

Dans le deuxième chapitre, nous présenterons les systèmes d'inférence floue (FIS), nous commencerons par l'énonciation des fondements théoriques de la logique floue et des sous-ensembles flous. Ensuite, nous donnerons la structure générale d'un régulateur flou et nous décrivons en détail tous ses composants.

Dans le troisième chapitre, nous présenterons un état de l'art sur l'apprentissage par renforcement en commençant par la présentation de ses fondements mathématiques et ses concepts de base. Ensuite, nous décrirons les principaux algorithmes utilisés.

Les deux derniers chapitres présentent les contributions de notre travail.

Dans le quatrième chapitre, nous décrivons les régulateurs testés à base de la logique floue pour la navigation autonome d'un robot mobile. Aussi, nous décrivons l'application des régulateurs flous entraînés par apprentissage par renforcement pour la même mission de navigation et d'évitement. Nous allons proposer des systèmes de commande réactifs en décomposant la tâche globale en un ensemble de sous-tâches secondaires.

Dans le dernier chapitre, nous effectuerons l'implémentation d'une navigation flou à l'aide des techniques V-SLAM sur un robot mobile réel. Nous décrivons le matériel utilisés pour la réalisation et nous traitons les solutions offertes par cette technique. Cette mise en oeuvre sera utilisée pour le comportement d'atteinte d'un but.

Le long des deux derniers chapitres, des exemples de simulations et d'expérimentations sont fournis afin de mettre en évidence les résultats des méthodes proposées pour la navigation autonome d'un robot mobile. Ensuite, ces résultats seront présentés, commentés et comparés.

Nous terminerons ce mémoire par une conclusion générale résumant les principaux résultats obtenus et exposant quelques perspectives autour du travail que nous avons amené.

Chapitre 1

Robotique Mobile et Vision

1.1 Introduction

La robotique qui est un domaine de recherche important qui fait appel aux connaissances croisées de plusieurs disciplines ; son objectif étant de permettre au robot d'interagir rationnellement avec son environnement sans intervention humaine [LAU01]. Les robots mobiles (figure 1.1) ont une place particulière en robotique. Leur intérêt réside dans leur mobilité qui a un aspect particulier qui impose une complexité technologique et méthodologique, cette dernière s'ajoute en général aux problèmes rencontrés par les robots manipulateurs. La résolution de ces problèmes passe par l'emploi de toutes les ressources disponibles tant au niveau technologique (capteurs, motricité, énergie) qu'à celui du traitement d'informations par l'utilisation des techniques de l'intelligence artificielle ou de processeurs particuliers [FIL16].

L'intérêt indéniable de cette discipline est d'avoir permis d'augmenter considérablement nos connaissances sur la localisation, la cartographie et la navigation des systèmes autonomes avec la notion de vision et plus précisément en utilisant des caméras ou ce que l'on appelle le SLAM visuel. Cet axe de recherche étend le domaine d'application de la navigation autonome à toutes les sortes d'environnement non structurés, due principalement à la nécessité d'aider ou de remplacer l'intervention humaine.

Le premier chapitre sert à donner un bref exposé sur le domaine de la robotique mobile. D'abord nous présenterons l'autonomie du robot mobile, les grandes classes et les types des capteurs utilisés. Puis, nous verrons aussi la partie vision en traitant les techniques V-SLAM les plus utilisées.

1.2 Robot mobile

La robotique est un très bon exemple de domaine pluridisciplinaire qui implique de nombreuses thématiques telles que la mécanique, la mécatronique, l'électronique, l'automatique, l'informatique ou l'intelligence artificielle. En fonction du domaine d'origine des auteurs, il existe donc diverses définitions du terme robot, mais elles tournent en général autour de celle-ci [FIL16] :

Un robot est une machine équipée de capacités de perception, de décision et d'action qui lui permettent d'agir de manière autonome dans son environnement en fonction de la perception qu'il en a.

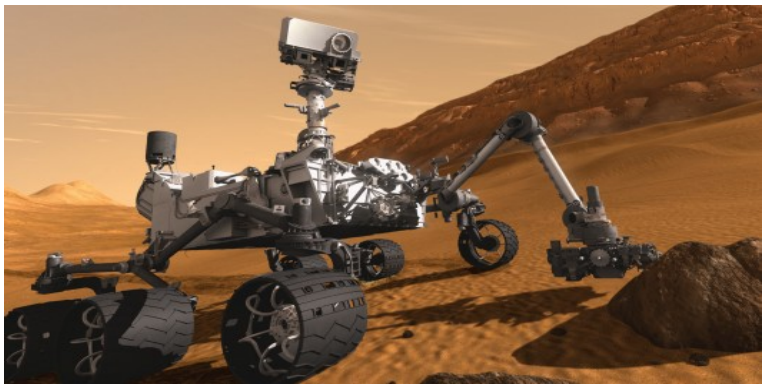


Figure 1.1 – Exemple d'un robot mobile (Curiosity on Mars)

Cette définition s'illustre par un schéma classique des interactions d'un robot avec son environnement (figure 1.2). La manière dont le robot mobile gère ces différents éléments fait appel à un modèle interne de l'environnement ou une stratégie intelligente pour lui permettre de planifier ses actions à long terme.

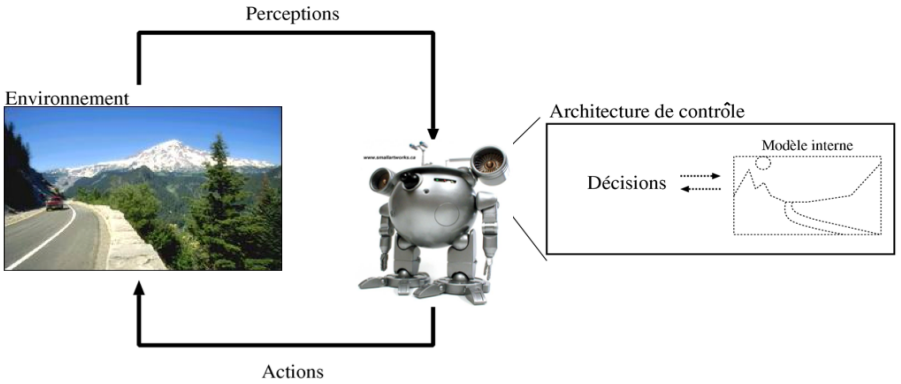


Figure 1.2 – Schéma d'interaction entre le robot et l'environnement

1.3 Autonomie des robots mobiles

Le problème posé par l'étude de l'autonomie d'une machine peut être résumé en la détermination à chaque pas de temps la commande qui doit être envoyée aux actionneurs, connaissant d'une part la mission à accomplir et d'autre part l'acquisition des valeurs retournées par les différents capteurs. Il s'agit de déterminer un lien entre la perception et l'action connaissant les buts à atteindre. Pour cela, le robot doit suivre le schéma correspondant au paradigme (Percevoir/Décider/Agir) [CHE14].

Nous considérons qu'un système est autonome si :

- Il est capable d'accomplir sans intervention humaine les objectifs pour lesquels il a été conçu.
- Il est capable de choisir ses actions afin d'accomplir ces missions.

Il existe plusieurs architectures au niveau de la satisfaction de ce paradigme. Pour cela, l'activité d'un tel robot se ramène aux tâches illustrées sur la figure 1.3 et énoncées ci-après :

- **Percevoir** : le robot doit acquérir des informations sur l'environnement dans lequel il évolue par l'intermédiaire de ses capteurs. Ces informations permettent de mettre à jour un modèle de l'environnement (architectures hiérarchiques ou délibératives) ou peuvent être directement utilisées comme entrées de comportement de bas niveau (architecture purement réactive).

- **Décider** : le robot doit définir des séquences d'actions résultant d'un raisonnement appliqué sur un modèle de l'environnement ou répondant de manière réflexe à des stimuli étroitement liés aux capteurs (architecture purement réactive).

- **Agir** : le robot doit exécuter les séquences d'actions élaborées en envoyant des consignes aux actionneurs par l'intermédiaire des boucles d'asservissements.

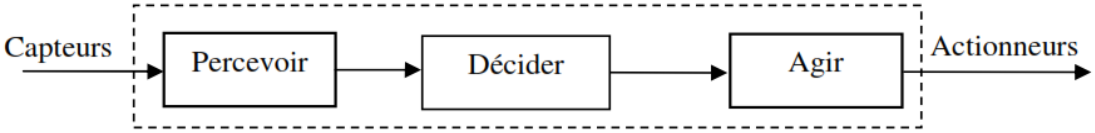


Figure 1.3 – Étapes de traitement automatique

1.4 Présentation des robots mobiles

De manière générale, on regroupe sous l'appellation robots mobiles l'ensemble des robots à base mobile. L'usage veut néanmoins que l'on désigne le plus souvent par ce terme, les robots mobiles à roues. Les robots mobiles sont classés généralement selon le type de locomotion utilisé en quatre groupes distincts :

1.4.1 Robots mobiles à roues

Compte tenu de la simplicité du mécanisme de locomotion utilisé, ce type de robot est le plus répondu actuellement. La plupart des robots mobiles à roues opèrent dans des sites aménagés ou des environnements intérieurs ; mais il existe également des applications en environnements extérieurs, comme l'exploration spatiale. Les modèles des robots mobiles à roues peuvent être classés en plusieurs types avec des propriétés intéressantes [FIL16, BAY07].

1.4.2 Robots mobiles à chenille

Lorsque le terrain est accidenté, les roues perdent leur efficacité de locomotion. Ceci limite la capacité de mouvement du robot mobile équipé de ce type de système de locomotion. Dans ces conditions, les chenilles sont plus intéressantes car elles permettent d'augmenter l'adhérence au sol et de franchir des obstacles plus importants [CHE14].

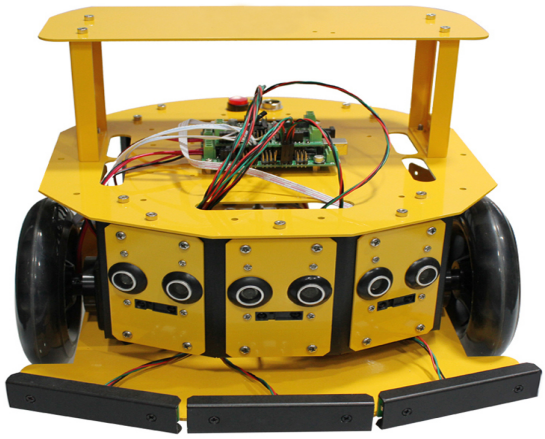
1.4.3 Robots mobiles marcheurs

Ils sont destinés à réaliser des tâches variées dont le terrain est avec grandes différences de hauteur, ce qui rend les deux types précédents moins efficaces. La conception et la commande de ce type de robots sont très complexes. En plus, la vitesse d'évolution est généralement très réduite.

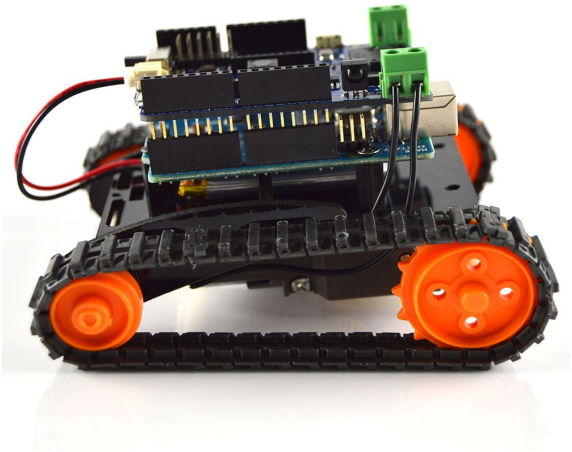
1.4.4 Autres moyens de locomotion

Cette catégorie englobe les robots mobiles qui utilisent un moyen de locomotion différent des trois précédents. Par exemple, les robots mobiles qui se déplacent par reptation, les robots sous-marins et les robots volants,...etc. Les applications et la commande de ces robots sont très spécialisées, l'architecture est en général spécifique à l'application visée [BAY07].

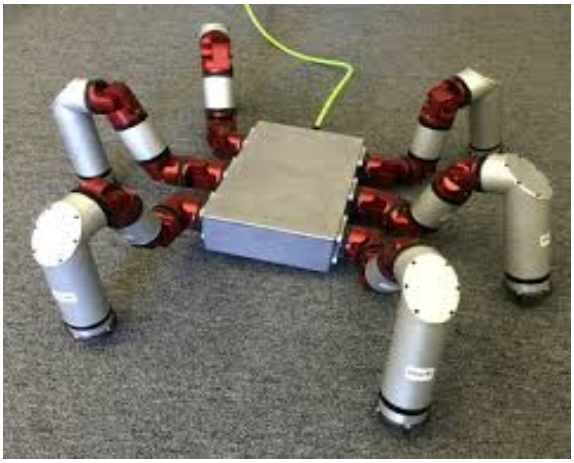
Les figures suivantes (figure 1.4a à la figure 1.4d) illustrent les différents types utilisés dans le domaine de la robotique mobile.



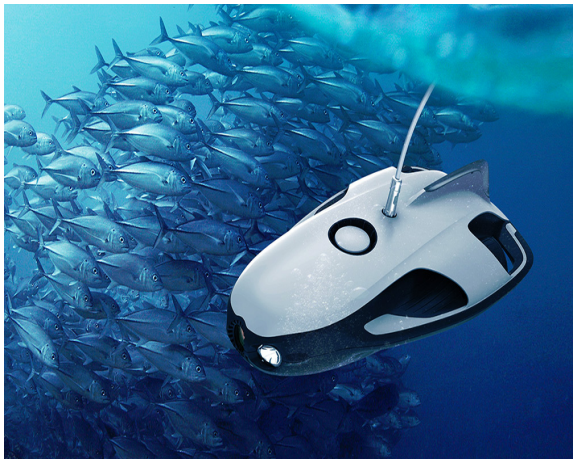
(a) Robot mobile à roues



(b) Robot mobile à chenille



(c) Robot mobile marcheur



(d) Robot mobile sous-marin

Figure 1.4 – Différents robots mobiles

1.5 Capteurs en robotique mobile

Le système de perception (capteurs) est très important pour la sécurité du robot si l'environnement est encombré d'obstacles fixes ou bien mobiles (autres robots). En robotique mobile, on classe traditionnellement les capteurs en deux catégories selon qu'ils mesurent l'état du robot lui-même ou l'état de son environnement.

1.5.1 Capteurs proprioceptifs

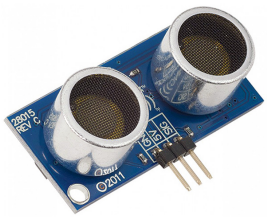
Dans le premier cas, à l'image de la perception chez les êtres vivants, on parle de proprioception et donc de capteurs proprioceptifs. On trouve par exemple dans cette catégorie les capteurs de position ou de vitesse des roues et les capteurs de charge de la batterie. Ce sont des capteurs que l'on peut utiliser directement. Nous citons par exemple : l'odomètre, radar doppler, systèmes inertiels [FIL16, BAY07].

1.5.2 Capteurs extéroceptifs

Les capteurs renseignant sur l'état de l'environnement proche du robot, donc de ce qui est extérieur au robot, sont appelés capteurs extéroceptifs. Ils fournissent des mesures caractéristiques de la position que le robot peut acquérir dans son environnement par la détection des objets qui le contourne. Ils peuvent être de natures très variée. Nous citons comme exemple les télémètres à ultrason, infrarouge, laser, les caméras,...etc [FIL16, BAY07].

On peut voir quelques exemples de capteurs utilisés en robotique mobile sur la figure 1.5.

(a) Roues avec encodeurs



(b) Télémètre à ultrason

(c) Caméra RGB-D

Figure 1.5 – Exemples de capteurs

1.6 Vision artificielle

En robotique, au cours des deux dernières décennies, les innovations technologiques concernant la fabrication de caméras a permis d'intégrer des systèmes complexes de vision dans les robots mobiles pour la navigation autonome. La vision artificielle revêt une importance particulière, car elle permet de fournir à la machine les capacités nécessaires pour réagir avec son environnement ; elle fournit les représentations à partir desquelles le robot prend des décisions.

1.6.1 Caméras pour la vision

L'utilisation d'une caméra pour percevoir l'environnement est une méthode attractive proche des celles utilisées par les humains et fournit une grande quantité d'informations sur l'environnement. Le traitement des données volumineuses et complexes fournies par ces capteurs est cependant souvent difficile. Ces capteurs peuvent se classifier en trois catégories [MEI07].

1.6.1.1 Caméras classiques

Elles peuvent être utilisées pour la navigation d'un robot mobile pour détecter des amers visuels à partir desquels il sera possible de calculer la position du robot. Cependant elles ont un angle de vue assez faible (40 degrés).

1.6.1.2 Caméras stéréoscopiques

La stéréovision offre la possibilité d'estimer directement la position des amers dans la scène et de connaître le facteur d'échelle. Autrement dit, Lorsque l'on dispose de deux caméras observant la même partie de l'environnement à partir de deux points de vue différents, il est possible d'estimer la distance des objets et d'avoir ainsi une image de profondeur. La résolution et l'écartement des deux caméras imposent également les profondeurs minimale et maximale qui peuvent être perçues, ce qui peut être limitatif pour la vitesse de déplacement du robot [FIL16].

1.6.1.3 Caméras panoramiques

Elles sont constituées d'une caméra standard pointant vers un miroir de révolution. L'image recueillie permet d'avoir une vision de l'environnement sur 360 degrés autour de la caméra. Elles deviennent de plus en plus populaires en robotique notamment en SLAM. On peut voir les caméras décrites précédemment sur la figure 1.6.

(a) Caméra monoculaire

(b) Caméra stéréoscopique

(c) Caméra omnidirectionnelle

Figure 1.6 – Exemples de caméras

1.6.2 V-SLAM

Le SLAM basé sur la vision a pour but d'effectuer simultanément la construction d'une carte de l'environnement et la localisation du robot à l'aide des caméras. Elle s'inspire des travaux de SLAM classiques qui se basent généralement sur des capteurs de distance. Cette approche basée sur la vision est apparue il y a une dizaine d'années avec les travaux d'Andrew Davison [DAV07]. Elle est très utilisée aujourd'hui car elle représente un moyen pratique et efficace d'effectuer de la cartographie et de la localisation en utilisant uniquement la vision.

1.6.3 SLAM monoculaire

Différentes questions sont impliquées dans le problème du SLAM visuel avec une caméra et de nombreuses approches existent afin de le résoudre. Nous allons présenter quelques travaux réalisés.

1.6.3.1 MonoSLAM

MonoSLAM est le premier algorithme de SLAM Monoculaire à être capable de fonctionner en temps-réel. Il est basé sur l'utilisantion d'une carte probabiliste des caractéristiques de l'environnement, qui représentent la position et l'orientation estimés de la caméra et de tous les points d'intérêt (regroupés dans un vecteur d'état), ainsi que l'incertitude de ces estimées (regroupées dans une matrice de covariance). Le vecteur d'état et la matrice de covariance augmentent à chaque nouvelle observation et les estimées sont mises à jour grâce à un filtre de Kalman étendu (EKF). L'un de ses inconvénients est le temps de calcul qui est proportionnel au carré du nombre de points d'intérêt, ce qui le rend inutile pour de grands environnements.

1.6.3.2 Parallel Tracking And Mapping

PTAM [KLE07] exécute la détection de la position et de l'orientation de la caméra et la cartographie sur l'environnement en parallèle sur différents threads d'un processeur multi-cœur, ce qui veut dire que l'algorithme réduit les coûts de calcul et lui permet de fonctionner en temps-réel (figure 1.7) ; un thread est responsable de l'estimation de la pose d'une caméra et l'autre s'occupe de la construction de la carte.

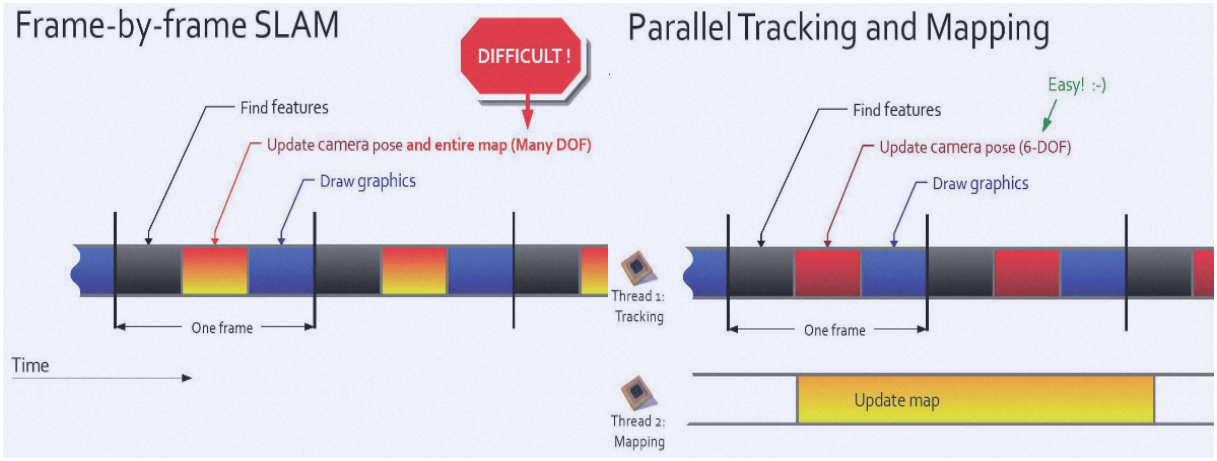


Figure 1.7 – Compariason entre PTAM et frame par frame algorithme.

Le suivi et la cartographie sont séparés et fonctionnent en parallèle La cartographie est basée sur des images clés, qui sont traitées par lots en utilisant l'ajustement de faisceaux (Bundle Ajustement). La carte est initialisée en utilisant une paire d'image avec l'algorithme de 5 points stéréo, par conséquent elle représente une limitation car l'initialisation stéréo d'une manière très spécifique nécessite l'intervention de l'utilisateur. Cette approche est également caractérisée par le grand nombre d'amers utilisés dans la cartographie.

1.6.3.3 Large-Scale Direct Monocular SLAM

LSD-SLAM [ENG14] est une technique de SLAM monoculaire directe, c'est-à-dire, qu'au lieu d'utiliser des points d'intérêt dispersés dans l'environnement, il opère directement sur les intensités d'images à la fois pour le suivi et la cartographie. La caméra est suivie à l'aide d'alignement d'image direct, tandis que la géométrie est estimée sous la forme de cartes de profondeur semi-denses obtenues par filtration sur de nombreuses comparaisons faites pixel par pixel. Ce SLAM consiste de 3 modules : tracking (suivi), l'estimation de la profondeur de la carte et l'optimisation de la carte.

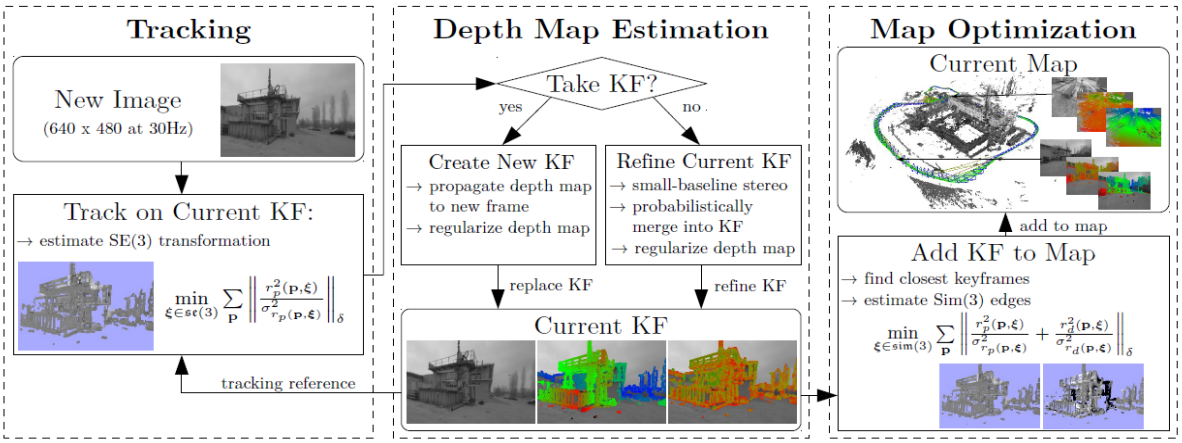


Figure 1.8 – Aperçu sur l'algorithme de LSD-SLAM

1.6.3.4 ORB-SLAM

L'algorithme de ORB-SLAM (Oriented fast and Rotated Brief SLAM) [MUR15] utilise un détécteur de zones d'intérêts et est capable de calculer en temps réel la trajectoire de la caméra et la reconstruction de la scène dans une grande variété d'environnements, allant de petites séquences tenues à la main d'un bureau à une voiture conduite autour de plusieurs pâtés de maisons.

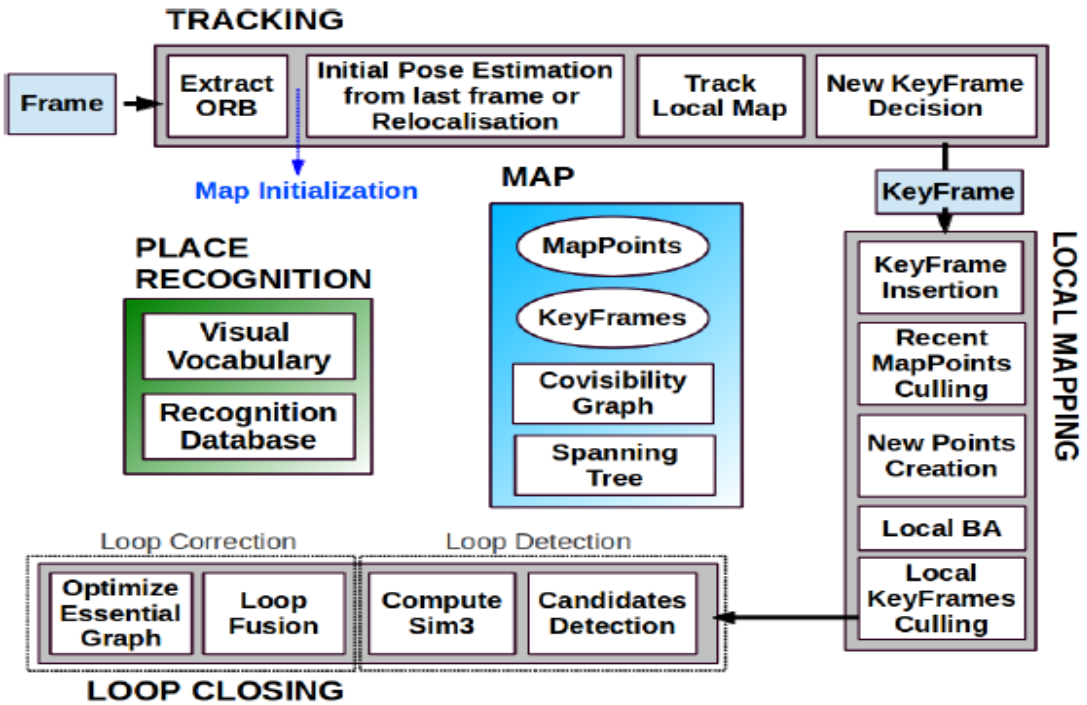


Figure 1.9 – Schéma des différents modules d'ORB-SLAM

L'ORB-SLAM consiste à recevoir en entrée les images et à exécuter 3 threads en parallèle : la localisation (Tracking), la cartographie locale (Local Mapping) et la détection de fermeture de boucle (Loop Closing) (figure 1.9).

La localisation : elle permet de localiser la caméra à partir des images. Ce thread décide également si une nouvelle image sera considérée comme la référence ou rejetée. Dans le cas où la localisation perd sa référence, le module de reconnaissance (Place Recognition) sert de refaire une localisation globale afin de retrouver la position de la caméra.

La cartographie locale : elle a pour objectif de traiter les nouvelles images de référence. Elle permet de faire une correspondance entre les nouveaux points ajoutés en faisant une triangulation. Lors de cette étape, un algorithme d'élimination est appliqué dans le but de ne garder que les points de bonnes qualités selon les critères de Recent Map Points Culling.

La fermeture de boucle : elle recherche dans toutes les nouvelles images ajoutées s'il existe une fermeture de boucle. Dans le cas où une fermeture est détectée, les 2 terminaux de boucle sont alignés et les points redondants sont fusionnés. Cela nous permet d'estimer l'erreur accumulée et de corriger l'erreur.

1.7 Conclusion

Le robot mobile à roues est le robot mobile le plus répandu, à cause de sa structure mécanique simple et ses commandes relativement faciles. La commande d'un robot mobile se divise généralement en trois étapes principales : perception, décision et action.

Dans ce chapitre, nous avons résumé toutes les notions de base nécessaires à la compréhension du domaine de la robotique mobile, ainsi qu'un panorama des types de robots et les capteurs utilisés. Nous avons aussi présenté les caméras utilisées pour la perception ainsi que les principaux algorithmes de SLAM basé sur la vision.

Après cet aperçu, dans le chapitre suivant nous présenterons une étude générale sur les systèmes d'inférence floue et les fondements théoriques des sous-ensembles flous. Nous décrivons la structure générale d'un régulateur flou avec ces composants pour l'utiliser par la suite face au problème de navigation autonome d'un robot mobile.

Chapitre 2

Théorie des Systèmes d'Inférence Floue

2.1 Introduction

La logique floue a été introduite en 1965 par le professeur Lotfi Zadeh [ZAD65] de l'université de Berkeley en Californie comme une généralisation de la logique binaire. Elle est issue de la capacité d'une personne à décider et agir de façon pertinente malgré le flou des connaissances disponibles. En effet, la logique floue a été introduite pour approcher le raisonnement humain à l'aide d'une représentation adéquate des connaissances.

En 1974, Mamdani [MAM74] était le premier à étudier la possibilité de commander un système dynamique par des règles floues. Puis un an après, Mamdani et Assilian [MAM75] développaient le premier régulateur par logique floue. Les premiers travaux de Mamdani introduisaient la méthode de raisonnement flou min-max de Zadeh-Mamdani. En 1985, Takagi et Sugeno [SUG85] introduisaient une description différente des sous-ensembles flous de la sortie.

Dans ce chapitre, Nous exposerons un bref rappel mathématique sur la théorie des sous-ensembles flous et les opérateurs flous, en insistant sur les idées utilisées en commande floue. Ensuite, Nous présenterons la structure générale des régulateurs flous.

2.2 Concepts de base de la logique floue

La logique floue peut être vue comme une extension de la logique booléenne. Elle permet de traiter des variables linguistiques dont les valeurs sont des expressions du langage naturel. Cette partie fournit un résumé des concepts de base essentiels pour l'étude de la logique floue [BOU09].

2.2.1 Sous-ensembles flous

Un ensemble classique (figure 2.1a) A de U est défini par une fonction caractéristique notée $\mu_A(x)$ telle que :

$$\mu_A(x) = \begin{cases} 1 & \text{si} \quad x \in A \\ 0 & \text{si} \quad x \notin A \end{cases} \tag{2.1}$$

Le concept de base de la théorie des ensembles flous est la notion des sous-ensemble flous. Cette notion provient du constat que « très souvent, les classes d'objets rencontrés dans le monde physique ne possèdent pas de critères d'appartenance bien définis ».

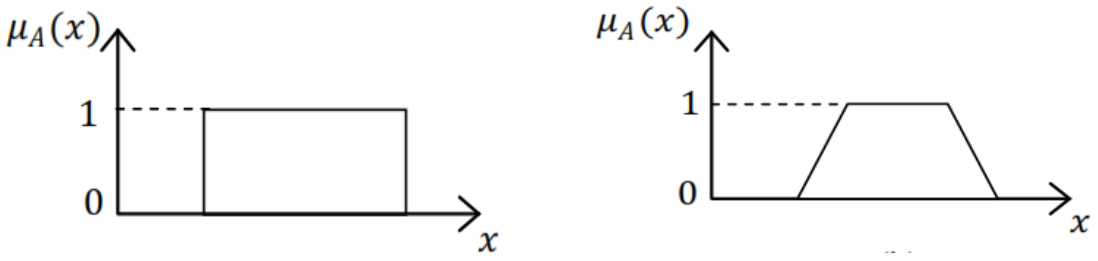
Soit $U = R^n$ une collection d'objets appelée univers de discours. Un ensemble flou A dans U est caractérisé par une fonction d'appartenance, notée $\mu_A(x)$:

$$\mu_A : U \rightarrow [0\ 1] \tag{2.2}$$

Cette fonction appliquée à un élément x de U, retourne un degré d'appartenance $\mu_A(x)$ de x à A (figure 2.1b). Un ensemble flou peut être considéré comme une généralisation de l'ensemble classique. La fonction d'appartenance d'un ensemble classique peut prendre seulement deux valeurs $\{0,1\}$. Un ensemble flou peut être représenté comme un ensemble de paires ordonnées :

$$A = \{(x, \mu_A(x) \,/\, x \in U)\} \tag{2.3}$$

Les ensembles flous ont le grand avantage de constituer une représentation mathématique des termes linguistiques largement utilisés dans l'expression des connaissances expertes, qualitatives et qui sont manipulées par la logique floue [BOU09].



(a) Logique classique (b) Logique floue

Figure 2.1 – Exemples de fonctions d'appartenance

2.2.2 Fonctions d'appartenance

Elles associent à chaque valeur x une valeur précise $\mu_A(x)$ désignée par le degré d'appartenance de x à A. Les fonctions d'appartenance les plus utilisées sont :

- **Fonction triangulaire :** Elle est définie par trois paramètres (a,b,c) (figure 2.2a).

$$\mu_A(x) = max\left\{min\left\{\frac{x-a}{b-a}, \frac{c-x}{c-b}\right\}, 0\right\} \tag{2.4}$$

- **Fonction trapézoïdale :** Elle est définie par quatre paramètres (a,b,c,d) (figure 2.2b).

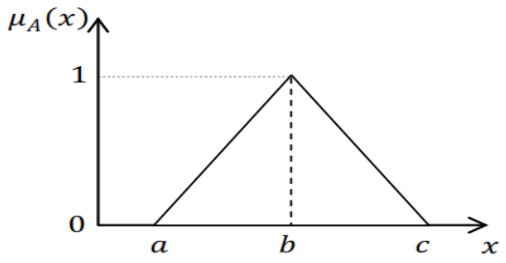
$$\mu_A(x) = max\left\{min\left\{\frac{x-a}{b-a}, 1, \frac{d-x}{d-c}\right\}, 0\right\} \tag{2.5}$$

- **Fonction gaussienne :** Elle est définie par deux paramètres (σ, m) (figure 2.2c).

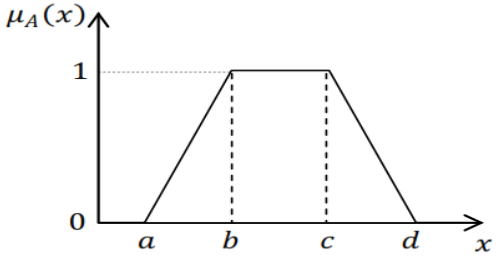
$$\mu_A(x) = e^{-\frac{(x-m)^2}{2\sigma^2}} \tag{2.6}$$

- **Fonction sigmoïde :** Elle est définie par deux paramètres (a,c) (figure 2.2d).

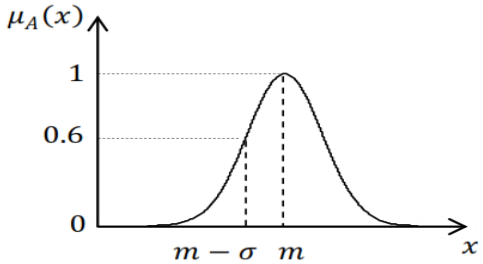
$$\mu_A(x) = \frac{1}{1+e^{-a(x-c)}} \tag{2.7}$$



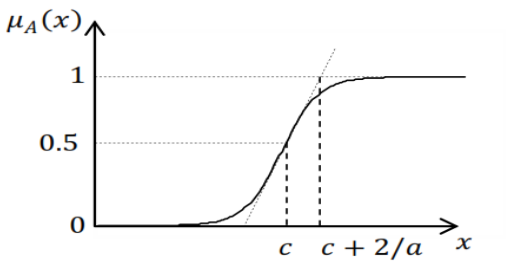
(a) Fonction triangulaire



(b) Fonction trapézoïdale



(c) Fonction gaussienne



(d) Fonction sigmoïde

Figure 2.2 – Formes des fonctions d'appartenance usuelles

2.2.3 Variable linguistique

C'est une variable dont les valeurs ne sont pas des nombres, mais des mots ou phrases exprimés en langage naturel. La raison pour laquelle on utilise cette représentation, est que le caractère linguistique est moins spécifique que le caractère numérique.

Une variable linguistique est généralement représentée par un triplet $(x, T(x), U)$ dans lequel :

— x est le nom de la variable linguistique (vitesse, erreur, position,...).

— $T(x)$ est l'ensemble des valeurs linguistiques qui sont utilisées pour caractériser x.

— U est l'univers de discours de la variable linguistique x.

Par exemple, si l'erreur est considérée comme une variable linguistique définie sur l'univers de discours $U = [-1\ +1]$, ses valeurs linguistiques peuvent être définies comme suit :

T(erreur)= Négatif Grand (NG), Négatif Petit (NP), Zéro (Z), Positif Petit (PP), Positif Grand (PG).

Ces symboles linguistiques peuvent être considérés comme des sous-ensembles flous dont les fonctions d'appartenance sont représentées sur la figure 2.3

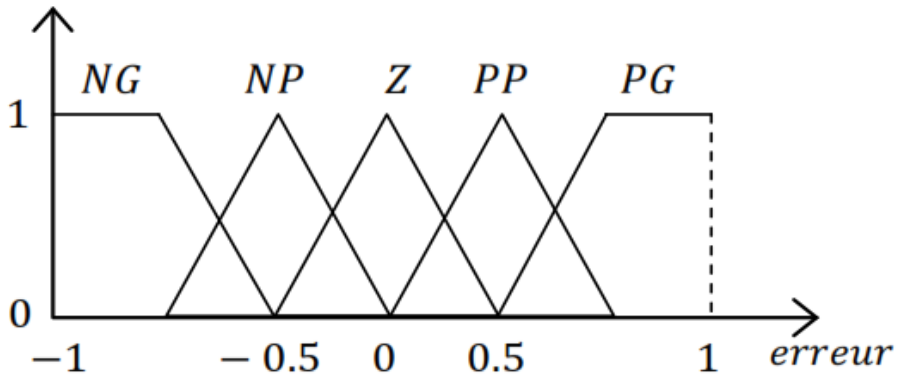


Figure 2.3 – Représentation des fonctions d'appartenance de la variable linguistique (erreur)

Le choix du nombre des sous-ensembles flous, de la forme des fonctions d'appartenance, du recouvrement de ces fonctions et de leur répartition sur l'univers de discours n'est jamais évident. Ainsi, la forme des fonctions n'a que peu d'influence, d'où le choix de la forme triangulaire pour notre travail, celle-ci étant relativement simple à mettre en œuvre.

2.2.4 Opérations sur les ensembles flous

Soit A et B deux sous-ensembles flous définis dans l'univers de discours U par les fonctions d'appartenance $\mu_A(x)$ et $\mu_B(x)$ respectivement.

Les opérateurs de la logique floue sont définis comme suit :

- **Complément floue :** Le complément $\bar{A}$ (figure 2.4a) de A est défini par la fonction d'appartenance :

$$\forall x \in U \ : \ \mu_{\bar{A}}(x) = 1 - \mu_A(x) \tag{2.8}$$

- **Union floue (disjonction) :** L'union de deux sous-ensembles flous A et B est un ensemble flou $(A \cup B)$ (figure 2.4b) dans U de fonction d'appartenance :

$$\forall x \in U \ : \ \mu_{A\cup B}(x) = \mu_A(x) \dot{+} \mu_B(x) \tag{2.9}$$

  Le symbole $\dot{+}$ représente la co-norme triangulaire. Les opérateurs d'union cités dans la littérature sont :

  — max

$$\mu_{A\cup B}(x) = max\{\mu_A(x), \mu_B(x)\} \tag{2.10}$$

  — somme algébrique

$$\mu_{A\cup B}(x) = \mu_A(x) + \mu_B(x) - \mu_A(x)\mu_B(x) \tag{2.11}$$

  L'opérateur *max* est le plus utilisé dans le domaine de la commande.

- **Intersection floue (conjonction) :** L'intersection de deux sous-ensembles flous A et B est un ensemble flou $(A \cap B)$ (figure 2.4c) dans U de fonction d'appartenance :

$$\forall x \in U \ : \ \mu_{A\cap B}(x) = \mu_A(x) * \mu_B(x) \tag{2.12}$$

  Le symbole $*$ représente la norme triangulaire. Les opérateurs d'intersection cités dans la littérature sont :

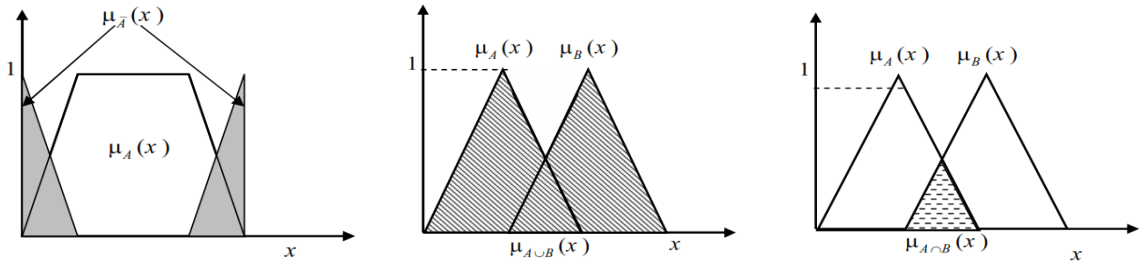
  — min

$$\mu_{A\cap B}(x) = min\{\mu_A(x), \mu_B(x)\} \tag{2.13}$$

  — produit algébrique

$$\mu_{A\cap B}(x) = \mu_A(x) \times \mu_B(x) \tag{2.14}$$

  L'opérateur *min* est le plus utilisé dans le domaine de la commande.



(a) Complément flou (b) Union floue (c) Intersection floue

Figure 2.4 – Opérateurs flous

- **Produit cartésien :** Soient $A_1, A_2, ..., A_n$ des ensembles flous respectivement dans $U_1, U_2, ..., U_n$, leur produit cartésien est un ensemble flou défini sur $U_1 \times U_2 \times ... \times U_n$, de fonction d'appartenance :

$$\mu_{A_1 \times A_2 \times ... \times A_n}(x_1, x_2, ..., x_n) = \mu_{A_1}(x_1) * \mu_{A_2}(x_2) * ... * \mu_{A_n}(x_n) \tag{2.15}$$

- **Relation floue :** Une relation floue représente le degré de présence ou d'absence d'une association entre les éléments de plusieurs ensembles flous. Une relation floue d'ordre n est un ensemble flou défini sur $U_1 \times U_2 \times ... \times U_n$ par l'expression suivante :

$$R_{U_1 \times ... \times U_n} = \{((x_1, ..., x_n), \mu_R(x_1, ..., x_n)) \,/\, (x_1, ..., x_n) \in U_1 \times ... \times U_n\} \tag{2.16}$$

- **Composition des relations floues :** Soient R_1 et R_2 deux relations floues définies respectivement dans $U \times V$ et $V \times W$. La composition de R_1 et R_2, est un ensemble flou dénoté par $R_1 \circ R_2$, de fonction d'appartenance :

$$\mu_{R_1 \circ R_2}(x, z) = \left\{(x, z), \underset{y \in V}{Sup} \{\mu_{R_1}(x, y) * \mu_{R_2}(y, z)\}\right\} \tag{2.17}$$

  Cette composition est appelée $min - max$

- **Implication floue :** C'est un opérateur qui permet d'évaluer le degré de vérité d'une règle R de la forme "Si x est A Alors y est B" à partir des valeurs de la prémisse (x est A) d'une part, et de celle de la conclusion (y est B) d'autre part.

$$\mu_R(x, y) = imp(\mu_A(x), \mu_B(y)) \tag{2.18}$$

  Les opérateurs les plus utilisés en commande floue sont les implications de Mamdani et de Larsen :

  — implication de Mamdani $\quad \mu_R(x, y) = min(\mu_A(x), \mu_B(y))$

  — implication de Larsen $\quad \mu_R(x, y) = \mu_A(x) * \mu_B(y)$

2.2.5 Raisonnement flou

En logique classique le modus ponens permet, à partir de la règle « Si x est A alors y est B », de conclure le fait « y est B » si « x est A », qui sera ajouté à la base des faits. Zadeh a étendu ce principe au cas flou, sous le nom de modus ponens généralisé [BOU09]. Le modus ponens et le modus ponens généralisé se résument comme suit :

	Modus Ponens	**Modus Ponens Généralisé**
Prémisse (fait)	x est A	x est A'
Règle (implication)	Si x est A alors y est B	Si x est A alors y est B
Conclusion (déduction)	y est B	y est B'

A partir de la règle « Si x est A Alors y est B» et du fait A', on déduit un nouveau fait B' qui est caractérisé par un sous-ensemble flou dont la fonction d'appartenance est donnée par :

$$\mu_{B'}(y) = Sup_x(\mu_{A'}(x) * \mu_R(x,y)) \tag{2.19}$$

Les fonctions d'appartenance $\mu_{A'}(x)$ et $\mu_R(x,y)$ caractérisent respectivement le fait A' et la règle.

2.3 Structure des systèmes d'inférence floue

Un régulateur flou peut être vu comme un système expert fonctionnant à partir d'une représentation des connaissances basées sur la théorie des ensembles flous [BOU09]. La figure 2.5 montre son schéma synoptique proposé par Mamdani. Nous allons rappeler dans ce qui suit une description sommaire de chaque module composant ce régulateur [LAB98] :

• **Fuzzification** : ce module transforme les données numériques de la grandeur physique d'entrée en une grandeur floue symbolique.

• **Base des règles floues** : elle contient les règles floues décrivant le comportement du système.

• **Moteur d'inférence floue** : il transforme la partie floue issue de la fuzzification en une nouvelle partie floue en utilisant la base des règles.

• **Défuzzification** : ce module transforme la grandeur floue issue de l'inférence en une grandeur physique.

Nous allons maintenant reprendre plus en détail ces différentes parties, afin de montrer le rôle de chacune.

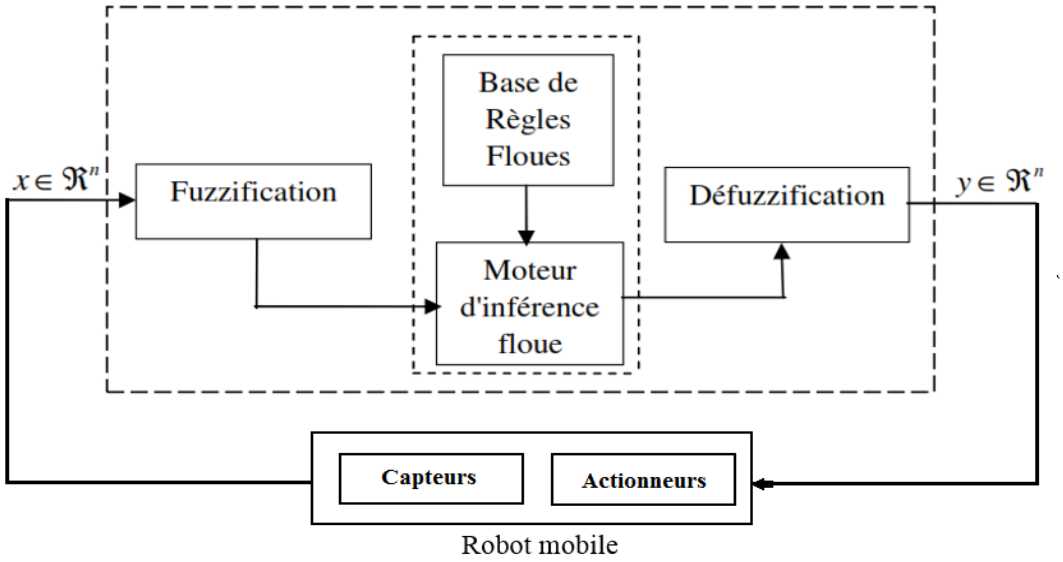


Figure 2.5 – Configuration de base d'un système flou

2.3.1 Fuzzification

La fuzzification consiste à relier le point numérique x_0 de U à l'ensemble flou A_x dans U. Il existe deux méthodes de fuzzification suivant la définition de A_x :

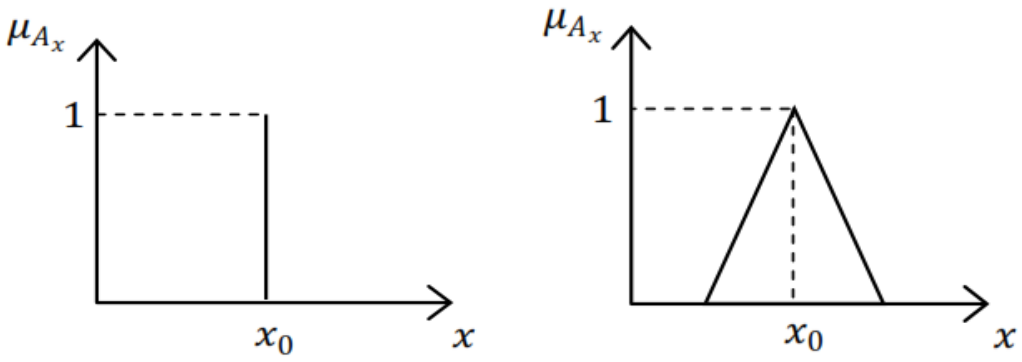
— A_x est un singleton flou défini par :

$$\mu_{A_x}(x) = \begin{cases} 1 & \text{si} \quad x = x_0 \\ 0 & \text{si} \quad x \neq x_0 \end{cases} \tag{2.20}$$

Dans ce cas, on considère que la valeur de x est précise et certaine (Figure 2.6a).

— A_x est un ensemble flou de fonction d'appartenance $\mu_{A_x}(x_0) = 1$ et $\mu_{A_x}(x)$ décroît lorsque x s'éloigne de x_0.

Dans ce cas, est pris en compte le comportement de la variable autour de la valeur x_0. Par exemple, une variable est modélisée par une fonction d'appartenance triangulaire présentée sur la Figure 2.6b.



(a) Singleton (b) Ensemble flou

Figure 2.6 – Méthodes de fuzzification

2.3.2 Base des règles floues

Une base de règles floues R est une collection de règles floues de la forme SI-ALORS, $R = [R_1, R_2, ..., R_m]$. Une règle floue R_i est donnée sous le modèle de :

— Mamdani

$$R_i : \text{Si } x_1 \text{ est } A_{i1} \text{ et } ... \text{ et } x_n \text{ est } A_{in} \text{ Alors } y \text{ est } B_i \tag{2.21}$$

— Takagi-Sugeno (TS)

$$R_i : \text{Si } x_1 \text{ est } A_{i1} \text{ et } ... \text{ et } x_n \text{ est } A_{in} \text{ Alors } y \text{ est } f_i(x) \tag{2.22}$$

où $f_i(x)$ est un polynôme souvent d'ordre zéro (TS0).

2.3.3 Moteur d'inférence floue

Le moteur d'inférence floue utilise la base des règles floues pour effecteur une transformation à partir des ensembles flous dans l'espace d'entrée vers les ensembles flous dans l'espace de sortie en se basant sur les opérations de la logique floue. L'antécédent de règle R définit un produit cartésien de $A_{i1}, A_{i1}, ..., A_{in}$, et la règle elle-même R_i est vue comme une implication. Soit A_x un ensemble flou dans U, alors chaque règle R_i détermine un ensemble flou B'_i ($B'_i = A_x \circ R_i$) dans V. La fonction d'appartenance de B' est donnée par la règle compositionnelle 2.24 :

$$\mu_{B'_i}(y) = \underset{x \in A_x}{Sup}(\mu_{A_x}(x) * \mu_{R_i}(x, y)) \tag{2.23}$$

L'ensemble des m règles constituant la base des règles floues sont liées par l'opérateur de disjonction "OU". Ainsi, l'ensemble flou final B'_i est donné par la relation suivante :

$$\begin{cases} B' = B'_1 \dot{+} B'_2 \dot{+} ... \dot{+} B'_m \\ \mu_{B'}(y) = \mu_{A_x \circ R_1}(y) \dot{+} \mu_{A_x \circ R_2}(y) \dot{+} ... \dot{+} \mu_{A_x \circ R_m}(y) \end{cases} \tag{2.24}$$

Dans le jeu des règles du système flou interviennent les opérateurs flous "ET" et "OU". L'opérateur "ET" s'applique aux variables à l'intérieur d'une règle, tandis que l'opérateur "OU" lie les différentes règles. Plusieurs types de raisonnement flou ont été proposés , les trois moteurs d'inférence floue les plus utilisés sont [BOU09] :

2.3.3.1 Méthode de Mamdani

Dans le modèle de Mamdani, les prémisses et les conclusions des règles sont symboliques ou linguistiques. Cette méthode se base sur l'utilisation de l'opérateur *min* pour l'implication floue et l'opérateur *max* pour l'agrégation des règles. La sortie nécessite l'utilisation d'une méthode de défuzzification qui est généralement le barycentre. Cette implémentation est appelée $(min, max, barycentre)$.

2.3.3.2 Méthode de Takagi-Sugeno

Dans ce cas, des règles floues de type Sugeno sont utilisées. En effet, les conclusions des règles floues sont des polynômes en fonction des variables d'entrée. L'implication floue est réalisée par l'opérateur *min* ou par le *produit algébrique*. La sortie finale est égale à la moyenne pondérée des conclusions des règles. Il est à noter que le système flou TS0 est équivalent au système flou de Mamdani utilisant une fuzzification singleton et une défuzzification du barycentre.

2.3.3.3 Méthode de Tsukumoto

Dans ce cas, des fonctions monotoniques sont associées aux variables de sortie. La sortie totale est une moyenne pondérée des degrés de confiance des règles floues et des valeurs de fonctions de variables de sortie.

La figure 2.7 illustre les types du raisonnement flou pour un système à deux entrées et une base de connaissances de deux règles. On constate que les différences viennent de la spécification de la partie conclusion d'une part, et de la méthode de défuzzification d'autre part.

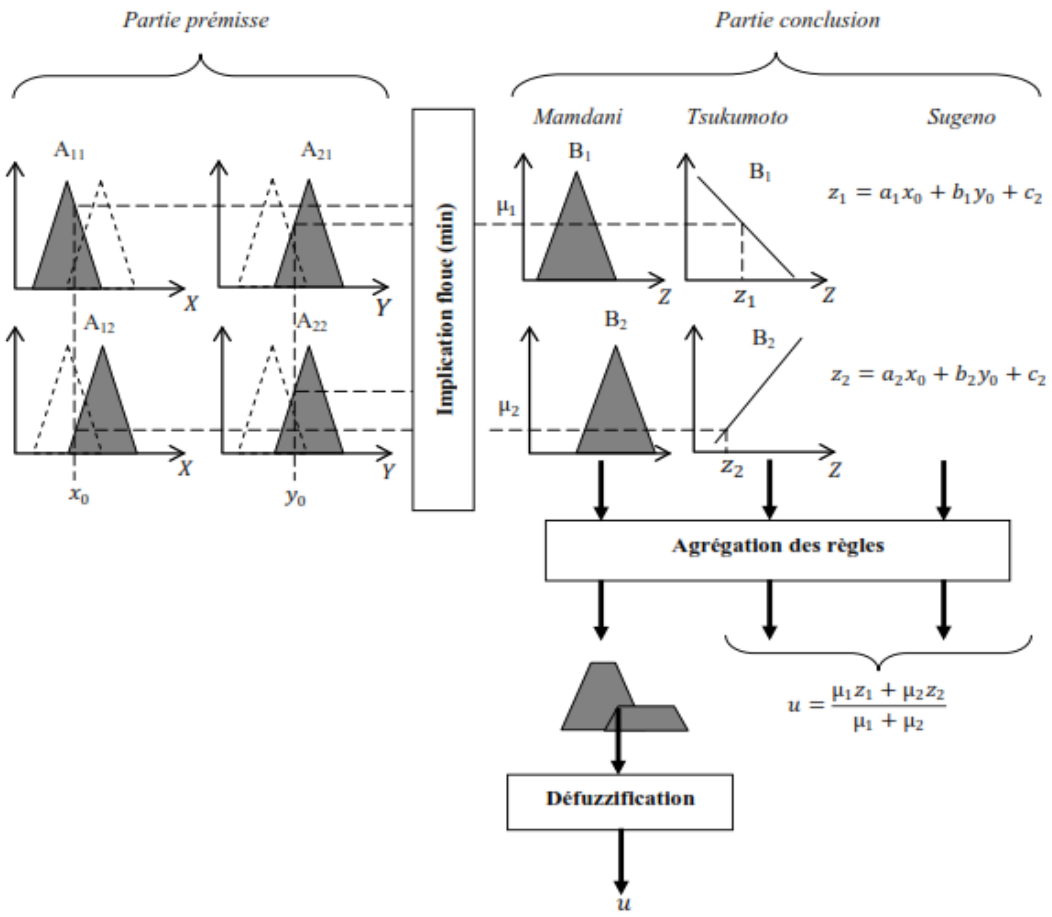


Figure 2.7 – Différents modèles d'inférence floue

2.3.4 Défuzzification

Le rôle de la défuzzification est de transformer la partie floue issue de l'inférence en une grandeur numérique applicable au système. Il existe plusieurs méthodes de défuzzification basées essentiellement sur la simplicité du calcul [BOU09].

2.3.4.1 Méthode de centre de gravité

Dans ce cas, on calcule le centre de gravité $\bar{y}$ de l'ensemble flou résultant B' et considère cette valeur comme résultat de défuzzification. Souvent on utilise la version discrète :

$$\bar{y} = \frac{\sum_{i=1}^{m} y_i \mu_{B'}(y_i)}{\sum_{i=1}^{m} \mu_{B'}(y_i)} \tag{2.25}$$

Il est à noter que le centre de gravité est généralement difficile à calculer. De ce fait, cette méthode est la plus coûteuse en temps de calcul [BOU09].

2.3.4.2 Méthode du maximum

La méthode du maximum examine l'ensemble flou B' issu de l'inférence et choisit comme sortie la valeur y pour laquelle $\mu_{B'}(y)$ est un maximum. Cependant, ce défuzzificateur présente un certain inconvénient lorsqu'il existe plusieurs valeurs pour lesquelles $\mu_{B'}(y)$ est un maximum.

2.3.4.3 Méthode de la moyenne des maxima

Cette méthode examine l'ensemble flou B' issu de l'inférence et et détermine en premier temps les valeurs y_i pour lesquelles $\mu_{B'}(y_i)$ est un maximum. Ensuite, elle calcule la moyenne de ces valeurs comme résultat de défuzzification.

Il existe d'autres méthodes qui pourront être exploitées, comme la méthode des hauteurs pondérées et la méthode des hauteurs pondérées modifiées.

2.4 Approximation des fonctions par les systèmes flous

Deux raisons principales amènent à utiliser les systèmes flous comme régulateurs des systèmes. Premièrement, ce type de systèmes flous a la propriété d'approximateur universel de fonctions continues avec un degré de précision quelconque à condition d'utiliser un nombre suffisant de règles floues. Deuxièmement, les systèmes flous sont construits à partir de règles floues de la forme Si-Alors, de ce fait, les informations linguistiques ou mathématiques disponibles, issues d'une expertise peuvent éventuellement être incorporées dans le régulateur.

Les systèmes flous sont des approximateurs universels, c'est-à-dire, pour toute fonction réelle continue f définie sur un compact C de R^n ; et pour toute constante positive e il existe un FIS tel que :

$$\forall x \in C, \|f(x) - FIS(x)\| < e \tag{2.26}$$

Notons, cependant, que la propriété d'approximation universelle ne donne pas une méthode de construction du système flou FIS, mais elle garantit seulement son existence. De plus, pour un degré de précision quelconque, il faut utiliser un nombre important de règles floues [LAB05, CHE14].

2.5 Conclusion

Dans ce chapitre, nous avons présenté en bref la théorie de la logique floue. Les notions de base de la logique floue les plus pertinentes sont exposées et l'architecture de base d'un régulateur flou est présentée. Les systèmes flous sont des approximateurs universels. En fait, ils peuvent approcher n'importe quelle fonction à partir de données numériques. Le fonctionnement d'un régulateur flou dépend d'un nombre important de paramètres (méthode de fuzzification, le type des fonctions d'appartenance, le type des règles floues, la méthode du raisonnement flou et la stratégie de défuzzification) qu'il faut déterminer lors de la conception.

Les systèmes d'inférence floue (structures basés sur des comportements flous) seront utilisés dans le chapitre 4 pour faire naviguer un robot mobile différentiel pour des tâches de convergence vers un but avec l'évitements d'obstacles.

Dans le chapitre 3 nous exposerons le principe de l'apprentissage par renforcement et la formalisation mathématique des Processus de Décision Markoviens (MDP). Nous présenterons aussi les différents algorithmes de cette méthode d'apprentissage.

Chapitre 3

Apprentissage par Renforcement

3.1 Introduction

L'apprentissage par renforcement [WAT89, SUT88] est une approche de l'apprentissage automatique (machine learning en anglais). Il est considéré comme un paradigme qui permet l'apprentissage d'un agent par l'interaction avec son environnement afin de trouver par un processus (essais/erreurs), l'action optimale à exécuter pour chacune des situations, dans le but de maximiser un signal de performance qui exprime un objectif à long terme [SUT98].

Ses notions telles que nous les connaissons aujourd'hui se sont développées à partir de 1988 par Sutton [SUT88] puis Watkins en 1989 [WAT89]. C'est une méthode de programmation où il n'est nul besoin de lui indiquer quoi faire, l'agent se charge d'apprendre par lui même en renforçant les actions qui s'avèrent les meilleures [TOU99].

Dans ce chapitre, nous présentons une introduction à l'apprentissage par renforcement avec ses fondements mathématiques (MDP) et nous décrivons les principaux algorithmes utilisés dans ce domaine de l'apprentissage automatique.

3.2 Principe

Dans la phase d'interaction, un agent (le robot) est situé dans un environnement. Le robot perçoit son environnement à l'instant t par une observation O_t qui va être traduite en un état S_t par un interpréteur, et peut agir en exécutant l'action A_t, et reçoit par conséquence une récompense R_t. Par contre, l'environnement reçoit l'action A_t exécutée par l'agent, et envoie la récompense immédiate R_{t+1} et l'observation O_{t+1} qui sera traduite en S_{t+1} [SIL15].

L'objectif est d'associer à chaque état s_t une action a_t qui permet de maximiser la récompense. L'intégration de cette approche permet de contrôler et d'améliorer les performances d'une stratégie de prise de décision pour le robot ; dans une application donnée ; après chaque interaction avec l'environnement dans lequel il évolue. Ces interactions ou observations sont généralement des mesures de capteurs [SUT98, CHE14].

L'interaction de l'agent avec son environnement est représentée sur la figure 3.1

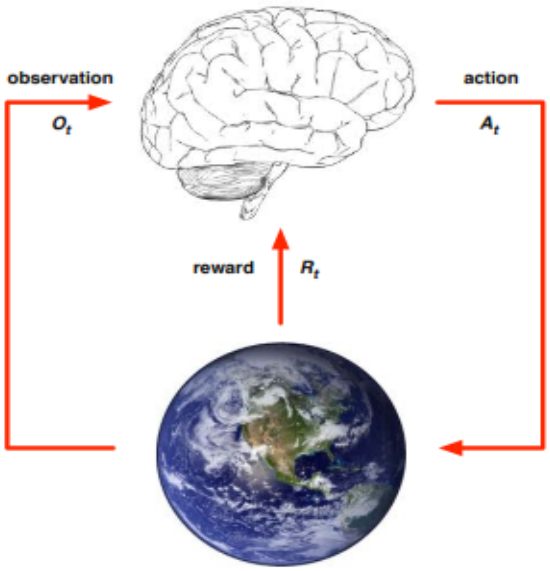


Figure 3.1 – Modèle standard de l'interaction agent/environnement

3.3 Processus de décision markovien (MDP)

Un MDP et ses dérivées représentent un outil permettant de décrire certains comportements dynamiques de problèmes décisionnels où un agent et les résultats ne sont pas déterminés. Dans les MDPs, l'évolution du système est supposée correspondre à un processus markovien. Il s'agit d'une classe de modèles de Markov.

Pour notre cas, les problèmes d'apprentissage par renforcement satisfaisant la propriété de Markov peuvent être décrits comme des MDPs. Pour cela, on suppose que l'évolution du robot dans son environnement est régie par un MDP qui décrit l'évolution de l'état et de la récompense en fonction des actions du robot [FIL16].

3.3.1 Formalisation

Un MDP est défini par un quintuplet $< \mathbb{S}, \mathbb{A}, \mathbb{P}, \mathbb{R}, \gamma >$ tel que :

— $\mathbb{S}$: un ensemble fini d'états.

— $\mathbb{A}$: un ensemble fini d'actions.

On note $\mathbb{A}(s)$ l'ensemble des actions possibles dans l'état s.

Avec :

$$\mathbb{A} = \bigcup_{s \in \mathbb{S}} \mathbb{A}(s)$$

— $\mathbb{P}$: une matrice (ou fonction) de probabilité de transition d'états.

$$\mathbb{P}^a_{s,s'} = P[S_{t+1} = s' \mid S_t = s, A_t = a] \tag{3.1}$$

Cette matrice donne la probabilité de passer à l'état s' lorsque l'agent effectue l'action a dans l'état s.

— $\mathbb{R}$: une fonction de récompense.

$$\mathbb{R}^a_s = E[R_{t+1} \mid S_t = s, A_t = a] \tag{3.2}$$

Elle donne la récompense moyenne lorsque l'agent exécute l'action a dans l'état s.

— γ : facteur de dépréciation $\gamma \in [0, 1]$.

Il peut exister un ensemble d'états finaux $\mathbb{F} \subset \mathbb{S}$; quand l'agent atteint l'un de ces états, sa tâche est terminée. C'est une tâche épisodique ou à horizon fini. Généralement les quatre ensembles sont fixes au cours du temps sinon, on dirait que le problème est non stationnaire [CHE14].

3.3.2 Propriété de Markov

C'est la propriété fondamentale qui doit être respectée par tous les MDPs. On dit que l'état (ou environnement) est markovien ou satisfait la propriété de Markov si l'état courant et le retour courant ne dépendent que de l'état précédent et de l'action qui vient d'être émise.

Définition 3.1. *Un état S_t est Markov si et seulement si :*

$$P[S_{t+1} \mid S_t] = P[S_{t+1} \mid S_1, ..., S_t] \tag{3.3}$$

Le futur est indépendant du passé ayant donné le présent

En d'autres termes, une fois l'état actuel est connu, tout l'historique pourrait être jeté. L'état courant rassemble toutes les informations utiles issues de l'historique. Il n'est alors nul besoin de mémoire pour prendre des décisions au mieux : seule la connaissance de l'état courant est utile. [SIL15, PRE07].

3.4 Notions de base de l'apprentissage par renforcement

3.4.1 Agent, Environnement et Historique

3.4.1.1 État de l'agent

L'état de l'agent à l'instant t, noté S_t^a, définit la représentation interne de l'agent exécuteur d'actions. Cette représentation englobe les informations qu'il utilise pour choisir la nouvelle action et ce sont les mêmes informations utilisées par les algorithmes d'apprentissage par renforcement [SIL15].

3.4.1.2 État de l'environnement

L'état de l'environnement à l'instant t, noté S_t^e, définit la représentation interne de l'environnement. Elle reflète les données qu'il utilise pour choisir le nouveau couple observation/récompense.

S_t^e n'est pas visible pour l'agent, et même s'il l'est, il pourrait contenir des informations hors de propos [SIL15].

3.4.1.3 Historique

Pour un problème d'apprentissage par renforcement, l'historique représente une séquence d'observations, d'actions et de récompenses. Il contient toutes les variables observables jusqu'à l'instant t. Ce qui rend les actions de l'agent et les observations/récompenses de l'environnement en dépendent.

$$H_t = A_0, O_1, R_1, A_1, ..., A_{t-1}, O_t, R_t, A_t$$

L'interpréteur dans ce contexte est l'outil utilisé pour déterminer l'état suivant. Formellement, il est en fonction de l'historique [AUS14].

$$S_t = f(H_t)$$

3.4.2 Politique

La politique π définit la manière dont l'agent se comporte dans un état donné s. En d'autres termes, elle détermine la règle de sélection des actions successives d'un agent dans l'état actuel du système (non l'historique). Une politique est dite déterministe (stationnaire) si elle spécifie la même action chaque fois qu'un état est visité. Une politique est dite non-déterministe (stochastique) si elle spécifie une action par la même distribution de probabilité sur l'ensemble des actions à chaque fois qu'un état est visité [BOU09].

$$\pi(s \mid a) = P[A_t = a \mid S_t = s] \tag{3.4}$$

Pour chaque état s, π donne la probabilité de choisir une action a.

3.4.3 Fonction de retour

Comme nous l'avons vu dans l'introduction, le but de l'agent est de maximiser les récompenses futures cumulées. La fonction de retour, ou tout simplement le retour G_t, est une mesure à long terme des récompenses. Il existe trois modèles principaux [BOU09] :

3.4.3.1 Retour à horizon fini

Dans ce cas, l'horizon correspond à un nombre fini de pas dans le futur. Il existe un état terminal et la suite d'actions entre l'état initial et l'état terminal est appelée un épisode.

$$G_t = \sum_{k=1}^{h} R_{t+k-1} = R_t + ... + R_{t+h-1} \tag{3.5}$$

Où h est le nombre d'itérations avant l'état terminal.

3.4.3.2 Retour à horizon infini

On l'utilise quand la suite des actions est infinie. On introduit un facteur appelé facteur de dépréciation. Le retour G_t est donc défini comme suit :

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} = R_{t+1} + \gamma R_{t+2} + ... \tag{3.6}$$

Le facteur γ permet de régler l'importance que l'on donne aux retours futurs par rapport aux retours immédiats, tel que si :

— γ près de 1 : les récompenses futures sont prises en compte.

— γ près de 0 : les récompenses futures sont négligées.

3.4.3.3 Retour moyen

L'agent cherche à maximiser la moyenne de tous les renforcements qu'il recevra :

$$G_t = \lim_{n\to\infty} \frac{1}{n} \sum_{k=0}^{n} \gamma^k R_{t+k} \tag{3.7}$$

3.4.4 Fonction de valeur

Dans cette section, nous allons définir la fonction de valeur associée à une politique donnée pour un MDP donné. Elle associe à chaque état (action) de l'environnement l'expectation du retour G_t à partir de l'instant t. Cette fonction est décrite sous forme de deux fonctions [SIL15] :

3.4.4.1 Fonction de valeur d'état

La fonction de valeur d'état s pour une politique π ; notée $V_\pi(s)$; d'un MDP est définie par la récompense espérée en démarrant de l'état s et en suivant la politique π par la suite. En utilisant le modèle de retour à horizon infini, la fonction V_π se définit alors comme suit :

$$V_\pi(s) = E_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s \right] \tag{3.8}$$

Où E_π représente l'espérance lorsqu'on suit la politique π.

3.4.4.2 Fonction de valeur d'action

Appelée aussi fonction de qualité ou fonction de valeur état-action. Cette fonction, notée $Q_\pi(s,a)$ est la récompense espérée d'un couple état-action lorsque l'on part de l'état s et que l'on exécute l'action a en s et que l'on suit la politique π par la suite. En utilisant le modèle de retour à horizon infini, $Q_\pi(s,a)$ se définit comme suit :

$$Q_\pi(s,a) = E_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a \right] \tag{3.9}$$

3.4.5 Modèle de l'environnement

Un modèle est une représentation de l'environnement et ainsi que sa dynamique. Il permet de prédire les résultats des actions ultérieures à l'aide des fonctions de :

- Transition $\mathbb{P}^a_{s,s'}$: prédire l'état suivant.
- Récompense $\mathbb{R}^a_s$: prédire la récompense suivante.

3.5 L'équation de Bellman

Les équations de $V_\pi(s), Q_\pi(s,a)$ pour une politique π vérifient une relation fondamentale appelée équation de Bellman [BEL57]. Pour tout $s \in \mathbb{S}$, l'équation de Bellman pour V_π est donnée par :

$$V_\pi(s) = \sum_{a \in \mathbb{A}(s)} \pi(a \mid s) \left(\mathbb{R}_s^a + \gamma \sum_{s' \in \mathbb{S}} \mathbb{P}_{s,s'}^a V_\pi(s') \right) \tag{3.10}$$

De la même manière, pour tout $s \in \mathbb{S}$ et tout $a \in \mathbb{A}(s)$ l'équation de Bellman pour Q_π est donnée par :

$$Q_\pi(s,a) = \mathbb{R}_s^a + \gamma \sum_{s' \in \mathbb{S}} \mathbb{P}_{s,s'}^a \sum_{a' \in \mathbb{A}(s')} \pi(a' \mid s') Q_\pi(s',a') \tag{3.11}$$

Remarquons qu'on a une forte relation entre ces deux fonctions ; où $V_\pi(s)$ étant la moyenne des $Q_\pi(s,a)$ pondérées par la probabilité de chaque action :

$$V_\pi(s) = \sum_{a \in \mathbb{A}(s)} \pi(a \mid s) Q_\pi(s,a) \tag{3.12}$$

La propriété essentielle de ces fonctions de valeur qui permet de construire les différents algorithmes d'apprentissage est la récurrence ; où cette fonction peut se définir en fonction de la valeur de tous les états de l'ensemble $\mathbb{S}$. Cette relation est connue par l'équation de Bellman [FIL16]

3.6 L'équation d'optimalité de Bellman

Le but de l'apprentissage par renforcement va être de trouver la politique optimale $\pi^\star$ maximisant la récompense à long terme [FIL16].

3.6.1 Politique optimale

Lorsque l'on est dans l'état s, la politique optimale d'un MDP consiste à choisir l'action a qui maximise la fonction de qualité. La fonction de valeur permet également de comparer les politiques. On dit qu'une politique π est optimale si et seulement si pour toute politique π' :

$$V_\pi(s) \geq V_{\pi'}(s), \forall s \in \mathbb{S} \tag{3.13}$$

— une telle politique est dite optimale et on la note $\pi^\star$

— pour tout MDP fini, il existe au moins une politique optimale qui se présente sous la forme d'une politique déterministe.

3.6.2 Fonction de valeur optimale

- La fonction de valeur d'état optimale $V^\star$ représente la fonction de valeur d'état maximale de toutes les politiques.

$$V^\star = \max_\pi V_\pi(s) \quad \forall s \in \mathbb{S} \tag{3.14}$$

- La fonction de valeur d'action optimale $Q^\star$ représente la fonction de valeur d'action maximale de toutes les politiques.

$$Q^\star = \max_\pi Q_\pi(s,a) \quad \forall s \in \mathbb{S}, \forall a \in \mathbb{A}(s) \tag{3.15}$$

Il existe une relation entre les deux fonctions optimales :

$$V^\star(s) = \max_{a \in \mathbb{A}(s)} Q^\star(s,a) \tag{3.16}$$

Le MDP est résolu, si on connaît la valeur de fonction optimale. Dans ce cas $Q^\star$ est connue, la politique optimale est déduite à partir des actions gloutonnes :

$$\pi^\star(a \mid s) = arg \max_{a \in \mathbb{A}(s)} Q^\star(s,a) \tag{3.17}$$

Intuitivement, cette fonction décrite par l'opérateur $\max_{a \in \mathbb{A}(s)}$ traduit le fait que la politique optimale choisit l'action qui maximise le retour [FIL16].

Tout le problème de l'apprentissage va donc être d'estimer $V^\star$ ou $Q^\star$ pour en déduire une politique optimale. On peut remarquer que si l'on connaît complètement le problème (c'est à dire si l'on connaît $\mathbb{P}^a_{s,s'}$ et $\mathbb{R}^a_s$), il est possible de calculer directement $V^\star$ ou $Q^\star$ en résolvant l'équation de Bellman. Cette solution est peu applicable en robotique car l'environnement est en général inconnu, et de plus, elle ne correspond pas à des caractéristiques souhaitables de l'apprentissage telles que la capacité à apprendre par essais et erreurs [SUT98, FIL16]

Il est également possible de décrire une relation de récurrence pour la fonction de valeur optimale qui sera légèrement différente de l'équation de Bellman. On parle alors d'équation d'optimalité de Bellman, qui peut s'écrire sous deux formes :

- **Équation d'optimalité de Bellman pour $V^\star$:** elle est basée sur les relations (3.10 et 3.14)

$$V^\star(s) = \max_{a \in \mathbb{A}(s)} \left[\mathbb{R}^a_s + \gamma \sum_{s' \in \mathbb{S}} \mathbb{P}^a_{s,s'} V^\star(s') \right] \tag{3.18}$$

- **Équation d'optimalité de Bellman pour $Q^\star$:** elle est basée sur les relations (3.11 et 3.15)

$$Q^\star(s,a) = \mathbb{R}^a_s + \gamma \sum_{s' \in \mathbb{S}} \mathbb{P}^a_{s,s'} \max_{a' \in \mathbb{A}(s)} Q^\star(s',a') \tag{3.19}$$

— l'équation d'optimalité de Bellman est non linéaire et donc ne possède ; en général ; pas une solution exacte.

3.7 Méthodes de l'apprentissage par renforcement

3.7.1 Méthodes de la programmation dynamique (DP)

La programmation dynamique (DP) [BEL57] est un ensemble de techniques algorithmiques permettant de calculer une politique optimale dans un MDP connu en utilisant les propriétés des équations de Bellman. L'idée clé de DP est basée sur l'utilisation de fonction de valeur dans le but d'estimer $V^\star$ afin d'en déduire $\pi^\star$ [SUT98, FIL16].

Nous présentons dans ce qui suit deux algorithmes de DP, l'itération de la valeur et l'itération de la politique. Cette dernière a deux fonctions, qui sont l'évaluation et l'amélioration de la politique.

3.7.1.1 Algorithme d'itération de la politique (Policy Iteration)

L'idée de l'algorithme d'itération de la politique est d'évaluer une certaine politique π, et en fonction de cette évaluation, on calcule une nouvelle politique qui serait meilleure que l'ancienne et ainsi de suite.

- **Fonction d'évaluation de la politique :** Elle utilise l'équation 3.10 comme étape de mise-à-jour permettant de calculer une suite de fonctions V_k convergeant vers V_π.

$$V_{k+1}(s) = \sum_{a \in \mathbb{A}(s)} \pi(a \mid s) \left(\mathbb{R}_s^a + \gamma \sum_{s' \in \mathbb{S}} \mathbb{P}_{s,s'}^a V_k(s') \right) \tag{3.20}$$

  Les V_k représentent la suite des estimations de V_π engendrées par la fonction d'évaluation de la politique. Autrement dit, à chaque itération, V_k s'approche de la valeur recherchée qui est V_π [PRE07]

  — Méthode de résolution :

  1. Initialisation arbitraire de V_0.
  2. Calcul successif de $V_0, V_1, V_2, \ldots$.
  3. Arrêt quand un critère de convergence est rempli.

  Le plus souvent on arrête le calcul lorsque $\max_{s \in \mathbb{S}} |V_{k+1} - V_k|$ est suffisamment petit à un seuil de précision ξ [GER07]. Car la convergence vers V_π est asymptotique, *i.e.* il faut une infinité d'itérations pour obtenir la vraie valeur de V_π.

  Pour un MDP et une politique fixés, l'algorithme d'évaluation de la politique converge asymptotiquement vers V_π.

- **Fonction d'amélioration de la politique :** Son principe d'amélioration consiste à construire une meilleure politique π' à partir de π. Considérons un état $s \in \mathbb{S}$, et supposons que π est telle que dans s l'action a est choisie. Afin de déterminer si une autre action a' est meilleure que a dans s nous calculons la fonction de valeur d'action qui estime l'utilité de prendre a' dans s et suivre la politique π par la suite :

$$Q_\pi(s,a') = \mathbb{R}_s^a + \gamma \sum_{s' \in \mathbb{S}} \mathbb{P}_{s,s'}^a V_\pi(s) \tag{3.21}$$

Plus généralement, Si $\forall s \in \mathbb{S}, Q_\pi(s,a')) \geq V_\pi(s)$ Alors $V_{\pi'}(s) \geq V_\pi(s)$ et $\pi' > \pi$, et la nouvelle politique π' est une amélioration de la politique originale π. Par l'extension de cette idée à tous les états, on obtient l'algorithme pour l'étape d'amélioration, qui permet de trouver une nouvelle politique gloutonne π' telle que :

$$\pi'(s) = arg \max_{a \in \mathbb{A}(s)} Q_\pi(s,a) \tag{3.22}$$

L'amélioration de la politique produit des politiques strictement meilleures et quand ce n'est plus le cas, c'est que la politique considérée est déja optimale [GER07].

Les fonctions vues précédemment fournissent l'idée d'un algorithme pour produire une politique optimale pour un MDP donné. Donc, partant d'une politique π_0 le processus d'itération de la politique mène à une politique optimale $\pi^\star$.

— Méthode de résolution :

1. Démarrer avec une politique π_0.
2. Calculer sa fonction de valeur V_0.
3. Construire une politique meilleure que la précédente.
4. Refaire ces étapes jusqu'à la politique optimale $\pi^\star$.

$$\pi_0 \xrightarrow{e} V_{\pi_0} \xrightarrow{a} \pi_1 \xrightarrow{e} V_{\pi_1} \xrightarrow{a} \pi_2 \xrightarrow{e} \ldots \xrightarrow{a} \pi^\star \xrightarrow{e} V^\star$$

— $\xrightarrow{e}$ représente une évaluation.

— $\xrightarrow{a}$ représente une amélioration.

L'algorithme d'itération de la politique converge vers $\pi^\star$ au bout d'un nombre fini d'itérations [PRE07]. La figure 3.2 résume le processus d'itération de la politique.

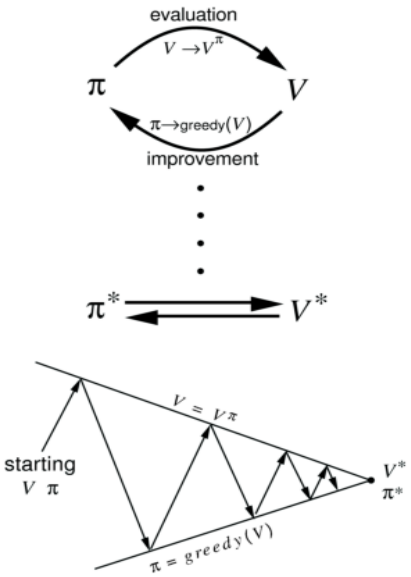


Figure 3.2 – Schéma de l'algorithme Policy Iteration

3.7.1.2 Algorithme d'itération de la valeur (Value Iteration)

Dans cette méthode, on ne cherche pas à déterminer explicitement la politique mais l'action de valeur optimale pour chaque état.

On définit alors $Q_\pi(s,a)$ et la règle de mise à jour pour l'itération sur les valeurs devient alors :

$$V_{k+1}(s) = \max_{a\in\mathbb{A}(s)} Q_\pi(s,a) = \max_{a\in\mathbb{A}(s)} \left(\mathbb{R}_s^a + \gamma \sum_{s'\in\mathbb{S}} \mathbb{P}_{s,s'}^a V_k(s') \right) \tag{3.23}$$

Pour tout MDP, l'algorithme d'itération de la valeur converge vers $V^\star$ et donc converge vers $\pi^\star$[PRE07].

Les algorithmes que l'on vient de voir sont simples à implémenter mais reposent sur la connaissance de l'environnement. Donc DP a une capacité limitée d'apprentissage et le fait de devoir connaître l'environnement pour apprendre une stratégie rend les méthodes de programmation dynamique peu utiles en robotique. En outre, elle pose des problèmes en temps de calcul [SUT98].

Dans ce qui suit, l'agent ne dispose pas d'un MDP, mais il cherche de le construire en interagissant avec l'environnement.

3.7.2 Méthode de Monte-Carlo (MC)

La méthode de Monte-Carlo va utiliser les mêmes idées de DP, mais en ayant recours à des expériences réalisées dans l'environnement plutôt qu'à un modèle. L'estimation de la fonction de valeur est réalisée à partir d'un ensemble de séquences réalisées par l'agent [FIL16].

Une façon simple de réaliser cette estimation consiste à simuler des épisodes à partir de chacun des états s jusqu'à l'état terminal. Si l'on note $G_k(s)$ la somme cumulée obtenue le long de la séquence k en suivant la politique π, alors une estimation de la fonction de valeur en s après $k+1$ séquences est donnée par :

$$\forall s \in \mathbb{S}, V_{k+1}(s) = \frac{G_1(s) + G_2(s) + ... + G_k(s) + G_{k+1}(s)}{k+1} \tag{3.24}$$

Pour ne pas stocker chacun des G_k reçus, on montre que tout simplement qu'un tel calcul peut se reformuler de manière incrémentale :

$$\forall s \in \mathbb{S}, V_{k+1}(s) = V_k(s) + \frac{1}{k+1}\left[G_{k+1}(s) - V_k(s)\right] \tag{3.25}$$

La dernière formule vérifie pour un nombre de séquences infini :

$$\lim_{k \to \infty} V_k(s) = V_\pi(s) \tag{3.26}$$

Cette méthode s'applique de la même façon pour la fonction Q.

$$\forall s \in \mathbb{S}, \forall a \in \mathbb{A}(s), Q_{k+1}(s,a) = Q_k(s,a) + \alpha\left[G_{k+1}(s) - Q_k(s,a)\right] \tag{3.27}$$

Les deux méthodes précédentes ont chacune un avantage important. La méthode de Monte-Carlo permet d'apprendre à partir d'expériences et la programmation dynamique pour sa part possède la propriété intéressante d'utiliser les estimations des états successeurs pour estimer la valeur d'un état.

3.7.3 Méthodes de différence temporelle (TD)

Nous allons nous tourner à présent vers les méthodes de différence temporelle (Temporel Difference methods) qui combinent les deux méthodes précédentes. Cette caractéristique permet une convergence beaucoup plus rapide (en termes de séquences) que la méthode de Monte-Carlo. Les méthodes d'apprentissage par différences temporelles vont réunir ces deux propriétés et constituent les méthodes les plus utiles en robotique [SUT88].

Si les estimations des fonctions de valeur aux états s_t et s_{t+1} notées $V(s_t)$ et $V(s_{t+1})$, étaient exactes, on aurait :

$$V(s_t) = R_t + \gamma R_{t+1} + ...$$

$$V(s_{t+1}) = R_{t+1} + \gamma R_{t+2} + ...$$

Donc on aura :

$$V(s_t) = R_t + \gamma V(s_{t+1}) \Rightarrow R_t + \gamma V(s_{t+1}) - V(s_t) \rightarrow 0 \tag{3.28}$$

L'équation 3.29 définit la différence temporelle δ_k.

$$\delta_k = R_t + \gamma V(s_{t+1}) - V(s_t) \tag{3.29}$$

On voit que cet algorithme repose sur une comparaison (l'erreur de différence temporelle δ_k) entre la récompense que l'on reçoit effectivement $V(s_t)$ et la récompense que l'on s'attend à recevoir $V(s_{t+1})$ en fonction des estimations construites précédemment [CHE14]. La méthode TD a conduit à plusieurs algorithmes, on essaye d'en citer trois :

3.7.3.1 L'algorithme TD(0)

L'algorithme le plus simple de différence temporelle est appelé TD(0). Lors des premiers épisodes les corrections sont importantes car l'estimation de $V_\pi(s)$ est mauvaise ; cependant, au fur et à mesure, la précision de cette estimation augmente et les corrections deviennent moins significatives. Pour cela le taux d'apprentissage α doit être initialement proche de 1, et diminue ensuite pour tendre vers 0. Son équation de mise à jour est la suivante :

$$V_{k+1}(s_t) \leftarrow (1 - \alpha_k(s_t))V_k(s_t) + \alpha_k(s_t)\,(R_t + \gamma V_k(s_{t+1})) \tag{3.30}$$

où $\alpha_k(s_t) \in [0,1]$ représente le taux d'apprentissage de l'itération k associé à l'état s_t.

Pour un MDP $< \mathbb{S}, \mathbb{A} >$ et une politique π, l'algorithme TD(0) converge vers V_π à condition que :

— chaque état soit visité une infinité de fois théoriquement.

— le taux d'apprentissage respecte les conditions suivantes :

$$\sum_k \alpha_k(s_t) = +\infty \quad \text{et} \quad \sum_k \alpha_k^2(s_t) < +\infty \quad \forall s \in \mathbb{S}$$

avec

$$\alpha_k(s_t) = \frac{1}{1+\text{nombre de visites de l'état } s_t \text{ depuis le début de l'épisode}}$$

3.7.3.2 L'algorithme SARSA

L'algorithme SARSA travaille sur les valeurs des couples (s,a) plutôt que sur la valeur des états [PRE07]. Son équation de mise à jour est identique à celle de TD(0) en remplaçant la fonction de valeur d'état par la fonction de valeur d'action (pour traiter un problème d'amélioration) :

$$Q(s_t,a_t) \leftarrow (1-\alpha)Q(s_t,a_t) + \alpha\left[R_t + \gamma Q(s_{t+1},a_{t+1})\right] \tag{3.31}$$

L'information nécessaire pour réaliser une telle mise à jour est la réalisation d'une transition $S_t,A_t,R_t,S_{t+1},A_{t+1}$, d'où découle le nom de l'algorithme [CHE14]. Effectuer ces mises à jour implique que l'agent détermine avec un pas de regard en avant quelle est l'action a_{t+1} qu'il réalisera lors du pas de temps suivant, lorsque l'action a_t dans l'état s_t l'aura conduit dans l'état s_{t+1} [SUT98, PRE07].

L'algorithme SARSA converge vers une politique quasi-optimale $\pi^\star$.

3.7.3.3 L'algorithme Q-learning

L'algorithme du Q-learning a été introduit par Watkins [WAT89]. Il est sans doute l'algorithme d'apprentissage par renforcement le plus connu et le plus utilisé en raison des preuves formelles de convergence et des applications existantes [SUT98]. L'algorithme Q-learning se présente comme une simplication de l'algorithme SARSA par le fait qu'il n'est plus nécessaire pour l'appliquer de déterminer un pas de temps à l'avance quelle sera l'action réalisée au pas de temps suivant.

Le principe du Q-learning consiste à apprendre la fonction de qualité en interagissant avec l'environnement. Trois fonctions principales participent au Q-learning (figure 3.3) : une fonction d'évaluation, une fonction de renforcement et une fonction de mise à jour [TOU99].

- **Fonction d'évaluation :** Les résultats de l'algorithme Q-learning sont stockés dans la fonction qualité (état-action), qui est souvent sous la forme de table à deux dimensions : situation et action. La fonction d'évaluation parcourt, pour une situation donnée, les valeurs de Q associées aux actions et sélectionne l'action avec la plus grande qualité.
- **Fonction de renforcement :** Cette fonction fournit pour chaque paire une évaluation qualitative de son intérêt par rapport au comportement désiré par l'opérateur. Il est important de remarquer que le retour de la fonction de renforcement estime l'intérêt de la situation perçue, ce retour est utilisé pour définir l'utilité d'effectuer l'action précédente dans la situation précédente.

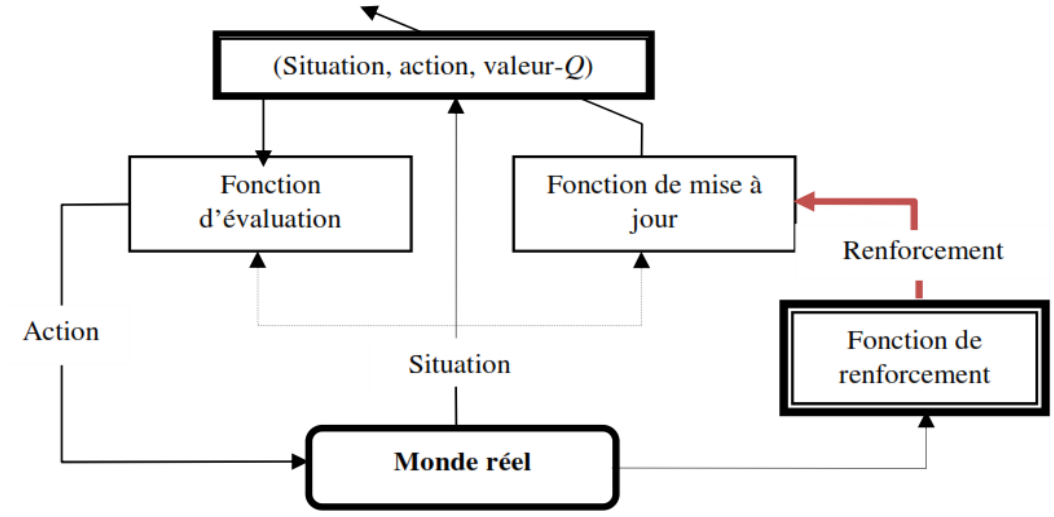


Figure 3.3 – Modèle de l'algorithme Q-learning

- **Fonction de mise à jour :** Le principe de Q-learning est de mettre à jour itérativement le fonction courante Q_t pour le couple (s_t, a_t) à la suite de chaque transition (s_t, a_t, R_t, s_{t+1}). Cette mise à jour se fait sur la base de l'observation des transitions instantanées et de leurs récompenses associées par l'équation suivante :

$$Q(s_t, a_t) \leftarrow (1-\alpha)Q(s_t, a_t) + \alpha \left[R_t + \gamma \max_{a \in \mathbb{A}(s_{t+1})} Q(s_{t+1}, a) \right] \tag{3.32}$$

L'algorithme Q-learning converge vers une politique optimale $\pi^\star$.

3.8 Dilemme exploration/exploitation

L'un des défis de RL est le compromis à trouver entre exploration et exploitation. Pour obtenir plus de récompense, un agent RL doit préférer des actions qu'il a essayées avant et trouvées être efficaces par les récompenses qu'elles ont entraînées. L'agent doit exploiter ce qu'il connaît déjà pour obtenir une récompense, mais il doit aussi explorer pour faire de meilleures actions dans le futur. Le dilemme est que ni l'exploration ni l'exploitation ne peuvent être poursuivies exclusivement sans faillir à la tâche [BOU10]. Naturellement, l'exploration doit initialement être importante (quand l'apprentissage est encore très partiel, on explore) puis diminuer au profit de l'exploitation quand l'apprentissage a été effectué pendant une période suffisamment longue, donc l'agent doit suivre une politique d'exploration/exploitation lui indiquant quelle action entreprendre dans un état donné [SUT98, PRE07]. Plusieurs stratégies sont utilisées classiquement :

3.8.1 Gloutonne (Greedy)

Elle consiste toujours à choisir l'action qui, à chaque état, maximise la fonction de qualité. Elle est utilisée une fois pour obtenir la politique optimale.

$$a_{gloutonne} = \max_{a \in \mathbb{A}(s)} Q(s,a) \tag{3.33}$$

3.8.2 ε-Gloutonne (ε-Greedy)

Elle consiste à choisir l'action gloutonne avec une probabilité $1-\varepsilon$, et à choisir une action au hasard avec une probabilité ε. Un cas particulier est la sélection 0-gloutonne qui consiste à choisir une action gloutonne.

3.9 Conclusion

Ce chapitre nous a permis de nous initier au domaine de l'apprentissage par renforcement, de présenter ses fondements théoriques et de décrire les principaux algorithmes standards. Ces derniers sont basés sur les processus markoviens et peuvent utiliser plusieurs approches. Ils permettent de travailler directement dans des MDPs continus en utilisant un approximateur de fonction comme les réseaux de neurones et les systèmes d'inférence floue pour résoudre plusieurs problèmes et notamment en robotique mobile (la navigation d'un robot mobile).

Toutefois, l'apprentissage par renforcement étant en plein développement et de nombreuses méthodes n'ont pu être présentées. On peut citer par exemple l'algorithme de l'acteur-critique, les travaux effectués sur les méthodes d'apprentissage par renforcement indirectes, citons Dyna, Prioritized Sweeping et E^3. Aussi les algorithmes d'apprentissage par renforcement avec les traces d'éligibilité : $TD(\lambda), SARSA(\lambda)$ et $Q(\lambda)$.

Nous présenterons, au chapitre 4, comment ces algorithmes peuvent être utilisés pour le réglage des paramètres d'un régulateur flou.

Chapitre 4

Navigation Autonome

4.1 Introduction

La navigation autonome d'un robot mobile nécessite une structure de contrôle visant à le doter d'un mécanisme de raisonnement lui permettant de se déplacer de manière autonome [TUN97]. Dans la majorité des applications de la logique floue pour la navigation, le modèle mathématique de l'environnement n'est pas nécessaire pour la conception du régulateur [JAN07]. Pour cela, les stratégies réactives basées sur des informations relatives aux interactions ont été proposées comme une alternative aux stratégies classiques [KHA86]. L'idée principale de cette approche est d'identifier des comportements différents selon les données sensorielles. Cette approche a été proposée par Brooks [BRO86] avec son architecture de subsumption.

Malgré le succès de la logique floue, plusieurs méthodes de conception, permettant le réglage des différents paramètres d'un régulateur flou ont été proposées dans la littérature [BER92, GOD97, BOU09]. Elles ont donné naissance aux systèmes hybrides. Les algorithmes RL sont capables d'automatiser le processus d'ajustement des paramètres des FISs par la modification de la base des règles afin de générer des solutions optimales ou quasi-optimales.

Dans ce chapitre, nous allons traiter le problème de navigation d'un robot mobile dans des environnements inconnus en lui permettant le déplacement vers une position désirée tout en évitant les obstacles. D'une part, nous utilisons la logique floue et d'autre part l'apprentissage par renforcement (Q-learning flou) pour le réglage des régulateurs flous (les conclusions des règles floues) dans des espaces continus. Pour ce faire, nous développons une architecture de contrôle réactive à base de deux comportements flous (convergence vers un but, évitement d'obstacles)

4.2 Navigation réactive par logique floue

Le principe de la navigation réactive est basé sur la décomposition de la tâche de navigation globale en un ensemble de comportements élémentaires d'action (comportement 1, comportement 2,..., comportement n). Ces comportements sont indispensables à l'exécution des sous-tâches spécifiques pour un robot mobile (éviter les obstacles, atteindre un but). L'architecture de subsumption, présentée sur la figure 4.1, utilise une hiérarchie de comportements arrangés qui se déclenchent selon un ordre de priorité en fonction des données de perception (fusion des comportements).

La logique floue est donc utilisée pour la représentation et la coordination des comportements. Le système de commande du robot mobile est donc basé sur des comportements flous décomposés en sous-tâches simples (comportements indépendants), ou chaque bloc est considéré comme un régulateur flou défini par un ensemble de règles floues visant à atteindre un objectif [FAT06].

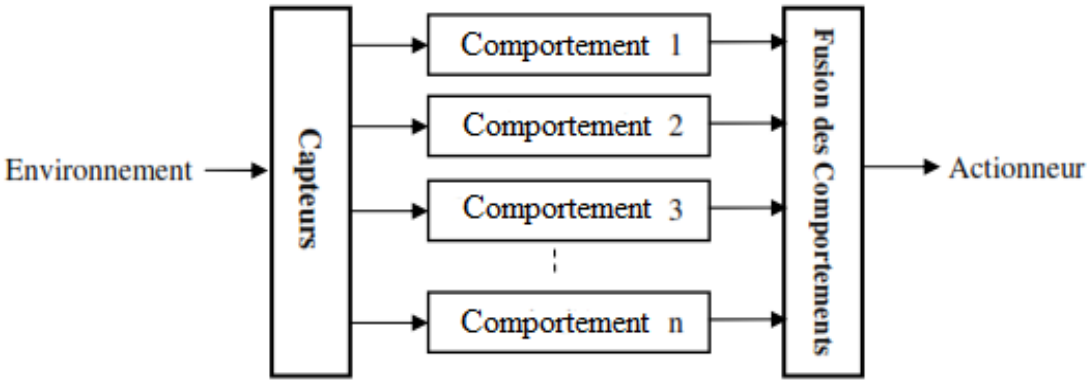


Figure 4.1 – Architecture de subsumption

4.3 Modélisation cinématique du robot différentiel

Une des configurations les plus utilisées pour les robots mobiles d'intérieur est la différentielle qui comporte deux roues (munis de deux moteurs avec deux encodeurs odométriques) commandées indépendamment et peuvent l'entraîner à une vitesse ω_D et ω_G. Une ou plusieurs roues folles sont ajoutées pour assurer sa stabilité et permet de plus au robot de tourner sur place. Cette possibilité permet de traiter dans certains cas le robot comme un robot holonome, ce qui va simplifier la planification de déplacement et la commande du robot [FIL16, BAR11].

Le robot est équilibré et son centre de masse se situe au milieu du segment qui relie les centres des deux roues. Nous noterons ce centre P, les coordonnées du robot dans l'espace de travail sont celles du centre de masse et sont notées (x,y). On suppose que l'origine de l'espace de travail est connue et correspond à $O=(0,0)$. On note $R_0(O,X,Y,Z)$ le repère fixe et $R_1(P,X_R,Y_R,Z_R)$ le repère mobile lié au robot [CAN18].

La figure 4.2 présente la configuration du robot mobile utilisé et les variables calculées dans l'espace de travail.

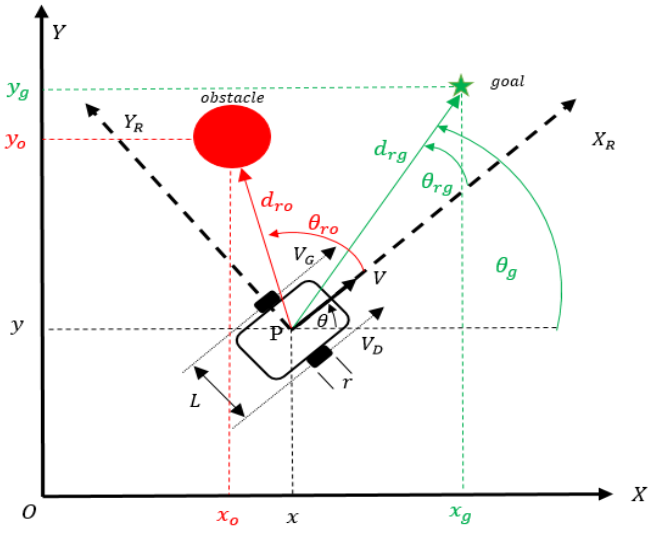


Figure 4.2 – Configuration du robot dans l'environnement

Avec :

— V : la vitesse linéaire du robot en m/s.

— Ω : la vitesse angulaire du robot en rad/s.

— $P=(x,y,\theta)$: la position et l'orientation du robot dans R_0.

— V_D, V_G : les vitesses linéaires respectives des roues droite et gauche en m/s.

— ω_D, ω_G : les vitesses de rotation respectives des roues droite et gauche en rad/s.

— r : le rayon des roues en m.

— L :la distance entre les axes des deux roues en m.

— d_{rg} : la distance du but par rapport au robot mobile en m.

— d_{ro} : l'angle du but par rapport au robot mobile en m.

— θ_{rg} : l'angle entre l'orientation du robot et celle de la cible en rad.

— θ_{ro} : l'angle entre l'orientation du robot et celle de l'obstacle en rad.

— θ_g : l'angle désiré en rad.

Le modèle cinématique du robot mobile est donné dans R_0 par la version discrète suivante :

$$\begin{aligned} x(k+1) &= x(k) + r\frac{T}{2}(\omega_D(k) + \omega_G(k))\cos(\theta(k)) \\ y(k+1) &= y(k) + r\frac{T}{2}(\omega_D(k) + \omega_G(k))\sin(\theta(k)) \\ \theta(k+1) &= \theta(k) + r\frac{T}{2}(\omega_D(k) - \omega_G(k)) \end{aligned} \tag{4.1}$$

On pose :

- T : temps d'échantillonnage
- $u_1 = \omega_D$: la commande de la roue droite.
- $u_2 = \omega_G$: la commande de la roue gauche.

4.4 Régulateurs flous de convergence vers un but

Dans ce travail, on utilise deux régulateurs flous Mamdani notés respectivement FLC20 et FLC49 selon le nobre de règles. La définition du point à atteindre est effectuée par l'intermédiaire de deux variables d'entrées (d_{rg}, θ_{rg}) et deux variables de sorties $\omega_D(u_1), \omega_G(u_2)$. Le schéma fonctionnel de commande proposé pour ce comportement est donné par la figure 4.3.

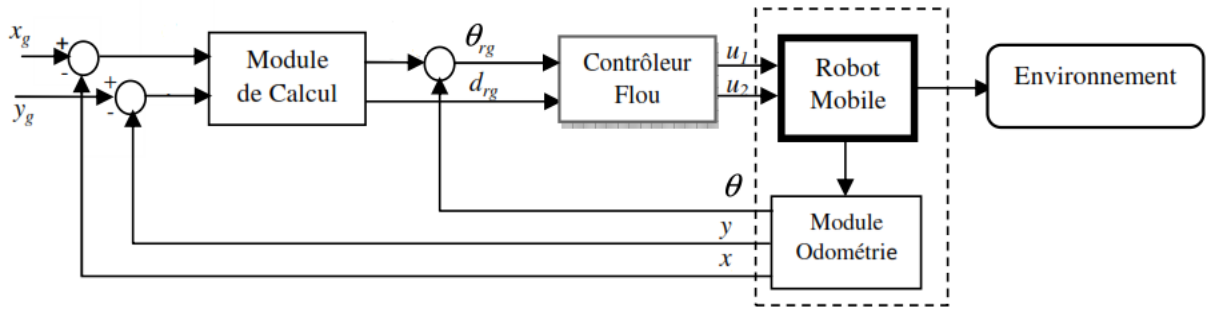


Figure 4.3 – Schéma de commande floue pour la convergence vers un but

Le module de calcul calcule la distance robot-but d_{rg} et l'angle désiré θ_g. Puis cette valeur d'angle est comparée avec l'orientation actuelle du robot θ pour calculer l'angle entre l'axe du robot et le but θ_{rg}. Ces calculs sont basés sur les équations suivantes :

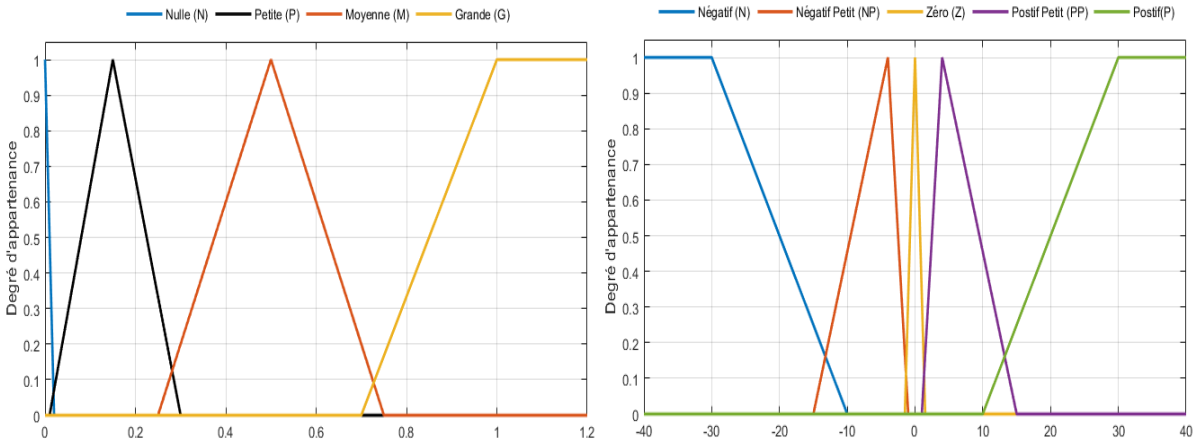
$$d_{rg} = \sqrt{(x_g - x)^2 + (y_g - y)^2} \tag{4.2}$$

$$\theta_{rg} = \arctan(\frac{y_g - y}{x_g - x}) - \theta \tag{4.3}$$

4.4.1 Synthèse du FLC20

4.4.1.1 Fonctions d'appartenances entrées/sorties

Dans cette étape, on spécifie les fonctions d'appartenance utilisées pour fuzzifier les variables d'entrée. Elles sont données sur la figure 4.4 avec des formes triangulaires et trapézoïdales. Les deux actions de commande sont réduites à des singletons (figure 4.5).



(a) Distance robot-but d_{rg} (m)

(b) Angle robot-but θ_{rg} *(degré)*

Figure 4.4 – Fonctions d'appartenance des entrées de FLC20 (Convergence)

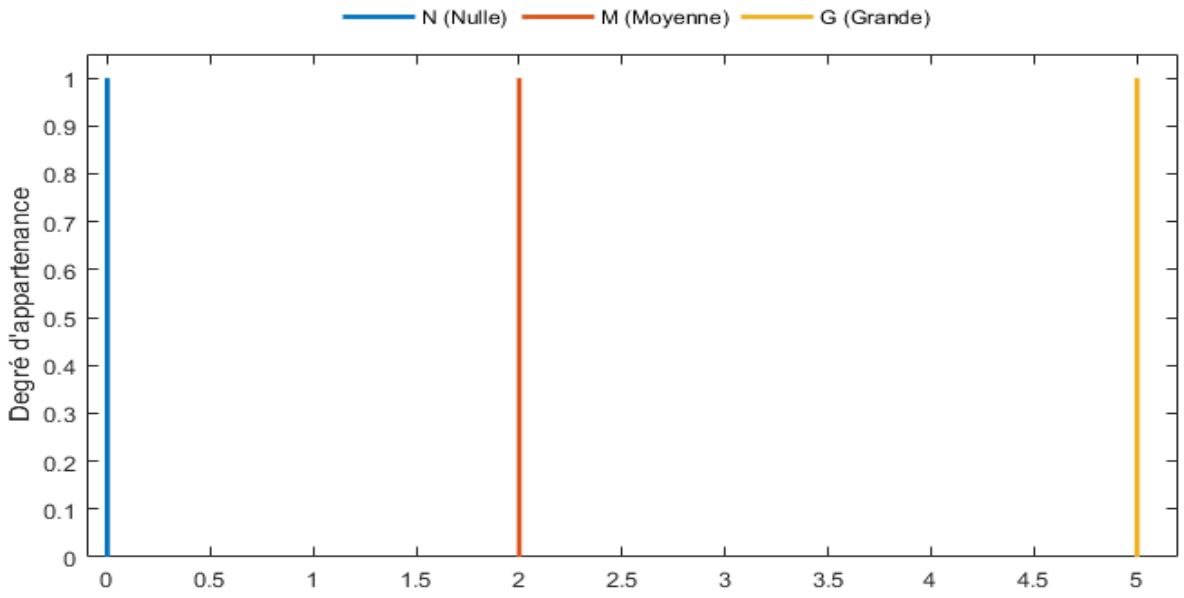


Figure 4.5 – Fonctions d'appartenance des sorties de FLC20 (Convergence)

4.4.1.2 Base des règles floues

Sur la base de la description du système avec des variables linguistiques et des fonctions d'appartenance (figures 4.4 et 4.5), on peut établir les règles d'inférence qui font appel à 20 règles représentant les liens entrées-sorties.

Le tableau (4.1) décrit les règles floues utilisées pour ce comportement.

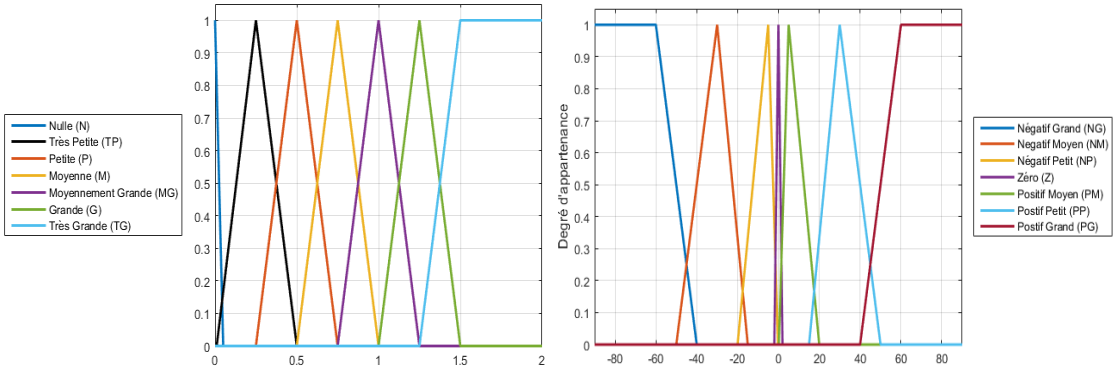
d_{rg} \ θ_{rg}		**N**	**NP**	**Z**	**PP**	**P**
N	ω_D	M	M	N	N	N
	ω_G	N	N	N	M	M
P	ω_D	G	G	M	M	M
	ω_G	M	M	M	G	G
M	ω_D	G	G	M	M	M
	ω_G	M	M	M	G	G
G	ω_D	G	G	M	M	M
	ω_G	M	M	G	G	G

Tableau 4.1 – Les règles de FLC20 (convergence vers un but)

4.4.2 Synthèse du FLC49

4.4.2.1 Fonctions d'appartenances entrées/sorties

Les fonctions d'appartenance utilisées pour fuzzifier les variables d'entrée sont données sur la figure 4.6. Les deux actions de commande sont réduites à des singletons (figure 4.7).



(a) Distance robot-but d_{rg} (m) (b) Angle robot-but θ_{rg} *(degré)*

Figure 4.6 – Fonctions d'appartenance des entrées de FLC49 (Convergence)

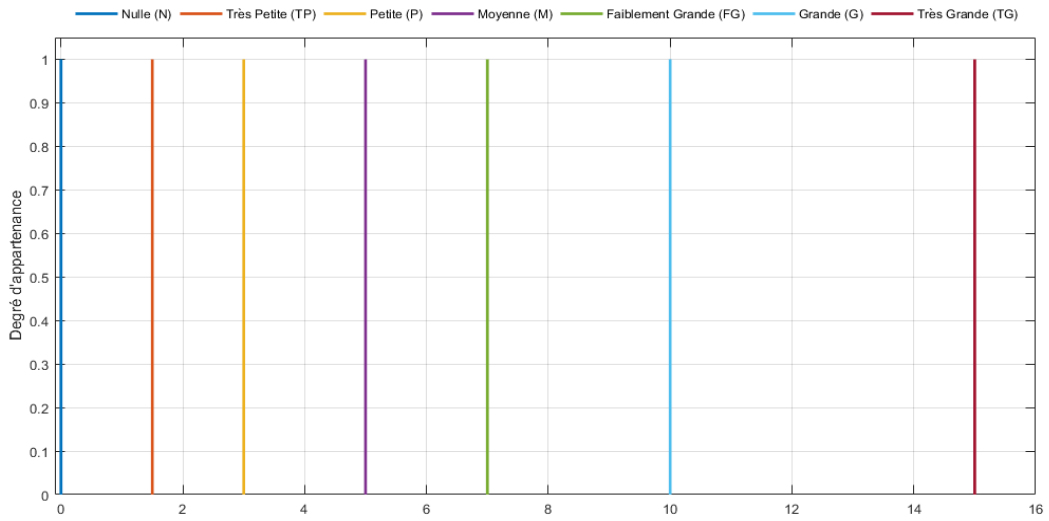


Figure 4.7 – Fonctions d'appartenance des sorties de FLC49 (Convergence)

4.4.2.2 Base des règles floues

L'étape d'inférence fait appel à 49 règles représentant le lien entre les différentes variables entrée-sortie. Le tableau (4.2) décrit les règles floues utilisées pour ce comportement.

d_{rg} \ θ_{rg}		**NG**	**NM**	**NP**	**Z**	**PP**	**PM**	**PG**
N	ω_D	N	N	N	N	P	P	M
	ω_G	M	P	P	N	N	N	N
TP	ω_D	TP	TP	N	TP	M	FG	G
	ω_G	G	FG	M	TP	N	TP	TP
P	ω_D	TP	TP	TP	P	FG	G	TG
	ω_G	TG	G	FG	P	TP	TP	TP
M	ω_D	TP	TP	TP	M	G	G	TG
	ω_G	TG	G	G	M	TP	TP	TP
MG	ω_D	TP	TP	P	FG	FG	G	TG
	ω_G	TG	G	FG	FG	P	TP	TP
G	ω_D	TP	TP	M	G	FG	G	TG
	ω_G	TG	G	FG	G	M	TP	TP
TG	ω_D	TP	TP	P	TG	FG	G	TG
	ω_G	TG	G	FG	TG	P	TP	TP

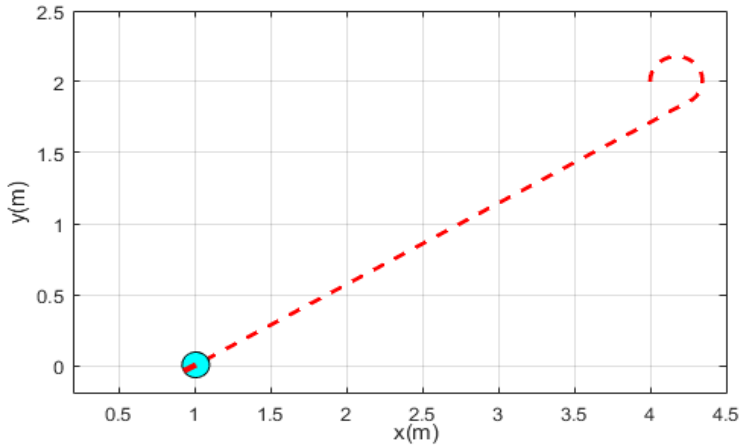
Tableau 4.2 – Les règles de FLC49 (convergence vers un but)

Une fois la mise en place des fonctions d'appartenance et l'établissement des règles définissant le comportement du régulateur ont été effectués, on passe à la sélection d'une méthode de défuzzification. Pour FLC20 et FLC49, nous avons utilisé la méthode de la moyenne des maxima.

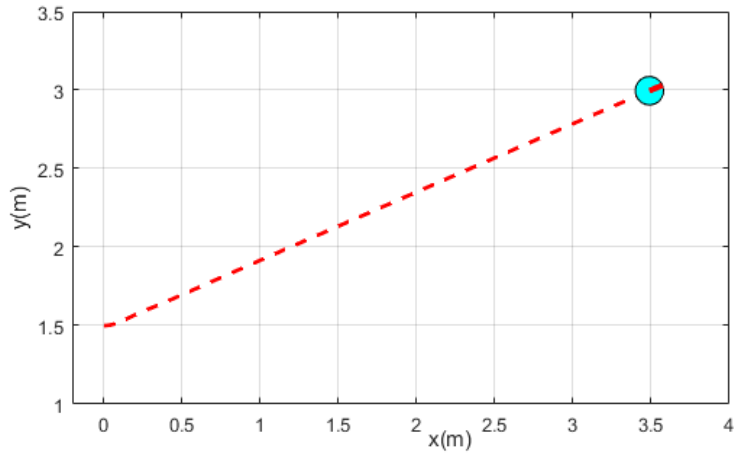
4.4.3 Résultats des simulations

Dans ce qui suit, des exemples de navigation du robot mobile différentiel dans des environnements libres seront présentés pour vérifier la validité et l'efficacité des régulateurs synthétisés.

Les graphes montrés sur les figures 4.9 et 4.10 présentent des navigations autonomes du robot avec des courbes montrant les commandes et l'évolution dans le temps.

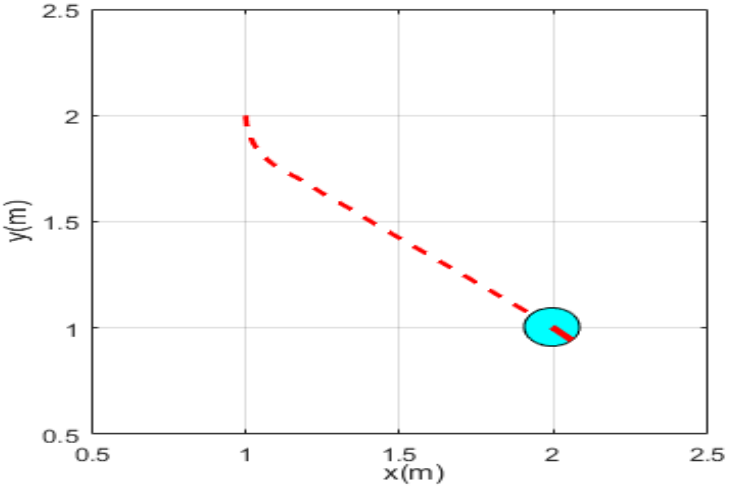


(a) Trajectoire du robot avec FLC20

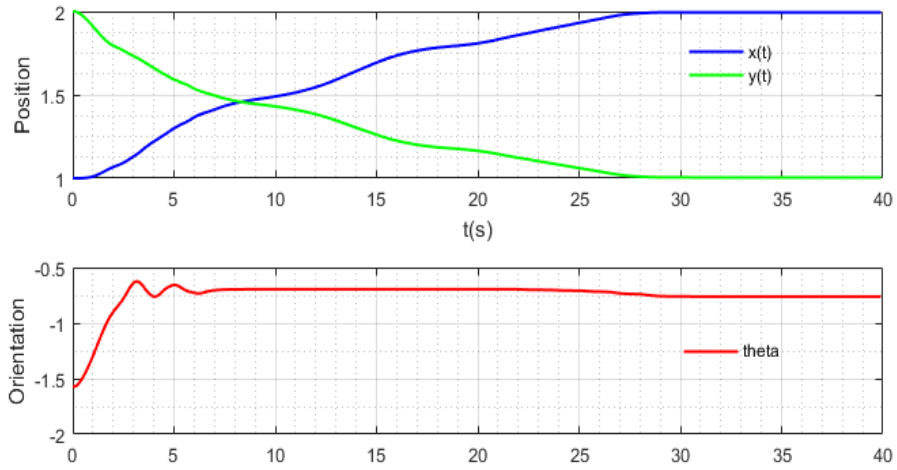


(b) Trajectoire du robot avec FLC49

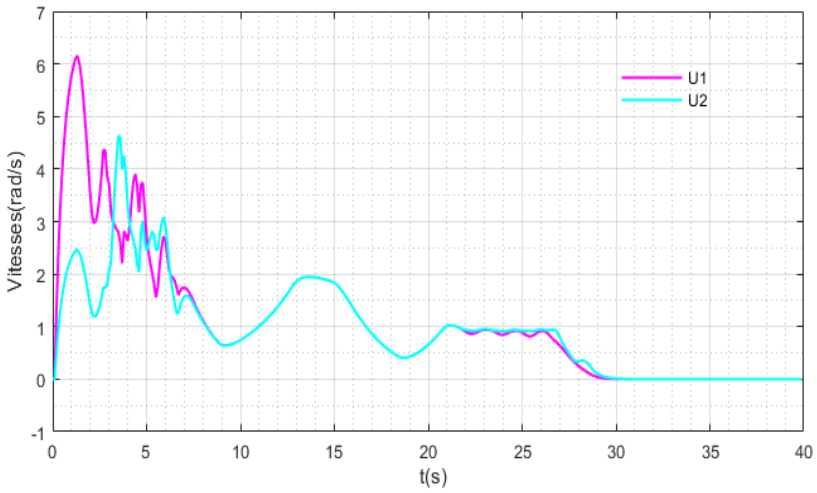
Figure 4.8 – Exemples de navigation libre



(a) Trajectoire du robot

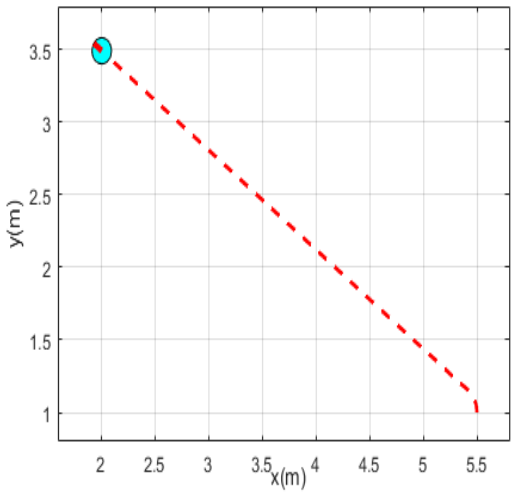


(b) Évolution de la position et de l'orientation

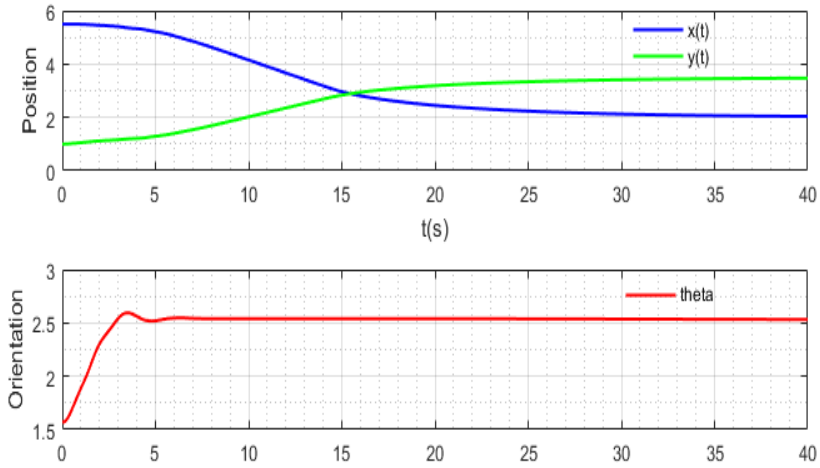


(c) Commandes appliquées

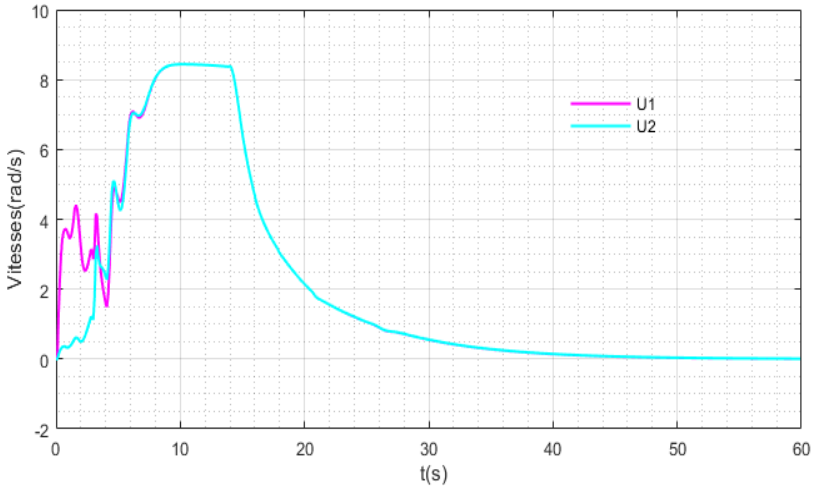
Figure 4.9 – Navigation libre avec FLC20



(a) Trajectoire du robot



(b) Évolution de la position et de l'orientation

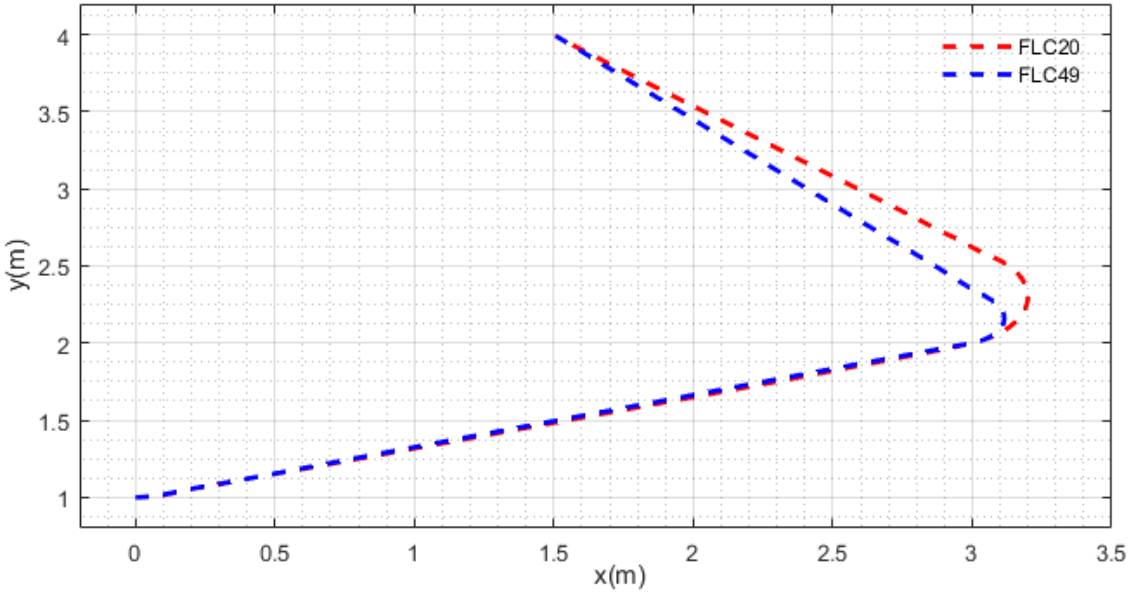


(c) Commandes appliquées

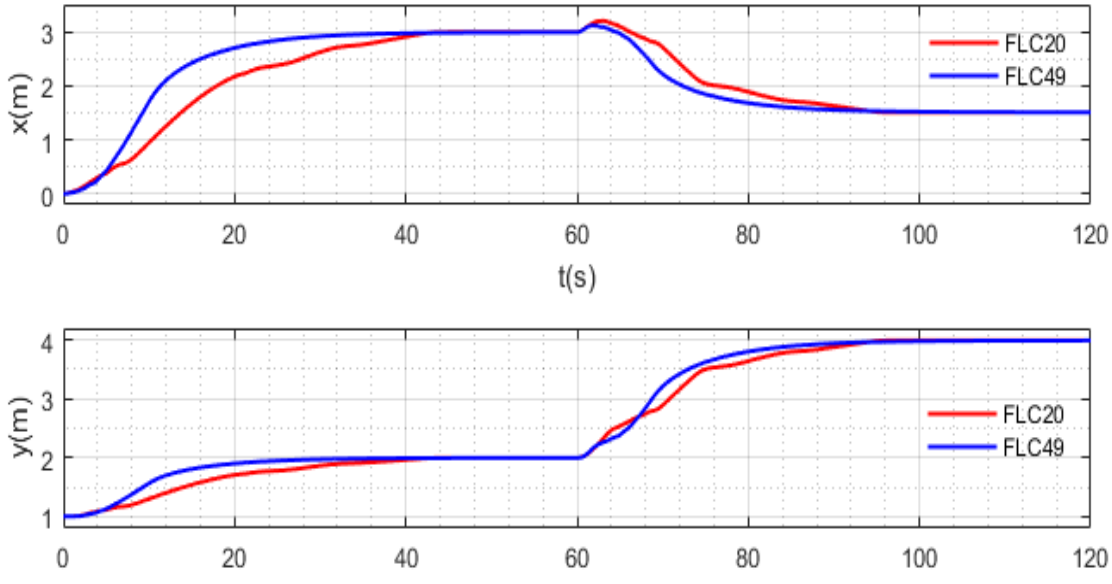
Figure 4.10 – Navigation libre avec FLC49

4.4.4 Étude comparative

D'autres tests montrent une comparaison entre les régulateurs FLC20 et FLC49 pour une navigation autonome avec une variation de but au cours de l'évolution, partant de $P_0 = (0,1,0)$ au point $P_{g_1} = (3,2)$, puis $P_{g_2} = (1.5,4)$.



(a) Comparaison des trajectoires



(b) Comparaison de coordonnées

Figure 4.11 – Comparaison entre FLC20 et FLC49

4.4.5 Commentaires

— Comme présenté sur les figures, les régulateurs flous élaborés sont capable de générer les commandes appropriées pour accomplir la tâche de navigation.

— Le FLC49 donne de meilleures performances par rapport au FLC20, et ceci revient à la précision fournie par le nombre de variables linguistiques caractérisant les entrées/sorties.

— Les vitesses des roues augmentent jusqu'à une valeur maximale lorsque le but est loin, diminuent et s'annulent lorsque le but est atteint. Les résultats obtenus sont très satisfaisants et montrent l'efficacité du comportement proposé.

4.5 Régulateur flou d'évitement d'obstacles

Le comportement présenté dans la section précédente permet de réaliser une navigation autonome libre, une telle condition est restrictive vu la nature de l'environnement. Le régulateur flou de cette section est conçu pour permettre au robot mobile d'éviter les obstacles lors de la convergence vers un but. Pour le comportement de convergence, on utilise le FLC49.

4.5.1 Synthèse du régulateur

Dans ce qui suit, on va synthétiser un régulateur flou basé sur la génération des commandes adéquates pour l'évitement d'obstacles en se basant sur les mesures des distances par des capteurs télémétriques. Le schéma de commande proposé est donné par la figure 4.12 où on a exploité deux comportements flous avec un bloc de coordination simple pour le choix entre les deux régulateurs.

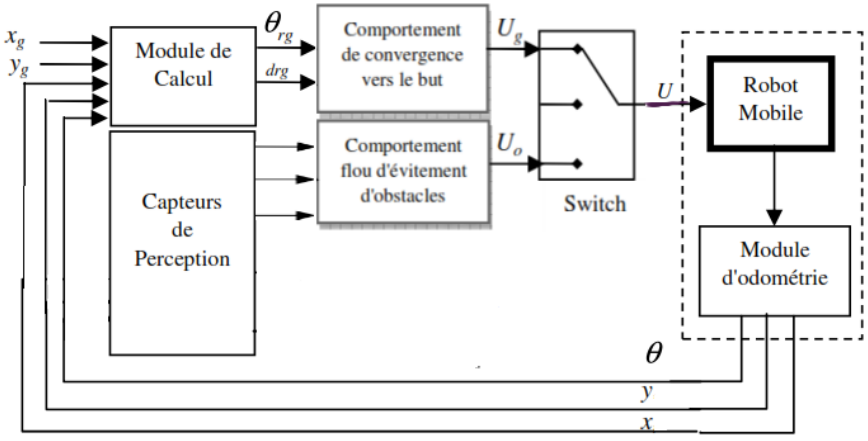


Figure 4.12 – Schéma global de navigation floue

4.5.1.1 Fonctions d'appartenances entrées/sorties

Dans cette étape, on spécifie les fonctions d'appartenance utilisées pour fuzzifier les variables d'entrée. Pour cela, on suppose que le robot mobile est équipé de trois capteurs de distance pour la détection des obstacles dans les trois directions (en avant, à droite et à gauche). Leurs fonctions d'appartenances (elles sont identiques) sont données sur la figure 4.13 avec des formes triangulaires et trapézoïdales. Les deux actions de commande (vitesses des roue droite et gauche) sont réduites à des singletons (figure 4.14).

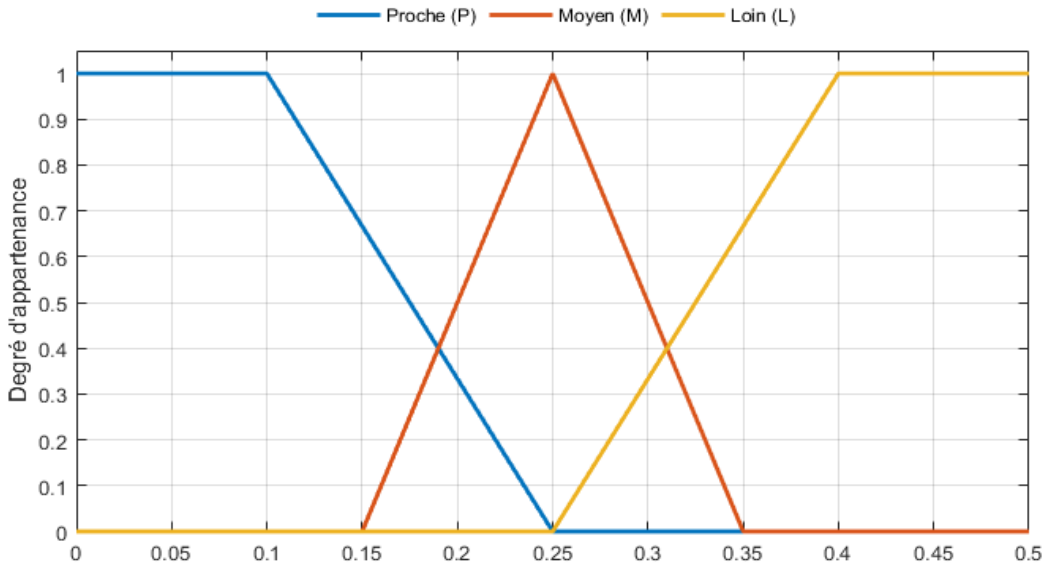


Figure 4.13 – Fonctions d'appartenance des entrées (évitement)

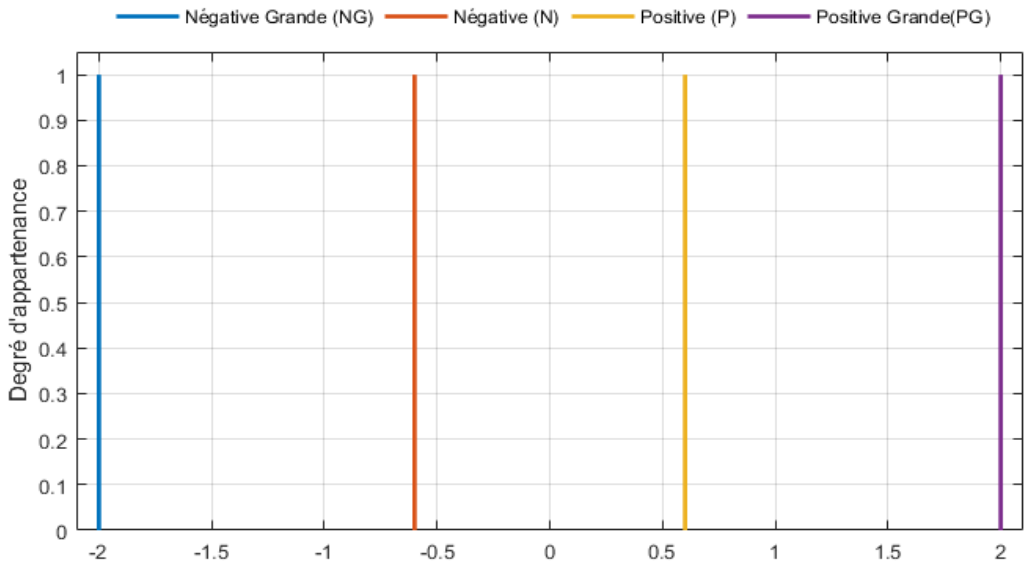


Figure 4.14 – Fonctions d'appartenance des sorties (évitement)

4.5.1.2 Base des règles floues

Sur la base de la définition des variables linguistiques et des fonctions d'appartenance (figures 4.13 et 4.14) pour les variables d'entrée et de sortie, l'étape d'inférence ; pour ce comportement, fait appel à 27 règles représentant le lien entre les différentes variables entrée-sortie. Ces règles sont représentées sur le tableau (4.3) ci-dessous.

Entrées			**Sorties**	
d_{ro}^{G}	d_{ro}^{A}	d_{ro}^{D}	ω_G	ω_D
P	*P*	*P*	*NG*	*NG*
P	*P*	*M*	*PG*	*NG*
P	*P*	*L*	*PG*	*NG*
P	*M*	*P*	*NG*	*NG*
P	*M*	*M*	*PG*	*N*
P	*M*	*L*	*PG*	*N*
P	*L*	*P*	*NG*	*NG*
P	*L*	*M*	*P*	*N*
P	*L*	*L*	*P*	*N*
M	*P*	*P*	*NG*	*PG*
M	*P*	*M*	*NG*	*PG*
M	*P*	*L*	*PG*	*NG*
M	*M*	*P*	*N*	*PG*
M	*M*	*M*	*NG*	*NG*
M	*M*	*L*	*PG*	*N*
M	*L*	*P*	*N*	*PG*
M	*L*	*M*	*P*	*P*
M	*L*	*L*	*PG*	*P*
L	*P*	*P*	*NG*	*PG*
L	*P*	*M*	*NG*	*PG*
L	*P*	*L*	*PG*	*NG*
L	*M*	*P*	*N*	*PG*
L	*M*	*M*	*P*	*PG*
L	*M*	*L*	*P*	*PG*
L	*L*	*P*	*N*	*PG*
L	*L*	*M*	*P*	*PG*
L	*L*	*L*	*P*	*P*

Tableau 4.3 – Les règles pour le comportement d'évitement d'obstacles

Avec :

- d_{ro}^{G} : la distance mesurée entre le robot et le premier obstacle rencontré à gauche.
- d_{ro}^{A} : la distance mesurée entre le robot et le premier obstacle rencontré en avant.
- d_{ro}^{D} : la distance mesurée entre le robot et le premier obstacle rencontré à droite.

Les positions des trois capteurs utilisés sur le robot mobile sont représentées sur la figure 4.15 ainsi que les différentes distances mesurées par rapport aux obstacles.

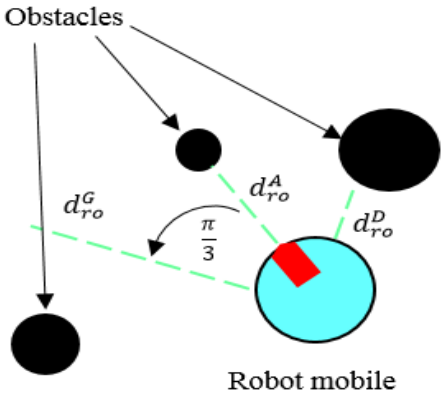


Figure 4.15 – Positions et arrangement des capteurs

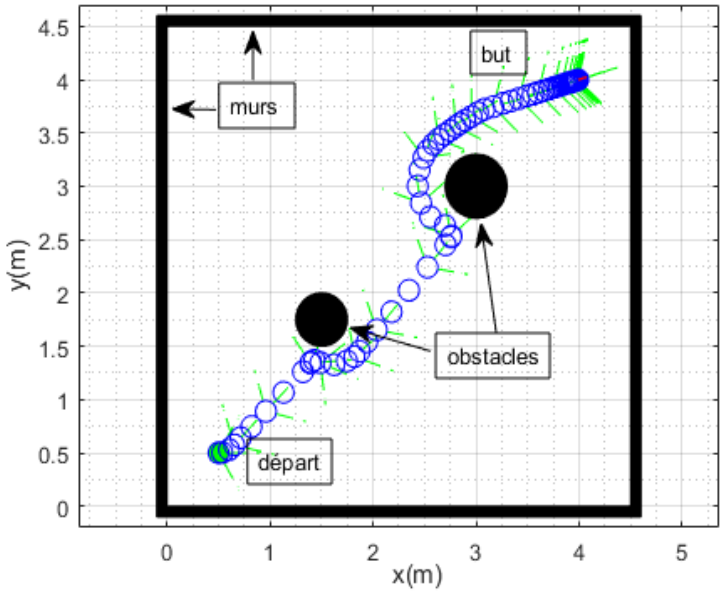
4.5.1.3 Défuzzification

Une fois la mise en place des fonctions d'appartenance et l'établissement des règles définissant le comportement du régulateur ont été effectués, on passe à la sélection d'une méthode de défuzzification. pour ce régulateur, nous avons utilisé la méthode du centre de gravité.

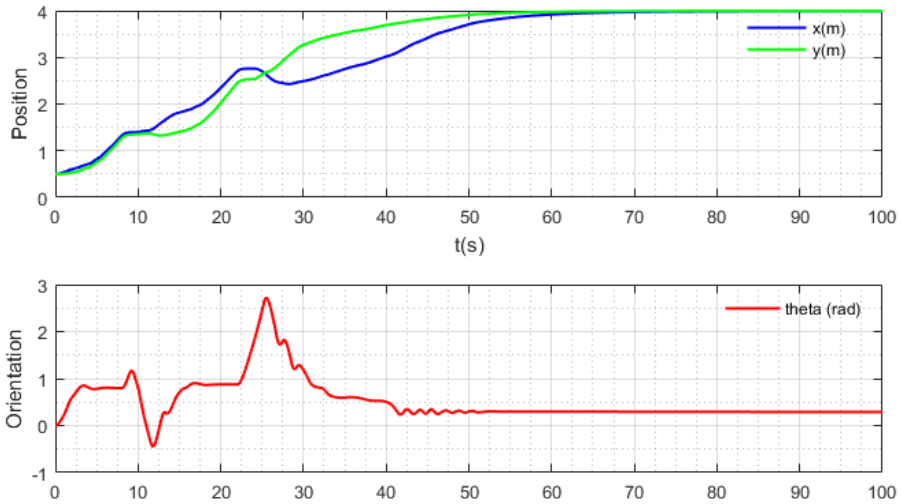
4.5.2 Résultats des simulations

Dans cette section, deux exemples de navigation du robot mobile dans des environnements encombrés d'obstacles seront présentés pour vérifier l'efficacité du régulateur. Durant le mouvement, sa mission de base est d'atteindre un but désiré. Lorsque le robot rencontre un obstacle, il doit avoir une capacité d'évitement d'obstacles selon les distances mesurées.

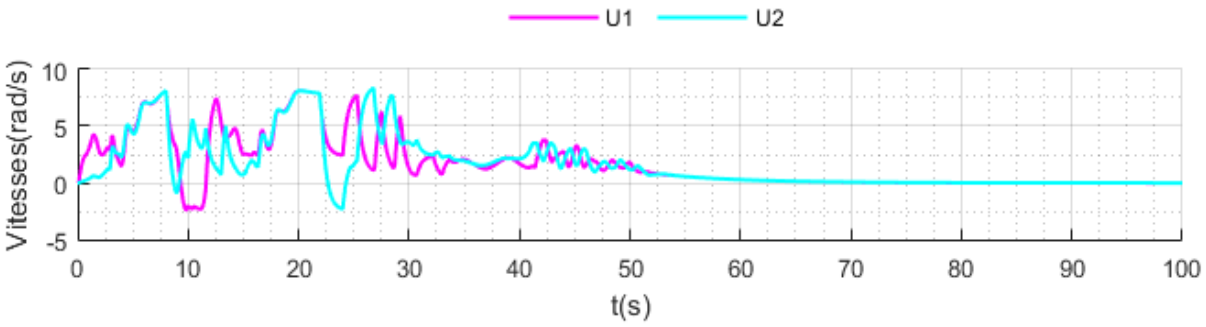
En choisissant différents scénarios du robot ainsi que différents environnements, les figures 4.16 et 4.17 présentent la trajectoire de navigation, l'évolution de la position et les commandes générées. La première configuration montre la navigation du point initial $P_0 = (0.5, 0.5, 0)$ au point final $P_g = (4, 4)$ avec deux obstacles. Tandis que la deuxième configuration teste le régulateur dans un environnement assez encombré d'obstacles, partant de $P_0 = (0.5, 4, \frac{\pi}{2})$ au point $P_g = (4, 0.5)$.



(a) Déplacement du robot

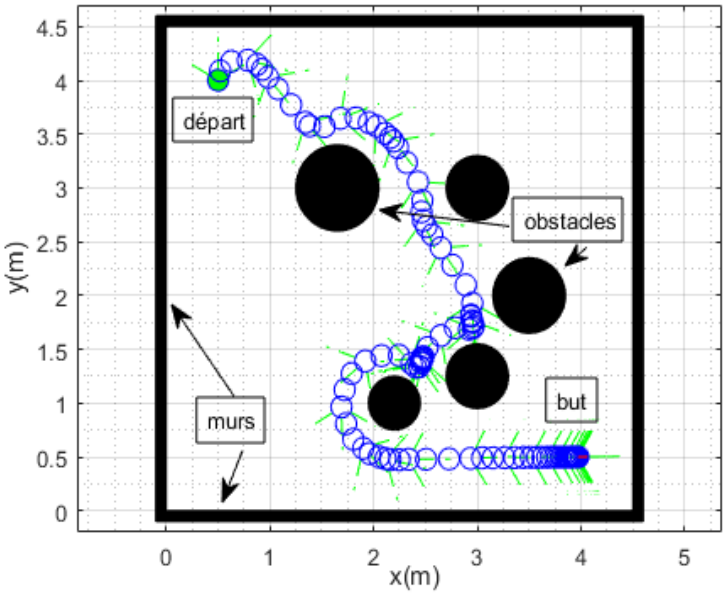


(b) Évolution de la position et de l'orientation

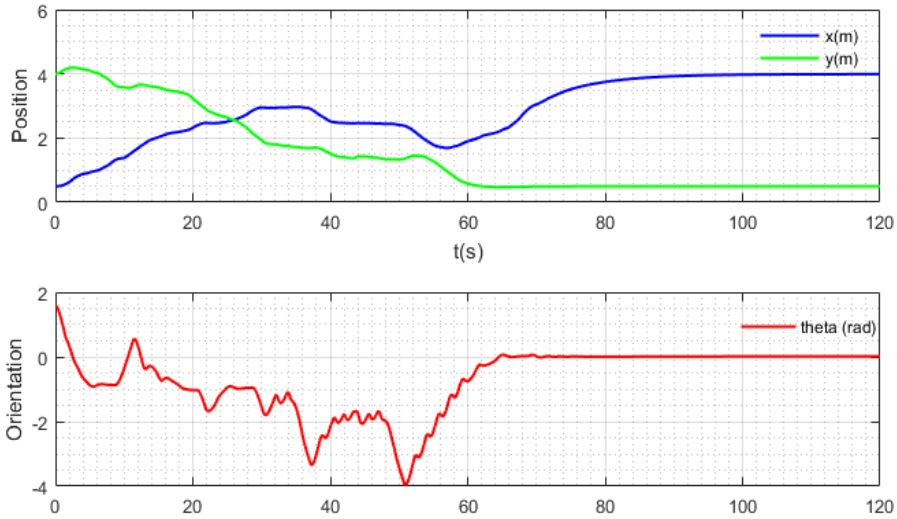


(c) Commandes appliquées

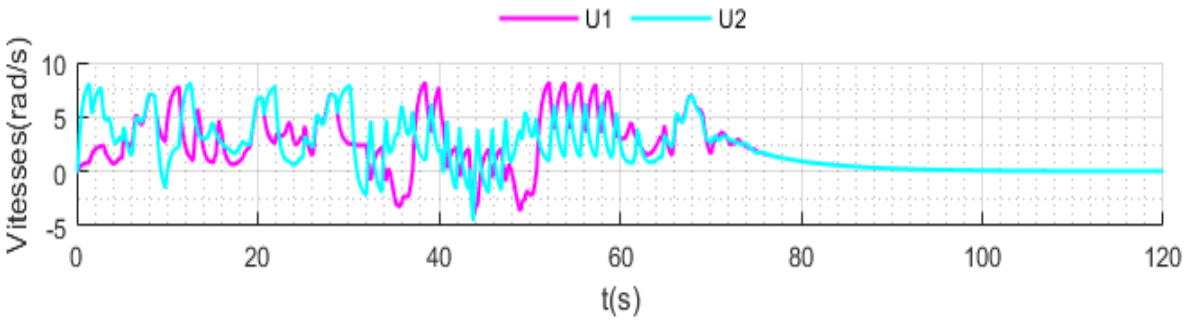
Figure 4.16 – Navigation avec évitement d'obstacles (premier environnement)



(a) Déplacement du robot



(b) Évolution de la position et de l'orientation



(c) Commandes appliquées

Figure 4.17 – Navigation avec évitement d'obstacles (deuxième environnement)

4.5.3 Commentaires

— Comme présenté sur les figures, dans tous les cas, le robot est capable de naviguer de manière autonome et peut atteindre son but efficacement en évitant les obstacles avec succès quelque soit sa position initiale. La trajectoire de mouvement obtenue et les actions générées montrent que le système de commande proposé donne de bonnes performances.

— Le robot se déplace vers l'objectif, lorsqu'un obstacle est détecté dans l'un des trois côtés (en face, à droite ou à gauche), le comportement d'évitement d'obstacle est activé afin de générer les actions appropriées pour éviter les collisions.

— L'utilisation de ce type de régulateurs donne des résultats acceptables dans les environnements encombrés d'obstacles.

Ces figures montrent l'efficacité des régulateurs synthétisés. Ils donnent des résultats très satisfaisants pour réaliser la tâche de navigation du robot mobile. Par contre, lors de la conception d'un régulateur flou, les choix sont effectués de façon relativement empirique et la mise au point d'un système de commande flou peut s'avérer longue et délicate, ce qui nous amène à faire appel aux méthodes d'apprentissage.

4.6 Extension de Q-learning

Nous avons vu dans le chapitre 3 que le Q-learning est l'algorithme le plus utilisé en pratique en raison de sa simplicité et de sa convergence. Dans cette application, le robot construit la fonction de qualité Q. L'utilisation de cet algorithme pose un problème structurel lié à la méthode de représentation des espaces d'états et d'actions. Son inconvénient est la difficulté d'obtenir une convergence permettant la distribution des séquences d'action aux d'états correspondants. L'apprentissage par renforcement est parfaitement adapté aux processus dont le nombre d'états et d'actions est fini. Mais en robotique mobile, ces espaces sont rarement discontinus. Dans ce cas, l'utilisation d'autres méthodes comme les approximateurs universels tels que les systèmes d'inférence floue [MYK05, KHR11]. Dans ce qui suit, pour objectif d'améliorer les performances des régulateurs et afin d'introduire une connaissance préalable à la conception de ses comportements, nous utilisons les systèmes d'inférence floue pour le passage aux espaces continus et pour que le comportement initial soit acceptable.

Une version floue de la fonction Q, dans des espaces continus appelée Q-FLC (Q-learning flou) a été proposée par Jouffe [JOU98]. Une fois les fonctions d'appartenance des entrées sont choisies, la tâche d'apprentissage consiste à optimiser les conclusions des règles.

Le principe de cet algorithme consiste à proposer plusieurs conclusions pour chaque règle, et à associer à chaque conclusion une fonction de qualité qui sera évaluée incrémentalement. Il s'agit alors de trouver la conclusion appropriée pour chaque règle. On résume ici les principaux points de l'algorithme [BOU09] :

— chaque règle possède plusieurs conclusions potentielles.

— chaque conclusion potentielle est associée à un indice de qualité.

— le régulateur évalue la qualité de chaque conclusion, selon une PEE donnée.

— à l'issue de l'apprentissage, chaque règle se voit associée à la meilleure conclusion.

La base des règles initiales est composée de M règles de la forme suivante :

$$\begin{array}{llllll} \text{Si "}s \text{ est } S_i\text{"} & \text{Alors} & c_i = c_{i1} & \text{avec} & Q_i = Q_{i1} \\ & \text{ou} & c_i = c_{i2} & \text{avec} & Q_i = Q_{i2} \\ & & \vdots & & \vdots \\ & \text{ou} & c_i = c_{iJ} & \text{avec} & Q_i = Q_{iJ} \end{array}$$

où c_{ij} représente la $j^{\text{ième}}$ conclusion potentielle et Q_{ij} la qualité associée. Dans ce dernier, la valeur de mise à jour est calculée par le système d'inférence flou et elle est commune à toutes les valeurs Q associées aux conclusions choisies. Dans notre algorithme, chaque valeur Q associée à une conclusion choisie est adaptée séparément par :

$$Q_{ik} \leftarrow Q_{ik} + \eta\,\alpha_i(s_t)(R_t + \gamma V(s_{t+1}) - Q(s_t,a)) \tag{4.4}$$

Avec :

$$V(s_{t+1}) = \sum_{i=1}^{M} \alpha_i(s_{t+1}) \underset{k}{max} Q_{ik}$$

Et :

$$Q(s_t,a) = \sum_{i=1}^{M} \alpha_i(s_t) Q_{ik}$$

— k : l'indice de la conclusion choisie par la PEE utilisée.

— η : taux d'apprentissage.

— Les autres valeurs Q_{ij} avec $1 \leq j \leq J$ et $j \neq k$ sont laissés sans changement.

— $\alpha_i(s_t)$: le degré d'activation de la règle i dans l'état s_t.

4.7 Régulateur RL-flou de convergence vers un but

Pour des raisons de complexité (la convergence est plus lente en utilisant FLC20, due au nombre important des règles floues utilisées.) et de mise en œuvre pratique sur un robot réel, on retient le FLC20 et on cherche à l'optimiser avec le RL. La politique d'exploration/exploitation ε-gloutonne est utilisée pour la sélection d'une action. Après un temps d'apprentissage suffisant, le processus peut être arrêté et la stratégie gloutonne (greedy) sera utilisée pour une exploitation finale des résultats d'apprentissage.

4.7.1 Synthèse du régulateur RL-FLC20

Dans ce qui suit nous proposons une méthode de réglage de FLC20 en utilisant l'algorithme Q-FLC pour le comportement de convergence vers un but. Dans cet algorithme, le robot possède une base de règles initiale qui définit les situations possibles pour le comportement désiré. Le principe consiste à proposer plusieurs conclusions de type singletons pour chaque règle, et à associer à chaque conclusion une fonction qualité qui sera optimisée au cours du temps.

Pour chaque règle ($i = 1...20$), on propose 9 conclusions selon la forme suivante :

$$\begin{array}{lll} \text{Si "}d_{rg}\text{ est }N\text{ et }\theta_{rg}\text{ est }N\text{"} & \text{Alors} & \omega_{Di} = \omega_{Di1} \text{ et } \omega_{Gi} = \omega_{Gi1} \text{ avec } Q_i = Q_{i11} \\ & \text{ou} & \omega_{Di} = \omega_{Di1} \text{ et } \omega_{Gi} = \omega_{Gi2} \text{ avec } Q_i = Q_{i12} \\ & & \vdots \qquad\qquad \vdots \\ & \text{ou} & \omega_{Di} = \omega_{Di3} \text{ et } \omega_{Gi} = \omega_{Gi3} \text{ avec } Q_i = Q_{i33} \end{array}$$

Pour le comportement de convergence vers un but, le robot reçoit les renforcements suivants :

Comportement du robot	**Renforcement reçu**
Le robot s'approche du but	1
Le robot s'éloigne du but	-1
Le robot atteint le but	10

Tableau 4.4 – Les renforcements reçus (convergence vers un but)

Afin d'optimiser le système de navigation renforcement-flou, les positions initiales sont choisies aléatoirement pendant la phase d'apprentissage. Chaque épisode commence par une position aléatoire et se termine lorsque le robot atteint le but.

4.7.2 Résultats des simulations

On remarque que le comportement du robot s'améliore pendant la phase d'apprentissage comme le montre la figure 4.18 représentant les renforcements reçus pour chaque épisode pour les deux scénarios. Les renforcements maximisés illustrent la convergence de l'algorithme utilisé. Les figures 4.19a et 4.19b représentent les trajectoires du robot obtenues après une phase d'entrainement pour cette mission avec des différentes situations, le robot est capable de rejoindre son but final de façon autonome quelque soit sa position initiale.

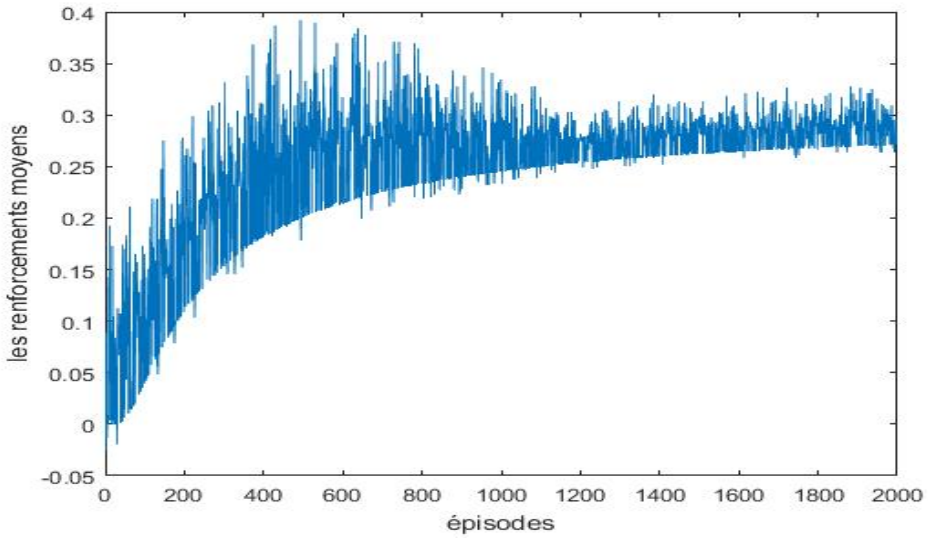
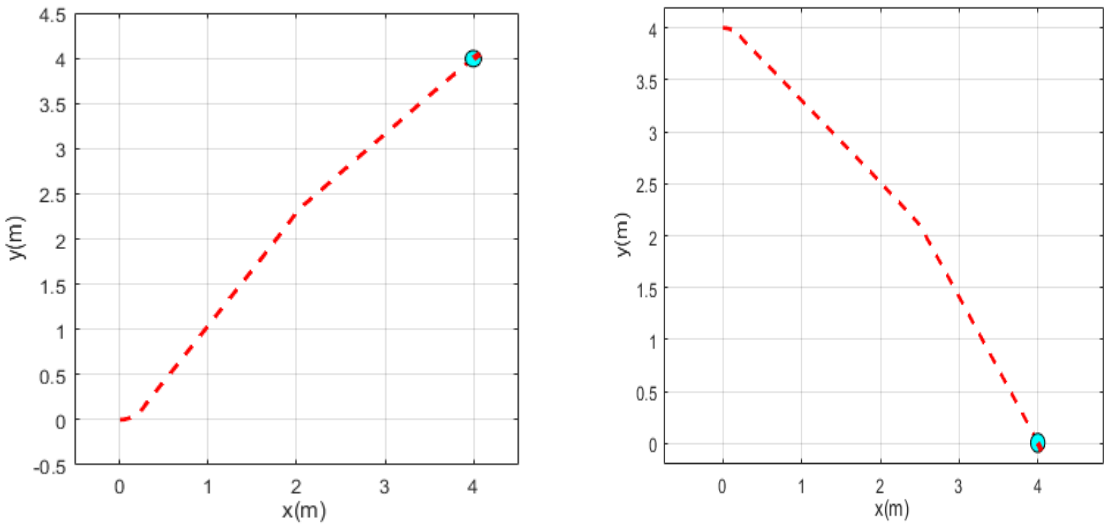


Figure 4.18 – Renforcements reçus par épisode (Convergence vers un but)



(a) Premier scénario

(b) Deuxième scénario

Figure 4.19 – Trajectoires après l'apprentissage (convergence vers un but)

4.7.3 Étude comparative

En se basant sur les deux régulateurs conçus, un exemple de comparaison des trajectoires du robot sont présentées sur la figures 4.20 avec ses comportements évolutifs en fonction du temps sur la figure 4.21. Les figures montrent que le FLC20 a de meilleures performances par rapport au RL-FLC20, ce qui est traduit par une capacité d'optimisation réduite dans l'application envisagée. La capacité de cette algorithme reste une capacité d'apprentissage.

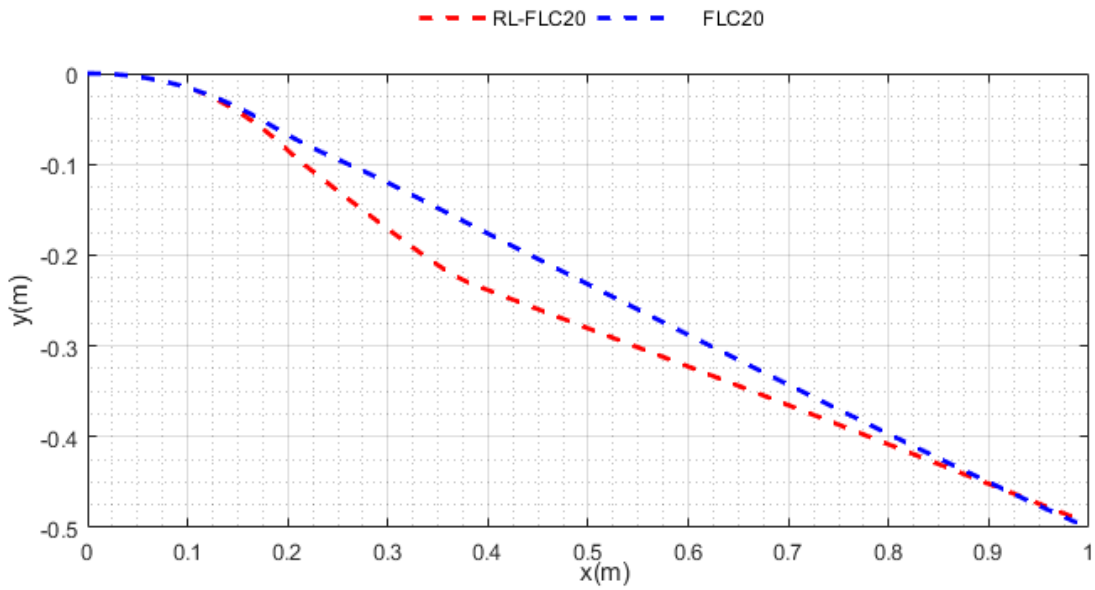


Figure 4.20 – Comparaison des trajectoires (Convergence vers un but)

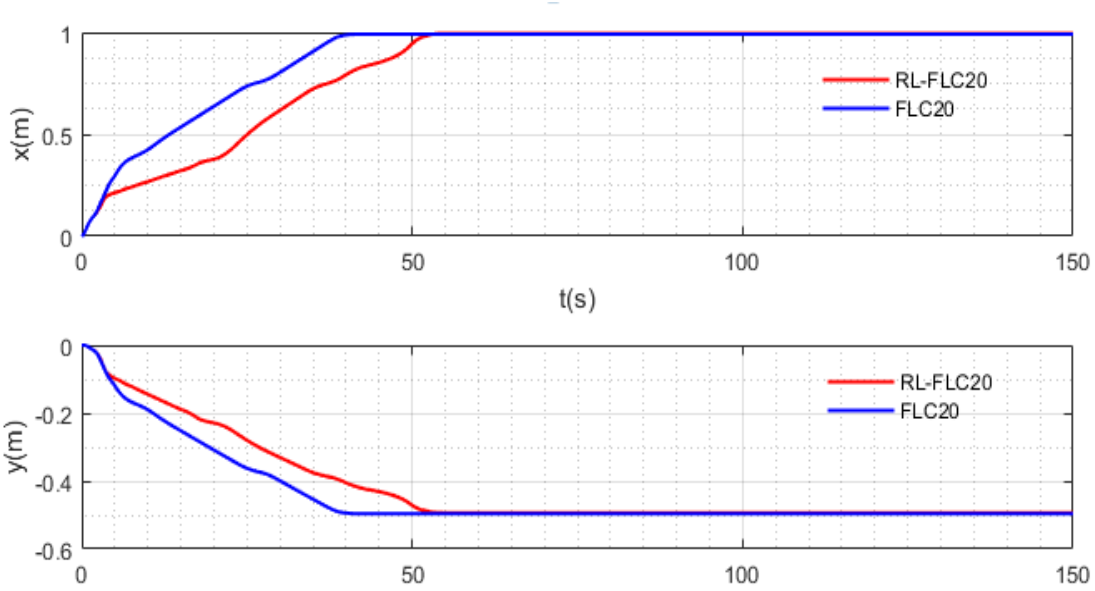


Figure 4.21 – Comparaison d'évolution de coordonnées (Convergence vers un but)

4.8 Régulateur RL-flou d'évitement d'obstacles

Dans ce travail, On suppose que le robot est capable de mesurer les distances à un obstacle en trois cotés (en avant, à droite et à gauche) à base d'un système de perception efficace. Le système de navigation autonome basé sur le Q-learning flou utilise comme entrées, les distances aux obstacles dans les trois directions, et doit générer après une phase d'apprentissage les vitesses des roues droite et gauche pour éviter les obstacles rencontrés.

4.8.1 Synthèse du régulateur

Dans ce qui suit nous proposons une méthode de réglage du régulateur conçu précédemment pour le comportement d'évitement d'obstacles seulement en utilisant l'algorithme Q-learning flou. Dans cet algorithme, le robot possède une base de règles initiale qui définit les situations possibles pour le comportement désiré. Le principe consiste à proposer plusieurs conclusions de type singletons pour chaque règle, et à associer à chaque conclusion une fonction qualité qui sera optimisée au cours du temps.

Pour chaque règle ($i = 1...27$), on propose 16 conclusions (quatre actions pour chaque roue) selon la forme suivante :

$$\begin{array}{lll} \text{Si "}o \text{ et } O_i\text{"} & \text{Alors} & \omega_{Di} = \omega_{Di1} \text{ et } \omega_{Gi} = \omega_{Gi1} \text{ avec } Q_i = Q_{i11} \\ & \text{ou} & \omega_{Di} = \omega_{Di1} \text{ et } \omega_{Gi} = \omega_{Gi2} \text{ avec } Q_i = Q_{i12} \\ & & \vdots \qquad\qquad \vdots \\ & \text{ou} & \omega_{Di} = \omega_{Di4} \text{ et } \omega_{Gi} = \omega_{Gi4} \text{ avec } Q_i = Q_{i44} \end{array}$$

Pour le comportement d'évitement, le robot mobile reçoit les renforcements suivants :

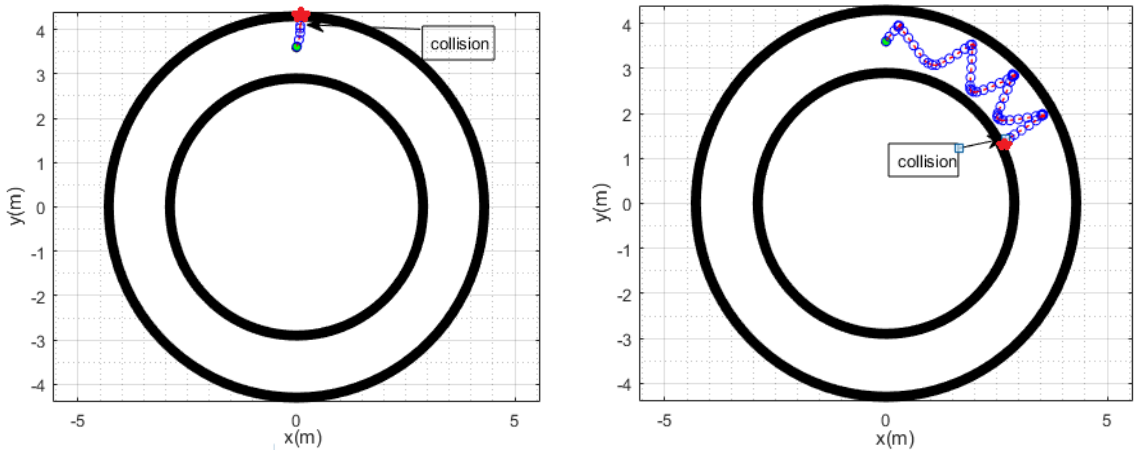
Comportement du robot	**Renforcement reçu**
Le robot s'éloigne de l'obstacle	1
Le robot s'approche de l'obstacle	-1
Le robot percute un obstacle	-10

Tableau 4.5 – Les renforcements reçus (évitement d'obstacles)

Dans cette application, la valeur de renforcement va servir à déterminer la meilleure conclusion parmi les conclusions proposées pour les deux commandes. On suppose que le robot au départ est dans une zone libre d'obstacles.

4.8.2 Résultats des simulations

Durant l'apprentissage, le robot doit explorer les actions possibles pour chaque situation perçue. Dans les figures 4.22a et 4.22b, on présente les trajectoires du robot dans les premiers épisodes (épisode 1 et 500). On observe des collisions avec les obstacles sont produites au début de l'apprentissage. Mais à partir de l'épisode 1000, le robot obtient le meilleur comportement pour éviter l'obstacle comme le montre la figure 4.23. Après la phase d'apprentissage, l'algorithme est arrêté et les conclusions des règles avec les meilleures qualités sont choisies. Les récompenses moyennes obtenues pour les phases d'apprentissage sont représentées sur la figure 4.24.



(a) Trajectoire à l'épisode 1

(b) Trajectoire à l'épisode 500

Figure 4.22 – Trajectoire pendant l'apprentissage (évitement d'obstacles)

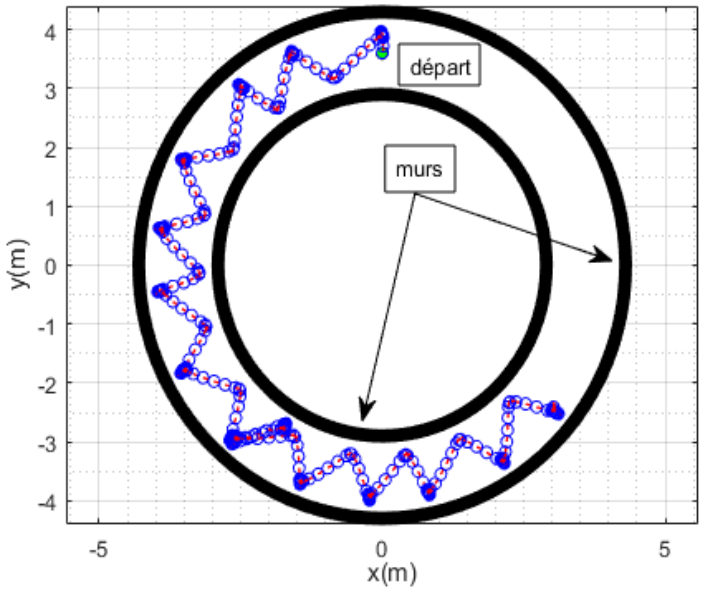


Figure 4.23 – Trajectoires après l'apprentissage (évitement d'obstacles)

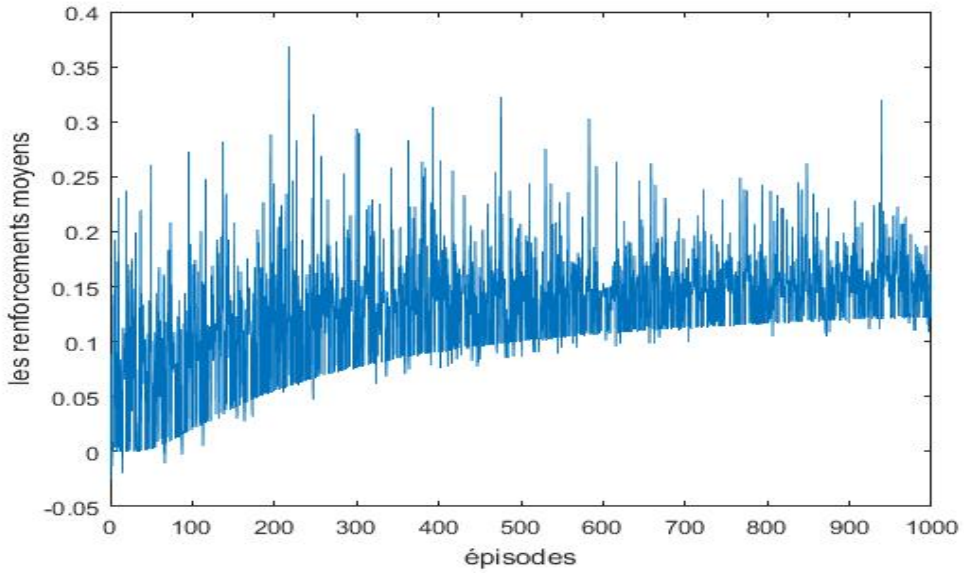
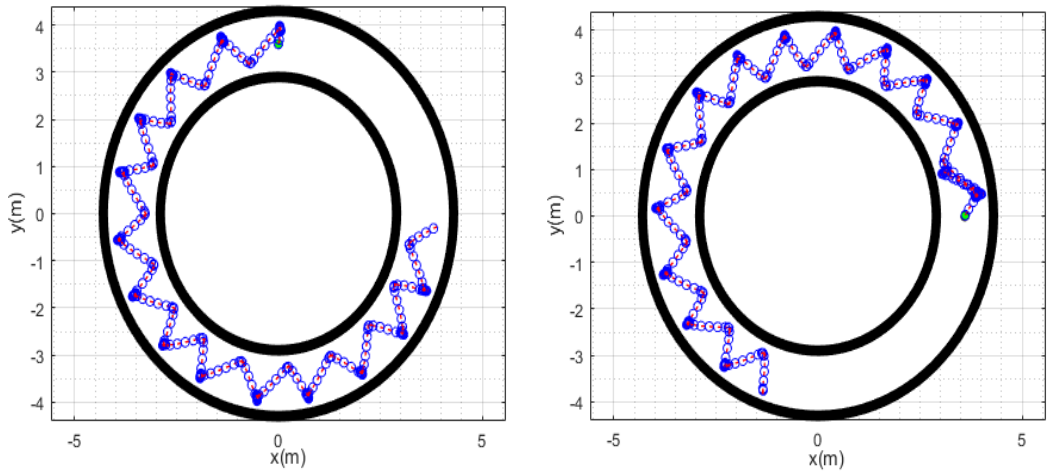


Figure 4.24 – Renforcements reçus par épisode (évitement d'obstacles)

4.8.3 Étude comparative

Dans cette section, on présente une comparaison entre les résultats obtenus par le navigateur renforcement-flou proposé et le régulateur flou présenté précédemment. Pour le même environnement de navigation, le comportement du robot dans les deux situations (flou et renforcement-flou) sont illustrés sur les figures 4.25a et 4.25b respectivement. Les résultats sont acceptables et le régulateur renforcement-flou réagit d'une manière satisfaisante pour cette mission. la simulation actuelle utilise le même environnement ce qui présente un risque de suraprentissage.



(a) Régulateur flou (b) Régulateur renforcement-flou

Figure 4.25 – Comparaison de performances (évitement d'obstacles)

4.9 Conclusion

Dans ce chapitre, nous avons présenté des systèmes de commande intelligents basés sur les techniques de l'intelligence artificielle pour la navigation autonome d'un robot mobile. Ces systèmes sont basés sur des comportements décomposés en un ensemble de sous-tâches simples. En premier lieu, nous avons utilisé un raisonnement par la logique floue. Les résultats des simulations sont satisfaisants et permettent au robot de se déplacer en évitant les obstacles.

Cependant, un régulateur flou pourrait fournir de bons résultats pour cet environnement et ne pas répondre correctement face à une situation nouvelle. C'est la raison pour laquelle nous avons opté, en deuxième lieu pour l'étude des possibilités offertes par l'apprentissage par renforcement pour l'ajustement et la génération règles floues. La méthode proposée est utilisée en particulier pour l'optimisation des régulateurs flous présentés précédemment. La tâche d'apprentissage consiste à améliorer en ligne la base de règles en changeant la partie conclusion. Des exemples de simulations en utilisant l'algorithme Q-FLC ont été fournis pour différents comportements (convergence vers un but, évitement d'obstacles). Dans les deux cas, la méthode a montré ses performances en cas d'apprentissage et ses limitations dans le cas d'optimisation en ligne.

Une comparaison du régulateur renforcement-flou avec le régulateur flou a été abordée pour vérifier l'efficacité des comportements obtenus. D'après les résultats présentés, on a montré que l'apprentissage par renforcement est une solution possible pour l'apprentissage des règles floues mais reste limité pour l'optimisation des régulateurs flous destinés à la navigation autonome.

Dans le chapitre 5 nous exposerons le matériel utilisés et les résultats obtenus lors de l'implémentation d'un régulateur flou avec des techniques V-SLAM pour la navigation visuelle d'un robot mobile.

Chapitre 5

Implémentation et Résultats Pratiques

5.1 Introduction

De nos jours, les recherches en navigation robotique se penchent vers les approches visuelles. Le compromis entre la puissance de calcul des systèmes embarqués dans les robots et la complexité des algorithmes de vision a poussé les chercheurs à produire des algorithmes plus efficaces et plus intelligents. Le but de la vision artificielle est de donner à une machine des capacités visuelles de façon à pouvoir acquérir des informations sur l'environnement. Une ou plusieurs caméras sont nécessaires pour acquérir les images qui vont être utilisées par la suite pour la navigation. Le choix du modèle de la caméra est d'une grande importance pour les techniques localisation et cartographie visuelles, la qualité de l'image ainsi que les résultats du traitement en dépendent.

Dans ce chapitre, nous allons nous intéresser à la mise en œuvre pratique d'une technique V-SLAM sur un robot mobile pour une navigation floue. Nous allons tout d'abord commencer par une présentation de la plateforme expérimentale, du matériel utilisé et de l'architecture électronique. Enfin nous passerons aux tests après implémentation de l'approche. Les résultats expérimentaux seront discutés et commentés, à travers différents tests.

5.2 Présentation du robot

Dans cette section on présente brièvement le robot sur lequel on effectue les tests, celui-ci à été réalisé par l'équipe ECEborg de l'ECE Paris (figure 5.1). La configuration de ce robot est différentielle dont le rayon et l'inter-axe sont 3.5 *cm* et 28 *cm* de longueur respectivement.

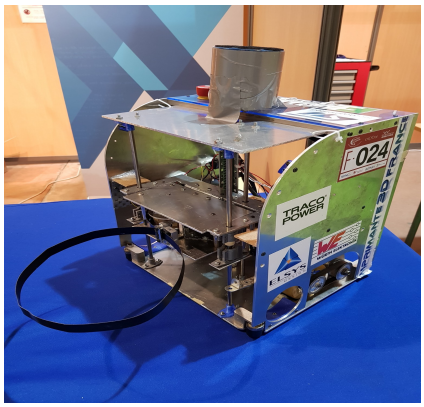


Figure 5.1 – Robot de ECEborg

Le matériel disponible pour l'application développée dans ce mémoire est constitué de :

- un châssis en aluminium.
- deux moteurs à courant continu (maxon) avec deux encodeurs optiques (AMT10 series).
- trois capteurs infrarouge Sharp GP2D
- une caméra RGB-D Kinect et une caméra monoculaire Go Pro HERO 3.
- une carte d'asservissement (Arduino Mega) et une carte d'évitement.
- une carte de navigation/localisation (Intel NUC i5).
- deux circuits imprimés (carte alimentation et carte puissance).

5.2.1 Châssis

C'est la structure sur laquelle se fixe toutes les autres parties du robot. Le choix de celle-ci influe grandement sur les performances, la robustesse et la stabilité de la base roulante différentielle. Pour avoir de bonnes performances de navigation, il faut choisir un châssis (voir figure 5.2) qui soit symétrique et qui offre le moins de déformation et de flexion possibles.

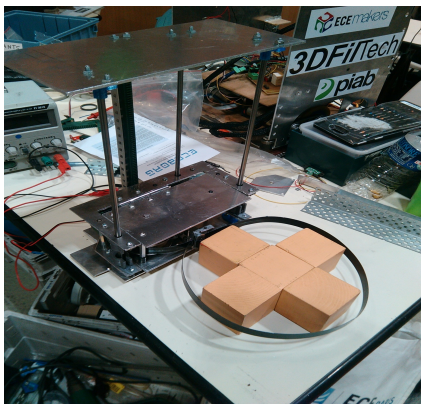


Figure 5.2 – Châssis

5.2.2 Motorisation

La base roulante fonctionne avec 2 moteurs électriques à courant continu *Maxon* avec une puissance de $11W$ avec réducteurs de vitesses. Les moteurs sont commandés par des PID classiques (figure 5.3) dans le but de faire avancer le robot.

Figure 5.3 – Moteur à courant continu Maxon ECX

5.2.3 Télémètres IR

Le robot est équipé de 3 capteurs infrarouge Sharp GP2D (figure 5.4), chacun dispose d'un circuit intégré de traitement de signal et une sortie analogique de type tension. Ils sont utilisés pour la fonctionnalité d'évitement d'obstacles sans V-SLAM.

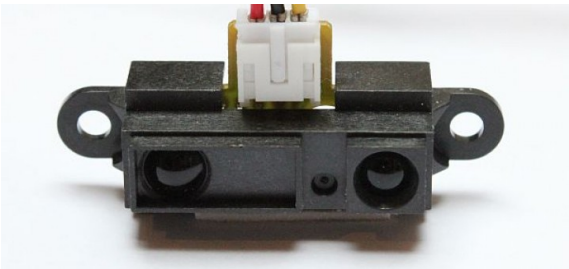


Figure 5.4 – Capteur IR Sharp GP2D

Ces capteurs ne sont généralement pas utilisés pour repérer l'ensemble des corps présents dans l'environnement car leur portée est trop faible. Ils servent plutôt à mesurer la distance des corps proches du robot. Ils ont comme caractéristiques, les points décris dans le tableau 5.1 ci-dessous :

étendue de mesure	$10\,cm$ - $80\,cm$
durée du cycle d'émission de la LED	$32\,ms$
consommation moyenne du courant	$33\,mA$

Tableau 5.1 – Caractéristiques d'un Capteur IR Sharp GP2D

5.2.4 Encodeurs

La plupart des encodeurs pour robots mobiles utilisent des capteurs optiques (mais il existe des encodeurs utilisant une information mécanique ou magnétique). Pour notre application, des encodeurs optiques AMT10 sont placés sur chaque roue, avec transmission directe avec les moteurs (figure 5.5)

Figure 5.5 – Encodeurs optiques AMT10

5.2.5 Caméras

Dans la phase de l'implémentation on a utilisé deux types de caméras, Kinect et Go Pro. La Kinect (figure 5.6a) est un périphérique de Microsoft constitué d'une barre horizontale connectée à sa base via un un pivot motorisé, celui-ci permet à la caméra d'effectuer des petits mouvements vers le haut ou le bas, afin d'adapter sa perception. La barre horizontale est équipée d'une caméra RGB, d'un capteur de profondeur "3D depth sensor" et d'une série de microphones multi-réseaux.

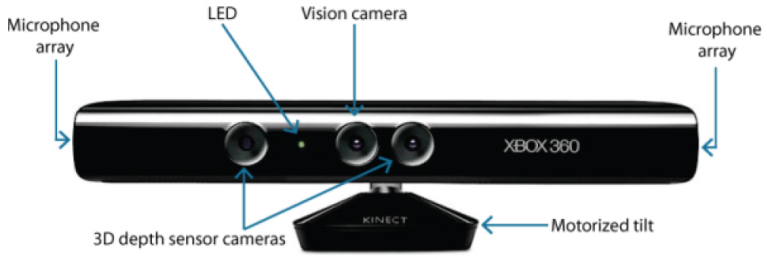
Le robot a été équipé, pour des essais en mode monoSLAM, par une caméra Go Pro HERO 3 (figure 5.6b), dont la matrice de calibration obtenue par l'algorithme de calibration d'une caméra de OpenCV est :

$$M = \begin{pmatrix} 857 & 0 & 960 \\ 0 & 876 & 540 \\ 0 & 0 & 1 \end{pmatrix}$$

La kinect a comme caractéristiques, les points décrits dans le tableau 5.2 ci-dessous :

étendue de mesure	$1.2m$ - $3.5m$
champ de vision horizontal	57°
champ de vision vertical	43°

Tableau 5.2 – Caractéristiques de la Kinect



(a) Kinect

(b) Go Pro HERO 3

Figure 5.6 – Caméras utilisées

5.2.6 Architecture électronique

L'électronique de notre robot est constituée de quatre parties (figure 5.7).

— La première faisant la commande en tension des moteurs droit et gauche (carte puissance) à partir des consignes générées de la carte d'asservissement.

— La partie asservissement s'occupe de la régulation de la consigne de tension appliquée en entrée de la carte puissance, elle reçoit des consignes de vitesses depuis la carte de navigation à travers une liaison série, calcule les consignes selon des algorithmes PID et les envoie sous forme de PWM à la carte moteur.

— La partie évitement est responsable de récupérer les distances mesurées par les capteurs infrarouge et les fournir à la carte asservissement dans le cas de présence d'obstacles.

— Enfin, la carte navigation intègre un programme de ORB-SLAM2 qui reçoit de la Kinect les images à chaque nouvelle frame, calcule selon l'algorithme ORB-SLAM2, puis envoie la localisation et la cartographie au programme "Navigation" (Simulink). Ce denier, reçoit les données de ORB-SLAM2 sous forme de fichier et reçoit manuellement une position désirée, calcule à base de la logique floue les nouvelles consignes de vitesses.

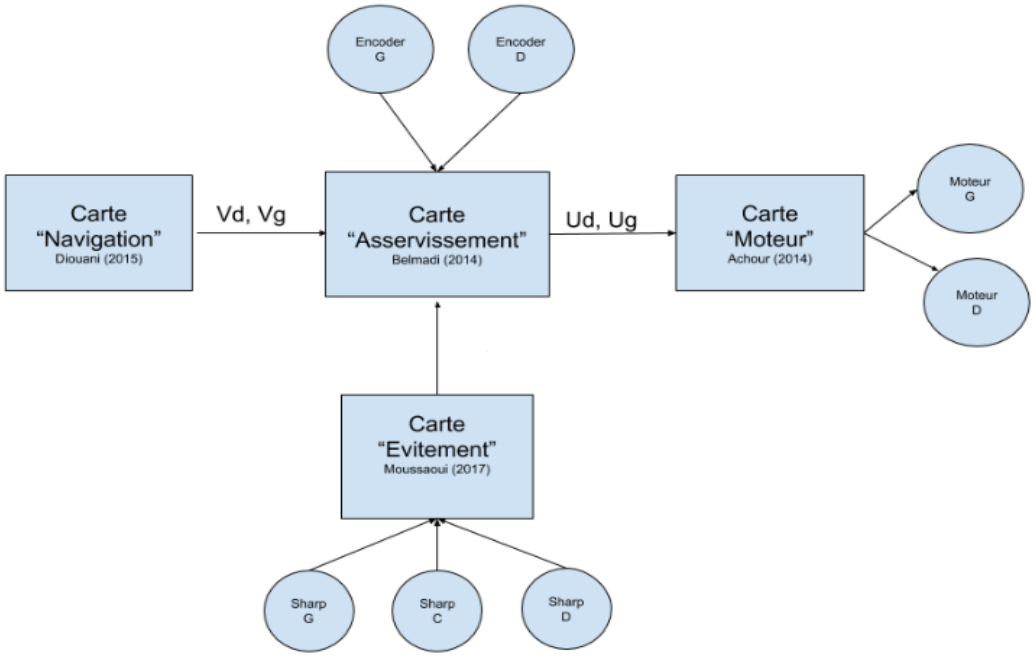


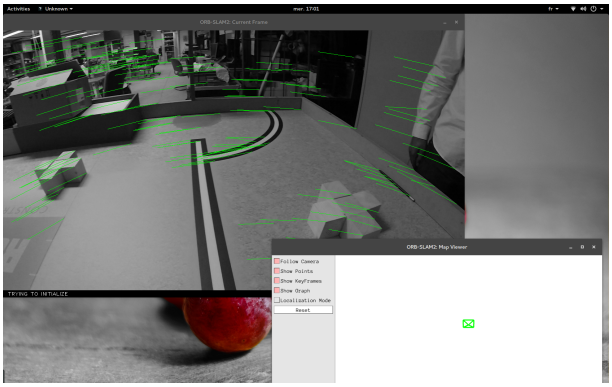
Figure 5.7 – Architecture électronique

5.3 Localisation par V-SLAM

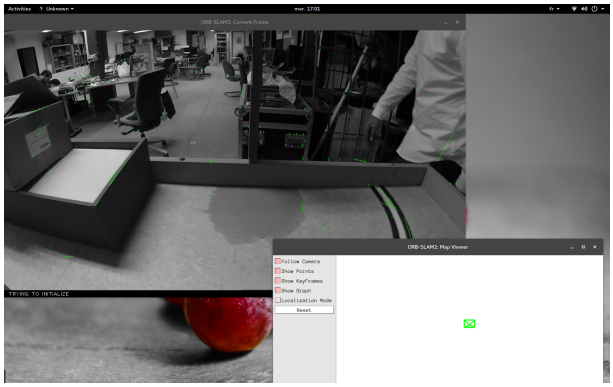
Dans cette section, on va implémenter l'algorithme ORB-SLAM2 dans le cas monoculaire et RGB-D. Il sert à calculer la trajectoire de la caméra et sa localisation en temps réel. Deux exemples de ORB-SLAM2 ont été fournis pour le cas de RGB-D et le cas monoculaire.

5.3.1 Test sur le monoSLAM

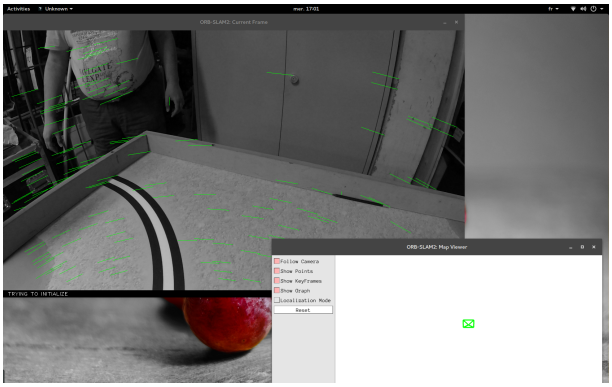
Des tests dans une salle ont été effectués sur ORB-SLAM2 en mode monoSLAM afin de localiser la caméra Go Pro dans cet environnement. Les résultats de localisation de ORB-SLAM2 en mode monoSLAM sont représentés sur la figure 5.8, avec une image chaque 5 secondes. Ils n'étaient pas concluants, il y a eu beaucoup de difficultés à l'initialiser (temps d'initialisation très important) plus la présence des pertes de localisation en cas de rotation pure ou translation pure. Cet algorithme dans le cas monoculaire est intéressant mais sans précautions, ses performances sont insuffisantes pour être reste utilisable en temps réel pour une boucle de régulation (comme une navigation) et ce malgré un paramétrage d'un nombre élevé de features.



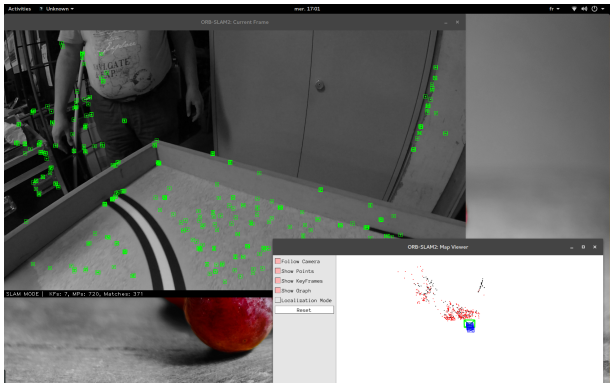
(a) $t_0 = 0\,s$



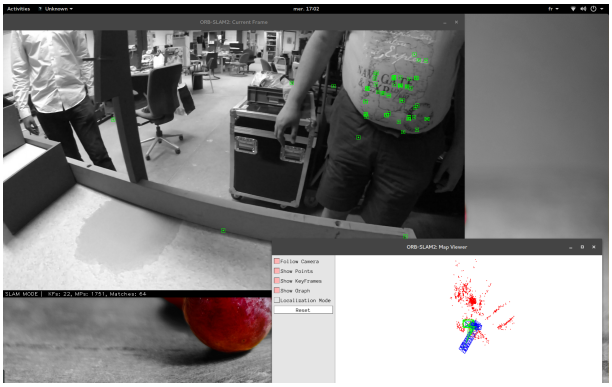
(b) $t_1 = 5\,s$



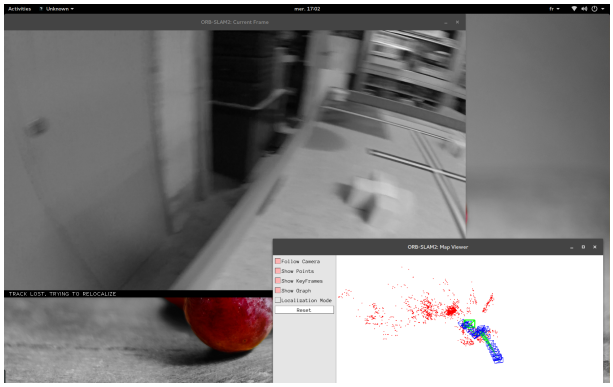
(c) $t_2 = 10\,s$



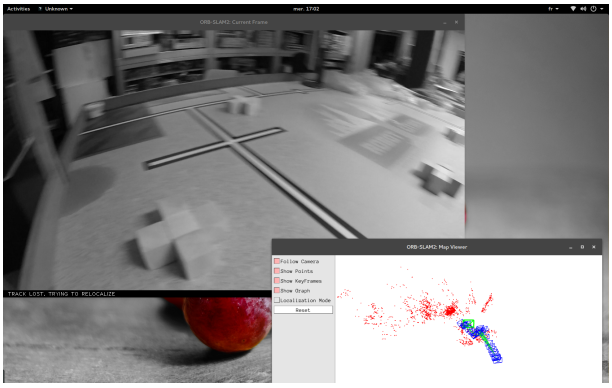
(d) $t_3 = 15\,s$



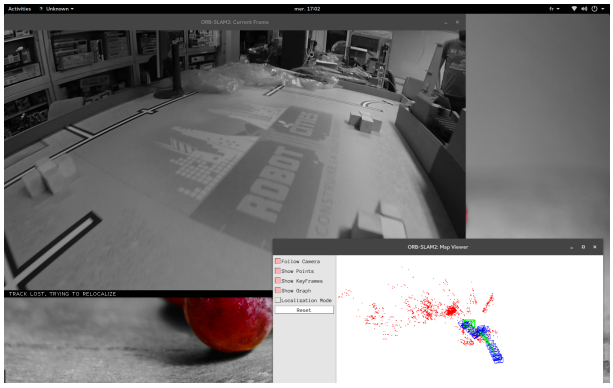
(e) $t_4 = 20\,s$



(f) $t_5 = 25\,s$



(g) $t_6 = 30\,s$



(h) $t_7 = 35\,s$

Figure 5.8 – Localisation ORB-SLAM2 en mode monoSLAM

5.3.2 Test sur le RGB-D SLAM

Le même test a été effectué en mode RGB-D SLAM. Nous avons déplacé le robot entre les positions $P_0 = (0,0,0)$ et $P_1(40,40,0)$ en faisant plusieurs aller/retour et avons récupéré à chaque fois les valeurs pour observer s'il y a eu une dérive. Le tableau 5.3 compare les résultats obtenus avec RGB-D SLAM avec ceux obtenus par Cyborg en 2017 (Fusion de données multicapteurs pour localisation de robot mobile 2017). Pour compliquer le tout : entre chaque aller/retour, nous avons pris des chemins compliqués avec plusieurs rotations et obstacles (voir figure 5.9). Les résultats étaient beaucoup plus concluants (initialisation immédiate et localisation quasi-parfaite).

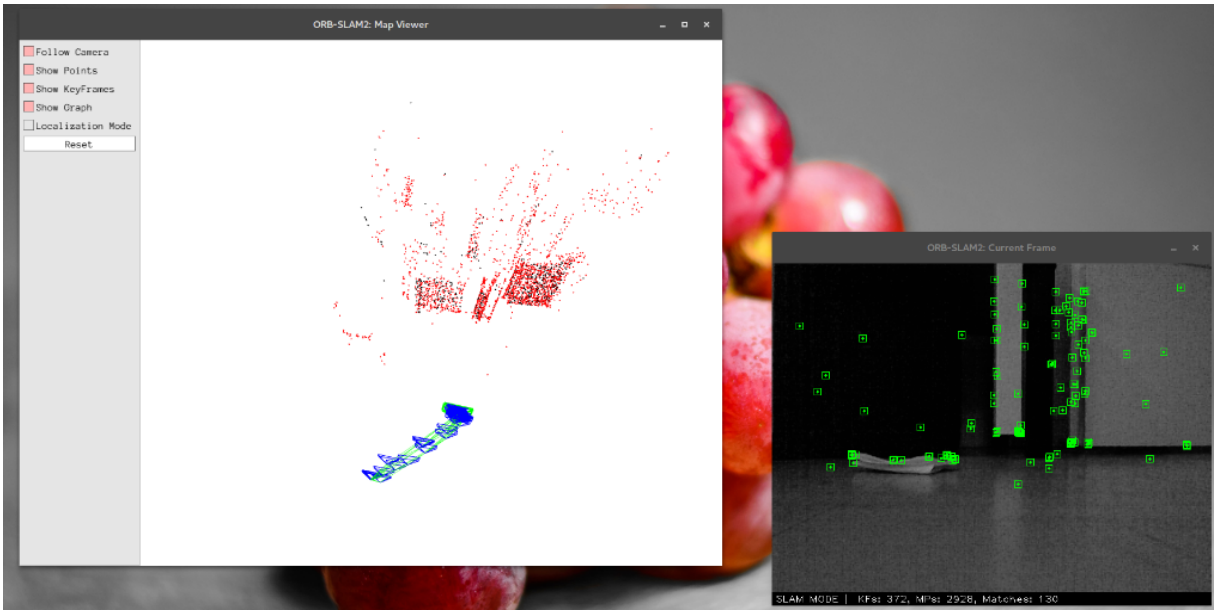


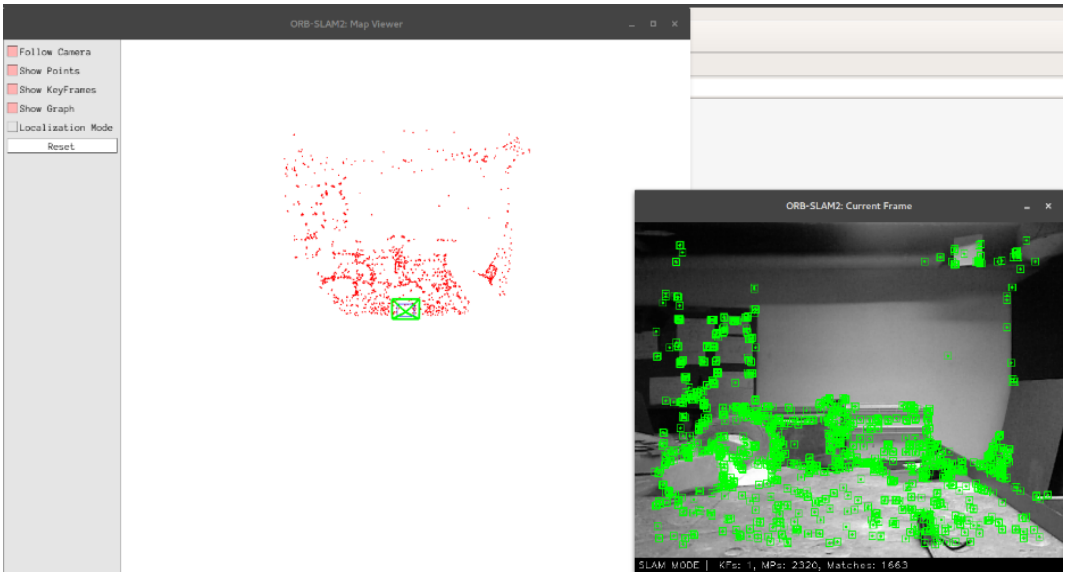
Figure 5.9 – Résultats de localisation ORB-SLAM2 en mode RGB-D SLAM

Valeurs réelles			**RGB-D SLAM**			**Odométrie (Cyborg 2017)**		
x	y	θ	x	y	θ	x	y	θ
0	0	0	−0.0370	0.0200	-1.7094^{-4}	0	0	0
40	40	0	39.9858	40.6389	−0.0975	40.53	42.87	0.01
0	0	0	−0.5027	−0.6823	−0.0111	−2.06	2.71	0.01
40	40	0	40.8477	40.5749	−0.0064	38	43.76	0
0	0	0	0.0313	−4.2988	−0.0264	−4.75	5.96	−0.01
40	40	0	40.5742	41.2686	−0.0954	37.01	46.09	−0.02

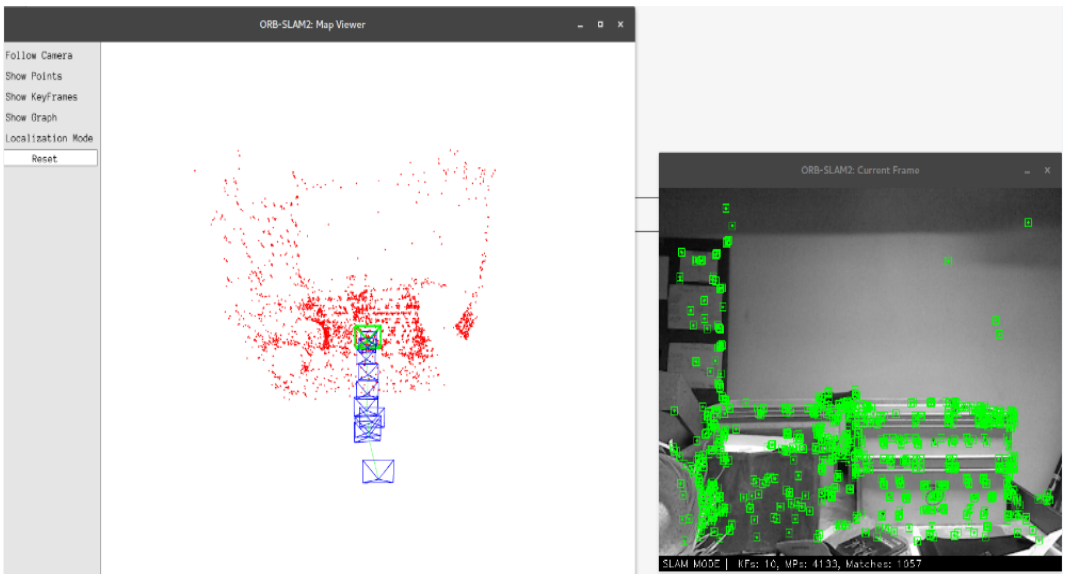
Tableau 5.3 – Comparatif de localisation

5.4 Navigation floue par V-SLAM

L'algorithme implémenté de ORB-SLAM2 fournit une interface graphique afin de permuter entre le mode SLAM et le mode Localisation. Les tests effectués cette fois-ci étaient en mode RGB-D SLAM pour la localisation et la cartographie. Les résultats obtenus sont moyennement satisfaisants pour une navigation floue avec FLC20. Les figures 5.10a et 5.10b représentent les résultats d'une navigation floue de $P_0(0,0,0)$ à $P_f(1,0)$ avec V-SLAM.



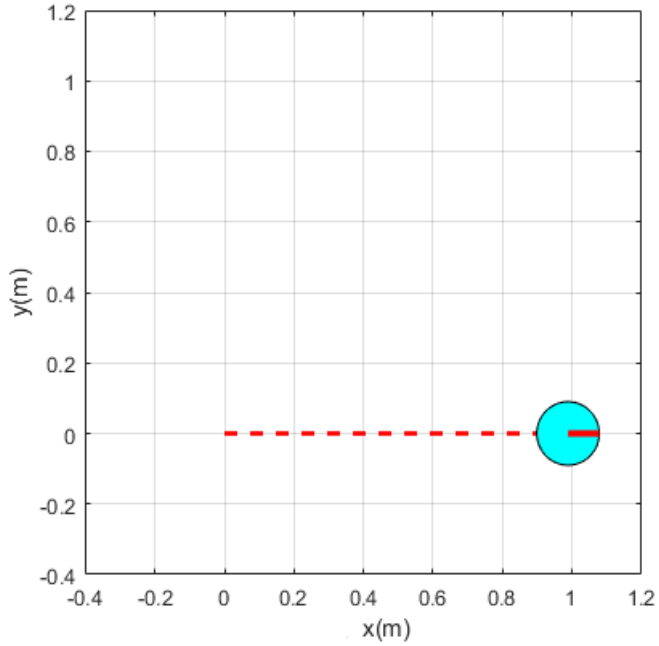
(a) Début de navigation floue



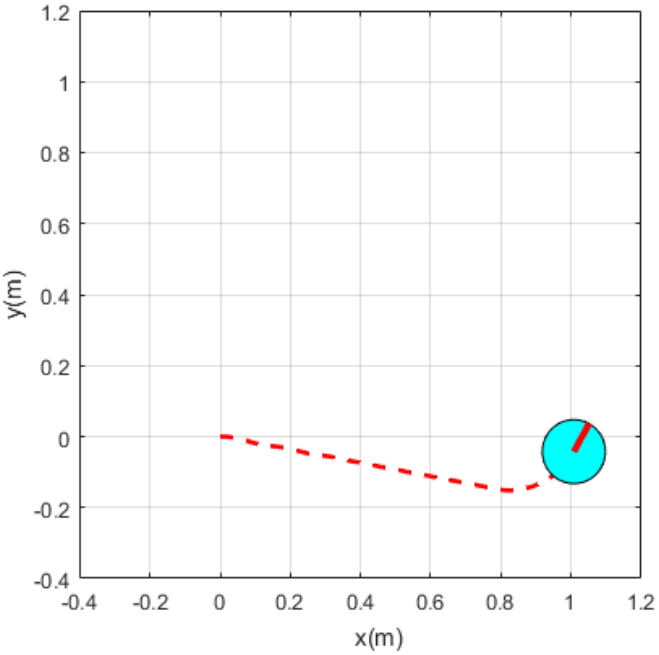
(b) Fin de navigation floue

Figure 5.10 – Résultats de navigation floue avec ORB-SLAM2 en mode RGB-D SLAM

Une comparaison de navigation floue de $P_0(0,0,0)$ à $P_f(1,0)$ avec des différentes techniques de localisation est illustrée sur les figures 5.11a et 5.11b. La première est avec une localisation basée sur des mesures odométriques et la deuxième utilise le V-SLAM pour une cartographie et une localisation simultanées visuelles.



(a) Navigation floue avec odométrie



(b) Navigation floue avec V-SLAM

Figure 5.11 – Comparaison d'une navigation floue

5.5 Conclusion

Dans ce chapitre, nous nous sommes intéressés à la mise en œuvre pratique des différents modes de V-SLAM sur un robot mobile et à l'implémentation d'un régulateur flou pour une tâche d navigation autonome. Nous avons commencé par une présentation des capteurs et actionneurs utilisés et de la structure électronique mise en question. Puis, nous avons exploité les techniques de V-SLAM pour la localisation et aussi pour la localisation et la cartographie simultanées visuelles pour obtenir de bonnes mesures.

Enfin, nous avons implémenté un navigateur flou pour l'asservissement visuel d'une part et pour un asservissement odométrique d'autre part. Les essais réalisés pour tester les performances de nos techniques nous ont permis de remarquer que les performances sont satisfaisantes. En effet, les résultats sont quasiment toujours meilleurs que ceux obtenus par les méthodes avancées.

Conclusion Générale et Perspectives

Le travail présenté dans ce mémoire concerne le développement des stratégies intelligentes de navigation autonome d'un robot mobile basées sur les systèmes flous et l'apprentissage par renforcement à l'aide des informations visuelles acquises à partir du capteur RGB-D et prévenantes de l'environnement qui l'entoure. Ceci est afin de lui permettre de mieux comprendre la scène dans laquelle il se déplace. La tâche de base que doit réaliser robot mobile est d'atteindre un but d'arrivée à partir d'un point de départ sans collision avec les obstacles et sans intervention humaine. Pour ce faire, nous avons utilisé des régulateurs flous et renforcement-flous et avons implménté une technique V-SLAM.

Ce travail se divise en deux parties :

Dans une première partie, nous avons présenté les concepts de ce travail à savoir ; la robotique mobile, la vision artificielle, la logique floue et l'apprentissage par renforcement.

Au début, un état de l'art sur le domaine de la robotique mobile, la vision par ordinateur et le SLAM visuel a été mis en œuvre.

Puis, la théorie de la logique floue a été exposée et l'architecture de base d'un régulateur flou a été présentée. Le fonctionnement d'un régulateur flou dépend d'un nombre important de paramètres qu'il faut déterminer afin d'optimiser ses performances. Les régulateurs flous présentent la possibilité d'incorporer des connaissances expertes dans leurs structures, ce qui peut faciliter le bon choix des paramètres.

Par la suite, nous avons exposé les principes de l'apprentissage par renforcement, cette méthode permet d'explorer l'espace des solutions possibles pour les problèmes d'apprentissage et d'optimisation. Ces méthodes et ces algorithmes ont été proposés et appliqués pour la résolution des problèmes d'apprentissage dans le domaine de la robotique mobile.

Dans la deuxième partie, les contributions de ce travail sont présentées.

Dans un premier lieu, nous avons utilisé deux régulateurs flous de Mamdani pour la navigation autonome d'un robot différentiel avec une méthode de conception spécifique : pour l'opération de fuzzification des entrées, nous avons utilisé des fonctions d'appartenances triangulaires et trapézoïdales, pour la sortie, nous avons utilisé des singletons, min comme opérateur d'implication flou et les méthodes du centre de gravité et la moyenne des maxima pour la défuzzification. La méthode de conception est présentée et la structure analytique de ces régulateurs est développée. La commande à base de la logique floue a démontré son efficacité pour les différentes missions du robot mobile (convergence vers un but, évitements d'obstacles). Ces régulateurs flous présentent aussi des avantages considérables par rapport à leur conception de base ; à savoir leurs paramètres sont interprétables physiquement et leurs structures présentent une faculté à introduire des connaissances expertes. L'inconvénient majeur de cette technique est la nécessité d'avoir suffisamment de connaissances à priori pour déduire les règles linguistiques adéquates.

En l'absence d'expertise, on utilise généralement les techniques d'apprentissage pour la construction de la base de règles. Puis, nous nous sommes penchés sur les méthodes d'apprentissage inspirées de la nature humaine qui ont été gâtées par un intérêt considérable de la communauté des sciences techniques pendant ces dernières années. En particulier, les algorithmes d'apprentissage par renforcement ont eu un grand succès à travers plusieurs problèmes d'optimisation et d'apprentissage. Nous avons proposé des méthodes sur la base de ces approches pour l'apprentissage des paramètres flous pour les comportements de convergence vers un but et d'évitement d'obstacles.

Pour cette méthode deux algorithmes basés sur le Q-learning ont été proposées avec une combinaison avec les systèmes flous pour la commande automatique du robot mobile. L'application de Q-learning flou se résume en l'implémentation d'un régulateur flou, la base des règles initiale est améliorée en ligne par une procédure d'apprentissage utilisant un signal de renforcement.

C'est une solution prometteuse pour le problème de navigation d'un robot mobile où cet algorithme permet le réglage des paramètres d'un FIS et représente une extension de Q-learning au cas continu en se basant sur un signal de renforcement. Ce signal de retour permet au navigateur d'ajuster sa stratégie pour améliorer ses performances.

Les performances des algorithmes proposés sont montrées, pour les différents comportements du robot mobile par des simulations où une version de l'algorithme Q-learning flou est utilisée pour le réglage des conclusions des règles floues pour la navigation réactive du robot mobile.

Une comparaison entre les navigateurs flous et hybrides renforcement-flous a été présentée. Les résultats obtenus montrent que les structures étudiées avec des meilleures performances pour la navigation autonome d'un robot mobile.

En second lieu, notre travail s'est orienté vers une implémentation pratique des régulateurs flous avec des données visuelles. Nous nous sommes intéressés au début aux matériels mis en disposition ainsi que les modes de V-SLAM applicables en navigation.

A l'issue de ce travail, ce mémoire ouvre de nouvelles perspectives de recherche parmi lesquelles nous citons :

- L'application des architectures présentées pour des environnements dynamiques.
- L'optimisation des régulateurs flous par des algorithmes évolutionnaires.
- L'hybridation de la logique floue et les réseaux de neurones artificiels dans le but d'accélérer la recherche et d'améliorer les performances.
- Développement d'une navigation basée sur des régulateurs flous type-2.

Bibliographie

[AUS14] A. Aussem, *"Théorie des jeux : Apprentissage par renforcement, Notes de cours"*, université de Lyon1, 2014.

[BAR11] A. Barrera, *"Advances in Robot Navigation"*, InTech, Vienna, 2011.

[BAY07] B. Bayle,, *"Robotique Mobile"*, Ecole Nationale Supérieure de Physique de Strasbourg, Université Louis Pasteur, 2007.

[BEL57] R. E. Bellman, *"Dynamic Programming"*, Princeton University Press, Princeton, New Jersey, USA, 1957.

[BOU09] H. Boubertakh, *"Contribution à l'Optimisation par Algorithmes Evolutionnaires des Contrôleurs Flous"*, Thèse de Doctorat en Automatique, ENP, 2009.

[BOU10] B. Bouzy, *"Apprentissage par renforcement"*, Cours d'apprentissage automatique, Université ParisV, 2010.

[BER92] H. R. Berenji et P. Khedkar, *"Learning and Tuning Fuzzy Logic Controllers Through Reinforcements"*, IEEE Transactions on Neural Networks, vol. 3, no. 5, pp. 724-740, 1992.

[BRO86] R. Brooks, *"A Robust Layered Control System for a Mobile Robot"*, IEEE Journal of Robotics and Automation, vol. 2, no.1, pp.14-23, 1986.

[BUH94] H. Bühler, *"Réglages par logique floue"*, Presses polytechniques et Universitaires Romandes, 1994.

[CAN18] J. Cano, *" Cinématique d'un robot à montage différentiel et algorithme de suivi de chemin, Notes de cours"*, E-Gab Centrale Marseille, 2018.

[CHE14] L. Cherroun, *"Navigation Autonome d'un Robot Mobile par des Techniques Neuro-Floues "*, Thèse de Doctorat en Automatique, Université Mohamed Khider, Biskra, 2014.

[DAV07] A. J. Davison, I. D. Reid, N. D. Molton, et O. Stasse, *"Monoslam : Realtime single camera slam"*, IEEE transactions on pattern analysis and machine intelligence, 29(6) :1052–1067, 2007.

[ENG14] J. Engel, T. Schöps et D. Cremers, *"Lsd-slam : Large-scale direct monocular slam"*, In European Conference on Computer Vision, pages 834–849, Springer, 2014.

[FAT06] A. Fatmi, A. Al Yahmadi, L. Khriji, et N. Masmoudi, *"A Fuzzy Logic Based Navigation of a Mobile Robot"*, World Academy of Science, Engineering and Technology, vol. 22, pp. 169-174, 2006.

[FIL16] D. Filliat, *"Robotique Mobile"*,Ecole Nationale Supérieur des Techniques Avancées ENSTA, Octobre 2016.

[GER07] P. Gérard, *" Apprentissage par Renforcement : Apprentissage Numérique, Notes de cours"*, l'université de Paris 13, LIPN, 2007.

[GOD97] J. Godjevac, *"Neuro-Fuzzy Controllers, Design and Application"*, Collection Meta, Presses Polytechniques et Universitaires Romandes, Lausanne, 1997.

[JAN07] J. Jantzen, *"Foundations of Fuzzy Control"*, West Sussex, England, 2007.

[JOU98] L. Jouffe, *"Fuzzy Inference System Learning by Reinforcement Methods"*, IEEE Transactions on Systems, Man, and Cybernetics, vol.28, no.3, pp. 338-355, 1998.

[KHA86] O. Khatib, *"Real-Time Obstacle Avoidance for Manipulators and Mobile Robots"*, International Journal of Robotics Research, vol. 5, no. 1, pp. 90-98, 1986.

[KHR11] L. Khriji et Al Yahmedi, *"Mobile Robot Navigation Based on Q-learning Technique"*, International journal of advanced Robotic System, vol. 8, no. 1, pp 45-51, 2011.

[KLE07] G. Klein et D. Murray, *"Parallel tracking and mapping for small ar workspaces, In Mixed and Augmented Reality"*, 6th IEEE and ACM International Symposium, pages 225–234, 2007.

[LAB98] S. Labiod, *"Commande Adaptative par les Systèmes Flous ; application aux robots manipulateurs"*, Thèse de Magister en Automatique, ENP, 1998.

[LAB05] S. Labiod, *"Contribution à la Commande Adaptative Floue des Systèmes Non Linéaires"*, Thèse de Doctorat en Automatique, ENP, 2005.

[LAU01] J.P. Laumond, *"La Robotique Mobile"*, Editions Hermès, 2001.

[LEF06] O. Lefebvre, *"Navigation autonome sans collision pour robots mobiles nonholonomes"*, Thèse de Doctorat, Institut National Polytechnique de Toulouse, 2006.

[MAM74] E.H. Mamdani, *"Application of Fuzzy Algorithms for Control of Simple Dynamic Plant"*, Proc. of IEEE, Vol. 121, No. 12, pp. 1585-1588, 1974.

[MAM75] E. H. Mamdani and S. Assilian, *"An Experiment in Linguistic Synthesis with a Fuzzy Logic Controller"*, International Journal of Man-Machine Studies, vol. 7, No. 1, pp. 1-13, 1975.

[MEI07] C. Mei et P. Rives, *"Cartographie et localisation simultanée avec un capteur de vision"*, Journées Nationales de la Recherche en Robotique, 2(1.9), 2007.

[MUR15] R. Mur-Artal, J. M. M. Montiel et J. D. Tardos, *"Orb-slam : a versatile and accurate monocular slam system"*, IEEE Transactions on Robotics, 31(5) : 1147–1163, 2015.

[MYK05] K. Mykhaylo et al, *"Using fuzzy Inference System as a Function Approximator of a State Action Table"*, Proceedings of Advanced Aspects of Theoretical Electrical Engineering, Sozopol, Bulgaria, October 2005.

[PRE07] P. Preux, *"Apprentissage par renforcement, Notes de cours"*, GRAPPA, cours de l'université de Lille1, octobre 2007.

[SIL15] D. Silver, *"Lectures in reinforcement learning"*, 2015.

[SUG85] M. Sugeno and G. T. Kang, *"Fuzzy identification of systems and its applications to modeling and control"*, IEEE Trans. Syst., Man, Cybern., vol. 15, pp. 116-132, 1985.

[SUT88] R. S. Sutton, *"Learning to Predict by the Methods of Temporal Differences"*, Machine Learning, vol. 3, pp.9-44, 1988.

[SUT98] R. S. Sutton et A.G. Barto. *"Reinforcement Learning : An Introduction"*, MIT Press, Cambridge, 1998.

[TOU99] C. Touzet, *"L'apprentissage par Renforcement"*, Paris : Masson, 1999.

[TUN97] E.Tunstel, *"Behaviour Hierarchy for Autonomous Mobile Robots : Fuzzy-Behaviour Modulation and Evolution"*, Intl J. of Inteligent automation and soft computing, 1997.

[WAT89] C. J. C. H. Watkins, *"Learning from Delayed Rewards"*, Thèse de doctorat, Université de Cambridge, L'Angleterre, 1989.

[WAT92] C. J. C. H. Watkins et P. Dayan,*"Technical Note, Q-learning"*, Machine Learning, vol. 8, pp. 279-292, 1992.

[ZAD65] L. A. Zadeh, *"Fuzzy sets"*, Information and Control, vol. 8, pp.338-353, 1965.