République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Kasdi Merbah Ouargla
Faculté des Sciences Appliquées
Département de Génie électrique

THÈSE

Présentée pour l'obtention du diplôme de
DOCTORAT en SCIENCES
Spécialité : Génie électrique

Par : BENMAKHLOUF Abdeslam

Thème

Navigation Intelligente Autonome Pour Robot Mobile

Soutenu le 05/10/2023 devant le jury composé de :

Dr. BOUREK Yacine	Président	MCA	Univ Ouargla
Dr. MERAOUMIA Abdallah	Rapporteur	Pr	Univ Tébessa
Dr. LOUAZENE Med Lakhdar	Co-rapporteur	MCA	Univ Ouargla
Dr. DIB Djalel	Examinateur	Pr	Univ Tébessa
Dr. MEGHNI Billel	Examinateur	Pr	Univ Annaba
Dr. BENYOUSSEF Lakhdar	Examinateur	MCA	Univ Ouargla

Dédicaces

A ma famille et mes amis

Remerciements

Je souhaite exprimer ma gratitude envers tous ceux qui ont apporté leur contribution, qu'elle soit directe ou indirecte, à l'achèvement de ce modeste projet.

Résumé

Les robots mobiles à roues présents de nos jours dans plusieurs secteurs d'activités, sont des machines dotées des capacités de perception, de réflexion et d'actions pour naviguer d'une façon autonome dans leur environnement en toute sécurité. Cette compétence de navigation autonome requiert une combinaison de ressources matérielles et logicielles pour accomplir des tâches élémentaires telles que la planification de chemin, l'évitement d'obstacles et le contrôle du mouvement en fonction de la situation de navigation.

La navigation dans des environnements statiques ou structurés ne pose pas de problèmes aux techniques de navigation classiques. Cependant, les environnements dynamiques présentent plusieurs défis où le robot doit être capable de s'adapter rapidement à de nouvelles situations en temps réel pour garantir la sûreté de ses mouvements. Les approches classiques de contrôle de robots mobiles n'assurent pas dans l'ensemble une navigation réussie dans ces conditions d'où vient la nécessité de nouvelles techniques de navigation basées sur les algorithmes de l'intelligence artificielle.

L'apprentissage par renforcement est parmi les méthodes intelligentes adoptées pour faire face aux défis de la navigation autonome dans des environnements dynamiques. C'est une technique basée sur l'interaction entre un agent qui a comme but d'apprendre une politique d'action et son environnement.

Dans cette thèse, l'apprentissage par renforcement a été combiné avec des méthodes classiques de navigation pour développer deux stratégies de navigation : globale et locale. Pour la navigation globale, l'algorithme *TD3* d'apprentissage par renforcement a été combiné avec l'algorithme *PRM* de planification globale de chemin. Le résultat est un algorithme qui peut planifier un chemin sans collision reliant la position de départ à la position cible en exploitant la carte de l'environnement et qui a la compétence de détection et d'évitement en temps réel d'obstacles dynamiques. Pour la navigation locale, un module de contrôle de vitesse qui peut s'adapter aux changements dans l'environnement a été développé pour éviter les obstacles dynamiques en combinant les algorithmes *DWA* et *DQN*.

Nous avons également étudié un aspect fondamental de la commande des robots mobiles à roues, qui est le suivi de chemin. Dans ce contexte, nous avons conçu un contrôleur de suivi de chemin pour les robots mobiles à roues à commande différentielle basé sur le modèle d'inférence floue à *SIRM* connecté dynamiquement. Ce type spécifique de modèle d'inférence est utilisé pour résoudre le problème de l'augmentation du nombre de règles d'inférence dans la commande floue classique pour les systèmes de commande multi-entrées. La structure simple du contrôleur basé sur les *SIRM* permet également de réduire le temps de traitement. Le contrôleur peut conduire un robot mobile pour suivre une trajectoire de référence discrète en ajustant les vitesses angulaire et linéaire du robot.

Mots clés : *Robot mobile, Navigation autonome, Environnement dynamique, Apprentissage par renforcement, Planification de chemins, Evitement d'obstacles, Suivi de chemin, Logique floue, SIRM.*

Abstract

Wheeled mobile robots, which are present in several fields of activities nowadays, are machines equipped with perception, reasoning, and action capabilities to navigate autonomously and safely in their environments. This autonomous navigation skill requires a combination of hardware and software resources to perform basic tasks such as path planning, obstacle avoidance, and motion control with respect of the current navigation situation.

Navigation in static or structured environments does not pose problems for classical navigation techniques. However, dynamic environments present several challenges where the robot must be able to quickly adapt to new situations in real time to ensure the safety of its movements. Classical approaches to mobile robot control do not ensure successful navigation in these conditions, hence the need for new navigation techniques based on artificial intelligence algorithms.

Reinforcement learning is one of the intelligent methods adopted to address the challenges of autonomous navigation in dynamic environments. It is a technique based on the interaction between an agent whose goal is to learn an action policy and its environment.

In this thesis, reinforcement learning has been combined with classical navigation methods to develop two navigation strategies: a global and a local one. For global navigation, the reinforcement learning algorithm *TD3* has been combined with the global path planning algorithm *PRM*. The result is an algorithm that can plan a collision-free path connecting the starting position to the target position by exploiting the environment map and that has the competence of real-time obstacle detection and avoidance. For local navigation, a speed control module that can adapt to changes in the environment has been developed to avoid dynamic obstacles by combining the *DWA* and *DQN* algorithms.

We also studied a fundamental aspect of wheeled mobile robots control, which is the path following. In this context, we designed a path following system for differential drive mobile robots based on the dynamically connected fuzzy inference *SIRM* model. This specific type of inference model is used to solve the problem of increasing the number of inference rules in classical fuzzy control for multi-input control systems. The simple structure of the controller based on *SIRM* also reduces processing time. The controller can drive a mobile robot to follow a discrete reference trajectory by adjusting the robot's angular and linear velocities.

Keywords: *Mobile robot, Autonomous navigation, Dynamic environment, Reinforcement learning, Path planning, Obstacle avoidance, Path following, Fuzzy logic, SIRM.*

ملخص

الروبوتات المتحركة على العجلات، والتي توجد في عدة مجالات في عصرنا هذا، هي آلات مجهزة بقدرات الاستشعار والتفكير والحركة للتنقل بشكل مستقل وآمن في بيئتها. تتطلب هذه المهارة في الانتقال الذاتي مزيجًا من التجهيزات والبرامج لأداء المهام الأساسية مثل تخطيط المسار وتجنب العوائق والتحكم في الحركة بناءً على الحالة الملاحية.

لا تشكل الملاحة في البيئات الثابتة أو المنظمة مشكلة بالنسبة لتقنيات الملاحة الكلاسيكية. في حين، تمثل البيئات الديناميكية تحديات عدة حيث يجب أن يكون الروبوت قادرًا على التكيف بسرعة مع الحالات الجديدة في الوقت الحقيقي لضمان سلامة حركته. لا تضمن الطرق الكلاسيكي للتحكم في الروبوت المتحرك ملاحة ناجحة في هذه الظروف، ومن هنا فإن هناك حاجة لتقنيات ملاحة جديدة تستند إلى خوارزميات الذكاء الاصطناعي.

التعلم المعزز هو أحد التقنيات الذكية المعتمدة لمواجهة تحديات الملاحة الذاتية في البيئات الديناميكية. إنها تقنية تستند إلى التفاعل بين عامل وبيئته المحيطة بهدف تعلم سياسة للتحرك والتفاعل مع المحيط من أجل انجاز مهامه بنجاح.

في هذه الاطروحة، تم دمج التعلم المعزز مع أساليب الملاحة الكلاسيكية لتطوير استراتيجيتين للملاحة: شاملة ومحلية. للملاحة الشاملة، تم دمج خوارزمية التعلم المعزز مع خوارزمية التخطيط المسار الكلي *PRM*. النتيجة هي خوارزمية تستطيع التخطيط لمسار خالٍ من الاصطدام يصل بين نقطة الانطلاق ونقطة الوصول عن طريق استغلال خريطة المحيط ولديها قدرة على تفادي العوائق في الوقت الحقيقي. أما بالنسبة للملاحة المحلية، تم تطوير وحدة تحكم في السرعة تستطيع التكيف مع تغييرات البيئة لتفادي العوائق الديناميكية عن طريق دمج خوارزميتي *DWA* و*DQN* .

لقد درسنا أيضًا جانبًا أساسيًا في التحكم في الروبوتات الذاتية الحركة ذات العجلات، وهو تتبع المسار. في هذا السياق، قمنا بتصميم نظام متابعة للمسار للروبوتات الذاتية الحركة ذات القيادة التفاضلية باستخدام نموذج الاستدلال الضبابي المتصل ديناميكيًا. يتم استخدام هذا النوع المحدد من النموذج الاستدلالي لحل مشكلة زيادة عدد قواعد الاستدلال في التحكم الضبابي الكلاسيكي لنظم التحكم المتعددة المداخل. كما يخفض بناء نظام التحكم الذي يعتمد على النموذج ديناميكيًا وقت المعالجة، حيث يستطيع المتحكم قيادة الروبوت الحركي لمتابعة مسار مرجعي عن طريق ضبط سرعات الروبوت الدورانية والخطية.

الكلمات المفتاحية: الروبوت المتحرك، الملاحة التلقائية، البيئة الديناميكية، التعلم بالمعزز، تخطيط المسار، تجنب العقبات، تتبع المسار، المنطق الضبابي، وحدة القاعدة أحادية المدخل.

Table des Matières

Liste des Figures VII
Liste des Tableaux IX
Liste des Algorithmes X
Liste des Abréviations XI
1. Introduction Générale 1
1.1. La navigation autonome de robots mobiles 1
1.2. La navigation intelligente de robots mobiles 2
1.3. Organisation de la thèse 5
2. Etat de l'Art 6
2.1. Introduction 6
2.2. La robotique mobile 6
2.3. La navigation autonome 8
2.3.1. Navigation basée sur une carte 10
2.3.2. Navigation sans carte 12
2.4. L'apprentissage automatique et la navigation autonome 13
2.5. Planification de trajectoire 17
2.6. Classification des algorithmes de planification de trajectoire 17
2.6.1. Planification globale 18
2.6.2. Planification locale et évitement d'obstacles 19
2.7. L'apprentissage par renforcement dans la planification de trajectoire 22
2.8. Contrôle de mouvement 24
2.9. Conclusion 27
3. Apprentissage par Renforcement 29
3.1. Introduction 29
3.2. Réseaux de neurones et apprentissage profond 29
3.2.1. Le neurone artificiel 29
3.2.2. Réseau de neurones artificiels 30
3.2.3. Réseau de Neurones Convolutif 31
3.2.4. Apprentissage et rétropropagation 32
3.2.5. Apprentissage supervisé 33
3.2.6. Apprentissage par renforcement 34
3.3. Définitions et notions 34
3.3.1. Définitions 34
3.3.2. Les Processus de Décision Markovien (MDP) 35
3.3.3. Politique et fonction de valeur 37

3.3.4. Les Processus Décisionnels de Markov Partiellement Observables 40
3.3.5. Méthodes avec ou sans modèle 43
3.3.6. Exploration vs Exploitation 43
3.3.7. On-Policy vs Off-Policy 44
3.3.8. Objectifs de l'apprentissage par renforcement 45
3.4. Classification des principaux algorithmes d'apprentissage par renforcement 45
3.4.1. Les méthodes tabulaires 45
3.4.1.1. La programmation dynamique : algorithmes basés sur un modèle 47
3.4.1.2. Itération de politique 48
3.4.1.3. Itération de valeur 49
3.4.2. Apprentissage par Renforcement : Algorithmes sans modèle 50
3.4.2.1. Les méthodes de Monte Carlo 51
3.4.2.2. Les méthodes de la différence temporelle 52
3.4.3. Approximation de fonction 53
3.4.3.1. Algorithmes basés sur la valeur 53
3.4.3.1.1. Deep Q-learning (DQN) 54
3.4.3.1.2. Améliorations et variantes 55
3.4.3.2. Le gradient de politique (Policy Gradient) 58
3.4.3.3. Acteur-Critique 62
3.4.3.3.1. Gradient de politique déterministe : (D)DPG, RDPG, D4PG) 64
3.4.3.3.2. Apprentissage multi-threads : A3C, A2C, ACER 65
3.4.3.3.3. Algorithmes de région de confiance : TRPO, ACKTR, PPO 67
3.4.3.3.4. L'entropie avec l'apprentissage par renforcement : SQL, SAC 70
3.5. Conclusion 71
4. Planification de Chemins 72
4.1. Introduction 72
4.2. L'algorithme PRM 73
4.3. Configuration du système de navigation 75
4.4. Configuration de l'agent TD3 76
4.4.1. Définition de l'état 79
4.4.2. Définition des actions 79
4.4.3. Définition de la fonction récompense 79
4.5. Configuration de l'environnement DRL 80
4.5.1. Environnement d'apprentissage 80
4.5.2. Environnement de test 81
4.6. Apprentissage et résultats de simulation 82
4.6.1. Test 1 84

4.6.2. Test 2 86
4.7. Conclusion 88
5. Suivi de Chemins **90**
5.1. Introduction 90
5.2. Le modèle cinématique du robot mobile 92
5.3. Modèle d'inférence floue à SIRM connecté dynamiquement 93
5.4. La stratégie du suivi de chemin 96
5.5. Le contrôleur du robot mobile 98
5.5.1. Le contrôleur d'orientation 99
5.5.2. Contrôleur de vitesse 100
5.6. Résultats de simulation 102
5.7. Conclusion 105
6. Evitement d'Obstacles **107**
6.1. Introduction 107
6.2. Principes de l'approche de la fenêtre dynamique 108
6.3. Configuration du système d'évitement d'obstacles 109
6.4. Configuration de l'agent DQN 111
6.4.1. Définition de l'état 111
6.4.2. Définition des actions 113
6.4.3. Définition de la fonction récompense 114
6.5. Configuration de l'environnement DRL 116
6.5.1. Environnement d'apprentissage 116
6.5.2. Environnement de test 117
6.6. Apprentissage et résultats de simulation 118
6.6.1. Test 1 120
6.6.2. Test 2 121
6.7. Conclusion 122
7. Conclusion Générale **124**
Bibliographie 127

Liste des Figures

Figure 2.1 : *Différents robots mobiles pour différentes applications.* 7

Figure 2.2 : *Architecture de système de navigation classique.* 11

Figure 2.3 : *Architecture de système de navigation locale (sans carte).* 12

Figure 2.4 : *Pipeline traditionnel de contrôle de robot.* 15

Figure 2.5 : *Les méthodes classiques de planification de chemin.* 19

Figure 2.6 : *Les méthodes VO (a : espace de travail, b : $VO_{A|b}$, c : $RVO_{A|b}$, d : $HRVO_{A|b}$).* 21

Figure 3.1 : *Neurone Artificiel.* 30

Figure 3.2 : *Réseau de Neurones Artificiels multicouches.* 31

Figure 3.3 : *Réseau convolutif pour la reconnaissance d'écriture manuscrite.* 32

Figure 3.4 : *Interaction agent-environnement.* 35

Figure 3.5 : *Réseau de décision d'un MDP fini.* 37

Figure 3.6 : *Schéma du POMDP.* 42

Figure 3.7 : *Classification des principaux algorithmes d'apprentissage par renforcement.* 46

Figure 3.8 : *Itération généralisée de politique.* 48

Figure 3.9 : *Deep Q learning.* 54

Figure 3.10 : *Architecture Dueling.* 57

Figure 3.11 : *Echantillonnage de trajectoire.* 62

Figure 3.12 : *Algorithme A3C.* 66

Figure 4.1 : *Exemple de planification de chemin par PRM.* 74

Figure 4.2 : *Architecture du système de navigation globale.* 76

Figure 4.3 : *Le robot mobile Pioneer 3DX.* 76

Figure 4.4 : *Les réseaux TD3.* 77

Figure 4.5 : *L'évaluation de la convergence des algorithmes RL.* 78

Figure 4.6 : *Environnements d'apprentissage.* 81

Figure 4.7 : *Environnement de test.* 81

Figure 4.8 : *L'organigramme de la procédure d'apprentissage.* 82

Figure 4.9 : *Processus d'apprentissage progressif.* 84

Figure 4.10 : *Planification de trajectoire dans une scène à petite échelle.* 85

Figure 4.11 : *Planification de trajectoire dans une scène à grande échelle.* 87

Figure 4.12 : *Planification de trajectoire basée sur PRM+TD3.* 87

Figure 5.1 : *Robot mobile.* 92

Figure 5.2 : *Structure d'un contrôleur flou.* 94

Figure 5.3 : *Structure d'un contrôleur flou à SIRM.* 96

Figure 5.4 : *Paramètres du suivi de trajectoire.* 97

Figure 5.5 : *Architecture du contrôleur du robot mobile.* 98

Figure 5.6 : *Structure du contrôleur d'orientation.* 99

Figure 5.7 : *Fonctions d'appartenance des SIRM du contrôleur d'orientation.* 99

Figure 5.8 : *Fonctions d'appartenance des DID du contrôleur d'orientation.* 100

Figure 5.9 : *Structure du contrôleur de vitesse.* 101

Figure 5.10 : *Suivi d'une trajectoire en forme de 8.* 103

Figure 5.11 : *Commande d'orientation (a : flou, b : SIRM).* 104

Figure 5.12 : *Signaux de la commande d'orientation (a : flou, b : SIRM).* 104

Figure 5.13 : *Signaux de la commande de vitesse (a : flou, b : SIRM).* 105

Figure 5.14 : *Suivi d'une trajectoire sinusoïdale.* 105

Figure 6.1 : *Exemple de la DWA.* 109

Figure 6.2 : *Architecture du système de navigation locale.* 110

Figure 6.3 : *Le module DQN.* 111

Figure 6.4 : *Environnements d'apprentissage.* 116

Figure 6.5 : *Environnements de test.* 118

Figure 6.6 : *Taux de réussite en fonction des récompenses.* 119

Figure 6.7 : *Taux moyens de réussite (Test 1).* 121

Figure 6.8 : *Taux moyens de réussite (Test 2).* 122

Liste des Tableaux

Tableau 2.1 : *Classification des algorithmes de planification de trajectoire.* 18

Tableau 3.1 : *Fonction valeur.* 46

Tableau 3.2 : *Fonction Q.* 47

Tableau 4.1 : *Paramètres d'évaluation.* 85

Tableau 4.2 : *Indices de planification dans un environnement à petite échelle.* 86

Tableau 4.3 : *Indices de planification dans un environnement à grande échelle.* 88

Tableau 5.1 : *Règles floues des SIRM du contrôleur d'orientation.* 100

Tableau 5.2 : *Règles floues des SIRM du contrôleur d'orientation.* 100

Tableau 5.3 : *Règles floues du module SIRM 3.* 101

Tableau 5.4 : *Règles floues du module SIRM 4.* 101

Tableau 5.5 : *Règles floues du module DID 3.* 102

Tableau 5.6 : *Les paramètres des contrôleurs.* 102

Tableau 6.1 : *Paramètres des fonctions de récompenses.* 118

Liste des Algorithmes

Algorithme 3.1 : *Itération de politique.* 48

Algorithme 3.2 : *Itération de valeur.* 50

Algorithme 3.3 : *Deep Q-learning avec répétition d'expérience.* 55

Algorithme 3.4 : *REINFORCE.* 60

Algorithme 3.5 : *Acteur-Critique.* 63

Algorithme 3.6 : *DDPG.* 64

Algorithme 3.7 : *Algorithme A3C.* 67

Algorithme 6.1 : *Algorithme de détection d'obstacles.* 113

Algorithme 6.2 : *Algorithme de calcul de vitesse.* 114

Liste des Abréviations

AI :	*Artificial Intelligence*
ML :	*Machine learning*
DL :	*Deep Learning*
RL :	*Reinforcement Learning*
DRL :	*Deep Reinforcement Learning*
ANN :	*Artificial Neural Networks*
CNN :	*Convolutional Neural Networks*
MDP :	*Markov Decision Process*
POMDP :	*Partially Observable Markov Decision Process*
DP :	*Dynamic Programming*
TD :	*Temporal Difference*
DQN :	*Deep Q-Network*
DDQN :	*Double Deep Q Network*
DDPG :	*Deep Deterministic Policy Gradient*
RDPG :	*Recurrent Deterministic Policy Gradient*
D4PG :	*Distributed Distributional Deterministic Policy Gradients*
A2C :	*Advantage Actor Critic*
A3C :	*Asynchronous Advantage Actor Critic*
ACER :	*Actor-Critic with Experience Replay*
TRPO :	*Trust Region Policy Optimization*
ACKTR :	*Actor-Critic using Kronecker-Factored Trust Region*
PPO :	*Proximal Policy Optimization*
SQL :	*Soft Q-Learning*
SAC :	*Soft Actor-Critic*
RRT :	*Rapidly exploring Random Tree*
PRM :	*Probabilistic Roadmaps*
DWA :	*Dynamic Window Approach*
SIRM :	*Single Input Rule Module*
FLC :	*Fuzzy Logic Control*
LiDAR :	*Light Detection and Ranging*

1. Introduction Générale

1.1. La navigation autonome de robots mobiles

Les robots mobiles sont des machines capables de se déplacer dans leur environnement, souvent en utilisant des roues, des pattes ou d'autres moyens de locomotion. Ils peuvent être équipés de capteurs pour percevoir leur environnement, de systèmes de navigation pour se déplacer en toute sécurité, et de systèmes de contrôle pour prendre des décisions et effectuer des tâches. La robotique mobile est un domaine en constante évolution, avec des applications de plus en plus diverses et une technologie de plus en plus avancée. Elle a le potentiel de transformer la manière dont les humains interagissent avec leur environnement et de résoudre de nombreux problèmes pratiques auxquels ils sont confrontés dans la vie quotidienne. Elle concerne la conception, la construction et l'utilisation de robots mobiles [1].

Les robots mobiles ont de nombreuses applications pratiques, telles que la surveillance, la maintenance, l'exploration, la livraison de colis, la sécurité et la logistique. Ils peuvent être utilisés dans divers environnements, tels que les usines, les entrepôts, les chantiers de construction, les champs agricoles et les espaces extérieurs. Les avancées technologiques en matière de capteurs, de traitement de données, de contrôle et de mécanique ont permis de développer des robots mobiles de plus en plus autonomes et capables de s'adapter à des situations variées.

La navigation autonome d'un robot mobile désigne la capacité d'un robot à se déplacer et à naviguer dans son environnement sans intervention ou assistance humaine. Cela comprend des tâches telles que la planification d'un trajet, l'évitement d'obstacles et l'ajustement du mouvement en fonction des entrées des capteurs. La navigation autonome requiert une combinaison de matériel et de logiciel, telle que des capteurs pour percevoir l'environnement, une puissance de traitement pour prendre des décisions basées sur cette perception et des actionneurs pour exécuter le mouvement. C'est une capacité fondamentale pour la majorité des applications de robots mobiles [2].

La navigation de robots mobiles dans des environnements dynamiques présente plusieurs défis, notamment la nécessité d'une perception et d'une prise de décision en temps

réel pour éviter les obstacles et réagir aux changements de l'environnement [3]. Le robot doit être capable de s'adapter rapidement à de nouvelles situations, telles que des obstacles en mouvement tout en maintenant une navigation sûre et efficace. De plus, le robot doit être capable de prédire avec précision le mouvement des autres objets de l'environnement pour éviter les collisions et se déplacer en douceur à travers les foules. D'autres défis comprennent la gestion du bruit des capteurs et du champ de vision limité, ainsi que la nécessité d'algorithmes et de matériel robustes pour prendre en charge un fonctionnement en temps réel. Dans l'ensemble, une navigation réussie dans des environnements dynamiques nécessite une combinaison de capteurs avancés, d'algorithmes intelligents et de matériel fiable [4].

1.2. La navigation intelligente de robots mobiles

L'intelligence artificielle (*AI*) joue un rôle crucial dans la navigation des robots mobiles. Avec les avancées des algorithmes d'apprentissage automatique et de la vision par ordinateur, les robots sont maintenant capables de percevoir leur environnement, de prendre des décisions en fonction de cette perception et d'exécuter des actions en conséquence. Les algorithmes de navigation basés sur l'*AI* permettent aux robots de gérer des environnements complexes, y compris des environnements dynamiques, et de prendre des décisions en temps réel pour éviter les obstacles et se déplacer efficacement [5-8].

D'autres techniques d'*AI*, telles que la vision par ordinateur et le traitement du langage naturel, ont également été utilisées dans la navigation des robots mobiles. La vision par ordinateur permet aux robots de percevoir leur environnement et de reconnaître les objets et les personnes, tandis que le traitement du langage naturel permet aux humains de communiquer avec les robots et de fournir des commandes de navigation de haut niveau. Ces techniques ouvrent de nouvelles possibilités pour la navigation des robots mobiles dans des environnements complexes et dynamiques [9].

Les techniques d'apprentissage automatique (*Machine Learning ML*) sont de plus en plus utilisées dans la navigation des robots mobiles, car elles permettent aux robots d'apprendre à partir des données et d'améliorer leur performance de navigation au fil du temps. La navigation basée sur l'*AI* implique l'utilisation d'algorithmes pour extraire des fonctionnalités utiles à partir de données de capteurs, apprendre des modèles qui convertissent

les données de capteurs en actions, et optimiser ces modèles en fonction de la performance du robot [10].

Une technique populaire dans la navigation de robots mobiles basée sur l'*AI* est l'apprentissage profond, qui consiste à entraîner des réseaux de neurones à effectuer des tâches de perception et de prise de décision. Par exemple, les *réseaux de neurones convolutifs* (*CNN*) peuvent être utilisés pour extraire des fonctionnalités à partir d'images de caméra, tandis que les réseaux de neurones récurrents (*RNN*) peuvent être utilisés pour apprendre des modèles qui font correspondre les données de capteurs aux actions adéquates [11].

Dans l'ensemble, la navigation de robots mobiles basée sur l'*AI* a le potentiel de permettre aux robots de s'adapter à des environnements nouveaux et changeants et d'améliorer leur performance de navigation au fil du temps. Cependant, ces techniques nécessitent de grandes quantités de données d'entraînement et de ressources informatiques, ainsi qu'un réglage minutieux des modèles pour assurer une navigation sûre et efficace.

L'apprentissage par renforcement est une sous-discipline de l'apprentissage automatique qui implique qu'un agent apprend à interagir avec un environnement pour maximiser un signal de récompense cumulatif. L'agent reçoit des retours d'information sous forme de récompenses ou de punitions en fonction de ses actions et utilise ces retours pour apprendre une politique qui associe des états à des actions. Par essais et erreurs, l'agent apprend à prendre des actions qui conduisent à des récompenses plus élevées, et avec le temps, il peut devenir compétent pour résoudre des problèmes complexes dans une large gamme de domaines.

L'apprentissage par renforcement (*RL*), a été largement utilisé dans la navigation des robots mobiles. Les algorithmes d'apprentissage par renforcement permettent aux robots d'apprendre de leurs expériences passées et d'adapter leur comportement en conséquence [12]. Par exemple, un robot peut apprendre à éviter les obstacles en recevant des récompenses pour des évitements réussis et des pénalités pour les collisions [13]. Cette approche permet au robot d'améliorer continuellement ses performances de navigation sans nécessiter de réglages manuels des algorithmes de navigation [14].

L'apprentissage par renforcement a été utilisé avec succès pour la planification de trajectoire dans la navigation de robots mobiles. Dans la planification de trajectoire basée sur

le *RL*, le robot apprend une politique qui fait correspondre son état actuel (*position et orientation*) à une action (*une commande de mouvement*) qui mène à l'objectif souhaité tout en évitant les obstacles [15]. La planification de trajectoire basée sur *RL* implique plusieurs composantes, notamment la représentation de l'espace d'état, la définition de l'espace d'action, la conception de la fonction de récompense et l'algorithme *RL* utilisé pour apprendre la politique [16].

La planification de trajectoire basée sur le *RL* s'est révélée efficace pour gérer les environnements complexes et dynamiques, car le robot peut apprendre et adapter en continu son comportement en fonction de ses expériences. Cette approche a le potentiel de permettre aux robots mobiles de naviguer dans des environnements réels de manière plus efficace que les méthodes traditionnelles basées sur des règles ou des heuristiques [17].

Le *RL* peut être utilisé pour l'évitement de collision dans la navigation de robot mobile. L'évitement de collision basé sur *RL* consiste à apprendre une politique qui associe l'état actuel du robot à une action qui évite les collisions avec les obstacles tout en se déplaçant vers la cible. L'algorithme *RL* apprend des expériences du robot en recevant des récompenses pour l'évitement réussi des collisions et des pénalités pour les collisions. La représentation de l'état comprend des informations sur la position, l'orientation et la vitesse du robot, ainsi que des informations sur les obstacles dans l'environnement.

En général, la représentation de l'espace d'état comprend des informations sur l'emplacement, la vitesse et l'orientation du robot, ainsi que des informations sur les obstacles et autres objets de l'environnement. L'espace d'action est défini en fonction des capacités physiques du robot, telles que sa vitesse maximale et son rayon de braquage [18]. La fonction de récompense est conçue pour fournir un renforcement positif pour les actions qui rapprochent le robot de l'objectif et un renforcement négatif pour les actions qui entraînent des collisions ou d'autres résultats indésirables. L'algorithme *RL* utilise ensuite ce signal de récompense pour apprendre la politique optimale pour le problème considéré.

Dans cette thèse, on s'intéresse à l'utilisation des algorithmes de *RL* dans les tâches de planification de chemin et d'évitement d'obstacles dynamiques dans un environnement partiellement connu. On s'intéresse aussi à la conception d'un contrôleur de suivi de chemin basé sur un modèle d'inférence floue simplifié [19].

1.3. Organisation de la thèse

Le reste de cette thèse est organisé comme suit :

Chapitre 2 : proposera une vue d'ensemble des approches de pointe liées à la navigation autonome basée sur apprentissage par renforcement.

Chapitre 3 : introduira les principes de l'apprentissage supervisé et de l'apprentissage par renforcement qui sont utilisés.

Chapitre 4 : montrera une méthode d'application des algorithmes de l'apprentissage par renforcement pour la navigation globale de chemins.

Chapitre 5 : présentera un contrôleur pour le suivi de chemin pour robot mobile à commande différentielle.

Chapitre 6 : focalisera sur le problème d'évitement d'obstacles dynamiques et propose une méthode hybride pour le résoudre.

Chapitre 7 : résumera les points clés de notre recherche et discute des perspectives possibles.

2. Etat de l'Art

2.1. Introduction

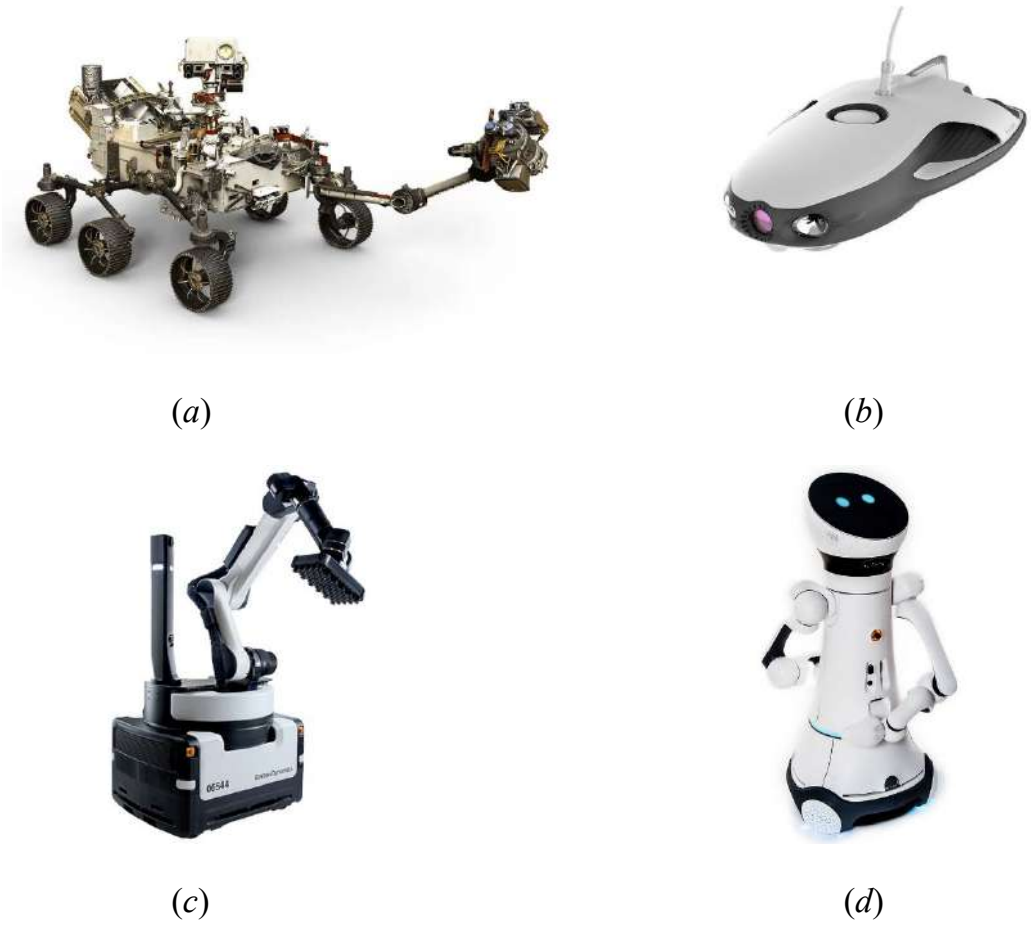
Le but de ce chapitre est de présenter les travaux de recherches existants liés à notre problématique, à savoir l'apprentissage par renforcement pour la navigation autonome de robot mobile dans un environnement d'intérieur partiellement connu. Nous nous intéressons aux trois aspects fondamentaux de la navigation autonome : la planification de chemin, l'évitement d'obstacles et le contrôle de mouvement. Nous avons divisé cet état de l'art en trois sections. La première couvre la navigation autonome, avec à la fois l'apprentissage supervisé et l'apprentissage par renforcement. La deuxième s'intéresse à la planification de trajectoire ou de chemin et aux travaux de recherches sur les méthodes d'évitement d'obstacles dans un environnement encombré. La troisième partie se concentre sur le suivi de chemin et les différentes techniques de commande de robot mobile.

2.2. La robotique mobile

Les robots se développent rapidement depuis les environnements industriels, où ils sont physiquement fixés à leurs lieux de travail, vers des machines de plus en plus complexes capables d'effectuer des tâches difficiles dans différents types d'environnements. Les robots industriels traditionnels utilisés dans les usines, où l'environnement est hautement contrôlé, sont généralement plus ou moins stationnaires. En revanche, les robots mobiles sont des systèmes robotiques capables de fonctionner dans des environnements non contraints et ayant la capacité de se déplacer librement à l'aide, par exemple, de roues. Ils peuvent fonctionner de manière autonome dans un environnement partiellement inconnu et imprévisible sans avoir besoin de dispositifs de guidage physiques ou électromécaniques (*robot mobile autonome (AMR)*). Alternativement, les robots mobiles peuvent se fier à des dispositifs de guidage qui leur permettent de suivre un itinéraire de navigation prédéfini dans un espace relativement contrôlé (*véhicule à guidage automatique (AGV)*).

Les robots mobiles ont été largement utilisés dans divers domaines, tels que l'exploration spatiale (*Figure (2.1 a)*), la surveillance sous-marine (*Figure (2.1 b)*), l'industrie (*Figure (2.1 c)*), les applications de services (*Figure (2.1 d)*), … etc.

Les progrès réalisés dans les robots mobiles autonomes ont également fourni des solutions à des tâches complexes qui étaient auparavant considérées comme réalisables uniquement par des humains. Avec les récents progrès technologiques, les robots mobiles autonomes sont déployés et utilisés dans différents domaines et scénarios. Les robots mobiles sont utilisés pour l'automatisation des entrepôts, dans les foyers pour le nettoyage, pour le service de livraison dans les restaurants, dans les environnements sociaux notamment pendant la pandémie de Covid-19, pour l'exploration extraterrestre et dans les missions de recherche et de sauvetage.



(*a*) (*b*)

(*c*) (*d*)

Figure 2.1 : *Différents robots mobiles pour différentes applications.*

Dans cette thèse, nous nous concentrons sur les robots mobiles autonomes à roues, des robots capables de prendre leurs propres décisions en fonction de la situation plutôt que de simplement exécuter une séquence prédéfinie de mouvements. En fait, puisque la plupart des robots équipés de telles capacités de prise de décision sont mobiles, nous pouvons considérer un robot mobile autonome comme un robot mobile ayant la capacité de prendre des décisions.

2.3. La navigation autonome

La navigation autonome de robots mobiles est un domaine de recherche en constante évolution en robotique qui vise à développer des méthodes et des algorithmes pour permettre aux robots de se déplacer de manière autonome dans des environnements inconnus et dynamiques. Les récents développements en matière d'apprentissage automatique ont ouvert de nouvelles perspectives pour la planification de trajectoire, la localisation, l'évitement d'obstacles, et le contrôle de robots mobiles dans des environnements dynamiques.

Un robot entièrement autonome peut recueillir des informations sur l'environnement, travailler pendant une période prolongée sans intervention humaine, se déplacer dans son environnement opérationnel sans assistance humaine, éviter les situations dangereuses pour les personnes et les biens. Un robot autonome peut également apprendre ou acquérir de nouvelles connaissances en s'adaptant à de nouvelles méthodes pour accomplir ses tâches ou en s'adaptant à des environnements incertains. Par conséquent, les robots mobiles doivent avoir les capacités d'autonomie et d'intelligence, ce qui nécessite la conception des algorithmes qui permettent aux robots de fonctionner de manière autonome dans des environnements non structurés, dynamiques, partiellement observables et incertains. Ces contraintes constituent un défi pour les chercheurs pour traiter des problèmes clés tels que l'incertitude (*à la fois dans la détection et l'action*), la fiabilité et la réponse en temps réel.

Dans chacun des domaines d'application des robots mobiles, la mobilité est presque inutile sans la capacité de navigation. Le mouvement aléatoire, qui ne nécessite pas de capacité de navigation, peut être utile pour certaines opérations de surveillance ou de nettoyage, mais pour la plupart des applications scientifiques ou industrielles de robots mobiles, la capacité de se déplacer de manière délibérée est requise. Par conséquent, la navigation autonome joue un rôle clé dans le succès des robots et constitue également la base des technologies relatives des robots mobiles autonomes.

Le sujet de la navigation des robots mobiles n'est pas nouveau et a été étudié depuis plus de quatre décennies [20]. Pour une navigation réussie, un robot mobile doit savoir où il se trouve, la destination et comment y arriver. La navigation comprend deux composantes, la recherche de chemin et la locomotion. La recherche de chemin consiste à planifier un chemin efficace pour atteindre l'objectif, tandis que la locomotion est l'exécution en temps réel des

commandes d'action tout en évitant les obstacles et les collisions le long du chemin en fonction des lectures des capteurs [21]. Bien que l'idée soit simple, la construction d'un système robuste capable de généraliser dans des situations variables dans un environnement réel est difficile.

La navigation des robots mobiles comprend approximativement les six compétences interdépendantes suivantes :

1. ***Perception :*** obtenir et interpréter les informations sensorielles ;
2. ***Exploration :*** la stratégie qui guide le robot pour choisir la prochaine direction à prendre ;
3. ***Cartographie :*** construire une représentation spatiale ou un modèle d'environnement en utilisant les informations sensorielles perçues ;
4. ***Localisation :*** la stratégie pour estimer la position du robot dans la carte spatiale qui se produit simultanément au contrôle de navigation ;
5. ***Planification de trajectoire :*** la stratégie pour trouver un chemin vers un emplacement de destination optimal ou non ;
6. ***Exécution de trajectoire :*** déterminer et adapter les actions motrices aux changements environnementaux, y compris l'évitement d'obstacles.

Un robot a besoin d'un mécanisme qui lui permet de se déplacer librement dans l'environnement, c'est-à-dire qu'il doit être capable de détecter et de réagir aux différentes situations. Ce sont les capteurs du robot qui jouent un tel rôle en tant que yeux des robots, et le robot sait où il est ou comment il arrive à un endroit, ou est capable de raisonner sur l'endroit où il est allé. Les capteurs peuvent être flexibles et mobiles pour mesurer la distance que les roues ont parcourue sur le sol, pour mesurer les changements inertiels et la structure externe de l'environnement. Les auteurs de [22] ont résumé que les capteurs peuvent être grossièrement divisés en deux classes :

- ***Les capteurs d'état interne,*** tels que les accéléromètres, les gyroscopes, qui fournissent des informations internes sur les mouvements du robot,
- ***Les capteurs d'état externe,*** tels que le *LiDAR*, les capteurs infrarouges, les sonars et les capteurs visuels, qui fournissent des informations externes sur l'environnement.

Les données des capteurs d'état interne peuvent fournir des estimations de la position du robot dans un espace *2D*. Les données des capteurs d'état externe peuvent être utilisées pour reconnaître directement un lieu ou une situation, ou être converties en informations dans une carte de l'environnement. Dans la plupart des cas, les lectures des capteurs sont imprécises et peu fiables en raison du bruit. Par conséquent, il est important pour la navigation des robots mobiles de traiter les données des capteurs avec des bruits. Comme les réseaux de neurones ont de nombreux nœuds de traitement, chacun avec des connexions principalement locales, ils peuvent fournir un certain degré de robustesse ou de tolérance aux défauts pour l'interprétation des données des capteurs.

Les techniques de navigation autonome de robot mobile peuvent être classées en deux catégories : la navigation basée sur une carte et la navigation sans carte.

2.3.1. Navigation basée sur une carte

La navigation d'un robot mobile basée sur une carte est un type de navigation autonome qui utilise des cartes préexistantes de l'environnement pour planifier un chemin. Le robot utilise ses capteurs, tels que les caméras et les *LiDAR*, pour se localiser sur la carte et planifier un chemin pour atteindre sa destination. Ce type de navigation est particulièrement utile dans des situations où une carte préexistante est disponible et fiable, comme dans les environnements intérieurs ou les environnements qui ne changent pas fréquemment.

La navigation basée sur une carte est une approche populaire en robotique en raison de sa capacité à naviguer efficacement et avec précision dans des environnements structurés. Cependant, elle nécessite des cartes précises de l'environnement et peut avoir des difficultés dans des situations où l'environnement est dynamique ou change fréquemment.

La navigation basée sur une carte est utilisée dans une variété d'applications, telles que les aspirateurs autonomes, les robots mobiles pour les applications industrielles et les véhicules autonomes.

Le système de navigation traditionnel des robots se compose de trois composants principaux : la cartographie, la localisation et la planification de trajectoire, comme le montre la Figure (2.2). La carte globale de l'environnement inconnu est construite soit par le système de cartographie, principalement en utilisant la localisation et la cartographie simultanées

(*SLAM*), soit manuellement conçue par des humains en utilisant des données de capteurs de mesure de distance et de vision [23, 24]. Le module de planification de trajectoire contient un planificateur global et un planificateur local. Le planificateur de trajectoire globale propose des points de passage au planificateur local pour une trajectoire optimale afin d'accomplir des tâches de navigation telles que l'atteinte d'une cible tout en évitant les obstacles. Le module de planification dépend d'une carte précise, d'une bonne estimation de la position, des données de capteurs et de la position cible pour déterminer la trajectoire optimale [25].

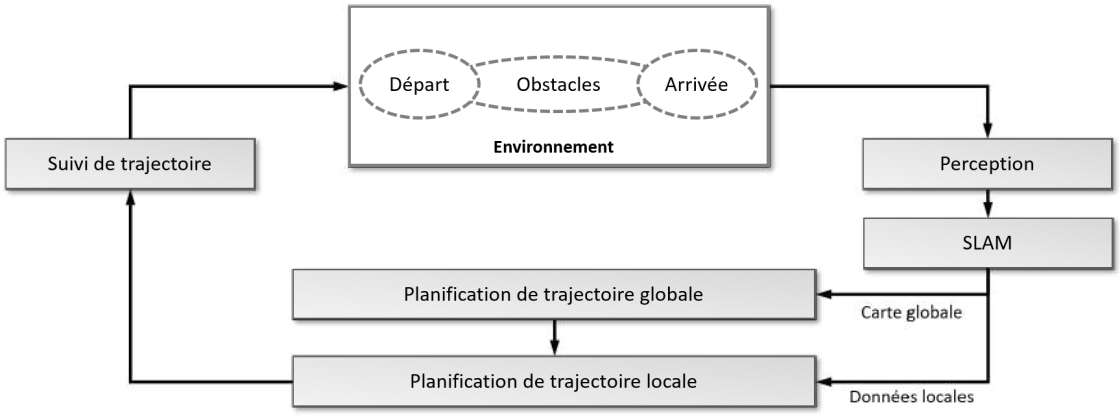


Figure 2.2 : *Architecture de système de navigation classique [23].*

Dans la navigation traditionnelle basée sur la carte, le fonctionnement du système de planification et de contrôle est traçable computationnellement à l'aide de l'estimation de la position et de la carte. De plus, l'utilisation d'un chemin optimal est assurée car la carte globale est disponible. Cependant, cette approche traditionnelle présente de nombreuses limitations et défis. La construction et la maintenance de la carte de l'environnement sont coûteuses en termes de calcul et nécessitent des capteurs laser précis et denses (*par exemple, dans le SLAM*) si elles sont effectuées dynamiquement, et sont sensibles au bruit des capteurs [26]. Sinon, c'est une tâche laborieuse et chronophage qui nécessite des ressources expertes si elle est effectuée manuellement, ce qui peut limiter son utilisation dans des environnements dynamiques ou inconnus [27]. Le cadre de navigation traditionnel comporte différents composants critiques qui méritent chacun une recherche approfondie, et l'intégration de ces composants agrège et amplifie les erreurs de calcul, ce qui conduit à de mauvaises performances [23].

2.3.2. Navigation sans carte

La navigation sans carte d'un robot mobile est un type de navigation autonome qui ne repose pas sur des cartes préexistantes de l'environnement. Au lieu de cela, le robot utilise ses capteurs, tels que les caméras et les *LiDAR*, pour percevoir l'environnement en temps réel et planifier son trajet en conséquence. Ce type de navigation est particulièrement utile dans des situations où une carte préexistante est indisponible ou peu fiable, comme dans les environnements extérieurs ou les environnements dynamiques qui changent fréquemment.

La navigation sans carte est un problème difficile en raison de la complexité des environnements réels et de la nécessité d'une perception et d'une planification en temps réel. Cependant, les progrès récents en matière d'apprentissage automatique, de vision par ordinateur et de robotique ont fait des avancées significatives dans ce domaine. La navigation sans carte est utilisée dans diverses applications, telles que les véhicules autonomes, les drones et les robots mobiles pour les missions de recherche et de sauvetage.

Ces dernières années, le développement de systèmes de contrôle de navigation sans carte pour les robots mobiles a suscité beaucoup d'attention. L'approche sans carte permet de se passer de l'information de carte globale, de sorte que la performance du système de navigation ne dépend plus de la qualité de la carte globale. De plus, cette approche modélise directement les informations des capteurs et la position relative de destination en actions robotiques en utilisant principalement des réseaux de neurones qui possèdent une forte capacité d'apprentissage avec une faible dépendance à l'exactitude des capteurs (*comme illustré dans la Figure (2.3)*).

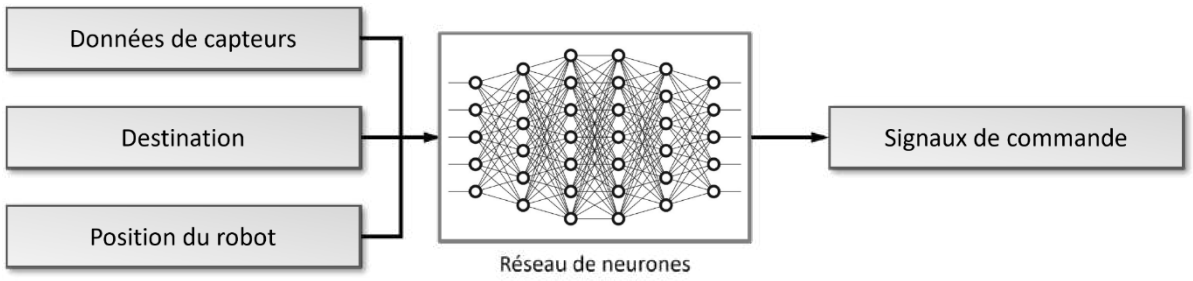


Figure 2.3 : *Architecture de système de navigation locale (sans carte).*

Les systèmes de contrôle de navigation sans carte sont principalement divisés en deux approches : les approches basées sur *LiDAR* et les approches basées sur la vision. Ces deux

approches capturent les données de l'environnement à partir de leurs capteurs respectifs et utilisent des réseaux de neurones pour apprendre la correspondance entre les images ou les données de nuage de points observées et les actions du robot pour effectuer des tâches de navigation dans un environnement inconnu sans information de carte globale. Cependant, trouver un chemin optimal est extrêmement difficile car la planification de chemin global ne peut pas être effectuée sans carte. Par conséquent, l'approche sans carte est couramment utilisée pour des tâches pour lesquelles il n'y a pas de destination explicite, telles que l'évitement d'obstacles ou lorsque la position de destination est donnée dans le référentiel de coordonnées locales du robot, comme dans la navigation locale.

En résumé, la tâche de navigation autonome consiste pour les robots mobiles à obtenir suffisamment d'informations sur l'environnement, à les traiter et à agir en se déplaçant en toute sécurité dans cet environnement, généralement selon un chemin prédéfini. La capacité de percevoir l'environnement qui les entoure est une exigence fondamentale pour tout système autonome. Toutes les décisions d'action sont prises en fonction des entrées sensorielles perçues par le robot.

2.4. L'apprentissage automatique et la navigation autonome

Les robots autonomes ne peuvent pas toujours être programmés pour exécuter des actions prédéfinies, car on ne sait pas toujours à l'avance les situations imprévues que le robot pourrait rencontrer. Aujourd'hui, cependant, la plupart des robots utilisés dans l'industrie sont préprogrammés et nécessitent un environnement bien défini et contrôlé. Reprogrammer de tels robots est souvent un processus coûteux nécessitant l'intervention d'un expert. En permettant aux robots d'apprendre des tâches soit par l'exploration autonome soit par l'enseignement d'un enseignant humain, l'installation et la reprogrammation de tâches de robot sont simplifiées. En revanche, les robots qui ne peuvent pas apprendre manquent l'un des aspects les plus intéressants de l'intelligence.

A l'ère de la technologie, les récents progrès en matière de puissance de calcul, d'intelligence artificielle et de vision par ordinateur ont permis la construction de véhicules et de robots autonomes capables d'apprendre à naviguer dans le monde physique et à effectuer des tâches complexes. Le fonctionnement autonome de ces véhicules, tels que les voitures autonomes, les robots mobiles et les drones, dépend d'un système de navigation fiable,

robuste et intelligent. La navigation est la capacité fondamentale de ces véhicules autonomes, qui leur permet de se déplacer d'un point à un autre dans l'espace bidimensionnel (*2D*) ou tridimensionnel (*3D*) sans aucune intervention humaine.

Les recherches récentes ont montré une tendance vers les approches d'intelligence artificielle pour améliorer la capacité autonome des robots basée sur les expériences accumulées, et les méthodes d'intelligence artificielle peuvent être moins coûteuses sur le plan informatique que les méthodes classiques. Les approches d'apprentissage automatique sont souvent appliquées pour alléger la charge des ingénieurs du système. L'apprentissage est donc devenu un sujet central de la recherche en robotique moderne.

L'apprentissage des robots est un domaine de recherche à l'intersection de l'apprentissage automatique et de la robotique. Il étudie les techniques permettant à un robot d'acquérir de nouvelles compétences ou de s'adapter à son environnement grâce à des algorithmes d'apprentissage. L'incarnation du robot, située dans un environnement physique, offre à la fois des difficultés spécifiques (*par exemple, une dimension élevée, des contraintes temporelles réelles pour la collecte de données et l'apprentissage*) et des opportunités pour guider le processus d'apprentissage.

L'apprentissage des robots consiste en une multitude d'approches d'apprentissage automatique, en particulier l'apprentissage par renforcement, l'apprentissage par imitation, l'apprentissage par renforcement inverse et les méthodes de régression, qui ont été suffisamment adaptées au domaine pour permettre l'apprentissage dans des systèmes robotiques complexes tels que les hélicoptères, le vol à ailes battantes, les robots bipèdes, les bras anthropomorphes et les robots humanoïdes. Alors que les approches classiques de la robotique ont souvent tenté de générer manuellement un ensemble de règles et de modèles qui permettent aux systèmes robotiques de détecter et d'agir dans le monde réel, l'apprentissage des robots repose sur l'idée qu'il est peu probable que nous puissions prévoir avec suffisamment de précision toutes les situations intéressantes du monde réel.

L'apprentissage par démonstration (*LfD : Learn from Demonstration*) ou l'apprentissage par imitation (*IL : Imitation Learning*) est une approche d'apprentissage pour les robots/agents qui prend en entrée des démonstrations d'un humain afin de construire des modèles d'actions ou de tâches. Il existe une large gamme d'approches qui relèvent de la

recherche en *LfD* [28]. Ces démonstrations sont généralement représentées sous forme de tuples *état-action*, et l'algorithme *LfD* apprend une politique de correspondance entre les états (*entrée*) et les actions (*sortie*) en se basant sur les exemples vus dans les démonstrations. L'apprentissage par renforcement inverse (*IRL*), en tant que branche importante des méthodes *LfD*, aborde le problème de l'estimation de la fonction de récompense d'un agent agissant dans un environnement dynamique.

Une autre approche consiste à fournir une correspondance entre les entrées sensorielles et les actions qui capturent statistiquement les objectifs comportementaux clés sans avoir besoin d'un modèle ou de connaissances détaillées sur le domaine [29]. De telles méthodes sont bien adaptées aux domaines où les outils disponibles pour apprendre à partir de l'expérience passée et s'adapter aux conditions émergentes sont limités.

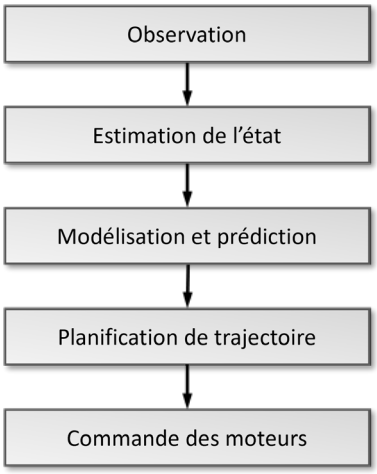


Figure 2.4 : *Pipeline traditionnel de contrôle de robot.*

Deep Reinforcement Learning (*DRL*) est un paradigme d'apprentissage automatique qui permet aux robots mobiles d'apprendre à exécuter leurs tâches et à contrôler leurs comportements complexes de manière autonome. Dans le cadre de *DRL*, des réseaux de neurones artificiels profonds sont utilisés comme approximateurs de fonctions pour l'apprentissage par renforcement [30]. L'agent mobile apprend en interagissant avec l'environnement sur une base d'essais et d'erreurs. *DRL* a montré des résultats remarquables dans la maîtrise de tâches complexes comme Go (*un jeu de plateau*) [31] et est capable d'apprendre à partir de données de haute dimension provenant de différents domaines tels que les images visuelles, les scanners laser et le langage naturel. Maintenant, il y a une tendance

croissante à utiliser *DRL* pour la navigation des robots mobiles dans des environnements inconnus.

DRL est une approche axée sur les données de bout en bout qui surpasse les approches traditionnelles de prise de décision pour le contrôle des robots, car des hypothèses préalables, des fonctionnalités conçues manuellement et des étiquettes ne sont pas nécessaires, ce qui peut introduire des biais et des erreurs [26, 27]. Les pipelines de contrôle de robots traditionnels sont constitués de modules séquentiels avec des règles conçues manuellement et chaque module est résolu individuellement (*comme le montre la Figure (2.4)*), ce qui entraîne une perte d'informations et une performance sous-optimale. De plus, la performance dépend fortement des règles, du concepteur et de la tâche.

Comparé à d'autres approches d'apprentissage automatique, telles que l'apprentissage supervisé ou l'apprentissage par imitation, l'apprentissage par renforcement est plus adapté à la tâche de navigation de robots mobiles dans des environnements inconnus car l'agent d'apprentissage a besoin d'une grande quantité de données pour apprendre une bonne politique de navigation, ce qui ne peut être obtenu que par interaction séquentielle avec l'environnement.

L'application de *DRL* dans les tâches de contrôle, bien qu'elle ait montré des résultats prometteurs, est une tâche difficile en raison de la structure de récompense complexe et des contraintes du monde réel. Un agent de robot mobile de navigation peut recevoir une récompense positive uniquement s'il parvient à atteindre sa destination sans heurter ou entrer en collision en chemin. La récompense finale dépend de nombreux facteurs tels que la distance parcourue, le temps de déplacement et les actions prises. Comme l'espace de recherche est grand, il est difficile d'entrainer des agents *DRL* et en fonction de la complexité de la tâche et de l'environnement, un agent peut prendre des jours pour être entrainé, ce qui rend son utilisation dans des scénarios réels difficile. Dans les sections suivantes, les défis liés au *DRL* sont discutés et nous cherchons des solutions à ces défis, car c'est l'objectif de cette thèse. Nous explorerons des solutions pour la navigation de robot mobile en utilisant une approche *DRL* dans un environnement contraint en ressources, qui sont non seulement efficaces à apprendre, mais sont également capables de se généraliser dans différents environnements.

2.5. Planification de trajectoire

La planification de trajectoire pour le robot mobile est le processus de détermination d'une trajectoire sans collision depuis sa position actuelle jusqu'à une destination ou un objectif souhaité. L'algorithme de planification de trajectoire prend en compte la dynamique du robot et l'environnement dans lequel il opère, tels que les obstacles, l'état du terrain et d'autres contraintes. L'objectif de la planification de trajectoire est de trouver une trajectoire optimale ou réalisable qui minimise une fonction de coût, telle que le temps, l'énergie ou la distance. La planification de trajectoire est un composant essentiel de la navigation des robots mobiles autonomes, permettant aux robots de naviguer en toute sécurité et efficacité à travers des environnements complexes pour accomplir leurs tâches.

Pour planifier une trajectoire pour un robot, il est nécessaire de fournir au planificateur des informations sur l'environnement dans lequel il doit évoluer. Ces informations peuvent inclure la présence d'obstacles, les caractéristiques de la surface ou d'autres éléments pertinents pour la planification. Les critères utilisés pour calculer la trajectoire sont déterminés par la manière dont le robot interagit avec cet environnement. Ainsi, si l'objectif est simplement de minimiser la longueur de la trajectoire et de connaître les zones franchissables ou non, cela peut suffire. Cependant, si l'objectif est de minimiser l'énergie consommée, il est important de prendre en compte des facteurs tels que la nature du sol et la manière dont le robot se déplace.

2.6. Classification des algorithmes de planification de trajectoire

Le Tableau (2.1) présente les différentes catégories de méthodes de la planification de trajectoire, en les divisant chacune en deux sous-catégories. Cette classification repose sur les principes et mécanismes fondamentaux utilisés pour construire et retourner une trajectoire. Cette classification prend en compte le fonctionnement des algorithmes de planification de trajectoire [32].

Tableau 2.1 : *Classification des algorithmes de planification de trajectoire.*

	Planification Globale		***Planification Locale***	
Approches stochastiques	*Recherche d'Espace de Configuration*		*Soft Computing*	
	Recherche de Graphe	Basés sur l'échantillonnage	Calcul Evolutionnaire	Intelligence Artificielle
Optimisation numérique	*Commande Optimale*		*Programmation Réactive*	
	Optimisation Globale	Résolution de PDE	Optimisation Locale	Manœuvre Réactive

2.6.1. Planification globale

La méthode de planification de trajectoire globale génère une trajectoire pour le robot en fonction de la carte globale et du point cible. Il est nécessaire de planifier la trajectoire que le robot peut suivre en utilisant les informations de la carte et de localisation.

Ce problème a été étudié dans un nombre important de travaux de recherche. Les algorithmes développés peuvent être divisés en trois catégories : les algorithmes basés sur la recherche de graphes [33], les algorithmes d'échantillonnage aléatoire [34] et les algorithmes bioniques intelligents [35]. Les algorithmes classiques basés sur la recherche de graphes comprennent principalement l'algorithme de *Dijkstra* [36], l'algorithme *A** [37], l'algorithme *DFS* (*Depth-First Search*) [38]et l'algorithme *BFS* (*Breadth First Search*) [39]. L'algorithme de *Dijkstra* et l'algorithme *A** ont été largement étudiés au cours des dernières décennies et ont démontré leur capacité en étant largement implémentés avec le système d'exploitation de robot (*ROS Robot Operating System*) pour les applications de robots réels [40].

Avec une stratégie de recherche heuristique, ces méthodes sont efficaces dans des environnements *2D* relativement simples. Toutefois, lorsque appliquées dans des environnements de grande taille ou de grande dimension, ces méthodes sont confrontées à une lourde charge de calcul.

En général, comme le montre la Figure (2.5), les algorithmes d'échantillonnage aléatoire comprennent les arbres d'informations par lots (*BIT*) [41], les arbres d'informations par lots accélérés régionalement (*RABIT*) [42], les arbres aléatoires d'exploration rapide (*RRT*) [43],

et l'arbre aléatoire à double risque basé sur l'exploration rapide (*Risk-DTRRT*) [44], etc. Comparés aux algorithmes de recherche de graphes, ces algorithmes sont plus efficaces et largement utilisés dans des environnements dynamiques ou à haute dimensionnalité.

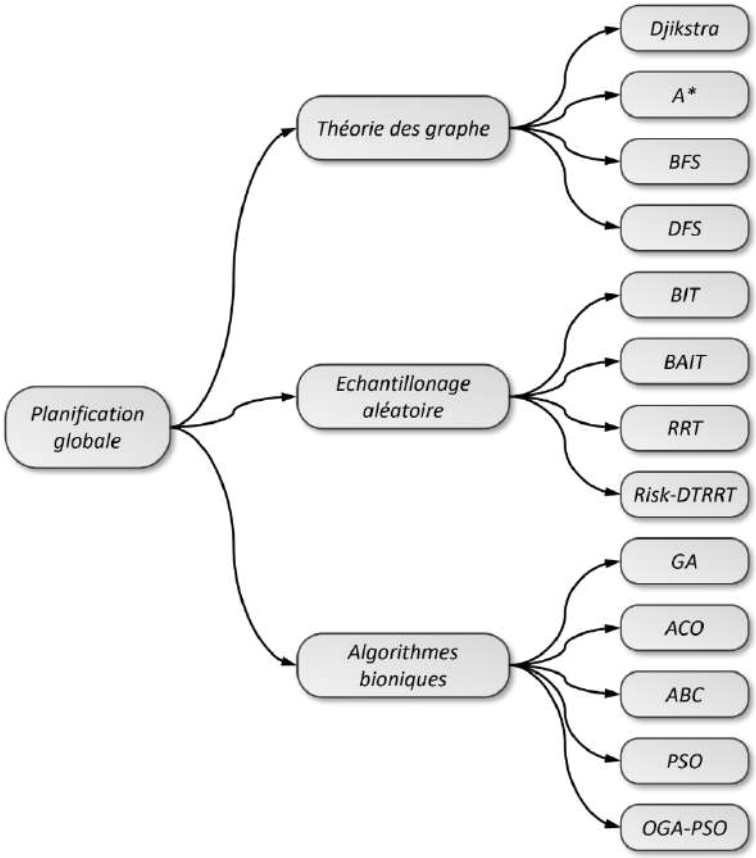


Figure 2.5 : *Les méthodes classiques de planification de chemin [45].*

Une autre branche importante des méthodes de planification globale de chemins est la méthode basée sur l'inspiration bionique intelligente, qui est un type d'algorithme intelligent qui simule les comportements évolutifs des insectes. Elle comprend généralement l'algorithme génétique (*GA*) [46], l'algorithme de colonie de fourmis (*ACO*) [47], l'algorithme de colonie d'abeilles artificielles (*ABC*) [48] et l'algorithme d'optimisation de l'essaim de particules (*PSO*) [49].

2.6.2. Planification locale et évitement d'obstacles

Le planificateur de trajectoire locale se concentre sur la génération d'une trajectoire locale en utilisant les informations disponibles sur l'environnement environnant du robot, de sorte que le robot puisse éviter les obstacles locaux de manière efficace. Les planificateurs de trajectoire locale sont largement utilisés car les informations capturées par le système de

perception changent en temps réel dans des environnements dynamiques. Comparé à la méthode de planification de trajectoire globale, la méthode de planification de trajectoire locale est plus efficace et pratique, servant de pont entre la trajectoire globale et le contrôle. Cependant, un inconvénient notable est que le planificateur local peut être piégé dans un minimum local.

Il existe de nombreux algorithmes classiques de planification locale pour obtenir un chemin optimal et éviter le problème des minimas locaux, tels que la méthode du champ de potentiel artificiel [50], la logique floue [51], l'algorithme de recuit simulé (*Simulated annealing*) [52], l'algorithme *PSO* chaotique [53], et une méthode hybride combinée avec l'algorithme génétique [54]. Cependant, ces méthodes ne tiennent pas compte du mouvement relatif entre l'agent et les objets dynamiques, et pire encore, il est parfois difficile d'acquérir explicitement les profils de vitesse des objets dynamiques.

Récemment, un planificateur local qui ne dépend pas des informations explicites de profil de vitesse a été développé. Dans ce système, la navigation de chaque agent dans l'environnement est indépendante, et un agent n'a pas besoin de communiquer avec d'autres agents [55]. En particulier, P. Fiorini [56] a proposé un algorithme qui utilise la notion de *VO* (*velocity obstacles*) pour éviter les collisions avec des objets en mouvement. Les *VOs* sont définis comme des ensembles de vitesses relatives qui entraînent une collision si les trajectoires actuelles du robot et de l'objet en mouvement ne sont pas modifiées. L'algorithme utilise ces *VOs* pour générer une zone sûre autour du robot, qui est définie comme l'ensemble de toutes les positions possibles du robot qui ne sont pas en collision avec les objets en mouvement. Les trajectoires possibles du robot sont ensuite générées en utilisant cette zone sûre et en recherchant les chemins optimaux pour atteindre la destination souhaitée. L'algorithme est capable de planifier des trajectoires dans des environnements dynamiques avec des objets en mouvement, tout en évitant les collisions et en trouvant des chemins optimaux pour atteindre la destination. Pour réduire le décalage de la vitesse actuelle et améliorer les performances, Van den Berg et *al.* [55] ont proposé la méthode d'obstacle de vitesse réciproque (*RVO*). Ils considéraient le nouveau profil de vitesse du robot comme la valeur moyenne de sa vitesse actuelle et de la vitesse qui se trouve en dehors de la *VO* des autres agents. Le *RVO* est suggéré comme une méthode utile pour planifier une trajectoire

fluide et sûre sans oscillations, dans la navigation à plusieurs agents. Cependant, il a toujours un inconvénient que plusieurs robots ne pourraient pas arriver à un consensus sur le côté à emprunter, ce qui cause un problème appelé *danse réciproque*. Pour résoudre ce problème, Snape [57] a étendu *RVO* à l'obstacle de vitesse réciproque hybride (*HRVO*). Cette approche a été appliquée pour la navigation multirobots en tenant compte de la cinématique et de l'incertitude des capteurs du robot.

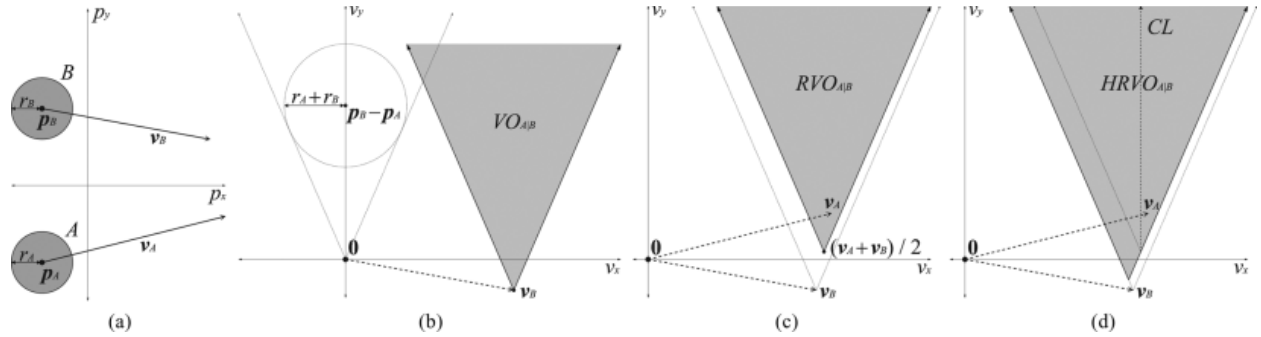


Figure 2.6 : *Les méthodes VO (a : espace de travail, b : $VO_{A|b}$, c : $RVO_{A|b}$, d : $HRVO_{A|b}$) [58].*

Cependant, si de nombreux objets dynamiques sont présents dans le scénario d'application, la vitesse du robot se rapprochera du point initial dans l'espace de vitesse [59]. Par conséquent, le robot peut rester coincé dans une zone. Ce problème peut être résolu en utilisant l'approche de troncature [58], avec laquelle les robots ne se heurteront pas à un moment donné après la troncature. Un exemple de *VO* est présenté dans la Figure (2.6), où les zones grises marquent le profil de vitesse qui peut provoquer une collision entre les agents. Plus de détails peuvent être trouvés dans [60]. Les nouvelles vitesses des agents doivent être choisies en dehors de ces zones grises. Pour ce faire, plusieurs méthodes dans différents aspects sont proposées. Trois méthodes couramment utilisées qui ont été proposées ces dernières années sont décrites ici. La première méthode est celle de *Optimal Reciprocal Collision Avoidance* (*ORCA*) introduite par Berg et *al.* [61]. Avec cette méthode, des demi-plans de vitesses sans collision peuvent être calculés et assignés à chaque agent. La région de vitesse optimale peut ensuite être définie en résolvant un problème de programmation linéaire. Les agents sélectionnent ensuite un profil de vitesse qui est le plus proche de la vitesse optimale et se déplacent avec elle ensuite. La deuxième méthode qui estime la vitesse sans collision est *ClearPath* présentée dans [62]. *ClearPath* est une méthode robuste et est meilleure que les méthodes basées sur *VO* pour l'évitement de collision. Il existe deux façons pour *ClearPath* de calculer la vitesse libre de collision. L'une consiste à choisir une vitesse à

l'intersection de deux lignes de frontière de *VOs* arbitraires. Une autre méthode choisit la vitesse déterminée par la projection du profil de vitesse préféré sur le *VO* le plus proche [61]. La troisième méthode est la méthode *Collision Avoidance with Localization Uncertainty* (*CALU*) introduite dans [63], qui combine les deux méthodes *Optimal Reciprocal Collision Avoidance* (*ORCA*) et *Non-holonomic robots Optimal Reciprocal Collision Avoidance* (*NH-ORCA*) [60] pour atténuer le besoin de connaissances préalables sur l'environnement.

Les *VOs* peuvent efficacement faire face aux problèmes causés par la localisation et la communication imprécises dans les environnements complexes. Cependant, cela nécessite des informations adéquates sur la forme, la vitesse et la position de l'agent. Toutefois, cela est inutile dans deux conditions :

- Le châssis du robot ne peut pas être traité comme un disque ;
- La distribution de croyance de la pose de l'*AMCL* dérive dans une direction.

Claes [59] a introduit l'évitement de collision sous l'incertitude de localisation bornée (*COCALU*) pour résoudre ce problème. Il propose une approche basée sur les techniques de contrôle stochastique, où l'incertitude dans la localisation est modélisée par une distribution de probabilité. Les obstacles sont modélisés par des polygones convexes, et le robot est représenté par un cercle avec un rayon de sécurité. L'algorithme de planification de mouvement est basé sur la méthode des champs de potentiel. Les obstacles sont considérés comme des sources de potentiel négatif, tandis que la destination est considérée comme une source de potentiel positive. Le robot suit le gradient de potentiel pour atteindre la destination tout en évitant les obstacles. Cependant, en présence d'incertitudes dans la localisation, le potentiel des obstacles peut ne pas être connu avec précision. Pour prendre en compte ces incertitudes, l'article propose d'ajouter une zone de sécurité autour de chaque obstacle, qui est également modélisée comme une distribution de probabilité. Le robot suit alors le gradient de potentiel en évitant non seulement les obstacles, mais également les zones de sécurité autour des obstacles.

2.7. L'apprentissage par renforcement dans la planification de trajectoire

Dans la robotique mobile, le *RL* utilise la rétroaction environnementale en tant qu'entrée pour la planification de chemin. Il produit une action pour le robot en interagissant continuellement avec l'environnement extérieur. En conséquence, une solution rationnelle

peut être générée pour le robot lorsqu'il rencontre des personnes et d'autres robots dans les environnements. De plus, comme dans tout type d'apprentissage, les performances s'améliorent avec l'expérience.

Les algorithmes de *RL* classiques dans la planification de chemin de robot incluent l'algorithme *Q-learning* [64], l'algorithme *SARSA* [65], l'algorithme *R-learning*. *Q-learning* est parmi les algorithmes de *RL* les plus étudiés. Habituellement, l'algorithme renvoie une valeur de récompense pour l'état et le mouvement du robot, grâce à la rétroaction que le robot obtient de l'environnement. En particulier, la valeur *Q* de l'action correcte augmente en diminuant le taux d'actions incorrectes. Ensuite, la méthode basée sur la valeur *Q* renvoie une politique optimale après que la valeur *Q* soit filtrée. L'algorithme *Q-learning* présente également certaines limitations. Tout d'abord, la demande de mémoire est grande. Deuxièmement, cela prendra beaucoup de temps pour l'apprentissage. Troisièmement, le taux de convergence est faible. Pour aborder les problèmes de *Q-learning*, Peng [66] a proposé l'algorithme *Q(λ)*, qui a utilisé l'idée de retour arrière. Dans cette approche, les données ultérieures peuvent être fournies dans le temps afin que la méthode puisse prédire le prochain comportement de manière efficace dans le temps. Le comportement erroné en cours d'acquisition est progressivement oublié dans le processus de mise à jour.

RL et ses variantes ont été largement utilisés pour la navigation des robots. Pour interagir de manière gracieuse avec les humains, les robots ont besoin de comprendre et de suivre certaines règles. Dans cet objectif, Kuderer [67] a proposé une approche pour modéliser le comportement de navigation coopératif des humains. Elle est capable d'obtenir des trajectoires humaines en temps réel. Récemment, l'apprentissage par renforcement inverse (*IRL*) a suscité beaucoup d'intérêt en recherche. Il contient une fonction de récompense pour le processus de prise de décision. Certains chercheurs ont appliqué l'*IRL* pour obtenir le modèle de confort humain pour la navigation collaborative [68]. Pour réaliser une planification de trajectoire gracieuse dans des environnements densément peuplés, Chen et *al.* [69] ont proposé un algorithme de collision multi-agent décentralisé basé sur le *DRL*, qui transfère efficacement le calcul en ligne à l'apprentissage hors ligne. Il est meilleur que l'algorithme *ORCA* bien que la méthode puisse conduire à une trajectoire oscillante (*marginalement stable*).

SA-CALDRL [70] a été proposé pour faire face à l'aléa du comportement humain, dans lequel une politique de navigation efficace en temps est utilisée. Notamment, cette méthode peut réaliser une navigation autonome à faible vitesse d'un véhicule autnome dans un environnement dynamique densément peuplé. Cependant, l'algorithme ne prend pas en compte la relation avec les piétons. Everett [71] a étendu l'algorithme *SA-CALDRL* en introduisant la méthode *Long Short-Term Memory* (*LSTM*) dans l'algorithme. L'avantage de l'algorithme est qu'il n'a pas besoin de supposer un modèle de comportement spécifique. De plus, il cherche à prédire la direction de course du robot de manière simple.

De plus, Ciou et *al.* [72] ont proposé un cadre d'apprentissage par renforcement composé (*CRL*). Dans ce cadre, le robot apprend une navigation sociale gracieuse à travers une entrée de capteur. Les expériences montrent que la méthode *CRL* peut apprendre à naviguer en toute sécurité dans l'environnement. Cependant, une connaissance préalable est nécessaire dans ce cadre, limitant son application généralisée dans le monde réel. Divers environnements et divers types de cadres d'entraînement sur scène ont été proposés par Long et *al*. [73]. Cette stratégie peut être bien étendue à de nouveaux scénarios qui n'apparaissent pas dans la phase d'entraînement.

2.8. Contrôle de mouvement

Le nombre d'applications possibles des robots mobiles à roues est énorme et est même en augmentation. Ainsi, des algorithmes de contrôle robustes pour de tels objets robotiques sont nécessaires. Il existe de nombreux articles relatifs aux algorithmes de contrôle pour les robots mobiles à roues publiés dans la littérature. Trois types de tâches de base réalisées par les plates-formes mobiles peuvent être distinguées :

- La stabilisation de point en point,
- Le suivi de trajectoire (*le robot doit suivre une courbe désirée qui est paramétrée par le temps*),
- Le suivi de chemin (*la tâche du robot est de suivre une courbe paramétrée par une distance curviligne à partir d'un point fixe*).

Le contrôle des robots mobiles à roues concerne la conception d'une loi de commande pour annuler le vecteur d'état d'erreur pour la stabilisation de posture ou le suivi de trajectoire. En se basant sur Brockett [74], plusieurs contrôleurs ont été proposés pour la stabilisation de

point [75-82]. Ces contrôleurs sont classés en tant que méthodes de contrôle non linéaires [75-77], contrôleurs discontinus [78-80] et rétroactions d'état variant dans le temps [81, 82].

Des contrôleurs non linéaires de stabilisation de point en point pour les robots mobiles de type monocycle sont abordés dans [75-77] en utilisant la méthode de stabilité de *Lyapunov*. Le patinage et la flottaison des roues sont analysés à l'aide du modèle cinématique dans [75]. Les dynamiques du système sont prises en compte dans les simulations et comparées aux expériences, où une stabilité asymptotique est obtenue avec des chemins non lisses. De plus, un modèle cinématique bijectif est utilisé dans [76] en introduisant un modèle à quatre états modifié. De plus, la transformation de l'état polaire est utilisée dans [77] pour la stabilisation de point et le suivi de trajectoire. Des trajectoires circulaires et linéaires sont utilisées pour montrer l'efficacité du contrôleur proposé, tandis que le contrôleur de suivi de trajectoire est singulier autour de l'origine.

Les contrôleurs discontinus sont abordés dans [78-80] pour asservir le robot mobile de type monocycle. La technique de *backstepping* est utilisée dans [78] et la stabilité est prouvée par la seconde méthode de *Lyapunov*, par la suite, des termes adaptatifs sont introduits pour surmonter la perturbation d'entrée. L'efficacité de cette méthode est illustrée par simulation. De plus, une fonction de *Lyapunov* étendue est présentée dans [79] en se basant sur le modèle cinématique. Le contrôleur proposé peut diriger le système vers le point désiré, cependant, il souffre de mouvements avant-arrière. Une représentation des états polaires est utilisée dans [80] pour la conception du contrôleur afin d'atteindre la régulation de point avec un chemin régulier et sans inversion de direction.

Les contrôleurs de direction sont traités dans [83, 84] en utilisant une vitesse longitudinale constante pour réaliser un suivi de trajectoire. Le contrôle de l'orientation du robot mobile est traité dans [14], en utilisant une approximation d'angle faible pour simplifier la non-linéarité du modèle, puis une constante de gain proportionnelle est utilisée pour réguler l'erreur d'angle désirée. De plus, les limites de l'actionneur sont prises en compte dans [84] par une fonction de saturation standard et un algorithme de régulation de la vitesse longitudinale dans les virages, tandis qu'un suivi approprié est montré par simulation.

Le contrôle en mode glissant (*SMC*) et les algorithmes *super-twisting* (*STA*) sont utilisés pour le suivi de trajectoire des robots mobiles, comme dans les articles [85-88]. Un *SMC*

robuste est abordé dans [85], et la stabilité est prouvée par la méthode directe de Lyapunov. L'efficacité de ce contrôleur est illustrée par des simulations de trajectoires circulaires. Un contrôleur de rétroaction d'état basé sur *STA* est proposé dans [86] pour SSMR. Des simulations et des expériences sont comparées avec une rétroaction d'état et un SMC de premier ordre. Le contrôle en mode glissant *super-twisting* est abordé dans [87] pour robot mobile, ce qui impose un mode glissant du second ordre suivi d'un contrôle PD pour éliminer les cliquetis et compenser la dynamique. Des expériences démontrent l'efficacité de ce contrôleur. En outre, un *STA* adaptatif pour robot mobile à quatre roues est proposé dans [88], les gains sont donc ajustés régulièrement pour maintenir l'effort de commande aussi minimal que possible. Cette méthode est validée expérimentalement en suivant une trajectoire circulaire.

Les techniques de contrôle adaptatif sont présentées dans [89-92] pour le suivi de trajectoire prédéfinie du robot mobile de type monocycle. Une loi de commande adaptative est conçue dans [89] avec deux observateurs pour estimer les entrées du système, suivi d'une simulation de suivi de cercle. Des contrôleurs basés sur la fonction de *Lyapunov* sont introduits dans [90-92] avec une planification de gain, où le glissement longitudinal est pris en compte dans les simulations, tandis que les perturbations externes et l'incertitude des paramètres sont pris en compte dans [92].

Des méthodes intelligentes telles que les réseaux de neurones (*NN*), les techniques d'optimisation heuristique et le contrôle prédictif basé sur un modèle (*MPC*) sont proposées dans [93-99] pour le contrôle de suivi de trajectoire des robots mobiles afin de surmonter les incertitudes des paramètres, les perturbations externes et les limitations de l'actionneur. L'*AMPC* est présenté dans [93] et étendu à l'évitement d'obstacles. Trois techniques d'optimisation heuristique sont évaluées et *PSO* est sélectionné pour une mise en œuvre réelle. L'*AMPC* avec apprentissage en ligne du modèle est introduit dans [94] pour générer une politique optimale basée sur le modèle *SSMR* appris, où les pneus et les conditions routières sont pris en compte avec une validation expérimentale. L'*AMPC* est proposé dans [95], où la fonction de coût quadratique inclut l'erreur de suivi et l'effort de contrôle qui est minimisé par le contrôleur proposé. Les capacités de suivi sont démontrées par simulation et validation de robot réel. Une fonction de composé de sinus *NN* et une fonction de composé de cosinus

améliorée *NN* sont introduites dans [96, 97] avec une structure simple et un algorithme d'apprentissage continu. Ces méthodes sont simulées pour des trajectoires circulaires et sinusoïdales. Un algorithme de suivi neuronal adaptatif basé sur l'apprentissage par renforcement est proposé dans [98], où le patinage et le glissement sont abordés et testés par des simulations. Un contrôleur *PID flou-neuronal* de type *2* adaptatif de *Takagi-Sugeno* est introduit dans [99] avec un algorithme d'apprentissage en ligne pour mettre à jour les paramètres de contrôle, compenser ainsi les perturbations. Ce contrôleur est testé dans le suivi de ligne.

L'autre approche est le suivi de trajectoire point à point [100-105], où la trajectoire désirée est représentée par un ensemble de points de passage ou d'arcs et de lignes. Un contrôle robuste à logique floue en mode glissant est abordé pour robot mobile dans [100], où une trajectoire de référence continue est générée à l'aide d'une courbe quadratique, en outre le contrôleur flou proposée est validé expérimentalement. Un ensemble de secteurs est défini dans [101] en utilisant les points de passage pour calculer l'orientation désirée du robot mobile. Le contrôle entre les points est réalisé par un contrôleur *PID*, puis une validation physique est menée sur des trajectoires en ligne droite, en marche et à trois points de passage. L'ajustement des arcs est proposé dans [102], où des arcs circulaires sont générés pour calculer le rayon de courbure désiré en fonction d'un modèle simplifié du robot mobile. La commande floue est utilisée pour suivre des trajectoires linéaires par morceaux dans [103], où la trajectoire de référence lisse est approximée par des lignes par morceaux, ensuite des simulations démontrent la faisabilité de cette commande. Un contrôleur PD linéaire piloté par l'erreur de suivi est présenté dans [104], et [105] a proposé une loi de contrôle à rétroaction lisse, où la trajectoire de référence est identifiée comme un ensemble de lignes ou d'arcs circulaires.

2.9. Conclusion

Dans ce chapitre, les principaux algorithmes de planification de trajectoire pour la navigation autonome des robots ont été passés en revue. Le problème de planification de trajectoire dans le cadre qui divise les méthodes de planification en planificateurs de trajectoire globaux et locaux a été examiné. Ces méthodes sont efficaces pour résoudre le problème de planification de trajectoire globale et locale en assurant l'évitement d'obstacles.

Nous avons également présenté les méthodes de contrôle cinématique pour le suivi de trajectoire pour robot mobile à roues.

3. Apprentissage par Renforcement

3.1. Introduction

Bien qu'ils existent depuis les années 1950, les réseaux de neurones ont gagné en popularité depuis 2012 avec l'avènement de l'apprentissage profond. Avec le développement de machines informatiques puissantes, *GPU* (*unité de traitement graphique*), qui permettent de paralléliser de nombreux calculs, couplées à la création de nombreux ensembles de données annotées (comme *ImageNet* [106]), l'apprentissage profond a permis de développer rapidement les méthodes d'apprentissage automatique.

L'apprentissage profond a une large gamme d'applications. De la compréhension du langage naturel à la vision par ordinateur, l'apprentissage profond est maintenant omniprésent. Le champ d'application de cette thèse se limite à la prise de décision pour la navigation autonome de robot mobile. Notre travail ici se concentre sur des algorithmes qui apprennent à un robot mobile à roues de naviguer dans un environnement d'intérieur.

Après une brève introduction sur les bases des réseaux de neurones et de l'apprentissage profond, l'objectif de ce chapitre est d'introduire les principes généraux de l'apprentissage profond et de l'apprentissage par renforcement. Pour des explications plus approfondies, nous encourageons le lecteur à se référer aux livres suivants : « *Deep Learning* » [107], « *Neural Networks and Deep Learning* » [108], « *Reinforcement Learning: An Introduction* » [30].

3.2. Réseaux de neurones et apprentissage profond

3.2.1. Le neurone artificiel

La première définition de neurone artificiel remonte à 1943 dans « *A logical calculus of the ideas immanent in nervous activity* » [109]. Inspiré des neurones biologiques, un neurone est un objet mathématique qui suit des règles spécifiques. En considérant une entrée $x = [x_0, \cdots, x_n]$, le neurone produit une sortie $y = \sigma\left(\sum w_i x_i + b_i\right)$, où w_i et b_i sont des paramètres internes du neurone, respectivement le poids et le biais. σ est une fonction d'activation qui peut ajouter de la non-linéarité. Les fonctions d'activation classiques sont :

- la sigmoïde $\sigma(x) = \frac{1}{1+e^{-x}}$,
- la tangente hyperbolique (*tanh*) $\sigma(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$,
- l'unité linéaire redressée (*Relu : Rectified linear unit*) $\sigma(x) = \max(0, x)$.

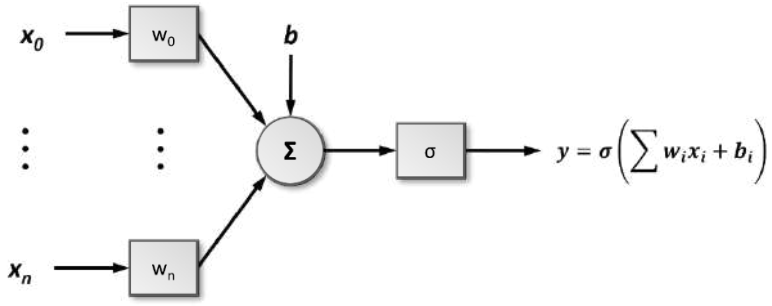


Figure 3.1 : *Neurone Artificiel.*

Initialement, le premier neurone artificiel était un perceptron. C'est un classificateur binaire dont la fonction d'activation est $sign()$, et il n'y a pas de biais (*voir l'Equation (1)*).

$$output = \begin{cases} 1 & si \sum w_i x_i > 0 \\ 0 & si\ non \end{cases} \quad (1)$$

3.2.2. Réseau de neurones artificiels

En raison de sa construction, le perceptron est seulement capable de classer des données qui peuvent être séparées par une seule ligne. Il n'est donc pas capable de modéliser des fonctions plus complexes telles que la fonction *XOR*. Pour modéliser cette dernière, plusieurs perceptrons doivent être utilisés. De cela est né le Perceptron Multicouches (*MLP : Multi Layer Perceptron*). Bien souvent confondu, un *MLP* est composé de plusieurs perceptrons - donc avec seulement des fonctions d'activation binaires $sign()$, tandis qu'un réseau de neurones artificiels (*ANN*) contient des neurones qui peuvent avoir toutes sortes de fonctions d'activation.

La Figure (3.2) montre un exemple de réseau de neurones artificiels à plusieurs couches. Chaque cercle représente un neurone. Ils sont généralement organisés en couches, où les sorties d'une couche L_i sont les entrées des neurones de la couche suivante L_{i+1}. Les couches internes sont appelées couches cachées. On parle d'apprentissage profond (*deep learning*) lorsqu'il y a plus de 2 couches cachées.

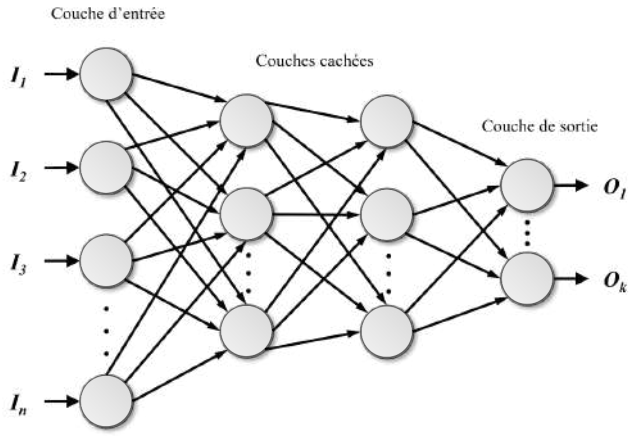


Figure 3.2 : *Réseau de Neurones Artificiels multicouches.*

3.2.3. Réseau de Neurones Convolutif

Lorsque les couches du réseau sont organisées comme précédemment, on parle de couches entièrement connectées, car chaque neurone est connecté à tous les neurones de la couche précédente. Cependant, cette architecture ne convient pas bien au traitement d'images. En effet, la taille des images nécessiterait un grand nombre de paramètres. Pour des images *RGB* de taille 224×224 - taille des images du jeu de données *ImageNet*, il faudrait déjà 150528 ($224 \times 224 \times 3$) poids pour chaque neurone de la première couche.

Les réseaux de neurones convolutifs (*CNN Convolutional Neural Networks*) ont émergé pour permettre le traitement d'images avec moins de paramètres que les couches entièrement connectées. Ils ont été introduits en 1998 par LeCun et *al.* dans « *Gradient-based learning applied to document recognition* » [110] pour permettre la reconnaissance d'écriture manuscrite (*voir Figure (3.3)*). Les *CNN* sont composés de couches de convolution suivies généralement de couches de *pooling*. Les premières traitent l'image par zone (*et non par pixel*) pour détecter des motifs. Les couches de *pooling* sont utilisées pour sous-échantillonner l'image. En plus de réduire le nombre de paramètres, un autre avantage des *CNN* est l'invariance de translation.

Les couches de convolution et de *pooling* sont généralement suivies par des couches entièrement connectées à la fin du *CNN*. Parmi les architectures classiques utilisées en vision par ordinateur, on peut citer *VGG* [111], *Resnet* [112] ou *Inception* [113]. La recherche de nouvelles architectures, plus légères et/ou plus performantes, est également très active.

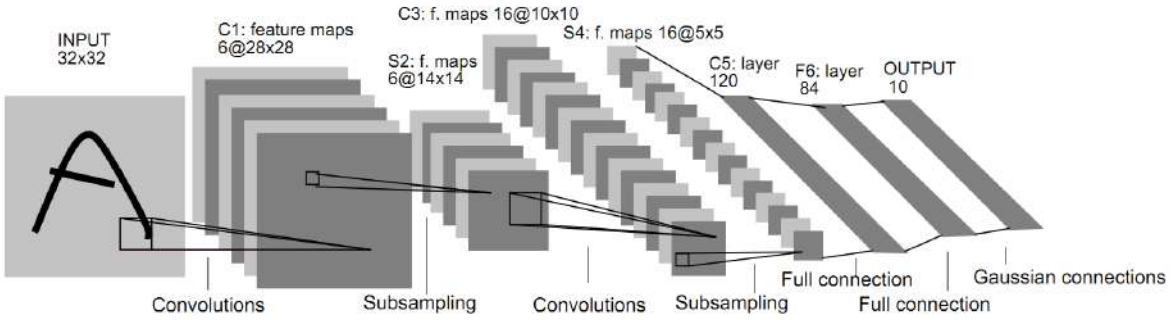


Figure 3.3 : *Réseau convolutif pour la reconnaissance d'écriture manuscrite [110].*

3.2.4. Apprentissage et rétropropagation

L'entraînement d'un réseau de neurones consiste à trouver les paramètres optimaux θ (*poids et biais du réseau de neurones*) pour une tâche donnée. A cette fin, une fonction de coût L est définie - dont la forme dépend du problème et du type d'apprentissage. Des exemples de fonctions de coût dans le cas de l'apprentissage supervisé ou par renforcement sont donnés dans les sections suivantes. Les paramètres du réseau seront modifiés en fonction de la dérivée de cette fonction de coût. Pour chaque θ_i dans l'ensemble de paramètres θ,

$$\Delta\theta_i = \frac{\partial L}{\partial \theta_i} \tag{2}$$

Le paramètre θ_i est mis à jour selon la règle de la descente de gradient (*on parle également de montée de gradient lorsque le coût doit être maximisée*) :

$$\theta_i \leftarrow \theta_i - \alpha\Delta\theta_i \tag{3}$$

Avec α le taux d'apprentissage. Les algorithmes d'optimisation les plus couramment utilisés sont *SGD* (*Stochastic Gradient Descent with moment* [114]) et *Adam* [115].

En apprentissage automatique, nous pouvons distinguer trois types de méthodes d'apprentissage : l'apprentissage supervisé, l'apprentissage par renforcement et l'apprentissage non supervisé. L'apprentissage supervisé apprend à partir de données annotées, l'apprentissage par renforcement utilise une fonction de récompense et l'apprentissage non supervisé traite des données non étiquetées. Dans cette thèse, nous nous intéressons uniquement à l'apprentissage par renforcement, qui est présenté dans les sections suivantes.

3.2.5. Apprentissage supervisé

Le type d'apprentissage le plus couramment utilisé est l'apprentissage supervisé, dans lequel le modèle apprend à partir de données annotées. La base de données est un ensemble (x_n, y_n) avec $n \in \mathbb{N}$, où x_i est l'entrée de notre modèle (*par exemple une image*) et y_i est la sortie souhaitée (*par exemple le type d'objet dans l'image d'entrée*). Le modèle peut être défini par une fonction f avec θ paramètres. Le but est d'apprendre les paramètres θ de manière à ce que $f(x_i, \theta) = y_i$.

En apprentissage supervisé, la fonction de coût classique est l'erreur quadratique moyenne (*MSE : Mean Squared Error*) (*Equation (4)*) :

$$L = MSE\left(f(x,\theta), y\right) = \frac{1}{N}\sum_{i=1}^{N}\left(f(x_i,\theta) - y_i\right)^2 \tag{4}$$

Pour les problèmes de classification, l'entropie croisée (*CE : Cross Entropy*), également appelée perte logarithmique, est souvent utilisée (*Equation (5)*). Dans ce cas, la sortie $f(x,\theta) = p$, où p est la probabilité d'appartenir à la classe considérée, et la vérité du terrain $y \in 0,1$. Pour la classification multi-classes, la fonction du coût est la somme sur toutes les classes i (*Equation (6)*).

$$L = CE(p, y) = -y\log(p) + (1-y)\log(1-p) \tag{5}$$

$$L = -\sum_{i} y_i \log(p_i) \tag{6}$$

L'apprentissage supervisé présente de nombreux avantages. Il est relativement rapide et stable, et offre généralement une bonne convergence. Cependant, il nécessite des données annotées, qui sont coûteuses. Les ensembles de données sont toujours limités et sont biaisés par l'homme. Lorsqu'il s'agit d'effectuer une succession d'actions (*comme dans le cas de la navigation autonome de robot mobile*), l'apprentissage supervisé peut également souffrir d'un décalage distributionnel, ce qui signifie que la distribution de l'ensemble de données n'est pas identique à celle rencontrée lors des tests. Dans ce cas, lorsque le système commet une erreur, il accumulera des erreurs et ne saura jamais comment récupérer de son erreur.

3.2.6. Apprentissage par renforcement

L'apprentissage par renforcement offre une alternative à l'apprentissage supervisé. Aussi appelé apprentissage par essais et erreurs, il n'utilise pas de données annotées, mais seulement une fonction de récompense qui indique un bon ou un mauvais comportement. Le biais externe est donc réduit, mais l'apprentissage peut prendre plus de temps, car aucune démonstration de bon comportement n'est donnée. En revanche, dans l'apprentissage par renforcement, l'apprenant ne reçoit qu'un retour d'information partiel sur les décisions qu'il prend. Par conséquent, dans le cadre de l'apprentissage par renforcement, l'apprenant est un agent de prise de décision qui entreprend des actions dans un environnement et reçoit une récompense (*ou une pénalité*) pour ses actions en essayant de résoudre un problème. Après une série d'essais et d'erreurs, il devrait apprendre la meilleure *politique*, qui est la séquence d'actions qui maximise la récompense totale [30]. Dans ce chapitre, nous présentons les bases structurelles des *MDP* et de l'apprentissage par renforcement.

3.3. Définitions et notions

3.3.1. Définitions

L'apprentissage par renforcement est généralement opéré dans un cadre d'interaction, illustré dans la Figure (3.4) : l'agent d'apprentissage interagit avec un environnement initialement inconnu et reçoit une représentation de l'état et une récompense immédiate en retour. L'environnement produit un état s_t à chaque étape t, et en recevant l'état actuel s_t l'agent réagit avec une action a_t qu'il calcule et exécute par la suite. L'agent agit selon une politique $\pi(a_t|s_t)$, qui représente la probabilité de prendre une action a_t lorsqu'il se trouve dans l'état s_t (*dans un environnement déterministe,* $\pi(s_t)=a_t$). Cette action entraîne une transition de l'environnement vers un nouvel état. L'environnement fournit le nouvel état s_{t+1} ainsi qu'une récompense r_t, qui indique à quel point le nouvel état est bon. L'agent reçoit la nouvelle représentation et la récompense correspondante, et tout le processus se répète.

Le but de l'agent est de maximiser la récompense cumulative : $\max\sum r_t$ (*les récompenses sont souvent pondérées pour éviter les sommes explosées*). La somme de récompense cumulée est appelée *retour* R_t.

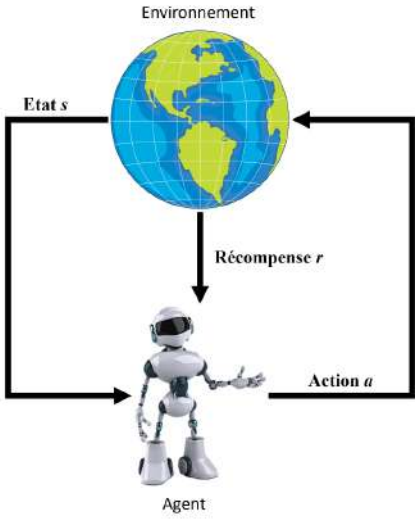


Figure 3.4 : *Interaction, agent-environnement.*

L'environnement en apprentissage par renforcement est généralement formulé comme un *Processus de Décision Markovien* (*MDP*), et l'objectif est d'apprendre une stratégie de contrôle afin de maximiser la récompense totale qui représente un objectif à long terme.

3.3.2. Les Processus de Décision Markovien (*MDP*)

Un Processus de Décision Markovien décrit un problème de prise de décision séquentielle dans lequel un agent doit choisir la séquence d'actions qui maximise un critère d'optimisation basé sur une récompense [30, 116]. Formellement, un *MDP* est un ensemble $M = \{S, A, T, r, \gamma\}$, où :

- $S = \{s_1, \cdots, s_N\}$ est un ensemble fini de N états qui représente l'environnement dynamique,
- $A = \{a_1, \cdots, a_k\}$ est un ensemble de k actions qui peuvent être exécutées par un agent,
- $T : S \times A \times S \to [0,1]$ est une fonction de probabilité de transition, ou un modèle de transition, où $T(s, a, s')$ représente la probabilité de transition d'état lors de l'application de l'action $a \in A$ à l'état $s \in S$ menant à l'état $s' \in S$, c'est-à-dire $T(s, a, s') = P(s' | s, a)$,
- $r : S \times A \to R$ est une fonction de récompense dont la valeur absolue est bornée par R_{max} ou $r(s, a)$ représente la récompense immédiate obtenue lors de l'exécution de l'action $a \in A$ à l'état $s \in S$,

- $\gamma \in [0,1]$ est un facteur d'actualisation (*rabais ou dévaluation*).

Etant donné un *MDP* M, l'interaction agent-environnement dans la Figure (3.1) se déroule comme suit : soit $t \in N$ le temps actuel, soit $S_t \in S$ et $A_t \in A$ représentent l'état aléatoire de l'environnement et l'action choisie par l'agent au temps t, respectivement. Une fois que l'action est sélectionnée, elle est envoyée au système, qui effectue une transition :

$$(S_{t+1}, R_{t+1}) \sim P(\cdot | S_t, A_t) \tag{7}$$

En particulier, S_{t+1} est aléatoire et $P(S_{t+1} = s' | S_t = s, A_t = a) = T(s, a, s')$ est vrai pour tout s, $s' \in S$ et $a \in A$. De plus, $\mathbb{E}[R_{t+1} | S_t, A_t] = r(S_t, A_t)$. L'agent observe ensuite l'état suivant S_{t+1} et la récompense R_{t+1}, choisit une nouvelle action $A_{t+1} \in A$ et le processus est répété.

Une propriété importante d'un *MDP* est que le processus est *Markovien*, c'est-à-dire que l'action optimale à prendre pour un état particulier ne dépend pas de l'historique des actions et des états que l'agent a précédemment visités. L'état actuel fournit toutes les informations dont l'agent a besoin pour agir.

L'hypothèse de Markov [30] implique que la séquence de paires *état-action* spécifie le modèle de transition T :

$$P(S_{t+1} | S_t, A_t, \cdots, S_0, A_0) = P(S_{t+1} | S_t, A_t) \tag{8}$$

La transition d'état peut être déterministe ou stochastique. Dans le cas déterministe, prendre une action donnée dans un état donné donne toujours le même état suivant, tandis que dans le cas stochastique, l'état suivant est une variable aléatoire.

L'objectif de l'agent d'apprentissage est de déterminer une théorie de choix des actions afin de maximiser la récompense totale actualisée attendue :

$$R = \sum_{t=0}^{\infty} \gamma^t R_{t+1} \tag{9}$$

Si $\gamma < 1$, alors les récompenses reçues loin dans le futur ont une valeur exponentiellement moins importante que celles reçues au premier stade.

3.3.3. Politique et fonction de valeur

L'agent sélectionne ses actions selon une fonction particulière appelée *politique*. Une politique est définie comme une fonction : $\pi : S \times A \to [0,1]$ qui associe à chaque état $s \in S$ une distribution $\pi(s,\cdot)$ sur A, satisfaisant $\sum_{a \in A} \pi(a|s) = 1, \forall s \in S$.

Une politique stationnaire déterministe est le cas où pour tout $s \in S$, $\pi(\cdot|s)$ est concentré sur une seule action, c'est-à-dire qu'à tout instant $t \in \mathbb{N}$, $A_t = \pi(S_t)$. Une politique stationnaire stochastique est une fonction qui associe à chaque état une distribution de probabilité sur les différentes actions possibles, c'est-à-dire que $A_t \sim \pi(\cdot|S_t)$. L'ensemble de toutes les politiques stationnaires stochastiques est noté Π.

L'application d'une politique est réalisée de la manière suivante. Tout d'abord, un état initial S_0 est généré. Ensuite, la politique π suggère l'action $A_0 = \pi(S_0)$ et cette action est exécutée. En fonction de la fonction de transition T et de la fonction de récompense r, une transition est effectuée vers l'état S_1, avec une probabilité $T(S_0, A_0, S_1)$ et une récompense $R_1 = r(S_0, A_0, S_1)$ est reçue. Ce processus se poursuit, produisant une séquence $S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, \cdots$ appelée *trajectoire*, comme indiqué dans la Figure (3.5).

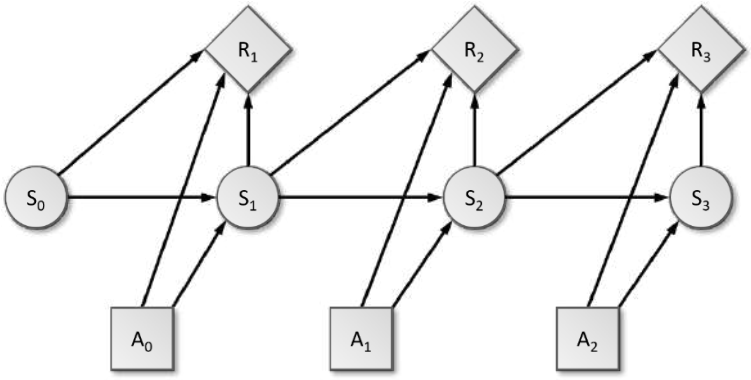


Figure 3.5 : *Réseau de décision d'un MDP fini.*

Les fonctions de valeur sont des fonctions d'états (*ou de paires état-action*) qui estiment à quel point il est avantageux pour l'agent d'être dans un état donné (*ou à quel point il est avantageux d'effectuer une action donnée dans un état donné*). La notion de *« quel est l'avantage »* ici est définie en termes de récompenses futures attendues, ou, pour être précis, en termes de retour attendu. Bien sûr, les récompenses que l'agent peut s'attendre à recevoir

dans le futur dépendent des actions qu'il prendra. En conséquence, les fonctions de valeur sont définies par rapport à des politiques particulières [30].

Etant donné une politique π, la fonction de valeur est définie comme étant une fonction $V^{\pi}: S \to \mathbb{R}$ qui associe à chaque état la somme attendue des récompenses que l'agent recevra s'il commence à exécuter la politique π à partir de cet état :

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t r(S_t, A_t) \mid S_0 = s\right], \forall s \in S \tag{10}$$

Où :

- S_t est la variable aléatoire représentant l'état au temps t,
- A_t est la variable aléatoire correspondant à l'action prise à cet instant telle que $P(A_t = a \mid S_t = s) = \pi(s,a)$.
- $(S_t, A_t)_{t\geq 0}$ est la séquence de paires *état-action* aléatoires générées en exécutant la politique π.

La fonction de valeur d'une politique stationnaire peut également être définie de manière récursive comme suit :

$$\begin{aligned} V^{\pi}(s) &= \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t r(S_t, A_t) \mid S_0 = s\right] \\ &= \mathbb{E}_{\pi}\left[r(S_0, A_0) + \sum_{t=1}^{\infty} \gamma^t r(S_t, A_t) \mid S_0 = s\right] \\ &= r(s, \pi(s)) + \mathbb{E}_{\pi}\left[\sum_{t=1}^{\infty} \gamma^t r(S_t, A_t) \mid S_0 = s\right] \\ &= r(s, \pi(s)) + \gamma \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t r(S_t, A_t) \mid S_0 \sim T(s, \pi(s), \cdot)\right] \\ &= r(s, \pi(s)) + \gamma \sum_{s' \in S} T(s, \pi(s), s') V^{\pi}(s') \end{aligned} \tag{11}$$

Où $\pi(s)$ est l'action associée à l'état s.

Si l'incertitude d'une politique stochastique $\pi(s)$ est prise en compte, $V^{\pi}(s)$ peut également être écrite de manière spécifique comme suit :

$$V^{\pi}(s) = \sum_{a \in A(s)} \pi(s,a) \left(r(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V^{\pi}(s') \right) \tag{12}$$

De même, la *Q-fonction* qui représente la fonction valeur d'action $Q^{\pi} : S \times A \to \mathbb{R}$ qui correspond à la valeur en partant d'un état s et en choisissant l'action a (*qui peut être différente de l'action choisie par la politique* π) puis en suivant la politique π jusqu'à atteindre un état terminal.

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi} \left[\sum_{t=0}^{\infty} \gamma^t r(S_t, A_t) \mid S_0 = s, A_0 = a \right] \tag{13}$$

Où S_t est distribué selon $\pi(S_t, \cdot)$ pour tout $t > 0$.

On appelle V^{π} la fonction *état-valeur* et Q^{π} la fonction *action-valeur*.

Finalement, la fonction d'avantage associée à π est définie comme suit :

$$A^{\pi} = Q^{\pi}(s,a) - V^{\pi}(s) \tag{14}$$

Une politique qui maximise la récompense totale attendue actualisée sur tous les états est appelée une politique optimale, notée π^*. Pour tout *MDP* fini, il existe au moins une politique optimale.

La fonction de valeur optimale V^* et la fonction de valeur-action Q^* sont définies par :

$$\begin{aligned} V^*(s) &= \sup_{\pi} V^{\pi}(s), \quad s \in S \\ Q^*(s,a) &= \sup_{\pi} Q^{\pi}(s,a), \quad s \in S, a \in A \end{aligned} \tag{15}$$

De plus, les fonctions de valeur optimales et de valeur d'action optimales sont reliées par les équations suivantes :

$$V^*(s) = \sup_{a \in A} Q^*(s,a), \quad s \in S \tag{16}$$

$$Q^*(s,a) = r(s,a) + \gamma \sum_{s' \in S} P(s' \mid s,a) V^*(s'), \quad s \in S, a \in A \tag{17}$$

Il s'avère que V^* et Q^* satisfont les équations d'optimalité de Bellman [117]. En particulier,

$$Q^*(s,a) = r(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) \max_{b \in A} Q^*(s',b) \tag{18}$$

$$V^*(s) = \max_{a \in A} r(s,a) + V^*(s') \tag{19}$$

Nous appelons une politique qui satisfait $\sum_{a \in A} \pi(a|s) Q(s,a) = \max_{a \in A} Q(s,a)$ pour tout état $s \in S$ *une politique gloutonne* (*greedy en anglais*) par rapport à la fonction Q. On sait que toutes les politiques qui sont gloutonnes par rapport à Q^* sont optimales et que toutes les politiques stationnaires optimales peuvent être obtenues de cette manière.

Ici, nous présentons les résultats importants suivants concernant les *MDP* [30] :

Théorème 2.1 (*Equations de Bellman*). Soit un problème de décision de Markov $M = \{S, A, T, r, \gamma\}$ et une politique $\pi : S \times A \to [0,1]$. Alors, pour tout $s \in S$, $a \in A$, V^π et Q^π satisfont :

$$V^\pi(s) = r(s, \pi(s)) + \gamma \sum_{s' \in S} T(s, \pi(s), s') V^\pi(s') \tag{20}$$

$$Q^\pi(s,a) = r(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V^\pi(s') \tag{21}$$

Théorème 2.2 (*Optimalité de Bellman*). Soit un problème de décision markovien $M = \{S, A, T, r, \gamma\}$ et une politique $\pi : S \times A \to [0,1]$ donnée. Alors, π est une politique optimale pour *M* si et seulement si, $\forall s \in S$,

$$\pi(s) = \arg\max_{a \in A} Q^\pi(s,a) \tag{22}$$

La probabilité de transition est : $T(s,a,s') = P(s'|s,a)$.

3.3.4. Les Processus Décisionnels de Markov Partiellement Observables

Dans le cadre d'un *MDP*, l'agent devrait obtenir des informations précises sur l'état de l'environnement à tout moment. Malheureusement, cette hypothèse peut ne pas être vérifiée en pratique car l'agent observe le monde à travers des récepteurs limités et imparfaits, et les informations contenues dans les perceptions ne sont pas suffisantes pour déterminer l'état. Les *Processus de Décision Markovien Partiellement Observables* (*POMDP*) [118] ont été introduits pour gérer de tels cas.

Formellement, un *POMDP* est un tuple $\langle S, A, T, O, Z, R \rangle$ où $\langle S, A, T, R \rangle$ est un *MDP*, et :

- O est un ensemble d'observations (*perceptions*),
- Z est une fonction d'observation : $Z(o, s, a)$ est la probabilité d'observer $o \in O$ si l'état du système est s et l'action qui a conduit à cet état est a.

Les observations peuvent être équivoques (*la même observation peut être observée dans différents états*) et stochastiques (*différentes observations peuvent être observées dans le même état*). Par conséquent, l'état du système ne peut pas être déterminé à partir des observations. Au lieu de cela, une observation peut être considérée comme une preuve sur l'état. La croyance de l'agent concernant l'état caché est une distribution de probabilité sur tous les états possibles, appelée l'état de croyance :

$$b_t = \left[\Pr\left(s_t = x^0\right), \Pr\left(s_t = x^1\right), \cdots, \Pr\left(s_t = x^{|S|-1}\right) \right]^T \tag{23}$$

L'agent commence avec un état de croyance initial b_0, et à chaque fois qu'une action a_t est exécutée et qu'une observation o_{t+1} est reçue, l'état de croyance b_t est mis à jour en utilisant la règle de Bayes :

$$\begin{aligned} b_{x+1} &= \Pr\left(s_{t+1} = s \middle| b_t, a_t, o_{t+1}\right) \\ &= \frac{\Pr\left(s_{t+1} = s, o_{t+1} \middle| b_t, a_t\right)}{\Pr\left(o_{t+1} \middle| b_t, a_t\right)} \\ &= \frac{\sum_{s' \in S} b_t(s') T(s, a_t, s') Z(o_{t+1}, s, a_t)}{\sum_{s' \in S} \sum_{s'' \in S} b_t(s') T(s, a_t, s'') Z(o_{t+1}, s'', a_t)} \end{aligned} \tag{24}$$

Par conséquent, l'état de croyance b_{t+1} est une fonction non linéaire de l'état de croyance précédent b_t, de l'action a_t et de l'observation o_{t+1} :

$$b_{t+1} = \tau\left(b_t, a_t, o_{t+1}\right) \tag{25}$$

Le vecteur d'état de croyance b_t à l'instant t nous permet de calculer la probabilité de toute observation o à l'instant $t+1$ comme suit :

$$\Pr\left(o_{t+1} = o \middle| b_t, a_t\right) = \sum_{s \in S} \sum_{s' \in S} b_t(s) T(s, a_t, s') Z(o, s', a_t) \tag{26}$$

La récompense attendue de l'exécution de l'action a pour un état de croyance b_t est donnée par :

$$r\left(a\middle|b_t\right)=\sum_{s\in S}b_t\left(s\right)R\left(s,a\right) \tag{27}$$

Un état de croyance markovien permet de formuler un *POMDP* comme un *MDP* où chaque croyance est un état. Le *MDP* de croyance résultant sera donc défini sur un espace d'états continu, car il y a une infinité de croyances pour tout *POMDP* donné [119]. Le *MDP* de croyance est défini comme un tuple $\left\langle B,A,\tau,r,\gamma\right\rangle$ où :

- B est l'ensemble des états de croyance sur les états du *POMDP*,
- A est le même ensemble d'actions que pour le *POMDP* d'origine,
- τ est la fonction de transition de l'état de croyance,
- $r:B\times A\to\mathbb{R}$ est la fonction de récompense sur les états de croyance,
- γ est le facteur de dévaluation égal à γ dans le *POMDP* d'origine.

Dans le *MDP* de croyance, τ et r doivent être dérivés du *POMDP* d'origine. Pour tous les $b,b'\in B$, $a\in A$:

$$\tau\left(b,a,a'\right)=\sum_{o\in O}\Pr\left(b'\middle|b,a,o\right)\Pr\left(o\middle|a,b\right) \tag{28}$$

$$r\left(b,a\right)=\sum_{x\in S}b\left(s\right)R\left(s,a\right) \tag{29}$$

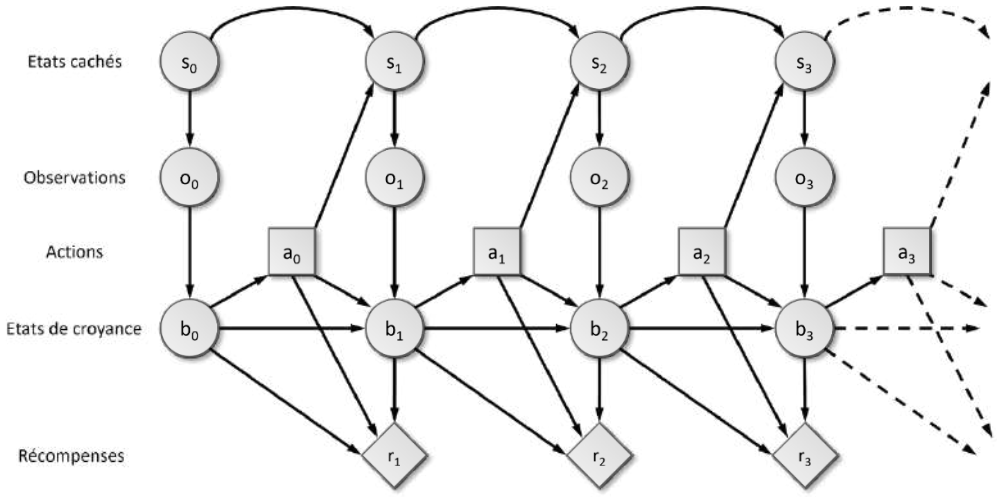


Figure 3.6 : *Schéma du POMDP.*

La Figure (3.6) illustre la série temporelle des états de croyance. La propriété de Markov implique que l'état de croyance et l'historique complet du système (*séquence d'actions et d'observations*) contiennent exactement les mêmes informations sur l'état courant, et par conséquent, sur tous les événements futurs.

3.3.5. Méthodes avec ou sans modèle

Les algorithmes d'apprentissage par renforcement peuvent être classés comme étant sans modèle ou basés sur un modèle. Dans les algorithmes basés sur un modèle, l'agent tente de modéliser le problème, c'est-à-dire d'apprendre le *MDP* (*car une fois que nous connaissons les fonctions de récompense et de transition, le problème peut être résolu par optimisation*). Dans les algorithmes sans modèle, l'agent apprend directement comment optimiser la récompense (*Q-learning ou Policy Gradient, par exemple*). Dans la navigation autonome, le modèle de l'environnement n'est pas connu (*on ne peut pas connaître les actions futures des acteurs environnants*) et il est difficile d'apprendre un modèle de l'environnement dans un espace de grande dimension.

3.3.6. Exploration vs Exploitation

L'Exploration versus Exploitation est un dilemme très connu en *RL*. L'exploitation vise à choisir la meilleure option à partir des connaissances actuelles, tandis que l'exploration consiste à tester d'autres solutions pour recueillir des informations, car une option non explorée pourrait être meilleure que les autres déjà testées. L'exploration et l'exploitation sont toutes deux cruciales en *RL*, et le défi consiste à trouver le bon équilibre entre ces deux notions. Il semble raisonnable d'explorer beaucoup au début de l'algorithme, lorsque tous les chemins n'ont pas encore été explorés, et d'explorer de moins en moins au fur et à mesure que l'algorithme s'améliore.

Deux approches courantes pour faire face au dilemme sont l'approche *ε-greedy* et *softmax*. L'approche *ε-greedy* consiste à choisir la meilleure action (*l'action la plus prometteuse*) avec une probabilité de $1-\varepsilon$ et une action aléatoire avec une probabilité ε (*des valeurs courantes pour ε vont de 0.01 à 0.1*). Avec l'approche *ε-greedy*, toutes les actions peuvent être choisies avec la même probabilité. L'approche *softmax* permet de différencier

les actions selon leur qualité $Q(a,s)$. La prochaine action est sélectionnée selon la distribution de *Boltzmann* :

$$\pi(a,s) \sim \frac{e^{\frac{Q(a,s)}{T}}}{\sum_{a_i \in A} e^{\frac{Q(a_i,s)}{T}}} \tag{30}$$

Où :

- a est l'action choisie au temps t,
- T est une constante appelée *température*.

Les approches *ε-greedy* et *softmax* fonctionnent toutes deux dans le cas d'actions discrètes. Dans le cas d'actions continues, l'exploration est généralement réalisée en échantillonnant des actions aléatoires autour de la sortie du modèle, souvent en utilisant une distribution gaussienne :

$$\pi(a,s) \sim N(f(s),\sigma) \tag{31}$$

Dans l'Equation (31) ;

- $f(s)$ est la sortie du modèle,
- σ est l'écart-type de la distribution gaussienne (*qui peut être fixé ou appris par le modèle*).

3.3.7. On-Policy vs Off-Policy

Les algorithmes de *RL* fonctionnent en deux phases : l'interaction avec l'environnement pour collecter des données, et la mise à jour de la politique en utilisant les données collectées. Les algorithmes *on-policy* utilisent toujours la politique actuelle pour collecter des données, tandis que les algorithmes *off-policy* utilisent une autre politique pour collecter des données (*une politique plus ancienne ou une politique gloutonne*). Les algorithmes *on-policy* sont théoriquement plus stables - car la politique est mise à jour avec des données collectées avec elle-même - mais sont généralement plus lents que les algorithmes *off-policy*. En effet, les algorithmes *off-policy* sont efficaces en termes d'échantillonnage - c'est-à-dire qu'une donnée peut être réutilisée plusieurs fois - et la collecte de données et la mise à jour de la politique peuvent être effectuées en même temps, ce qui accélère l'apprentissage.

3.3.8. Objectifs de l'apprentissage par renforcement

Le but de l'apprentissage par renforcement est de trouver une politique optimale π^* qui associe des états ou des observations à des actions afin de maximiser le retour attendu J, qui correspond à la récompense cumulative attendue. Dans un modèle à horizon fini, on cherche uniquement à maximiser la récompense attendue pour l'horizon H, c'est-à-dire pour les H prochaines étapes (*dans le temps*) h :

$$J = \mathbb{E}\left\{\sum_{h=0}^{H} R_h\right\} \tag{32}$$

Ce paramétrage peut également être appliqué pour modéliser des problèmes où l'on sait combien d'étapes restent. Alternativement, les récompenses futures peuvent être actualisées avec un facteur de rabais γ (avec $0 \leq \gamma < 1$) :

$$J = \mathbb{E}\left\{\sum_{h=0}^{\infty} \gamma^h R_h\right\} \tag{33}$$

Deux objectifs naturels se présentent pour l'apprenant. Dans le premier, nous essayons de trouver une stratégie optimale à la fin d'une phase d'apprentissage ou d'interaction. Dans le second, l'objectif est de maximiser la récompense sur l'ensemble du temps pendant lequel l'agent interagit avec l'environnement.

3.4. Classification des principaux algorithmes d'apprentissage par renforcement

Nous présentons ici un diagramme pour classer les algorithmes d'apprentissage par renforcement (*Figure (3.7)*). Nous pouvons d'abord séparer les algorithmes d'apprentissage par renforcement en deux catégories : les méthodes tabulaires et l'approximation de fonction. Les méthodes tabulaires supposent un *MDP* fini, c'est-à-dire avec un nombre fini d'états et d'actions. Nous présenterons ces méthodes dans le paragraphe suivant, puis nous approfondirons l'approximation de fonction, où le *MDP* peut être infini.

3.4.1. Les méthodes tabulaires

Cette section est inspirée du livre « *Reinforcement Learning: An Introduction* » de Richard Sutton et Andrew Barto [30]. Les premières méthodes présentées sont les méthodes tabulaires où le *MDP* est fini et les dimensions des espaces d'états et d'actions sont petites,

c'est-à-dire que la fonction de valeur et la fonction Q peuvent être représentées sous forme de tableaux (*Tableau (3.1) et Tableau (3.2)*).

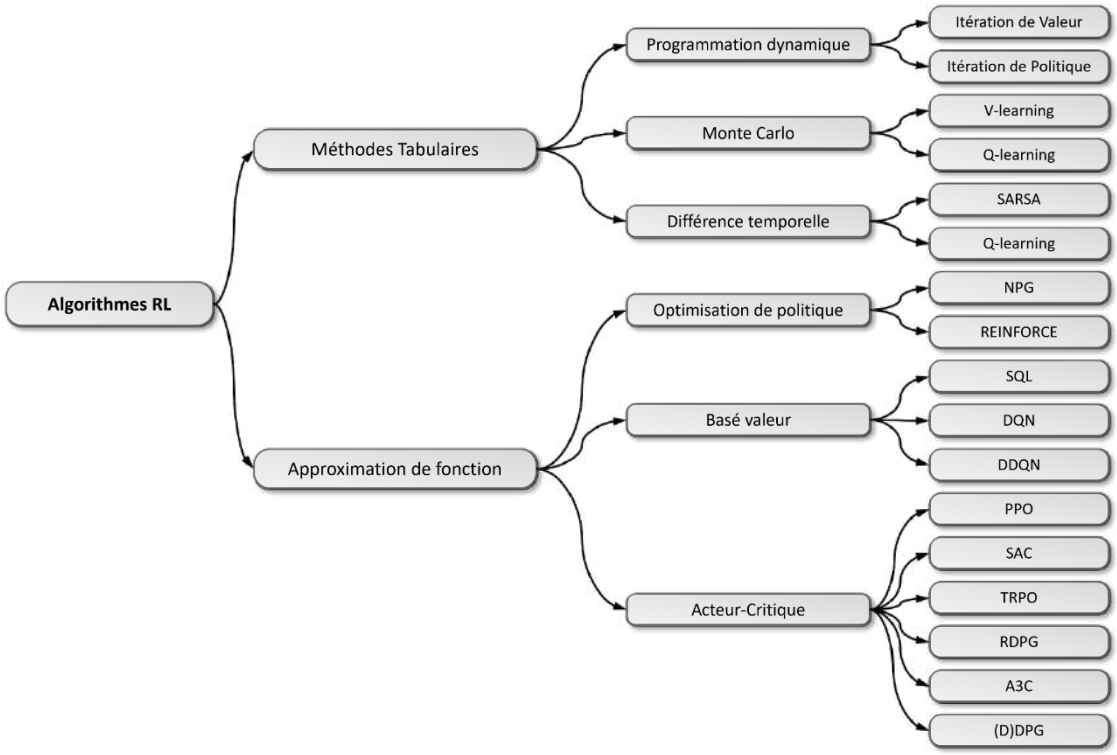


Figure 3.7 : *Classification des principaux algorithmes d'apprentissage par renforcement.*

L'objectif des méthodes tabulaires est de remplir ces tableaux, car une fois qu'ils sont connus, la politique optimale peut être déduite :

- De la fonction de valeur dans le cas où la fonction de transition est connue :

$$\pi^*(s) = \arg\max_a \sum_{s',r} p(s', r|s, a)\left[r + \gamma V(s')\right] \tag{34}$$

- De la fonction Q :

$$\pi^*(s) = \arg\max_a Q(a, s) \tag{35}$$

Tableau 3.1 : *Fonction valeur.*

Etat	Valeur
s_0	15
s_1	20
…	…
s_n	…

Tableau 3.2. *Fonction Q.*

		Actions			
		Haut	*Bas*	*Gauche*	*Droite*
Etats	s_0	-1	12	0	15
	s_1	3	20	4	14
	...	…	…	…	…
	s_n	…	…	…	…

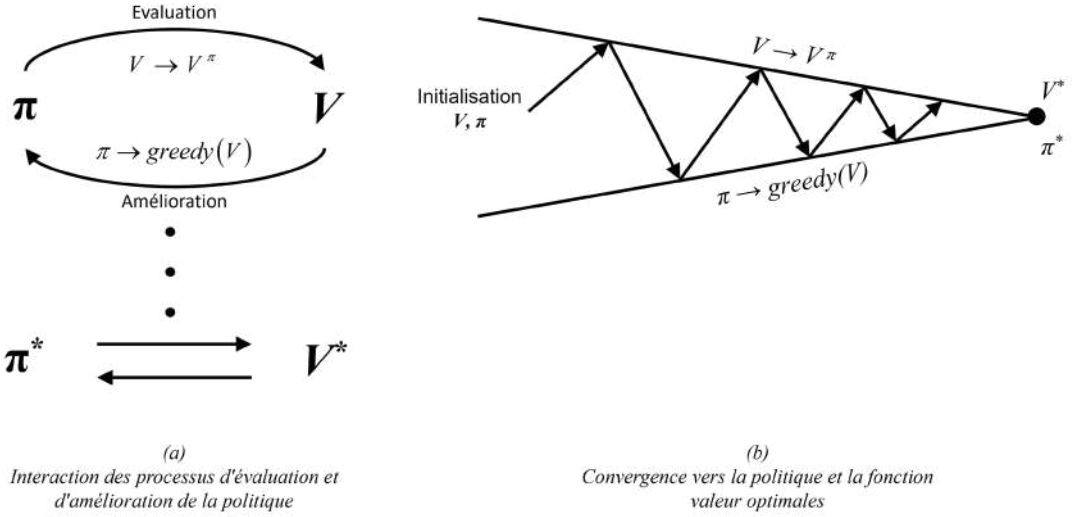
Nous pouvons diviser les méthodes tabulaires en 3 catégories : la programmation dynamique, les méthodes de Monte-Carlo et l'apprentissage par renforcement par différence temporelle (*TD-learning*).

3.4.1.1. La programmation dynamique : algorithmes basés sur un modèle

La programmation dynamique (*DP*) est une méthode de calcul de politique optimale π^* afin de résoudre un *MDP* donné. La programmation dynamique suppose une connaissance complète du *MDP*, y compris la dynamique de transition de l'environnement et la fonction de récompense [120]. Par conséquent, elle est classée comme un algorithme d'apprentissage basé sur un modèle.

Les algorithmes de programmation dynamique pour résoudre les *MDP* peuvent être catégorisés en l'une des deux familles : *l'itération de la valeur* (*VI*) et *l'itération de la politique* (*PI*) [30]. Les deux approches partagent un mécanisme sous-jacent commun, le principe d'itération généralisée de politique (*Generalized Policy Iteration GPI*) [30], représenté dans la Figure (3.8). Ce principe se compose de deux processus d'interaction. La première étape, l'évaluation de la politique, estime l'utilité de la politique actuelle π, c'est-à-dire qu'elle calcule la valeur V^{π}. Cette étape recueille des informations sur la politique pour calculer la deuxième étape, l'amélioration de la politique. Dans cette étape, les valeurs des actions sont évaluées pour chaque état, afin de trouver des améliorations possibles, c'est-à-dire d'autres actions dans des états particuliers qui sont meilleures que l'action que la politique actuelle propose. Cette étape calcule une politique améliorée π' à partir de la politique actuelle π en utilisant les informations de V^{π}. Tant que les deux processus continuent de mettre à jour tous les états, l'objectif final est de converger vers la fonction de valeur optimale

et une politique optimale. La Figure (3.8 b) présente une métaphore géométrique pour la convergence à la fois de la fonction de valeur et de la politique dans *GPI*.



(a)
Interaction des processus d'évaluation et d'amélioration de la politique

(b)
Convergence vers la politique et la fonction valeur optimales

Figure 3.8 : *Itération généralisée de politique.*

3.4.1.2. Itération de politique

L'itération de politique itère entre les deux processus *GPI*. Cela est répété jusqu'à la convergence vers une politique optimale. Cette méthode est représentée dans l'Algorithme (3.1).

Algorithme 3.1 : *Itération de politique [30]*

Entrée : *un modèle de* $MDP\langle S, A, T, r, \gamma \rangle$

/ Initialisation */*

1. $t = 0,\ k = 0$ *;*

2. *Pour tout* $s \in S$ *: Initialiser* $\pi_t(s)$ *avec une action arbitraire ;*

3. *Pour tout* $s \in S$ *: Initialiser* $V_k(s)$ *avec une valeur arbitraire ;*

4. Répéter

/ Evaluation de politique */*

5. Répéter

6. *Pour tout* $s \in S$ *:* $V_{k+1}(s) = r(s, \pi_t(s)) + \gamma \sum_{s' \in S} T(s, \pi_t(s), s') V_k(s')$ *;*

7. $k \leftarrow k + 1$ *;*

8. Jusqu'à ce que *pour tout* $s \in S$ *:* $\left| V_k(s) - V_{k-1}(s) \right| < \varepsilon$ *;*

/ Amélioration de politique */*

9. *Pour tout* $s \in S$ *:* $\pi_{t+1}(s) = \arg\max_{a \in A} \left[r(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V_k(s') \right]$ *;*

10. $t \leftarrow t + 1$ *;*

11. Jusqu'à ce que $\pi_t = \pi_t - 1$ *;*

12. $\pi^* = \pi_t$ *;*

Sortie : *une politique optimale* π^*.

Il s'agit de commencer par une politique π_t choisie au hasard et une initialisation aléatoire de la fonction de valeur correspondante V_k, pour $k=0$ et $t=0$ (*étapes 1 à 3*), puis de répéter de manière itérative les opérations d'évaluation et d'amélioration de la politique.

La phase d'évaluation de la politique (*étapes 5 à 8*) consiste à calculer la valeur d'action de la politique π_{t+1} en résolvant l'Equation (15) pour tous les états $s \in S$. Une méthode itérative efficace pour résoudre cette équation est d'initialiser la fonction de valeur de π_{t+1} avec la fonction de valeur V_k de la politique précédente, puis de répéter l'opération :

$$\forall s \in S : V_{k+1}(s) = r\left(s, \pi_t(s)\right) + \gamma \sum_{s' \in S} T\left(s, \pi_t(s), s'\right) V_k(s') \tag{36}$$

Jusqu'à ce que $\forall s \in S : \left|V_k(s) - V_{k-1}(s)\right| < \varepsilon$, pour un seuil d'erreur prédéfini ε.

L'amélioration de la politique (*étapes 9 à 10*) consiste à trouver la politique *greedy* π_{t+1} donnée par la fonction de valeur V_k :

$$\forall s \in S : \pi_{t+1}(s) = \arg\max_{a \in A} \left[r(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V_k(s') \right] \tag{37}$$

Ce processus s'arrête lorsque $\pi_t = \pi_t - 1$, et une politique optimale est obtenue, $\pi^* = \pi_t$.

En somme, l'itération de politique génère une séquence directe de politiques et de fonctions de valeur alternées :

$$\pi_0 \to V^{\pi_0} \to \pi_1 \to V^{\pi_1} \to \cdots \to \pi^* \to V^* \to \pi^*$$

Les processus d'évaluation de politique se produisent dans les transitions de $\pi_t \to V^{\pi_t}$ tandis que les conversions de V^{π_t} à π_{t+1} sont réalisées par les processus d'amélioration de politique.

3.4.1.3. Itération de valeur

Le principal inconvénient de l'itération de politique est qu'une évaluation de politique complète est requise à chaque itération. L'itération de valeur consiste à superposer les processus d'évaluation et d'amélioration.

Au lieu de séparer complètement les processus d'évaluation et d'amélioration, l'approche de l'itération de valeur interrompt l'évaluation après une seule itération. En fait, elle intègre immédiatement l'étape d'amélioration de politique dans ses itérations, en se concentrant principalement sur l'estimation directe de la fonction de valeur.

L'itération de valeur, décrite dans l'Algorithme (3.2), peut être écrite comme une simple opération de sauvegarde :

$$\forall s \in S : V_{k+1}(s) = \max_{a \in A} \left[r(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V_k(s') \right] \tag{38}$$

Cette opération est répétée (*étapes 3 à 6*) jusqu'à ce que $\forall s \in S : \left| V_k(s) - V_{k-1}(s) \right| < \varepsilon$, et dans ce cas la politique optimale est simplement la politique gloutonne (*greedy*) par rapport à la fonction de valeur V_k (*étape 7*).

L'itération de valeur produit la séquence suivante de fonctions de valeur :

$$V_0 \to V_1 \to V_3 \to V_4 \to V_5 \to \cdots \to \pi^*$$

Algorithme 3.2 : *Itération de valeur [30]*

Entrée : *un modèle de* $MDP\langle S, A, T, r, \gamma \rangle$

1. $k = 0$ *;*
2. *Pour tout* $s \in S$ *: Initialiser* $V_k(s)$ *avec une valeur arbitraire ;*
  3. ***Répéter***
    4. *Pour tout* $s \in S$ *:* $V_{k+1}(s) = \max_{a \in A} \left[r(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V_k(s') \right]$ *;*
    5. $k \leftarrow k+1$ *;*
  6. ***Jusqu'à ce que*** *pour tout* $s \in S$ *:* $\left| V_k(s) - V_{k-1}(s) \right| < \varepsilon$ *;*
  7. *Pour tout* $s \in S$ *:* $\pi^*(s) = \arg\max_{a \in A} \left[r(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V_k(s') \right]$ *;*

Sortie : *une politique optimale* π^*.

3.4.2. Apprentissage par Renforcement : Algorithmes sans modèle

Contrairement à la programmation dynamique qui suppose la disponibilité d'un modèle parfait de l'environnement, le *RL* est principalement préoccupé par la façon d'obtenir une politique optimale lorsque ce modèle n'est pas disponible. Par conséquent, le *RL* est sans modèle. De plus, le *RL* ajoute aux *MDP* une attention particulière à l'approximation et à

l'information incomplète, ainsi que la nécessité d'échantillonner et d'explorer pour recueillir des connaissances statistiques sur ce modèle inconnu.

Dans un problème *RL*, l'agent et son environnement peuvent être modélisés comme étant dans un état $s \in S$ et peuvent effectuer des actions $a \in A$, chacune pouvant appartenir à des ensembles discrets ou continus et pouvant être multidimensionnelles. Un état s contient toutes les informations pertinentes sur la situation actuelle pour prédire les états futurs. Une action a est utilisée pour contrôler l'état du système. A chaque étape, l'agent reçoit également une récompense R, qui est une valeur scalaire et supposée être une fonction de l'état et de l'observation. Elle peut également être modélisée comme une variable aléatoire qui dépend uniquement de ces variables. Dans la tâche de navigation, une récompense possible pourrait être conçue en fonction des coûts énergétiques pour les actions prises et des récompenses pour atteindre les cibles. L'apprentissage par renforcement est conçu pour trouver une politique π reliant les états aux actions, qui choisit une action a dans l'état donné s maximisant la récompense cumulative attendue. La politique π peut être déterministe ou stochastique. La première utilise toujours la même action exacte pour un état donné sous la forme $a = \pi(s)$, la seconde tire un échantillon d'une distribution sur les actions lorsqu'elle rencontre un état, c'est-à-dire $a \sim \pi(s,a) = P(a|s)$. L'agent d'apprentissage par renforcement doit découvrir les relations entre les états, les actions et les récompenses. Par conséquent, l'exploration est nécessaire, qui peut être directement intégrée dans la politique ou effectuée séparément et uniquement dans le cadre du processus d'apprentissage. Différents types de fonctions de récompense sont couramment utilisés, notamment des récompenses dépendant uniquement de l'état actuel $R = R(s)$, des récompenses dépendant de l'état et de l'action actuels $R = R(s,a)$, et des récompenses incluant les transitions $R = R(s',a,s)$.

3.4.2.1. Les méthodes de Monte Carlo

Les méthodes de *Monte Carlo* utilisent l'échantillonnage pour estimer la fonction de valeur et découvrir la politique optimale [30]. Cette procédure peut être utilisée pour remplacer l'étape d'évaluation de politique des méthodes de programmation dynamique décrites ci-dessus. Contrairement à la programmation dynamique, les méthodes de *Monte Carlo* ne supposent pas une connaissance complète de l'environnement, ce sont des méthodes

sans modèle, c'est-à-dire qu'elles n'ont pas besoin d'une fonction de transition explicite. Elles ne nécessitent que de l'expérience des séquences d'échantillons d'états, d'actions et de récompenses provenant d'interactions en ligne ou simulées avec un environnement. Apprendre à partir de l'expérience en ligne ne nécessite aucune connaissance préalable de la dynamique de l'environnement, mais peut encore atteindre un comportement optimal. Apprendre à partir de l'expérience simulée nécessite un modèle, mais le modèle doit simplement générer des transitions d'échantillons, pas les distributions de probabilité complètes de toutes les transitions possibles qui sont requises par les méthodes de programmation dynamique.

Les méthodes de *Monte Carlo* résolvent les problèmes d'apprentissage par renforcement en calculant la moyenne des retours d'échantillonnage. Elles effectuent des simulations en exécutant la politique courante sur le système, opérant ainsi sur la politique en cours. Les fréquences des transitions et des récompenses sont suivies et utilisées pour former des estimations de la fonction de valeur. Par exemple, dans un cadre épisodique, la valeur *d'état-action* d'une paire *état-action* donnée peut être estimée en faisant la moyenne de tous les retours qui ont été reçus en partant de ceux-ci.

3.4.2.2. Les méthodes de la différence temporelle

Les méthodes de la différence temporelle (*Temporal Difference TD*) sont une combinaison des méthodes de *Monte Carlo* et des méthodes de programmation dynamique [30]. Contrairement aux méthodes de *Monte Carlo*, les méthodes *TD* n'ont pas besoin d'attendre qu'une estimation du retour soit disponible (*c'est-à-dire à la fin d'un épisode*) pour mettre à jour la fonction de valeur. Au lieu de cela, elles utilisent des erreurs temporelles et doivent attendre jusqu'au prochain pas de temps. L'erreur temporelle est la différence entre l'ancienne estimation et une nouvelle estimation de la fonction de valeur, en prenant en compte la récompense reçue dans l'exemple actuel. Ces mises à jour sont effectuées de manière itérative et, contrairement aux méthodes de programmation dynamique, ne tiennent compte que des états successeurs échantillonnés plutôt que des distributions complètes sur les états successeurs. Comme les méthodes de *Monte Carlo*, ces méthodes sont sans modèle, car elles n'utilisent pas de modèle de la fonction de transition pour déterminer la fonction de valeur et peuvent apprendre directement à partir de l'expérience brute sans modèle de la

dynamique de l'environnement. Dans ce cadre, la fonction de valeur ne peut pas être calculée analytiquement mais doit être estimée à partir des transitions échantillonnées dans le *MDP*.

3.4.3. Approximation de fonction

Toutes les trois classes d'algorithmes présentées précédemment supposaient un *MDP* fini, ce qui peut être limitatif pour les problèmes continus, voire même pour les problèmes à haute dimension. Dans cette section, nous allons approfondir la deuxième classe d'algorithmes d'apprentissage par renforcement, appelée approximation de fonction. Cette fois, les algorithmes ne convergent pas nécessairement vers la solution optimale comme c'est le cas avec les méthodes tabulaires, car nous sommes maintenant dans une dimension infinie. Le terme *malédiction de la dimensionnalité* a été inventé par *Richard Bellman* en 1961 pour décrire les difficultés qui surgissent lorsqu'on augmente la taille des données. Les algorithmes d'approximation de fonction peuvent être divisés en trois catégories. La première est celle des algorithmes basés sur la valeur, qui reposent sur l'inférence de Q pour chaque paire *état-action*, puis déduisent la politique optimale à partir de cette inférence. Dans l'approximation de fonction, l'espace des états peut maintenant être très grand, mais pour utiliser des algorithmes basés sur la valeur, l'espace d'action doit encore être fini. La deuxième catégorie, les algorithmes basés sur la politique, permet d'avoir un espace d'action grand ou infini en calculant directement la politique, sans intermédiaire. Mais au lieu d'utiliser des algorithmes basés sur la politique, une troisième catégorie est plutôt utilisée car elle est plus stable. Il s'agit des *acteurs-critiques*, qui calculent à la fois la politique et la valeur d'état en même temps.

3.4.3.1. Algorithmes basés sur la valeur

Les algorithmes basés sur la valeur reposent sur le calcul de la valeur de chaque état, et plus précisément sur la valeur Q pour chaque paire (*état, action*). Ils se basent sur le fait que la connaissance de la fonction Q suffit pour obtenir la politique optimale $\pi^*(s) = \arg\max_a Q(a,s)$. L'apprentissage profond de Q (*Deep Q-learning*) est l'algorithme basé sur la valeur le plus connu. Il calcule Q dans le cas d'un espace d'état non fini. Nous présentons l'apprentissage profond de Q dans cette section, ainsi que certaines de ses variantes.

3.4.3.1.1. Deep Q-learning (*DQN*)

Deep Q-learning a été introduit par Mnih et *al*. dans « *Playing Atari with Deep Reinforcement Learning* » [121]. Ils ont mis en œuvre le *Q-learning* avec des entrées d'image. Dans ce cas, les dimensions (*en particulier les dimensions de l'espace d'état*) sont trop grandes pour les méthodes tabulaires.

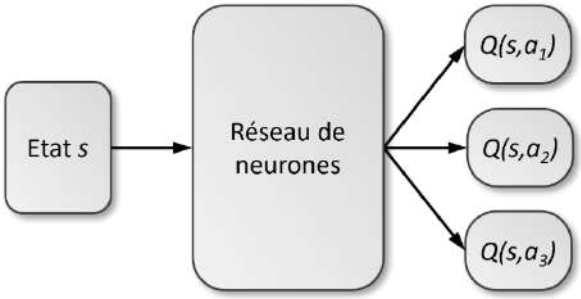


Figure 3.9 : *Deep Q learning.*

La fonction Q est représentée par un réseau de neurones convolutifs (*un réseau de neurones profond, d'où le nom de Deep Q Learning*), dont les entrées sont des images prétraitées (*état actuel + états précédents*), et les sorties sont les valeurs Q prédites pour chaque action possible. Le paramètre de Q est θ (*les poids du réseau de neurones*). La fonction de coût à optimiser dans *DQN* est :

$$L = \mathbb{E}\left[\left(y_i^{DQN} - Q(s,a,\theta)\right)^2\right] \tag{39}$$

Avec :

$$y_i^{DQN} = r_i + \gamma \max_{a'} Q\left(s_{i+1}, a' \middle| \theta\right) \tag{40}$$

Cependant, de nombreux algorithmes d'apprentissage supposent que les données sont indépendantes et identiquement distribuées, ce qui n'est pas le cas en apprentissage par renforcement. Comme nous traitons des états séquentiels, les états successifs sont fortement corrélés. Pour surmonter le problème des données corrélées, Mnih et *al*. utilisent la technique de répétition d'expérience (*Experience replay*) introduite par Lin dans sa thèse en 1993, et plus tard formalisée et testée par de nombreux chercheurs, comme Adam et *al*. dans « *Experience replay for real-time reinforcement learning control* » [122]. Pendant l'apprentissage, les échantillons sont stockés dans un tampon, et les mises à jour du modèle (*c'est-à-dire du réseau de neurones*) sont effectuées avec des *mini-lots* échantillonnés de

manière aléatoire à partir du tampon. Détaillé dans l'Algorithme (3.3), *Deep Q-learning* a obtenu des résultats de pointe sur sept jeux Atari.

Algorithme 3.3 : *Deep Q-learning avec répétition d'expérience [121]*

1. *Initialiser la capacité de la mémoire de répétition D à N ;*
2. *Initialiser la fonction d'action-valeur Q avec des poids aléatoires ;*
3. ***Pour*** *épisode = 1, M* ***faire :***
4. *Initialiser la séquence* $s_1 = x_1$ *et les séquences prétraitées* $\phi_1 = \phi(s_1)$ *;*
  5. ***Pour*** *t = 1 à T* ***faire :***
  6. *Avec une probabilité* ***ε****, sélectionner une action aléatoire* $\boldsymbol{a_t}$ *;*
  7. *Sinon, sélectionner* $a_t = \max_a Q^*(\phi(s_t), a; \theta)$ *;*
  8. *Exécuter l'action* a_t *dans l'émulateur et observer la récompense* r_t *et l'image* x_{t-1} *;*
  9. *Définir* $s_{t+1} = s_t, a_t, x_{t+1}$ *et prétraiter* $\phi_{t+1} = \phi(s_{t+1})$ *;*
  10. *Stocker la transition* $(s_{t+1}, a_t, r_t, \phi_{t+1})$ *dans D ;*
  11. *Échantillonner un mini-lot de transitions* $(s_j, a_j, r_j, \phi_{j+1})$ *aléatoirement à partir de D ;*
  12. $y_i = \begin{cases} r_j & \text{pour le terminal } \theta_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a', \theta) & \text{pour le non terminal } \theta_{j+1} \end{cases}$
  13. *Effectuer une étape de descente de gradient sur* $y_i - Q(\phi_j, a_j; \theta)$ *selon l'Equation* (*39*) *;*
  14. Fin pour *;*
15. Fin pour *;*

3.4.3.1.2. Améliorations et variantes

Plus tard en 2015, les auteurs Mnih et *al*. ont poussé plus loin le *Deep Q-learning* et ont réussi à l'implémenter avec succès sur les 49 jeux Atari disponibles dans « *Human-level control through deep reinforcement learning* » [123]. L'amélioration par rapport à la version précédente du *Deep Q-learning* est l'ajout d'un réseau cible utilisé pour les mises à jour, ce qui améliore la stabilité des algorithmes.

Introduit par Van Hasselt et *al*. dans « *Double Q learning* » [124], et plus tard dans « *Deep Reinforcement Learning with Double Q-Learning* » [64], l'apprentissage en *double Q* est l'ajout du réseau cible. Le problème du *DQN* classique réside dans l'Equation (39) car y_i (*dans l'Equation (40)*) est également une fonction de *Q*, ce qui peut entraîner un comportement instable et une surestimation des valeurs en raison du maximum dans l'Equation (40). Par conséquent, un réseau cible *Q* avec des paramètres θ' est utilisé pour le calcul de la cible y_i, de sorte que le réseau *Q* utilisé pour le calcul de y_i ne change pas à chaque itération. L'Equation (40) devient :

$$y_i^{DDQN} = r_i + \gamma \max_{a'} Q\left(s_{i+1}, a' \middle| \theta'\right) \tag{41}$$

Tous les *N* pas, les paramètres du réseau cible sont mis à jour avec $\theta' \leftarrow \theta$. Dans [125], Van Hasselt et *al*. comparent les estimations de valeur et les scores sur six jeux Atari avec *DQN* et *DDQN* (*Double Deep Q Network*), et *DQN* montre des scores et une stabilité plus faibles que *DDQN*, ainsi qu'un dépassement beaucoup plus important de l'estimation de la valeur *Q*. Le double apprentissage profond en *Q* est donc une version plus efficace et stable de *DQN*.

D'autres chercheurs ont également amélioré l'expérience de jeu avec « *Prioritized Experience Replay »* [126]. Dans cet article, les transitions se voient attribuer des probabilités d'être sélectionnées, de sorte que les plus intéressantes sont échantillonnées plus fréquemment à partir du tampon de relecture. L'importance d'un échantillon dépend de son erreur *TD :* $\delta = r_{t+1} + \gamma \max_{a'} Q_{\text{target}}\left(s_{t+1}, a'\right) - Q\left(s_t, a_t\right)$, afin qu'il puisse améliorer plus rapidement le réseau *Q*. Andrychowicz et *al*. ont proposé dans « *Hindsight Experience Replay »* [127] une autre manière (*appelée HER*) de traiter les récompenses rares sans ajuster la fonction de récompense. L'idée de *HER* est de répéter des épisodes infructueux tout en changeant l'objectif initial par l'état réellement atteint par l'agent. C'est particulièrement destiné à traiter les récompenses rares, dans des environnements où l'exploration ne peut pas toujours suffire à atteindre l'objectif. L'objectif est ensuite stocké et devient une variable de transition. Des tests sont effectués avec le simulateur *Mujoco* [128] avec trois tâches : pousser une boîte, faire glisser une rondelle ou ramasser et placer une boîte, avec des récompenses rares (*l'agent ne reçoit une récompense que lorsque la tâche est réussie*). Les auteurs montrent que la combinaison de *DDPG* + *HER* parvient à accomplir la tâche dans un pourcentage élevé des cas, tandis que *DDPG* seul n'a pas été en mesure d'apprendre dans certaines des tâches.

L'un des principaux problèmes en apprentissage par renforcement est l'exploration. L'exploration classique est *ε-greedy*, mais si l'objectif ne peut être atteint qu'après une succession spécifique d'actions (*comme dans l'environnement du véhicule de montagne, où la voiture doit gagner de l'élan pour monter une colline*), effectuer une action aléatoire de temps en temps peut ne pas suffire. Dans « *Deep Exploration via Bootstrapped DQN »* [129], Osband et *al.* proposent une version de *DQN* où les fonctions *Q* sont échantillonnées sur une

distribution au début d'un épisode. Par conséquent, l'agent peut explorer et être cohérent sur un épisode, ce qui est appelé exploration profonde.

D'autres chercheurs testent de nouvelles architectures : dans « *Dueling Network Architectures for Deep Reinforcement Learning* » [130], Wang et *al.* proposent une nouvelle architecture avec deux flux : l'un pour l'estimation de la valeur de l'état (*fonction V*) et l'autre pour l'estimation de l'avantage. Les deux flux sont combinés pour produire les valeurs Q (*voir Figure (3.10)*), car $Q(s,a) = V(s) + A(s,a)$.

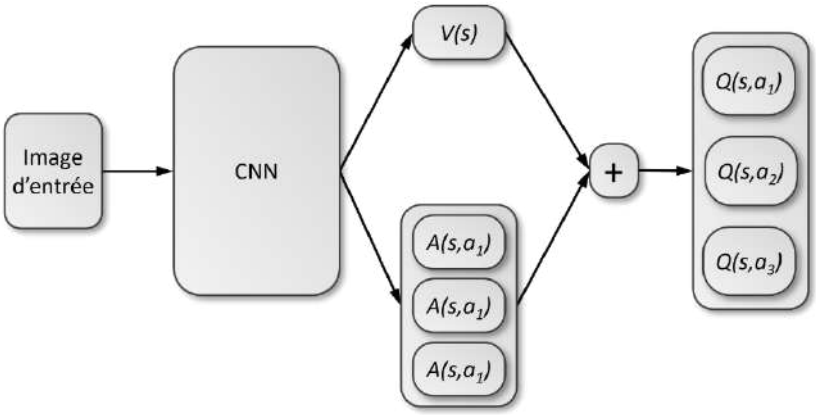


Figure 3.10 : *Architecture Dueling* [130].

L'idée clé derrière cette architecture est que dans de nombreux environnements, les actions ne sont importantes que parfois (*par exemple, dans le jeu Breakout, le mouvement de la raquette ne compte que lorsque la balle est près de la raquette*). En calculant explicitement à la fois la valeur et l'avantage au lieu de calculer directement Q, le réseau apprendra dans quel état l'action est réellement importante. En particulier, l'avantage estime si une action est meilleure que la moyenne et détecte donc si l'action a peu d'importance dans l'état considéré. Wang et *al*. comparent leur architecture *dueling* avec une architecture à un seul flux, et le résultat est que le réseau Q dueling converge plus rapidement et mène à de meilleures performances que celui à un seul flux. En particulier, sur les 57 jeux Atari, le score moyen par rapport aux performances humaines de l'architecture *dueling* est de 591,9%, tandis qu'il est de 341,2% pour le *DQN* de [123].

Plus récemment, une équipe de *Deep Mind* a combiné plusieurs améliorations de *DQN* pour mesurer la complémentarité de ces extensions comme l'apprentissage double Q [124]

ou la relecture d'expérience [126] dans « *Rainbow: Combining Improvements in Deep Reinforcement Learning* » [131].

3.4.3.2. Le gradient de politique (*Policy Gradient*)

Deep *Q-learning* est un algorithme efficace, mais ne peut être appliqué qu'à des espaces d'actions discrètes. Dans le cas d'actions continues, il est possible de les discrétiser, mais cela se heurte rapidement à la malédiction de la dimensionnalité : le nombre d'actions possibles croît exponentiellement avec le nombre de dimensions de l'action. Par exemple, si l'action a 3 dimensions et que chaque dimension est discrétisée en 10 intervalles, la dimension de sortie de la fonction Q est de $10^3 = 1000$, et plus la précision nécessaire est grande, plus il faut d'intervalles. Pour contourner cette malédiction de la dimensionnalité, *Policy Gradient* est une branche des algorithmes *RL* qui optimise directement la politique. Nous ne couvrons ici ni les algorithmes génétiques ni les algorithmes évolutionnaires, mais seulement le gradient de politique qui apprend la politique $p\left(action|état\right)$. Contrairement aux algorithmes *Q-learning* ou l'itération de valeur qui estiment d'abord la qualité d'un état, puis construisent une politique qui mène à des états de qualité, la politique est directement calculée. Les méthodes de *Policy Gradient* présentent différents avantages : tout d'abord, elles sont efficaces dans les espaces d'actions de haute dimension ou continues, elles ont de bonnes propriétés de convergence théorique, et elles peuvent traiter les politiques stochastiques, comme dans les jeux de *Poker* ou *Pierre-Papier-Ciseaux*. Dans les méthodes de Policy *Gradient*, la politique est directement paramétrée, avec un vecteur de paramètres θ (*classiquement les poids et les biais d'un réseau de neurones*) et prend un état s en entrée. Dans le cas d'un espace d'actions discrètes, la sortie du réseau de politique sera un vecteur de probabilités sur toutes les actions possibles, et dans le cas d'un espace d'actions continues, la sortie classique du réseau de politique est la moyenne (*et potentiellement la variance*) d'une distribution Gaussienne.

L'algorithme *REINFORCE* est le premier algorithme de gradient de politique détaillé dans « *Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning* » [132]. Cependant, depuis son développement en 2012, il y a eu de nombreuses améliorations pour rendre cet algorithme plus efficace. Cette section est inspirée des cours de John Schulman [133] et David Silver [134] que nous recommandons au lecteur. L'idée de

base de *REINFORCE* (*voir l'Algorithme (3.4)*) est de maximiser la somme attendue des récompenses sur une trajectoire $\tau : \mathbb{E}_{\pi}\left[R(\tau)\right]$ en utilisant l'ascension de gradient sur le paramètre de politique θ (*ascension de gradient car nous voulons maximiser la récompense, et non minimiser un coût*) :

$$\theta = \theta + \nabla_{\theta}\mathbb{E}_{\pi}\left[R(\tau)\right] \tag{42}$$

En utilisant la notation $J(\theta) = \mathbb{E}_{\pi}\left[R(\tau)\right]$, l'équation précédente devient :

$$\theta = \theta + \nabla_{\theta}J(\theta) \tag{43}$$

Le gradient $\nabla_{\theta}J(\theta)$ peut être dérivé comme suit :

$$\begin{aligned}
\nabla_{\theta}J(\theta) &= \nabla_{\theta}\mathbb{E}_{\pi}\left[R(\tau)\right] \\
&= \nabla_{\theta}\int p(\tau|\theta)R(\tau)d\tau \\
&= \int \nabla_{\theta}p(\tau|\theta)R(\tau)d\tau \\
&= \int p(\tau|\theta)\frac{\nabla_{\theta}p(\tau|\theta)}{p(\tau|\theta)}R(\tau)d\tau \\
&= \int p(\tau|\theta)\nabla_{\theta}\log p(\tau|\theta)R(\tau)d\tau \\
&= \mathbb{E}_{\pi}\left[\nabla_{\theta}\log p(\tau|\theta)R(\tau)\right]
\end{aligned} \tag{44}$$

En utilisant le fait que le gradient du logarithme de la probabilité de $p(\tau|\theta)$ est :

$$p(\tau|\theta) = p(s_0)\prod_{t=0}^{T-1}\left[\pi(a_t|s_t,\theta)P(s_{t+1},r_t|s_t,a_t)\right] \tag{45}$$

Où :

- $P(s_{t+1},r_t|s_t,a_t)$ est la probabilité de transition,
- $p(s_0)$ est la probabilité de l'état initial s_0.

Le gradient du logarithme de la probabilité de p est :

$$\log p(\tau|\theta) = \log p(s_0) + \sum_{t=0}^{T-1}\left[\log\pi(a_t|s_t,\theta) + \log P(s_{t+1},r_t|s_t,a_t)\right] \tag{46}$$

Lorsque nous dérivons par rapport à θ, comme $\log(s_0)$ et $\log P(s_{t+1}, r_t | s_t, a_t)$ ne dépendent pas de θ, nous obtenons :

$$\nabla_\theta \log p(\tau|\theta) = \nabla_\theta \sum_{t=0}^{T-1} \left[\log \pi(a_t | s_t, \theta) \right] \tag{47}$$

Donc :

$$\begin{aligned} \nabla_\theta J(\theta) &= \nabla_\theta \mathbb{E}_\pi \left[R(\tau) \right] \\ &= \mathbb{E}_\pi \left[R(\tau) \nabla_\theta \log p(\tau|\theta) \right] \\ &= \mathbb{E}_\pi \left[R(\tau) \nabla_\theta \sum_{t=0}^{T} \log \pi(a_t | s_t, \theta) \right] \\ &= \mathbb{E}_\pi \left[\left(\sum_{t=0}^{T} r_t(s_t, a_t) \right) \left(\nabla_\theta \sum_{t=0}^{T} \log \pi(a_t | s_t, \theta) \right) \right] \end{aligned} \tag{48}$$

Pour calculer une estimation de la valeur attendue, N trajectoires sont échantillonnées, et la valeur attendue est la moyenne de ces trajectoires.

Algorithme 3.4 : *REINFORCE*

1. *Initialiser* $\boldsymbol{\theta}$ *;*
2. ***Tant que*** *non terminé* ***faire*** *:*
   3. *Echantillonner* $\{\tau^i\}$ *à partir de (exécuter la politique pour échantillonner des trajectoires) ;*
   4. $\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_i^N \left[\left(\sum_{t=0}^{T} r(s_t^i, a_t^i) \right) \left(\nabla_\theta \sum_{t=0}^{T} \log \pi(a_t^i | s_t^i, \theta) \right) \right]$ *;*
   5. $\theta \leftarrow \theta - \alpha \nabla_\theta J(\theta)$ *;*
6. ***Fin tant que*** *;*

Le principe clé du gradient de politique est de donner une forte probabilité aux actions lorsque les récompenses sont élevées. Le problème de l'Equation (48) est que toutes les actions dans une trajectoire sont traitées de manière égale, même si certaines d'entre elles sont mauvaises et d'autres bonnes. De plus, le gradient de politique a une forte variance et une convergence lente.

La première astuce pour améliorer son fonctionnement est la causalité, ce qui signifie que la politique au temps t ne peut affecter les récompenses qu'au temps t' *si* $t' > t$.

$$\begin{aligned}\nabla_\theta J(\theta) &\approx \frac{1}{N}\sum_i^N\left(\nabla_\theta \sum_{t=0}^T \log \pi\left(a_t^i \middle| s_t^i, \theta\right)\right)\left[\left(\sum_{t=0}^T r\left(s_t^i, a_t^i\right)\right)\right] \\ &\approx \frac{1}{N}\sum_i^N\sum_{t=0}^T\left(\nabla_\theta \log \pi\left(a_t^i \middle| s_t^i, \theta\right)\right)\left[\left(\sum_{t'=0}^T r\left(s_{t'}^i, a_{t'}^i\right)\right)\right] \\ &\approx \frac{1}{N}\sum_i^N\sum_{t=0}^T\left(\nabla_\theta \log \pi\left(a_t^i \middle| s_t^i, \theta\right)\right)\left[\left(\sum_{t'=t}^T r\left(s_{t'}^i, a_{t'}^i\right)\right)\right]\end{aligned} \tag{49}$$

Avec cette nouvelle formule, les actions sont renforcées proportionnellement aux récompenses futures. Le deuxième truc pour réduire la variance est d'ajouter une *baseline* car nous ne renforçons que les actions qui sont meilleures que la moyenne. Une *baseline* très courante est la somme attendue des récompenses.

$$b = \frac{1}{N}\sum_{i=0}^N r\left(\tau^i\right) \tag{50}$$

Nous pouvons ajouter une référence car :

$$\begin{aligned}\mathbb{E}_\pi\left[\nabla_\theta \log \pi_\theta(\tau) b\right] &= \int \pi_\theta(\tau) \nabla_\theta \log \pi_\theta(\tau) b \, d\tau \\ &= \int \nabla_\theta \pi_\theta(\tau) b \, d\tau \\ &= b \nabla_\theta \int \pi_\theta(\tau) \, d\tau \\ &= b \nabla_\theta 1 \\ &= 0\end{aligned} \tag{51}$$

Le gradient devient alors :

$$\nabla_\theta J(\theta) = \frac{1}{N}\sum_i^N\sum_{t=0}^T\left(\nabla_\theta \log \pi\left(a_t^i \middle| s_t^i, \theta\right)\right)\left[\left(\sum_{t'=t}^T r\left(s_{t'}^i, a_{t'}^i - b\right)\right)\right] \tag{52}$$

La politique doit maintenant être définie pour que l'algorithme soit complet. Si l'espace d'action est fini, alors la politique peut être représentée par un réseau de neurones qui produit un vecteur de probabilité pour les différentes actions possibles. Dans le cas d'un espace d'action continu, la politique peut être représentée par un réseau de neurones qui produit la moyenne $\mu(s)$ (*et éventuellement la variance* $\sigma^2(s)$*, qui peut également être fixe*) d'une distribution gaussienne. L'action *a* sera échantillonnée à partir de cette distribution : $a \sim N\left(\mu(s), \sigma^2(s)\right)$. Voir la Figure (3.11) pour une illustration de l'échantillonnage de trajectoire dans le cas d'un espace d'action continu.

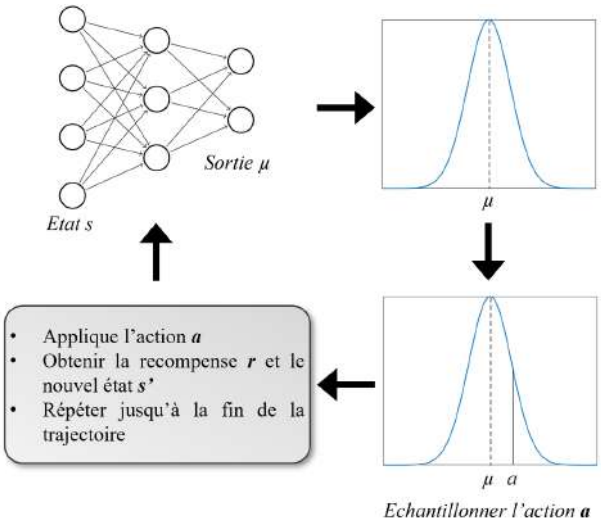


Figure 3.11 : *Echantillonnage de trajectoire.*

3.4.3.3. Acteur-Critique

Cette section est inspirée du cours de Sergey Levine [135]. L'Acteur-Critique combine le Gradient de Politique et l'apprentissage *Q*. A la fin de l'algorithme du Gradient de Politique (*sans prendre en compte la baseline b*), nous avons :

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_i^N \sum_{t=0}^T \left(\nabla_\theta \log \pi\left(a_t^i \middle| s_t^i, \theta\right)\right)\left[\left(\sum_{t'=t}^T r\left(s_{t'}^i, a_{t'}^i\right)\right)\right] \tag{53}$$

L'idée de l'équation ci-dessus est de suivre la direction où les bonnes récompenses se trouvent. Cependant, $\sum_{t'=t}^T r\left(s_{t'}^i, a_{t'}^i\right)$ est une estimation sur un seul échantillon des récompenses futures à partir de l'état s_t en prenant l'action a_t. Comme il s'agit d'une estimation sur un seul échantillon, il y a une forte variance dans le gradient de politique, et réduire cette variance conduira à une convergence meilleure et/ou plus rapide. Il serait plus efficace d'estimer les récompenses futures à partir de l'état s_t en prenant l'action a_t avec une meilleure précision, et un bon estimateur est la valeur *Q* telle que $Q_\pi(s,a) = \mathbb{E}_\pi\left[R_t \middle| S_t = s, A_t = a\right]$. L'Equation (53) devient :

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_i^N \sum_{t=0}^T \left(\nabla_\theta \log \pi\left(a_t^i \middle| s_t^i, \theta\right)\right) Q\left(s_t^i, a_t^i\right) \tag{54}$$

De plus, une référence est ajoutée pour réduire le biais. La somme des récompenses $b = \frac{1}{N}\sum_{i=0}^{N} r\left(\tau^i\right)$ est une référence communément utilisée. Mais dans ce cas, une moyenne plus précise peut être calculée : $b = \frac{1}{N}\sum_{i}^{N} Q\left(s_t^i, a_t^i\right) \approx \mathbb{E}_{a_t \sim \pi_\theta(a_t|s_t)}\left[Q\left(s_t, a_t\right)\right] = V\left(s_t\right)$. Le gradient devient alors :

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i}^{N}\sum_{t=0}^{T}\left(\nabla_\theta \log \pi\left(a_t^i \middle| s_t^i, \theta\right)\right)\left[Q\left(s_t^i, a_t^i\right) - V\left(s_t^i\right)\right] \tag{55}$$

Nous avons défini plus tôt l'avantage $A(s,a) = Q(s,a) - V(s)$ qui indique dans quelle mesure l'action a est meilleure que l'action moyenne.

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i}^{N}\sum_{t=0}^{T}\left(\nabla_\theta \log \pi\left(a_t^i \middle| s_t^i, \theta\right)\right)\left[A\left(s_t^i, a_t^i\right)\right] \tag{56}$$

Pour calculer l'avantage, l'approximation suivante est utilisée :

$$Q\left(s_t, a_t\right) \approx r\left(s_t, a_t\right) + \gamma V\left(s_{t+1}\right) \tag{57}$$

Qui mène à :

$$A\left(s_t, a_t\right) = Q\left(s_t, a_t\right) - V\left(s_t\right) = r\left(s_t, a_t\right) + \gamma V\left(s_{t+1}\right) - V\left(s_t\right) \tag{58}$$

Un exemple d'Acteur-Critique est décrit dans l'Algorithme (3.5).

En résumé, les algorithmes d'acteur-critique comportent deux composantes, un acteur, la politique π, avec le paramètre θ, et un critique (*Q, V ou A en fonction de l'algorithme*) avec le paramètre ϕ qui estime la qualité de la sortie de l'acteur. Dans la suite, nous présenterons les différents algorithmes d'Acteur-Critique.

Algorithme 3.5 : *Acteur-Critique*

1. *Initialiser* $\boldsymbol{\theta}$ *et* $\boldsymbol{\phi}$*;*
2. ***Tant que*** *non terminé* ***faire*** *:*
   3. *Echantillonner* $\left\{\tau^i\right\}$ *à partir de (exécuter la politique pour échantillonner des trajectoires) ;*
   4. *Ajuster* $V_\phi^\pi\left(s\right)$ *aux sommes de récompenses échantillonnées ;*
   5. $A^\pi\left(s_i, a_i\right) = r\left(s_i, a_i\right) + \gamma V_\phi^\pi\left(s_i'\right) - V_\phi^\pi\left(s_i\right)$
   6. $\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i}^{N} \nabla_\theta \log \pi_\theta\left(a_i \middle| s_i\right) A^\pi\left(s_i, a_i\right)$
   7. $\theta \leftarrow \theta - \alpha \nabla_\theta J(\theta)$ *;*
8. ***Fin tant que*** *;*

3.4.3.3.1. Gradient de politique déterministe : (D)DPG, RDPG, D4PG)

Les algorithmes de gradient de politique classiques traitent des politiques stochastiques. Cependant, dans les algorithmes de gradient de politique déterministe [136], Silver et *al*. ont construit un algorithme acteur-critique modifié pour considérer les politiques déterministes. Le gradient de politique déterministe est calculé plus efficacement car Q est dérivé uniquement par rapport aux actions. Les auteurs présentent un algorithme *off-policy* qui apprend une politique déterministe en utilisant une politique stochastique pour l'exploration.

Algorithme 3.6 : *DDPG [137]*

1. *Initialiser le réseau de critique* $Q(s,a|\phi)$ *et l'acteur* $\mu(s,\theta)$ *avec les poids* $\boldsymbol{\phi}$ *et* $\boldsymbol{\theta}$ *;*
2. *Initialiser le réseau de critique* Q' *et l'acteur* μ' *avec les poids* $\phi' \leftarrow \phi$ *et* $\theta' \leftarrow \theta$ *;*
3. *Initialiser le tampon de relecture* ***R*** *;*
4. ***Pour*** *chaque épisode = 1, M* ***faire*** *:*
   5. *Initialiser un processus aléatoire* ***N*** *pour l'exploration d'actions ;*
   6. *Recevoir une observation de l'état initial* $\boldsymbol{s_1}$ *;*
   7. ***Pour*** *chaque étape t = 1, T* ***faire*** *:*
      8. *Sélectionner l'action* $a_t = \mu(s_t|\theta) + N_t$ *selon la politique actuelle et le bruit d'exploration ;*
      9. *Exécuter l'action* $\boldsymbol{a_t}$ *et observer la récompense* $\boldsymbol{r_t}$ *et le nouvel état* $\boldsymbol{s_{t+1}}$ *;*
      10. *Stocker la transition* (s_t, a_t, r_t, s_{t+1}) *dans R ;*
      11. *Echantillonner un mini-lot aléatoire de N transitions* (s_i, a_i, r_i, s_{i+1}) *;*
      12. *Définir* $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta')|\phi')$ *;*
      13. *Mettre à jour le critique en minimisant le coût* $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\phi))$ *;*
      14. *Mettre à jour la politique de l'acteur en utilisant le gradient de politique échantillonné :*
          $\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_i \nabla_a Q(s,a|\phi)\big|_{s=s_i, a=\mu(s_i)} \nabla_\theta \mu(s|\theta)\big|_{s_i}$ *;*
      15. *Mettre à jour les réseaux cibles :*
          - $\phi' \leftarrow \tau\phi + (1-\tau)\phi'$
          - $\theta' \leftarrow \tau\theta + (1-\tau)\theta'$
   16. ***Fin Pour*** *;*
17. ***Fin Pour*** *;*

Comme une extension de *DPG* [136], Lillicrap et *al*. présentent dans « *Continuous Control with Deep Reinforcement Learning* » [137] l'algorithme *DDPG* (*Deep Deterministic Policy Gradient*), qui est une combinaison de *DPG* et *DQN* [123]. L'algorithme est basé sur l'acteur-critique pour la politique déterministe comme *DPG*, mais utilise les techniques *Deep Q Learning* pour estimer le critique (*valeurs Q*) : répétition de l'expérience et un réseau cible.

De plus, le réseau cible est lentement mis à jour : $\theta' \leftarrow \tau\theta + (1-\tau)\theta'$ avec $\tau \ll 1$. Le résultat de *DDPG* est une accélération remarquable sur la plupart des jeux Atari qui sont résolus en 20 fois moins d'étapes que *DQN*. Les études [138, 139] ont montré que *DDPG* était cependant très sensible aux hyperparamètres et moins stable que les algorithmes en batch (*DDPG met à jour la politique à chaque étape, alors que de nombreux algorithmes acteur-critique échantillonnent un lot de trajectoires avant la mise à jour*). L'implémentation de *DDPG* est présentée dans l'Algorithme (3.6).

Il existe une version récurrente de *DPG* appelée *RDPG* (*Recurrent Deterministic Policy Gradient*) présentée dans « *Memory-based control with recurrent neural networks* » [140] qui utilise également la répétition d'expérience et des réseaux cibles. *RDPG* est particulièrement efficace dans les environnements partiellement observables, où l'agent doit se souvenir de ce qui s'est passé dans les états précédents. Une autre extension de *DDPG* est *D4PG* : *Distributed Distributional Deterministic Policy Gradients* [141]. Les modifications consistent à utiliser plusieurs acteurs pour échantillonner des trajectoires, et à utiliser des mises à jour probabilistes de critique. Cela signifie que le retour est traité comme une variable aléatoire, et l'objectif est d'apprendre sa distribution au lieu du retour attendu. *D4PG* inclut également une mémoire de relecture d'expérience priorisée et un retour à *N* étapes, ce dernier produisant le meilleur gain de performance.

3.4.3.3.2. Apprentissage multi-threads : A3C, A2C, ACER

Nous avons précédemment vu l'algorithme *D4PG* qui propose d'utiliser plusieurs acteurs pour collecter des données. L'algorithme *A3C* proposé par *Google Deep Mind* dans « *Asynchronous Methods for Deep Reinforcement Learning »* [142] propose de paralléliser le calcul des gradients pour éviter l'utilisation d'un tampon de répétition. Au lieu de faire une propagation avec un lot d'épisodes, de plus petites rétropropagations sont effectuées en parallèle avec de plus petits lots. *A3C* (pour *Asynchronous Advantage Actor Critic*) est composé des éléments suivants :

- Un thread global,
- Plusieurs threads qui contiennent des réseaux locaux.

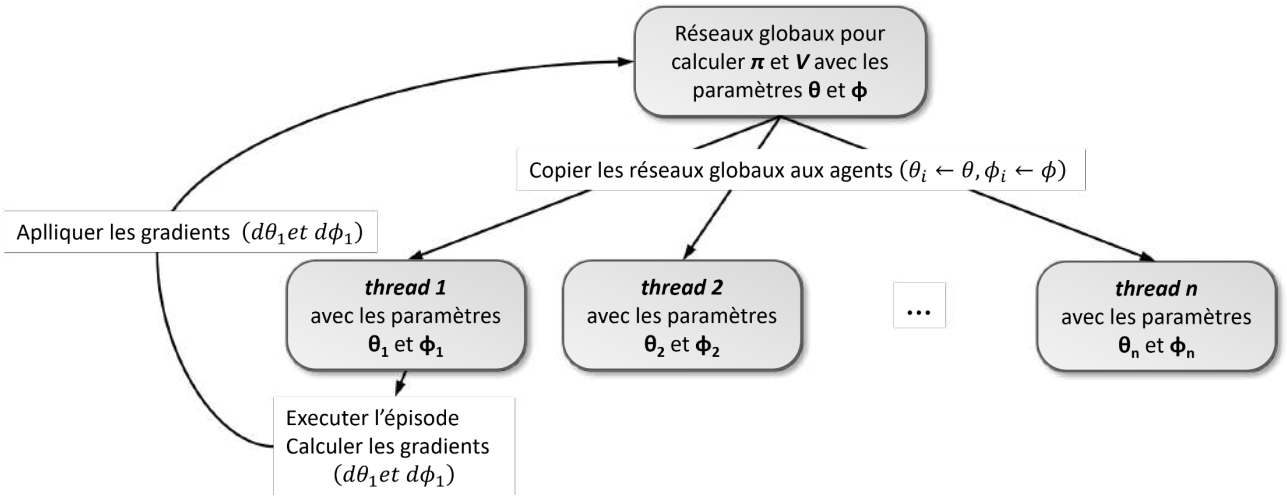


Figure 3.12 : *Algorithme de A3C.*

Le thread global contient des réseaux à jour. Chaque thread interagit avec sa propre instance de l'environnement. Chaque thread copie localement les réseaux globaux, exécute des épisodes et calcule les gradients correspondants. Ensuite, le réseau global est mis à jour avec les gradients locaux du thread, et le thread recommence, en copiant les réseaux globaux, en exécutant des épisodes, etc. Comme chaque thread le fait en parallèle, la mise à jour du réseau global est asynchrone. L'algorithme est un algorithme *acteur-critique*, donc deux gradients sont calculés, l'un pour la politique et l'autre pour la fonction de valeur, et le thread global et les threads locaux ont chacun deux réseaux (*ou un réseau partagé, selon l'architecture*), avec des paramètres θ et ϕ pour la politique et la valeur respectivement. Le schéma de la Figure (3.12) représente l'idée de l'algorithme *A3C*, et l'Algorithme (3.7) est le pseudo-code pour un agent. Pour des raisons de clarté, le parcours complet n'est montré que pour le premier thread.

L'efficacité de *A3C* a été démontrée sur les jeux Atari ainsi que sur la tâche de navigation en labyrinthe. Il existe différentes extensions du *A3C* :

- ***A2C.*** Le défaut de l'asynchronisme est que les threads travaillent potentiellement avec des versions anciennes de la politique. La version synchrone d'*A3C*, *A2C*, attend que tous les threads aient terminé avant de mettre à jour le thread global et de copier les poids à jour vers les threads. *A2C* fonctionne aussi bien, voire mieux, que *A3C*.
- ***ACER*** [143] (*Actor-Critic with Experience Replay*). C'est une extension hors politique de *A3C*, avec un tampon de lecture pour une efficacité d'échantillonnage accrue.

• ***Impala*** [144]. Contrairement à *A3C*, les acteurs ne calculent pas le gradient, mais envoient la transition complète au thread global qui effectue la mise à jour. L'acte est complètement séparé de l'apprentissage - qui peut également être divisée en plusieurs apprenants.

Algorithme 3.7 : *Algorithme A3C [145]*

1. *// On suppose que les paramètres partagés globaux* $\boldsymbol{\phi}$ *et* $\boldsymbol{\theta}$ *et le compteur partagé global T sont initialisés à 0.*
2. *// On suppose que les vecteurs de paramètres spécifiques à l'agent* $\boldsymbol{\phi_i}$ *et* $\boldsymbol{\theta_i}$ *sont initialisés.*
3. *Initialiser le compteur d'étapes de l'agent* $\boldsymbol{t}$ *à 1 ;*
4. ***Répéter :***
  5. *Réinitialiser le gradient :* $d\theta_i \leftarrow 0$ *et* $d\phi_i \leftarrow 0$ *;*
  6. *Synchroniser les paramètres spécifiques à l'agent* $\theta_i = \theta$ *et* $\phi_i = \phi$ *;*
  7. $t_{start} = t$ *;*
  8. *Obtenir l'état* s_t *;*
  9. ***Répéter*** *:*
    10. *Effectuer l'action* a_t *en fonction de la politique* $\pi_{\theta_i}\left(a_t \middle| s_t\right)$ *;*
    11. *Recevoir la récompense* r_t *et le nouvel état* s_{t+1} *;*
    12. $t \leftarrow t+1$ *;*
    13. $T \leftarrow T+1$ *;*
  14. Jusqu'à ce que *l'épisode soit terminé ;*
  15. $R = \begin{cases} 0 & \text{pour l'état final } s_t \\ V_{\phi_i}\left(s_t\right) & \text{si non} \end{cases}$
  16. Pour *chaque étape* $k = t-1,\ t_{start}$ *faire :*
    17. $R \leftarrow r_k + \gamma R$ *;*
    18. $d\theta_i \leftarrow d\theta_i + \nabla_{\theta_i} \log \pi_{\theta_i}\left(a_k \middle| s_k\right)\left(R - V_{\phi_i}\left(s_k\right)\right)$ *;*
    19. $d\phi_i \leftarrow d\phi_i + \partial\left(R - V_{\phi_i}\left(s_k\right)\right)^2 \Big/ \partial\phi_i$ *;*
  20. Fin Pour *;*
  21. *Effectuer une mise à jour asynchrone de* $\boldsymbol{\theta}$ *en utilisant* $d\theta_i$ *et de* $\boldsymbol{\phi}$ *en utilisant* $d\phi_i$ *;*
22. Jusqu'à ce que $T > T_{max}$ *;*

3.4.3.3.3. Algorithmes de région de confiance : TRPO, ACKTR, PPO

Les algorithmes de gradient de politique permettent à l'agent d'effectuer des actions dans un espace continu, cependant, la taille de pas α doit être choisie avec soin : si elle est trop petite, l'apprentissage sera très lent, et si elle est trop grande, elle sera submergée par le bruit, car le lot suivant sera collecté sous une mauvaise politique. Dans « *Trust Region Policy Optimization* » [146], Schulman et *al.* proposent un nouvel algorithme pour améliorer de

manière itérative la politique. A chaque mise à jour, la politique est modifiée pour améliorer les performances tout en restant proche de la politique précédente. L'objectif réel est de maximiser la fonction $\eta(\theta)=\mathbb{E}_{\pi}\left[\gamma^{t} r(s_t)\right]$, mais cet objectif est difficile à calculer, c'est pourquoi les auteurs introduisent une fonction de substitution

$$L(\theta)=\mathbb{E}\left[\frac{\pi_{\theta}(a_t|s_t)}{\pi_{\theta_{old}}(a_t|st_t)}A_t\right] \tag{59}$$

qui est une approximation locale de l'objectif réel et dont les dérivées de premier ordre sont égales.

$$\begin{aligned}\nabla_{\theta}L(\pi_{\theta})\big|_{\theta_{old}} &= \mathbb{E}_{s,a\sim\pi_{old}}\left[\frac{\nabla_{\theta}\pi_{\theta}(a|s)}{\pi_{\theta_{old}}(a|s)}A^{\pi_{old}}(s,a)\right]|\theta_{old} \\ &= \mathbb{E}_{s,a\sim\pi_{old}}\left[\nabla_{\theta}\log\pi_{\theta}(a|s)A^{\pi_{old}}(s,a)\right]|\theta_{old}\end{aligned} \tag{60}$$

Nous obtenons la formule de la dérivée du gradient telle que présentée dans l'Equation (56), donc au premier ordre, lorsque θ et θ_{old} sont proches :

$$\nabla_{\theta}L(\pi_{\theta})\big|_{\theta_{old}}=\nabla_{\theta}\eta(\theta)\big|_{\theta=\theta_{old}} \tag{61}$$

Cette fonction de substitution est précise uniquement lorsque π et π_{old} sont proches, de sorte que les deux distributions sont contraintes avec la divergence de *Kullback-Leibler*. Par conséquent, l'objectif de l'algorithme *TRPO* est :

$$\begin{cases}\max\limits_{\theta}\sum_{n}^{N}\dfrac{\pi_{\theta}(a_n|s_n)}{\pi_{\theta_{old}}(a_n|s_n)}A_n \\ \qquad tel\ que\ \overline{KL}[\pi_{old},\pi]<\delta\end{cases} \tag{62}$$

ACKTR (*Actor-Critic using Kronecker-Factored Trust Region*) [147] est une variante de l'algorithme *TRPO* qui utilise une courbure approximative factorisée de *Kronecker* au lieu de la divergence de *Kullback-Leibler* pour l'optimisation de la région de confiance.

Une équipe de *OpenAI* a proposé un algorithme de *PPG* (*Proximal Policy Optimization*) [148] qui modifie l'algorithme *TRPO* pour une version plus simple. Les auteurs suggèrent deux fonctions de substitution différentes qui contraignent la nouvelle politique π à être proche de l'ancienne politique π_{old}. La première est naturellement définie à partir de

l'Equation (62) : au lieu de la contrainte séparée, la pénalité *KL* est directement ajoutée dans la fonction de substitution L^{KLPEN}.

$$L^{KLPEN}(\theta) = \mathbb{E}_t\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|st_t)}A_t - \beta KL\left[\pi_{\theta_{old}}, \pi_\theta\right]\right] \tag{63}$$

Le paramètre β est mis à jour comme suit :

$$\begin{aligned} &Si \quad d < \frac{d_{\text{targ}}}{1.5}: \quad \beta \leftarrow \frac{\beta}{2} \\ &Si \quad d < d_{\text{targ}} \times 1.5: \quad \beta \leftarrow \beta \times 1.5 \end{aligned} \tag{64}$$

Avec : $d = \mathbb{E}_t\left[KL\left[\pi_{\theta_{old}}, \pi_\theta\right]\right]$

Cependant, cette fonction de substitution peut être remplacée par une fonction plus stable, une fonction de restriction appelée L^{CLIP}. Définissons $r_t(\theta)$ comme étant le rapport de probabilité $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|st_t)}A_t$ (et A_t est toujours l'avantage l'instant t). La fonction de restriction L^{CLIP} est la suivante :

$$L^{CLIP}(\theta) = \mathbb{E}_t\left[\min\left(r_t(\theta)A_t, clip\left(s_t(\theta)\right), 1-\varepsilon, 1+\varepsilon\right)A_t\right] \tag{65}$$

Cette fonction de contrainte permettra de maintenir le $r_t(\theta)$ dans l'intervalle $[1-\varepsilon, 1+\varepsilon]$, où ε est un petit nombre.

Cet algorithme utilise une méthode *Acteur-Critique*, donc la fonction de valeur V est également calculée. Si la politique et la fonction de valeur partagent les mêmes paramètres de réseau de neurones, la fonction coût est alors définie comme suit :

$$L(\theta) = \mathbb{E}_t\left[L_t^{PG}(\theta) + c_1 L_t^{VF}(\theta) - c_2 S\left[\pi_\theta(s_t)\right]\right] \tag{66}$$

Dans cette formule,

- L^{PG} peut être soit L^{CLIP} soit L^{KLPEN},
- L^{VF} est une fonction coût de valeur en erreur quadratique,
- S est une prime d'entropie pour encourager l'exploration.

Ces algorithmes sont comparés les uns aux autres en utilisant le benchmark de jeux Atari, et sur des tâches de robotique en utilisant le simulateur *Mujoco* [128], où les performances de l'algorithme *PPO* dépassent largement celles de *A2C* et *TRPO*. Sur les jeux Atari, les performances de *PPO* sont comparables à celles d'*ACER*, où il le dépasse dans le jeu *Enduro*. L'algorithme *PPO* présente également l'avantage d'être plus simple que *TRPO* et *ACER*, et de performer au moins aussi bien. Il permet également au réseau de valeurs et de politiques de partager des paramètres, ce qui n'est pas le cas dans *TRPO* en raison de la contrainte sur la politique.

3.4.3.3.4. L'entropie avec l'apprentissage par renforcement : SQL, SAC

Un problème classique en apprentissage par renforcement est que les agents deviennent très spécialisés, car ils ont été entraînés à résoudre une tâche très spécifique de la meilleure façon possible. Plus précisément, les agents n'apprennent qu'une seule façon de résoudre la tâche. Une façon de résoudre ce problème et de forcer l'agent à explorer et à apprendre toutes les façons possibles de résoudre la tâche est d'ajouter de l'entropie et de maximiser à la fois la somme des récompenses et l'entropie, comme présenté dans l'article « *Reinforcement Learning with Deep Energy-Based Policies »* [149]. La politique optimale devient :

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\pi}\left[\sum_t r_t + H\left(\pi\left(\bullet, s_t\right)\right)\right] \tag{67}$$

Avec cette méthode, appelée *Soft Q-Learning* (*SQL*), l'agent apprendra à la fois toutes les méthodes possibles de résoudre une tâche donnée, même s'il préférera une méthode optimale, et sera en mesure de passer rapidement à une méthode lorsque la méthode optimale sera indisponible.

Soft Q-Learning est le prédécesseur de *Soft Actor-Critic* (*SAC*) présenté dans l'article « *Soft Actor-Critic: Off-Policy Maximum entropy deep reinforcement learning with a stochastic actor »* [150]. SAC est la version acteur-critique de *SQL*, combinant des mises à jour hors politique et une maximisation de l'entropie.

Dans un travail ultérieur [150], *SAC* est étendu pour ajouter un réglage automatique des hyperparamètres, ce qui permet un apprentissage plus stable. De nombreux algorithmes (*A3C,*

PPO...) incluent désormais l'entropie dans leur fonction de substitution, pour une meilleure exploration et une robustesse améliorée [151].

3.5. Conclusion

Dans ce chapitre, nous avons présenté un aperçu des réseaux de neurones profonds, des algorithmes d'apprentissage supervisé et d'apprentissage par renforcement, en mettant l'accent sur l'apprentissage par renforcement profond sans modèle. Dans certains problèmes de décision séquentielle (*comme le jeu de Go*), l'apprentissage par renforcement a surpassé l'apprentissage supervisé, et l'apprentissage par renforcement ne nécessite pas non plus de données annotées. Cette thèse se concentre donc sur l'apprentissage par renforcement pour résoudre quelques problèmes de la navigation autonome d'un robot mobile. Dans le prochain chapitre, nous présentons un état de l'art de la robotique mobile et nous y présenterons les différentes méthodes de la navigation autonomes.

4. Planification de Chemins

4.1. Introduction

Ces dernières années, le *DRL* est devenu un outil puissant dont l'utilisation est très courante pour la navigation sans carte pour les robots mobiles navigant dans des environnements d'intérieur ou même d'extérieur [152]. L'élaboration d'algorithmes de planification de chemins basés sur le *DRL* offre de multiples avantages, notamment son autonomie par rapport à un modèle global (*carte*) préalable de l'environnement, d'un grand pouvoir de généralisation et d'une bonne adaptabilité aux différentes situations de navigation, etc…. Cependant, l'inconvénient majeur est que l'apprentissage pour la planification de chemins avec *DRL* prend généralement un temps excessivement long, ce qui limite sa large utilisation, en particulier dans des environnements où les ressources de calcul sont limitées. Dans la plupart des cas, les algorithmes de planification de chemins basés sur le *DRL* doivent être entrainés dans des simulateurs robotiques munis d'un moteur graphique *3D* et d'un moteur de calcul physique pour des résultats réalistes, où les mouvements du robot sont régis par des règles physiques. Le nombre requis d'interactions entre l'agent (*le robot mobile*) et l'environnement pour apprendre à planifier son chemin avec succès est très grand, ce qui augmente considérablement la durée de l'apprentissage [153]. Lorsque les environnements d'apprentissage sont complexes, il est difficile pour l'algorithme de garantir la convergence vers les objectifs souhaités avec une initialisation aléatoire des paramètres du réseau de neurones. Par exemple, les poids et les biais [154] peuvent encore détériorer les performances de la planification de chemin par *DRL*. De plus, étant donné que les actions sont généralement basées sur les états d'observation locaux et sont prises à chaque étape dans la plupart des algorithmes *DRL*, cela peut limiter la capacité du robot mobile à prendre des décisions globales satisfaisantes.

Ce chapitre vise à aborder les questions ci-dessus, et les principales contributions sont résumées comme suit :

- Nous proposons d'utiliser la méthode d'apprentissage progressif qui consiste à transférer les paramètres de l'agent *DRL* entre les environnements d'apprentissage

pour améliorer l'efficacité du développement et la convergence de l'algorithme de planification de chemins basée sur le *DRL* pour un robot mobile. Cette méthode peut également simplifier le processus de test et d'évaluation des algorithmes ainsi que l'architecture des réseaux de neurones utilisés, ce qui permet de perfectionner de manière pratique l'algorithme de planification.

- Nous combinerons l'algorithme *TD3* d'apprentissage par renforcement avec l'algorithme *PRM* de planification classique de chemins afin d'améliorer les performances de planification, et nous comparerons les résultats avec les algorithmes populaires pour une navigation intérieure de robot mobile, comme *A*+DWA* et *A*+TEB*.
- Nous utiliserons l'algorithme *TD3* pour apprendre au robot mobile d'effectuer deux tâches : la détection d'obstacles et la navigation locale.
- Nous évaluerons et analyserons de manière approfondie les performances de *PRM+TD3* sur différentes scènes d'application, par rapport à *A*+DWA*, *A*+TEB* et *TD3* selon le même environnement et les mêmes critères.

4.2. L'algorithme PRM

L'algorithme *PRM*, ou *Probabilistic Roadmap Method* (*Méthode des Cartes de Chemin Probabilistes en français*), est une technique de planification de mouvement utilisée en robotique et en informatique pour résoudre des problèmes de planification de mouvement dans des environnements complexes. L'objectif principal de l'algorithme *PRM* est de trouver un chemin sûr et faisable pour un robot ou un objet mobile dans un espace encombré.

La planification d'un chemin par l'algorithme *PRM* passe par les étapes suivantes :

- ***Création de l'espace de configuration (C-space) :*** Tout d'abord, l'environnement est modélisé sous la forme d'un espace de configuration, également appelé C-space. Cet espace représente toutes les positions possibles du robot ou de l'objet dans l'environnement, en prenant en compte les obstacles et les contraintes.
- ***Echantillonnage de points aléatoires :*** Des points aléatoires sont échantillonnés dans l'espace de configuration. Ces points représentent des positions potentielles du robot. Plus il y a de points échantillonnés, plus la solution trouvée sera précise, mais cela peut également augmenter la complexité de l'algorithme.

- ***Validation des points échantillonnés :*** Chaque point échantillonné est vérifié pour s'assurer qu'il est à l'extérieur des obstacles de l'environnement et qu'il respecte toutes les contraintes de mouvement.
- ***Construction du graphe probabiliste :*** Les points valides sont reliés entre eux pour former un graphe. Les arêtes du graphe représentent les chemins possibles entre les points. Les arêtes sont créées en tenant compte de la distance entre les points et en vérifiant que le chemin entre eux est réalisable.
- ***Recherche de chemin :*** Une fois que le graphe a été construit, on peut utiliser un algorithme de recherche de chemin, comme l'algorithme *A** ou *Dijkstra*, pour trouver un chemin entre le point de départ et le point d'arrivée dans le graphe probabiliste. Ce chemin est la solution au problème de planification de mouvement.
- ***Raffinement du chemin (facultatif) :*** Enfin, le chemin trouvé peut être raffiné pour s'assurer qu'il est sûr et praticable. Cela peut impliquer la vérification de collisions potentielles avec des obstacles en utilisant des techniques de détection de collision.

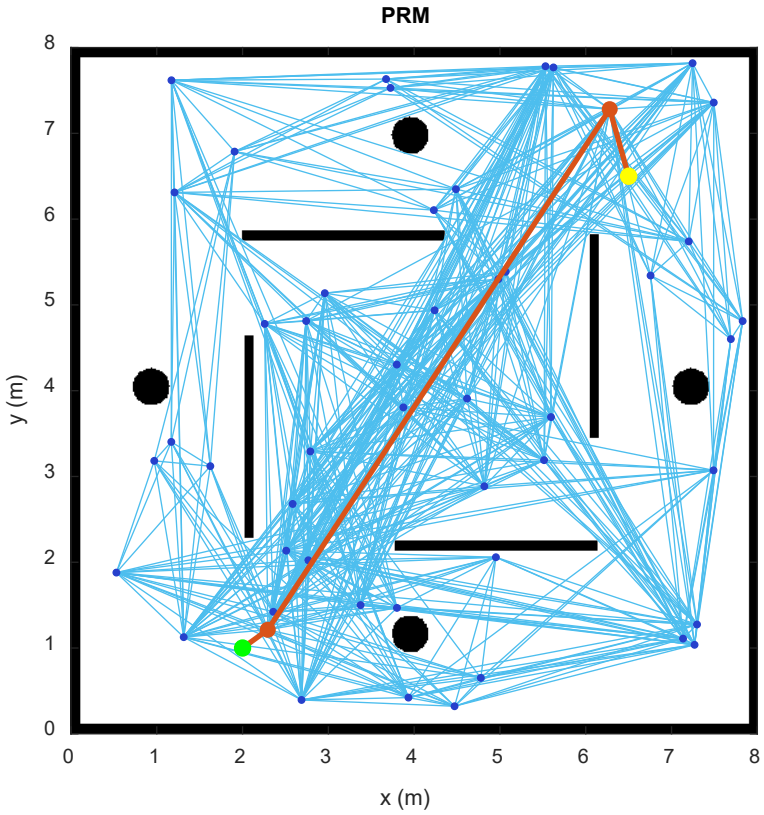


Figure 4.1 : *Exemple de planification de chemin par PRM.*

L'algorithme *PRM* est particulièrement utile pour résoudre des problèmes de planification de mouvement dans des environnements complexes où il y a de nombreux obstacles et contraintes. En échantillonnant de manière aléatoire et en construisant un graphe probabiliste, il permet de trouver des solutions de manière efficace tout en garantissant la faisabilité et la sécurité du chemin. La Figure (4.1) montre un exemple de planification par l'algorithme *PRM*. Les points de couleur verte, jaune et bleue représentent respectivement le point de départ, la cible et les points échantillonnés, tandis que la ligne rouge représente le chemin global.

4.3. Configuration du système de navigation

L'architecture du système de navigation globale est illustrée dans la Figure (4.2), il est composé des modules suivants :

- ***Le robot mobile et le LiDAR :*** recevant les consignes de vitesses de translation, et de rotation et fournissant les mesures odométriques et les données du *LiDAR*.
- ***Le module de localisation adaptative de Monte Carlo (AMCL) [155] :*** qui fusionne les données odométriques et celle du scan pour fournir la meilleure approximation possible de la position du robot dans son environnement (*supposant qu'une carte de l'environnement est disponible*).
- ***Le module PRM :*** ce module calcule un itinéraire global reliant le point de départ du robot à sa destination finale en se basant sur la carte globale préexistante de l'environnement et sur les données de localisation.
- ***L'agent TD3 :*** c'est l'agent d'apprentissage profond par renforcement qui apprend à détecter les obstacles et à analyser leur mouvement pour ajuster le chemin global afin d'assurer la sécurité du robot lors de son déplacement. L'agent *TD3* fournit la commande actuelle du robot (V^t_{robot}) composée des vitesses de translation et de rotation pour guider le robot le long de son trajet.

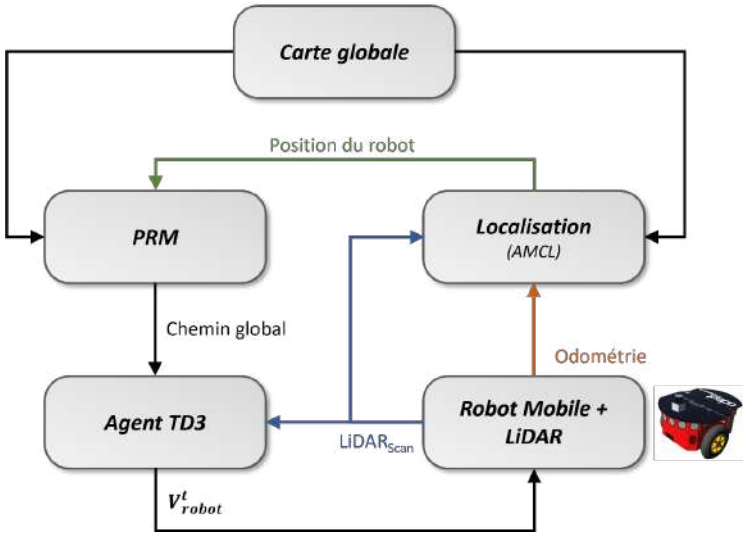


Figure 4.2 : *Architecture du système de navigation globale.*

Notre travail fait usage du robot mobile *Pioneer 3DX* équipé d'un *LiDAR* pour la perception de l'environnement, comme illustré à la Figure (4.3) ; il s'agit d'un robot de type différentiel capable de se déplacer à une vitesse linéaire maximale de 1 m/s et une vitesse de rotation maximale de ± 3 rad/s et une accélération linéaire maximale de 2.5 m/s^2 et une accélération angulaire maximale de 3.2 rad/s^2. Le robot est équipé d'un modèle simulé du *LiDAR Hokuyo URG 04LX* caractérisé par une distance de balayage variant entre 0.2 m et 4m et un angle de balayage dans l'intervalle [0º, 240º], à une résolution angulaire de 1° et un temps de balayage de 0.1 s.

Figure 4.3 : *Le robot mobile Pioneer 3DX.*

4.4. Configuration de l'agent TD3

L'algorithme *Twin-Delayed Deep Deterministic Policy Gradient* (*TD3*) est une méthode d'apprentissage par renforcement sans modèle, en ligne et hors politique. Un agent *TD3* est

un agent d'apprentissage par renforcement de type *acteur-critique* qui recherche une politique optimale maximisant la récompense cumulative à long terme attendue.

L'algorithme *TD3* est une extension de l'algorithme *DDPG*. Les agents *DDPG* peuvent surestimer les fonctions de valeur, ce qui peut produire des politiques *sous-optimales*. Pour réduire cette surestimation des fonctions de valeur, l'algorithme *TD3* inclut les modifications suivantes par rapport à l'algorithme *DDPG* :

- Un agent *TD3* apprend deux fonctions *Q-value* et utilise la valeur minimale de ces estimations de fonction lors des mises à jour de la politique.
- Un agent *TD3* met à jour la politique et les cibles moins fréquemment que les fonctions *Q*.
- Lors de la mise à jour de la politique, un agent *TD3* ajoute du bruit à l'action cible, ce qui rend la politique moins encline à exploiter des actions avec des estimations de *Q-value* élevées.

Les réseaux de neurones profonds (*DNN*) peuvent être utilisés pour l'approximation des fonctions non linéaires de valeur et de politique dans *DRL*. La structure du réseau de *TD3* [156],[157] adopté dans ce travail est illustré à la Figure (4.4), comprenant un réseau de la critique (*l'algorithme TD3 utilise deux réseaux pour le critique et la sortie de cette dernière est le minimum des sorties de ces deux réseaux, sur la Figure (4.4) est représenté un de ces deux réseaux*) et un réseau de l'acteur, dans lequel chaque couche est constituée du type de connexion (*entièrement connecté, FC : fully conncected*), les neurones et la fonction d'activation à l'exception de la couche d'entrée.

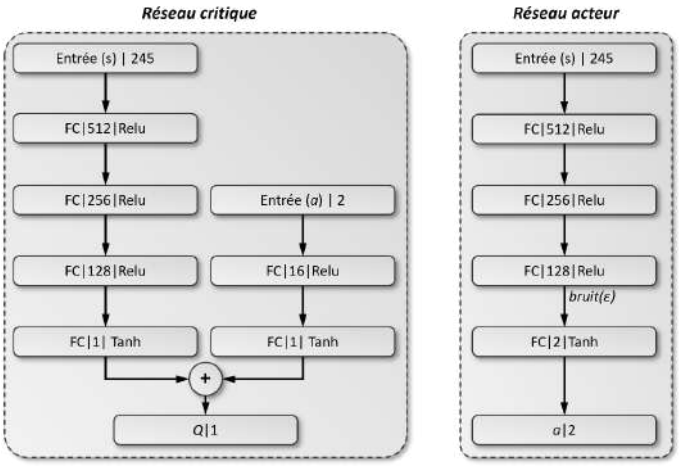


Figure 4.4 : *Les réseaux TD3.*

Pour l'implémentation de *TD3*, nous utilisons les fonctions fournies par les boites à outils apprentissage par renforcement et apprentissage profond.

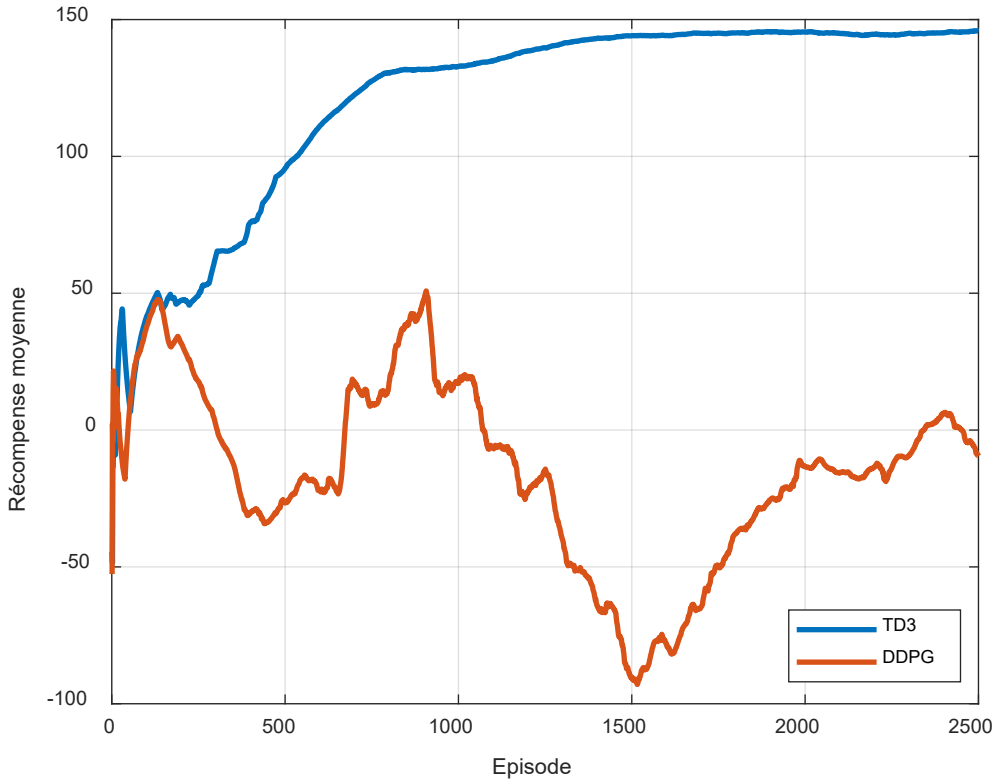


Figure 4.5 : *L'évaluation de la convergence des algorithmes RL.*

Afin de sélectionner l'algorithme optimal d'apprentissage par renforcement pour la planification de chemin, nous avons procédé à une comparaison de entre deux algorithmes populaires adaptés à l'espace d'état-action continu, à savoir le *DDPG* [137], et le *TD3* [156]. Cette évaluation a été effectuée au sein d'un environnement *2D*, avec la possibilité de pouvoir réutiliser facilement ces algorithmes dans un environnement *3D*. Les deux algorithmes ont été mis en œuvre à l'aide des fonctions des boîtes à outils *RL* et *DNN* de *Matlab*, et nous avons exécuté 2500 épisodes pour effectuer une comparaison exhaustive.

La Figure (4.5) montre les récompenses moyennes des deux algorithmes *RL*. En comparaison, *TD3* a une meilleure convergence, ce qui signifie que le robot mobile peut apprendre la capacité de planification de chemins pour atteindre la position cible sans collision beaucoup plus rapidement.

4.4.1. Définition de l'état

L'objectif d'apprentissage du modèle *DRL* est de maximiser la récompense totale *R* lors des interactions entre l'agent et l'environnement. A un pas de temps t donné, l'état s_t est composé :

- des données *LiDAR* d_m qui fournit un vecteur de 241 mesures de distance ordonnées selon l'angle de balayage,
- de la relation géométrique, c'est-à-dire le vecteur C contenant l'angle et de la distance entre l'agent et le point cible,
- de la vitesse linéaire v_{t-1} et vitesse angulaire ω_{t-1} au pas de temps précédent.

4.4.2. Définition des actions

Considérant l'agent *TD3* comme une fonction $f(*)$ qui fait correspondre les états d'entrée s_t au vecteur des signaux de commande actuelle du robot mobile. Ce vecteur est constitué par les vitesses de translation v_t et de rotation ω_t. L'Equation (68) décrit la fonction réalisée par l'agent de renforcement.

$$(v_t, \omega)_t = f(s_t) = f(d_m, C, v_{t-1}, \omega_{t-1}) \tag{68}$$

A chaque changement d'état d'observation s_t, l'environnement fournit une récompense instantanée r_t, qui mesure la qualité de la décision prise. La récompense est d'autant plus grande que la décision satisfait les critères de la tâche, et inversement. Comme l'environnement est inconnu à l'avance, la probabilité de transition d'un état à un autre n'est pas disponible, ce qui rend les algorithmes DRL sans modèle plus adaptés.

4.4.3. Définition de la fonction récompense

Dans ce paragraphe, nous élaborons la fonction de récompense qui supervise l'agent dans son apprentissage en vue d'obtenir la politique optimale. La récompense instantanée constitue un élément central du *DRL*, étant attribuée par la fonction de récompense à chaque étape temporelle. Ainsi, le modèle peut apprendre les tâches spécifiées, pour autant que la fonction de récompense soit judicieusement conçue.

Pour la planification du chemin du robot, l'objectif d'apprentissage est qu'un robot se déplace rapidement d'un point de départ à un point final sans collision. La fonction de récompense est définie dans l'Equation (69). En outre, il est à mentionner que nous avons ajouté une distance de sécurité pour assurer un meilleur évitement des obstacles.

$$r(s_t, a_t) = \begin{cases} 200 & \text{si cible atteinte} \\ -150 & \text{si collision} \\ -0.001 & \text{ailleurs} \end{cases} \tag{69}$$

L'agent *TD3* reçoit une récompense positive d'une valeur de 200 pour atteindre sa destination. Il est pénalisé (*une pénalité de* -150) s'il heurte un obstacle. Et à chaque pas de temps il reçoit une petite pénalité d'une valeur de -0.001 qui sert à minimiser le temps de navigation.

4.5. Configuration de l'environnement DRL

Généralement, pour un robot, il est courant de procéder à l'apprentissage préalable de son modèle *DRL* dans un environnement de simulation doté d'un moteur physique avant de le mettre en application réelle. Dans notre cas, nous utilisons le simulateur *CoppeliaSim*, une plate-forme robotique largement reconnue qui offre un environnement de simulation *3D* intégrant plusieurs moteurs physiques.

4.5.1. Environnement d'apprentissage

Le robot Pioneer *3DX* met en œuvre la planification de trajectoire pour atteindre des cibles prédéfinies tout en évitant les obstacles dynamiques et statiques présents dans son environnement. Il est important de noter que des activités telles que la vérification de l'algorithme *DRL* et l'ajustement des paramètres nécessitent inévitablement des ressources de calcul substantielles et prennent du temps dans un environnement de simulation *3D* équipé d'un moteur physique, ce qui peut entraîner des inefficacités.

Cela nous a conduit à construire 2 types d'environnements d'apprentissage :

- ***Environnement 2D :*** pour améliorer l'efficacité du développement, nous avons développé un environnement *2D* léger sans moteur physique à l'aide des boîtes à outils *Robotics System*, *LiDAR* et *Navigation* de Matlab comme le montre la Figure (4.6 a). Cet environnement simule la cinématique du robot ainsi que le *LiDAR*.

- ***Environnement 3D :*** Les caractéristiques de base de l'environnement *2D* sont les mêmes que celles des environnements *3D* dans la Figure (4.6 b) et la Figure (4.6 c), c'est-à-dire espace continu, évitement d'obstacles en temps réel, objectifs prédéfinis, environnement perceptible, et simulation du *LiDAR*. De plus, *CoppeliaSim* est programmable depuis Matlab, ce qui signifie que les codes d'algorithmes développés pour l'environnement *2D* peuvent être réutilisés facilement après adaptation du code dans *CoppeliaSim*.

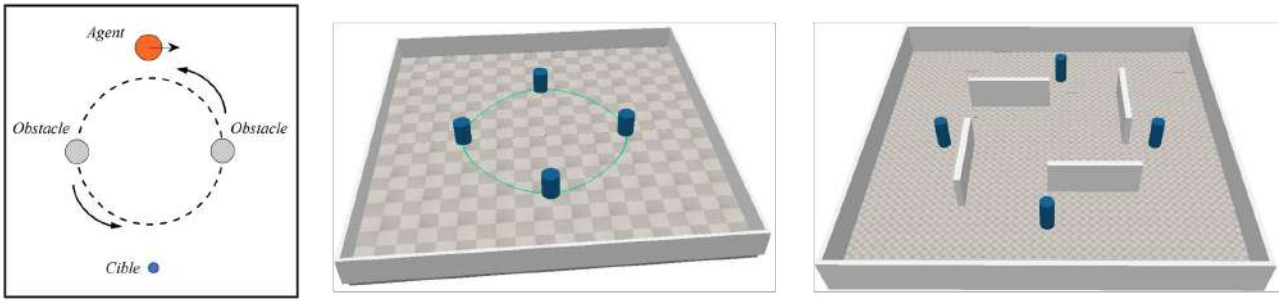


Figure 4.6 : *Environnements d'apprentissage.*

4.5.2. Environnement de test

Afin de comparer la capacité de convergence et de généralisation de l'algorithme *DRL* dans un environnement complexe, nous construisons l'environnement de test $15 \times 15\, m^2$ comme illustré dans la Figure (4.7). Nous avons aussi mené un test dans la $3^{ème}$ scène d'apprentissage (*Figure (4.6 c)*) en variant le nombre de cibles générées.

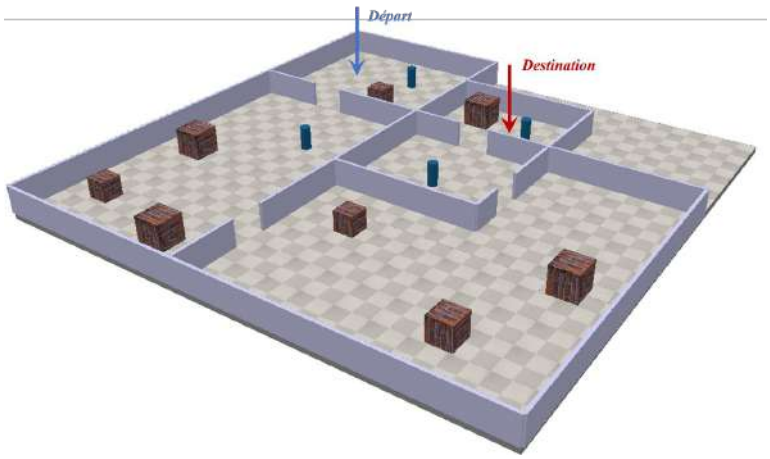


Figure 4.7 : *Environnement de test.*

4.6. Apprentissage et résultats de simulation

L'organigramme de la procédure d'apprentissage est montré sur la Figure (4.8). Après l'initialisation des paramètres de l'agent, de l'algorithme d'apprentissage et de l'environnement de simulation, l'épisode commence par la génération d'une cible aléatoirement dans l'espace de travail du robot. Pour chaque pas de temps, l'agent calcule les actions que va exécuter le robot pour naviguer vers sa destination. Si le robot heurte un obstacle ou atteint sa cible, un indicateur booléen (*done*) de fin d'épisode est mis à un. Dans ce cas, les réseaux *DNN* de l'agents sont mis à jour, le compteur d'épisode est incrémenté. On teste ensuite si on n'a pas atteint le nombre maximal d'épisodes. Si c'est le cas, l'environnement de simulation est réinitialisé, et un nouvel épisode est lancé. Si par contre l'indicateur *done* est égal à zéro, alors le temps de navigation est incrémenté d'un pas. Puis les réseaux *DNN* de l'agents sont mis à jour et on teste si on n'a pas atteint la fin de la durée de l'épisode (*200 s*). Si oui, la navigation vers la cible actuelle continue. Sinon, le compteur d'épisode est incrémenté, et on refait le même test que précédemment jusqu'à ce que le nombre maximal d'épisodes soit atteint et l'apprentissage est terminé. Bien sûr l'apprentissage peut se terminer si l'objectif fixé est atteint avant d'atteindre le nombre maximal d'épisodes (*non représenté dans l'organigramme*).

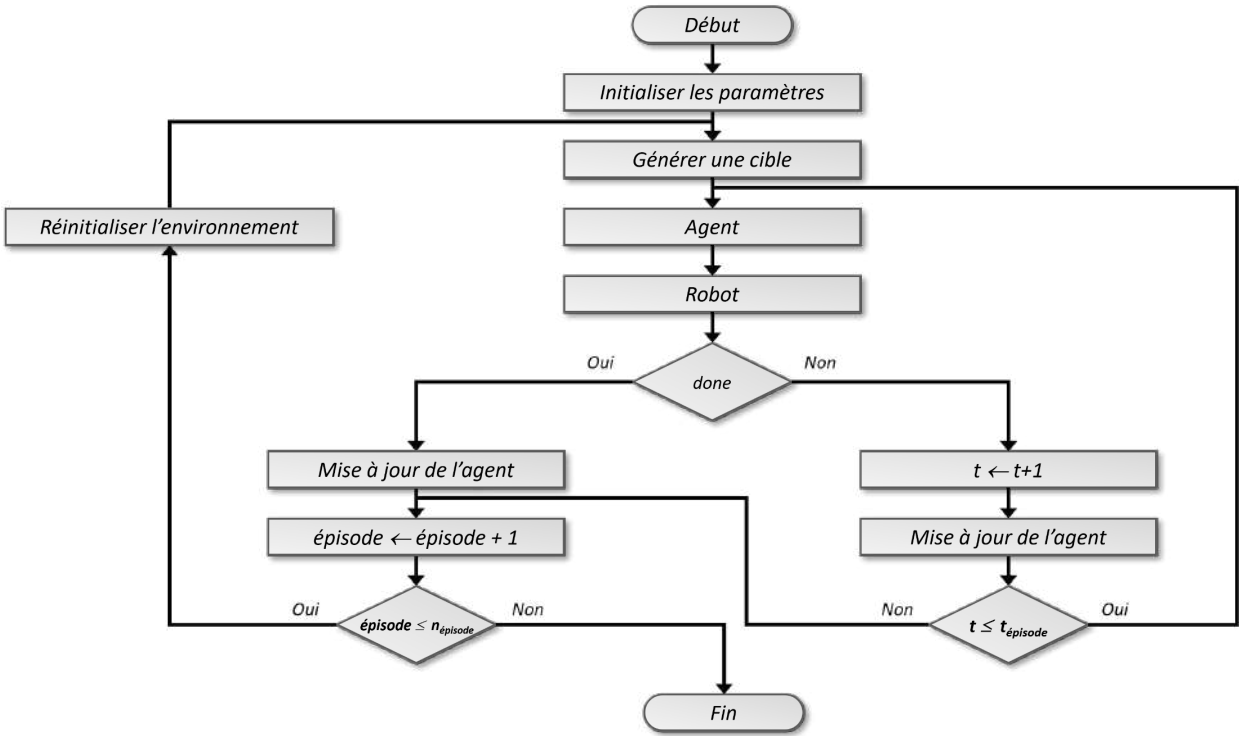


Figure 4.8 : *L'organigramme de la procédure d'apprentissage.*

Le mode d'apprentissage progressif que nous utilisons implique le transfert des paramètres tant du robot que de l'agent *DRL* (*notamment les paramètres du réseau de neurones*) entre les environnements d'apprentissage. Dans ce contexte, le terme progressif (*incrémental*) indique que le processus d'apprentissage et l'environnement d'apprentissage deviennent progressivement plus complexes au fur et à mesure des phases d'apprentissage.

Le principe fondamental du mode d'apprentissage progressif consiste à initialement effectuer des tâches complexes telles que la validation de l'algorithme *DRL* et le réglage des paramètres dans un environnement en *2D* pour un apprentissage préliminaire. Par la suite, la solution optimale ainsi obtenue est transférée vers l'environnement en *3D*, spécifiquement les scènes construites dans *CoppeliaSim*. Les expériences ont prouvé que cette approche a le potentiel d'accroître l'efficacité du développement de la planification de trajectoire basée sur le *DRL* jusqu'à un facteur de 5. Pour illustrer ce point, il est à noter qu'avec les mêmes tâches et les mêmes ressources matérielles, les exigences temporelles sont d'environ 26 heures en *3D* contre seulement 6 heures en *2D* [158].

Les réseaux de neurones profonds (*DNN*) utilisés dans le *DRL* sont sensibles aux valeurs initiales de leurs poids. L'initialisation aléatoire de ces paramètres peut causer une convergence lente, voire parfois une non-convergence. Pour éviter ce problème, nous adoptons le processus d'apprentissage progressif pour le *DRL* présenté dans la Figure (4.9). Avant d'entamer la phase formelle d'apprentissage du modèle en *3D*, nous l'entrainons initialement pendant 4000 épisodes dans un environnement *2D* simple de 5×5 m^2 (*voir Figure (4.6 a)*) où sont effectuées les étapes lentes telles que la définition de la fonction de récompense, l'évaluation des algorithmes de planification de chemin, l'ajustement des paramètres, … etc. Puis pendant 4000 épisodes dans un environnement *3D* de mêmes dimensions en présence de *4* obstacles dynamiques (*Figure (4.6 b)*). Les paramètres résultants de l'apprentissage en *2D* servent de paramètres initiaux pour l'apprentissage en *3D* après l'adaptation des codes de l'algorithme. Ensuite, un autre transfert progressif est opéré et le modèle est à nouveau entraîné pendant 4000 épisodes dans un environnement complexe de 8×8 m^2 (*voir Figure (4.6 c)*). Cette approche de transfert assure une meilleure convergence de l'algorithme *TD3*. A partir de ce point, nous pouvons évaluer plus en détail la capacité de généralisation des algorithmes *DRL*, conformément à ce qui est illustré dans la Figure (4.9).

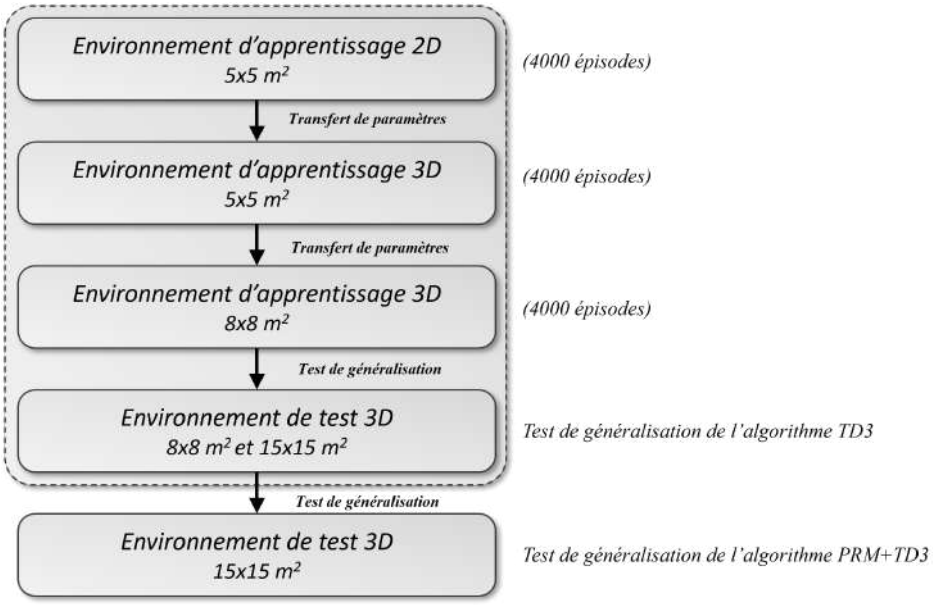


Figure 4.9 : *Processus d'apprentissage progressif.*

4.6.1. Test 1

D'abord, les paramètres d'expérimentation et les critères d'évaluation sont donnés. Ensuite, le planificateur de chemin *PRM+TD3* adopté est testé et comparé avec les méthodes *A*+DWA*, *A*+TEB* et *TD3* en utilisant les mêmes critères et environnements. Les résultats de la planification de chemins sont analysés ainsi que la généralisation dans différents scénarios d'application.

Pour l'application de la planification de trajectoire sur le robot mobile *Pioneer 3DX*, les algorithmes à comparer et à analyser sont divisés en trois groupes :

I. A*+DWA,
II. A*+TEB,
III. PRM+TD3.

Ici, le premier et le deuxième groupe sont les algorithmes de planification de chemins classiques, le troisième groupe représente la combinaison de l'algorithme de planification globale traditionnelle de chemins *PRM* et de l'algorithme d'apprentissage profond par renforcement *TD3*.

Pour comparer l'efficacité de la planification de chemins et de la capacité de généralisation de ces algorithmes, les scènes d'application ont été classées en scène à petite échelle (8×8 m²) et à grande échelle (15×15 m²), respectivement. Les deux scènes sont basées

sur le simulateur *CoppeliaSim*. Toutes les expériences sont menées sur un ordinateur de bureau avec la configuration suivante :

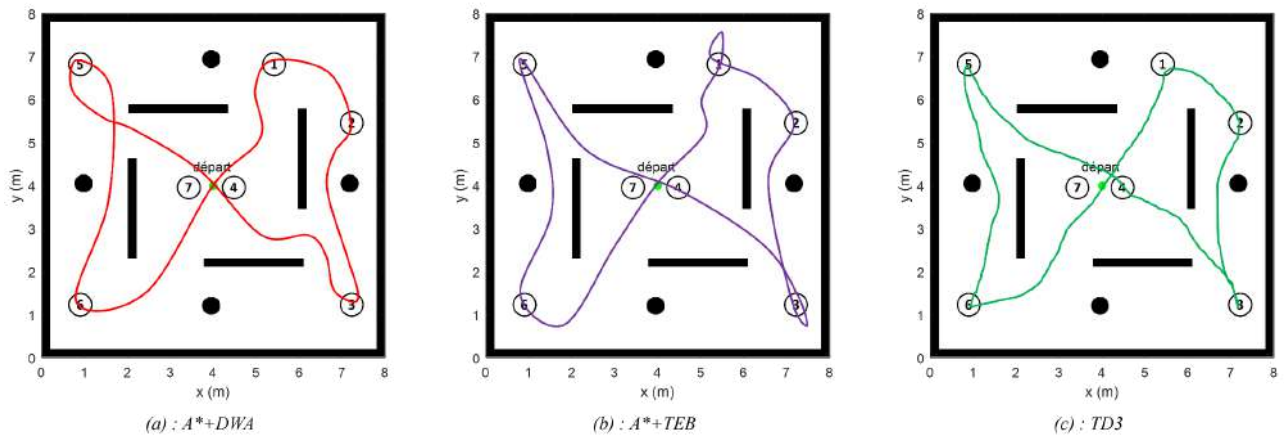
- Un processeur octa-cores *AMD Ryzen 7 3700X @ 3,6 GHz*,
- Une mémoire vive de *16 Go*,
- Une carte graphique *Nvidia RTX 2060* avec *6 Go* de *RAM* dédiée.

Les critères d'évaluation sont définis dans le Tableau (4.1).

Tableau 4.1 : *Paramètres d'évaluation.*

N	Paramètre	Désignation
1	*dep*	Dépendance ou non de la carte de planification de chemin (*oui/non*)
2	t_{trajet}	Durée de navigation du point de départ au point d'arrivée (*s*)
3	$t_{réaction}$	Durée entre l'entrée des données du capteur et la sortie de la commande d'action (*s*)
4	l_{trajet}	Longueur du trajet du point de départ au point final (*m*)
5	*T*	Taux de réussite de l'agent dans *20* expériences (*%*)

Pour vérifier les performances de planification de l'apprentissage par renforcement profond dans différents environnements, nous avons d'abord testé *TD3* dans une scène de petite surface (8×8 m^2), puis comparé *TD3* avec *PRM+TD3* dans une scène de grande surface (15×15 m^2). Au début, nous avons généré 7 points cibles en séquence pour effectuer une planification de chemin avec *A*+DWA*, *A*+TEB* et *TD3*, respectivement, alors que quatre obstacles tournaient dans le sens anti-horaire. Les résultats de simulation sont représentés dans la Figure (4.10), où les courbes rouge, violette et verte représentent la trajectoire planifiée avec les algorithmes I, II et III correspondants, respectivement. On peut constater que *TD3* est meilleur que les deux autres.



(a) : A+DWA* *(b) : A*+TEB* *(c) : TD3*

Figure 4.10 : *Planification de trajectoire dans une scène à petite échelle.*

Tableau 4.2 : Indices de planification dans un environnement à petite échelle.

N	***Algorithme***	***dep***	l_{trajet} *(m)*	$t_{réaction}$ *(s)*	t_{trajet} *(s)*	***T***
I	*A*+DWA*	Oui	36.96	0.5	76.3	75%
II	*A*+TEB*	Oui	35.11	0.7	73.1	85%
III	*TD3*	Non	30.70	0.1	71.5	90%

D'après le Tableau (4.2), le *TD3* ne nécessite pas de carte préalable contrairement aux algorithmes I/II. La planification de chemin traditionnelle nécessite une planification globale en premier lieu en utilisant la carte, puis une planification locale est effectuée après la génération de la carte locale à partir des données du *LiDAR*. C'est pourquoi le temps de réaction ($t_{réaction}$) des algorithmes I/II peut atteindre une grande valeur avec un environnement dynamique. Plus la carte locale est complexe, moins la planification est efficace. En revanche, la décision de *TD3* à chaque pas est générée par le de réseau de neurones, dont la durée reste inférieure aux deux autres algorithmes, offrant une meilleure réactivité aux environnements complexes. Dans un environnement dynamique et étroit, les algorithmes classiques I/II rencontrent parfois des problèmes de collision en raison de leurs faibles performances en temps réel, alors que *TD3* peut presque toujours accomplir la tâche de planification avec succès. En raison des caractéristiques mécaniques inhérentes, la collision est rarement observée dans la phase initiale, lorsque l'agent et les obstacles sont très proches.

4.6.2. Test 2

Dans la scène de 15×15 m^2, le groupe III utilise encore *TD3* seul au début. L'effet de planification des algorithmes est vérifié à travers une cible à longue distance, comme le montre la Figure (4.11). On peut constater que *TD3* (*Figure (4.11 c)*) ne peut pas accomplir la tâche sans la planification de chemin global pour fournir des points de passage intermédiaires, tandis que les algorithmes traditionnels I/II peuvent le faire (*Figure (4.11 a, b)*). Il est prouvé que bien que même si *TD3* ait une performance en temps réel supérieure, il est incapable d'effectuer une planification globale, en particulier dans des scènes complexes à grande échelle.

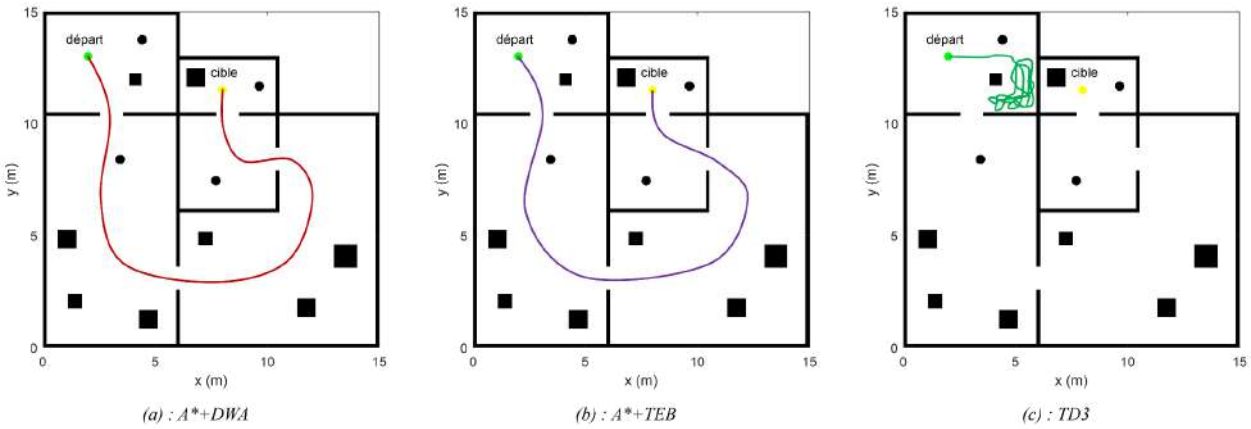


Figure 4.11 : *Planification de trajectoire dans une scène à grande échelle.*

Ensuite, le *PRM* est ajouté au groupe III, ce qui fournit les points de passage intermédiaires globaux à *TD3* et décompose le chemin en plusieurs sous chemins locaux, comme illustré dans la Figure (4.12 a). Il génère des échantillons de points sur la carte d'abord, puis effectue la recherche de chemin. La courbe rouge représente le chemin global planifié par *PRM*. *TD3* prend les points de passage intermédiaires du chemin global comme sous-objectifs et effectue la planification de chemin en plusieurs segments pour obtenir un chemin à grande échelle, représenté dans la Figure (4.12 b).

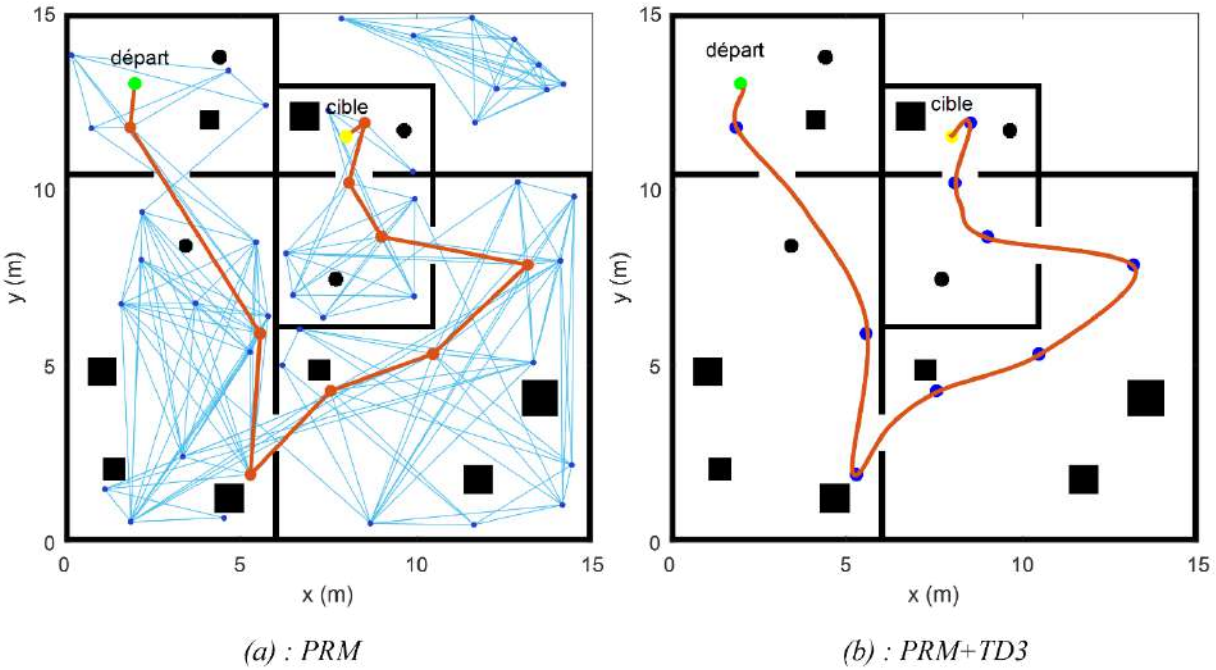


Figure 4.12 : *Planification de trajectoire basée sur PRM+TD3.*

A partir du Tableau (4.3), on peut voir que pour la planification de chemin à grande échelle, *TD3* nécessite que *PRM* recherche des points de passage intermédiaires dans la carte

globale, c'est-à-dire une dépendance à la carte ($dep = oui$). Etant donné que *PRM* est complètement probabiliste, les résultats de chaque planification sont différents et pourraient ne pas être optimaux. La longueur de chemin l_{trajet} et la durée totale du trajet t_{trajet} pour *PRM+TD3* sont plus longues que les deux autres algorithmes. Néanmoins, le temps de réaction $t_{réaction}$ pour *PRM+TD3* est toujours le plus court, ce qui offre une meilleure flexibilité dans les scènes dynamiques et étroites à petite échelle. Comme indiqué dans le Tableau (4.3), la scène de test de 15×15 m^2 n'a pas d'aire étroite dynamique (*présence de beaucoup d'obstacles dans une petite surface*) par rapport à la scène à petite échelle de 8×8 m^2, de sorte que le pourcentage accompli est plus élevé que pour les autres algorithmes. En même temps, étant donné que *PRM* peut fournir des points de passage intermédiaires permettant à *TD3* d'avoir de meilleures capacités de généralisation dans le scénario de test de 15×15 m^2, cela lui permet de réaliser la tâche de planification de chemin.

Tableau 4.3 : *Indices de planification dans un environnement à grande échelle.*

N	***Algorithme***	***dep***	l_{trajet} *(m)*	$t_{réaction}$ *(s)*	t_{trajet} *(s)*	***T***
I	A*+DWA	Oui	34.54	0.5	74.5	95%
II	A*+TEB	Oui	33.64	0.7	72.1	100%
III	PRM + TD3	Oui	35.60	0.1	80.24	100%

4.7. Conclusion

Dans le but de résoudre les problèmes d'efficacité de développement, de convergence médiocre et de faible capacité de généralisation des robots mobiles pilotés par *DRL* dans un environnement dynamique, un mode d'apprentissage progressif est employé pour la planification de chemins. Tout d'abord, nous effectuons les tâches complexes telles que la vérification de l'algorithme *DRL*, l'ajustement des paramètres et l'apprentissage préliminaire dans une scène simple de 5×5 m^2 dans un environnement *2D* léger. Ensuite, nous transférons la meilleure solution obtenue en *2D* vers deux scènes de 8×8 m^2 de complexité élevée pour continuer l'apprentissage en se servant à chaque phase des résultats de la phase précédente pour l'initialisation de l'agent *DRL*, ce qui peut réduire le problème de convergence de *TD3* causé par l'initialisation aléatoire des réseaux profonds de neurones. Ce mode d'apprentissage progressif permet d'améliorer l'efficacité du *DRL* pour de la planification de chemins pour

robot mobile de 5 fois. Etant donné que *TD3* s'adapte bien aux scènes à petite échelle, nous utilisons le planificateur de chemin *PRM+TD3* pour les scènes plus grandes. Les résultats de simulations montrent que le planificateur peut décomposer la navigation à longue distance en plusieurs sous-cibles et obtenir une meilleure généralisation à une nouvelle scène de grande taille. Même si le mode d'apprentissage progressif peut améliorer l'efficacité de développement, la convergence et la généralisation du robot mobile équipé de *DRL*, les résultats de chaque planification sont différents et pourraient ne pas être optimaux.

5. Suivi de Chemins

5.1. Introduction

Les robots mobiles à roues se caractérisent par une structure mécanique relativement simple et robuste, une capacité de charge élevée et des caractéristiques de mouvement flexibles. Ils sont considérés comme une catégorie importante de robots mobiles. Les robots mobiles à roues sont largement étudiés et sont devenus très populaires dans de nombreux domaines tels que la robotique industrielle et de service [159].

En effet, différents types de robots mobiles à roues existent, ils se différencient par leur structure mécanique, leur principe de locomotion qui dépend du type et du nombre de roues et d'actionneurs, etc… ; cependant, le type de robot mobile à roues le plus étudié est le robot mobile unicycle [160]. C'est un robot mobile à commande différentielle équipé de deux roues motrices coaxiales indépendantes entraînées par deux actionneurs distincts et d'un certain nombre de roues libres. Il présente plusieurs avantages tels que la simplicité de structure, la facilité de mise en œuvre, et la possibilité de changer de cap (orientation) en variant simplement les vitesses angulaires des roues motrices, sans en avoir besoin de système de direction supplémentaire [161].

La tâche élémentaire que doit effectuer un robot mobile est de se déplacer d'une position à une autre tout en suivant une trajectoire donnée. Les techniques de commande de mouvement se concentrent sur la façon de rendre un robot mobile autonome et indépendant de toute action externe. Différentes stratégies de suivi de trajectoire adoptant la commande floue ont été présentée dans de nombreux travaux de recherche [162-166]. L'objectif du contrôleur de suivi de trajectoire flou présenté dans [164] est de réduire le nombre de règles d'inférence en adoptant une structure simple ; il utilise deux modules flous avec 36 règles d'inférence floues au total.

Il est à noter que dans les systèmes d'inférence floue classiques, le fait d'augmenter le nombre de variables d'entrée dans les parties antécédentes de chaque règle d'inférence va augmenter de manière exponentielle le nombre total de règles d'inférence dans la base de règles. Par conséquent, la construction et l'expression claire de chaque règle devient difficile.

Pour résoudre ces problèmes et simplifier le processus de conception des systèmes conventionnels d'inférence floue, le modèle d'inférence floue à module de règle mono-entrée (*Single Input Rule Module SIRM*) connecté dynamiquement a été proposé pour le contrôle flou de processus à plusieurs entrées, dans lequel un *SIRM* est construit et un degré d'importance dynamique (*Dynamic Importance Degree DID*) est défini pour chaque variable d'entrée [167]. Cette approche a été appliquée avec succès pour la modélisation [168], la prédiction [169] et les applications de contrôle [169-175]. Les auteurs de [176] étendent le modèle flou à *SIRM* aux systèmes à *SIRM* neuronaux-flous connectés. Pour améliorer encore les performances des *SIRM*, un modèle d'inférence à *SIRM* basé sur la logique floue de type 2 a été introduit dans [177]. Les auteurs de [178] présentent une nouvelle méthode pour mesurer les degrés d'importance des *SIRM* en utilisant des fonctions multivariables de pondération. Les propriétés de continuité, de monotonie, de robustesse et de stabilité des *SIRM* floues de type 1 et de type 2 sont étudiées dans [179], [180] et [181]. L'étape la plus importante et la plus coûteuse en termes de ressources et de temps de traitement du processus de conception d'un système d'inférence floue basé sur les *SIRM* est la phase d'optimisation des paramètres ; qui est réalisé par différentes méthodes d'optimisation telle que la recherche aléatoire [173] ou les algorithmes génétiques [177].

Dans ce chapitre, on propose d'utiliser le modèle d'inférence floue de type 1 à *SIRM* connecté dynamiquement pour implémenter une stratégie de suivi de trajectoire qui a été proposée dans un article précédent [164]. La structure du contrôleur est divisée en deux unités de contrôle ; un contrôleur d'orientation mis en œuvre comme un système de contrôle à *SIRM* et un contrôleur de vitesse matérialisé à l'aide d'un contrôleur à *SIRM* modifié [19]. Le nombre total de règles d'inférence utilisées dans ce contrôleur est de 21 règles floues distribué sur 7 modules *SIRM* avec une seule variable dans la partie antécédente de chaque règle.

Les contributions de ce travail sont :

- L'application du modèle d'inférence floue à *SIRM* au problème de suivi de trajectoire pour robot mobile.
- La modification de la structure du modèle flou a *SIRM* dans le contrôleur de vitesse.

- La suppression de l'étape d'optimisation des paramètres du contrôleur en mettant toutes les valeurs de base à 0 et toutes les largeurs à 1 pour tous les modules *DID* utilisés.

5.2. Le modèle cinématique du robot mobile

Dans le présent travail, une configuration cinématique typique d'un robot mobile à commande différentielle est considérée (*Figure (5.1)*). Dans cette construction, le robot possède deux roues motrices coaxiales montées sur le châssis, chaque roue de rayon r. Dans cette conception, chaque roue motrice est équipée d'un moteur à courant continu pour l'actionnement et d'un encodeur incrémental pour le comptage des tours. Le robot utilise les mesures délivrées par les encodeurs pour calculer la vitesse de rotation de chaque roue.

Le robot mobile navigue sur un terrain horizontal plan. Le vecteur suivant spécifie la posture du robot mobile :

$$q_I = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \tag{70}$$

Où :

- x et y (en *m*) représentent les coordonnées du point P dans le repère global (X_I, Y_I)
- θ (en *rad*) représente l'angle fait par le repère mobile (X_R, Y_R) attaché au châssis du robot, et le repère fixe (X_I, Y_I), c'est l'orientation du robot mobile.

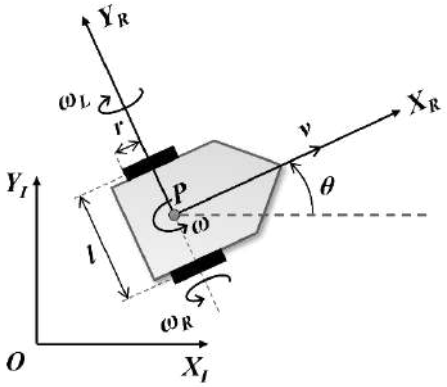


Figure 5.1 : *Robot mobile.*

Les vitesses linéaires des roues motrices sont calculées à l'aide de l'Equation (71), et la vitesse angulaire et la vitesse linéaire du robot mobile sont calculées à l'aide de l'Equation (72),

$$v_R = r\omega_R, v_L = r\omega_L \tag{71}$$

$$\omega = \frac{v_R - v_L}{l}, v = \frac{v_R + v_L}{2} \tag{72}$$

Ici :

- ω est la vitesse angulaire du centroïde P du robot (en *rad/s*).
- v est la vitesse du centroïde du robot (en *m/s*),
- v_L et v_R sont les vitesses linéaires des roues gauche et droite respectivement.
- ω_R et ω_L sont les vitesses de rotation de la roue droite et de la roue gauche, respectivement.

En combinant (71) avec (72), les expressions des vitesses linéaire et angulaire peuvent s'écrire comme suit :

$$\omega = \frac{r}{l}(\omega_R - \omega_L), v = \frac{v_R + v_L}{2} \tag{73}$$

De plus, le modèle cinématique du robot mobile peut être défini comme :

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos(\theta) & 0 \\ \sin(\theta) & 0 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{74}$$

Il est clair d'après l'Equation (74) que le robot mobile a deux entrées de commande et génère la dérivée de son vecteur de posture en sortie.

5.3. Modèle d'inférence floue à SIRM connecté dynamiquement

Cette section donne un bref aperçu du contrôle conventionnel par logique floue et présente le modèle d'inférence floue à *SIRM* connecté dynamiquement pour les systèmes de contrôle multi-entrées à sortie unique (*Multi Input Single Output MISO*) de n variables d'entrée et 1 élément de sortie.

La commande floue est une technique de contrôle heuristique intelligente très populaire. Elle utilise des informations linguistiques ou qualitatives pour intégrer les éléments clés du raisonnement approximatif et de l'expertise humaine dans la conception d'algorithmes de contrôle non linéaires [182]. La commande floue ne nécessite pas de modèles mathématiques précis, elle permet de gérer des entrées imprécises ou incertaines, des non-linéarités et présente une insensibilité aux perturbations supérieure à la plupart des algorithmes de contrôle non linéaires classiques. La commande par logique floue surpasse les autres algorithmes de contrôle dans la régulation des systèmes complexes, non linéaires ou non modélisés pour lesquels il existe des connaissances spécialisées adéquates.

La Figure (5.2) illustre la structure d'un contrôleur flou. Le traitement des valeurs réelles des entrées passe par 3 étapes : la fuzzification, la prise de décision ou l'inférence de règles, et la défuzzification comme moyen de générer une valeur réelle à la sortie [165].

Dans l'étape de fuzzification, les valeurs réelles nettes des variables d'entrée sont converties en variables linguistiques à l'aide de fonctions d'appartenance, c'est-à-dire la conversion des variables d'entrée en valeurs linguistiques appropriées. Les entrées floues sont ensuite manipulées par le moteur d'inférence pour générer une sortie floue selon la base prédéfinie de règles (*base de connaissances*). La base de règles représente les connaissances d'un expert exprimées sous la forme de règles d'inférence ***Si-Alors*** sur la façon de contrôler le système. Le moteur d'inférence (*l'interpréteur*) utilise des opérateurs flous pour inférer les règles actives en fonction des entrées floues pour prendre une décision. L'étape finale est la défuzzification, à ce stade, une des méthodes de défuzzification est utilisée pour transformer la sortie floue en une sortie nette réelle utilisée pour contrôler le système.

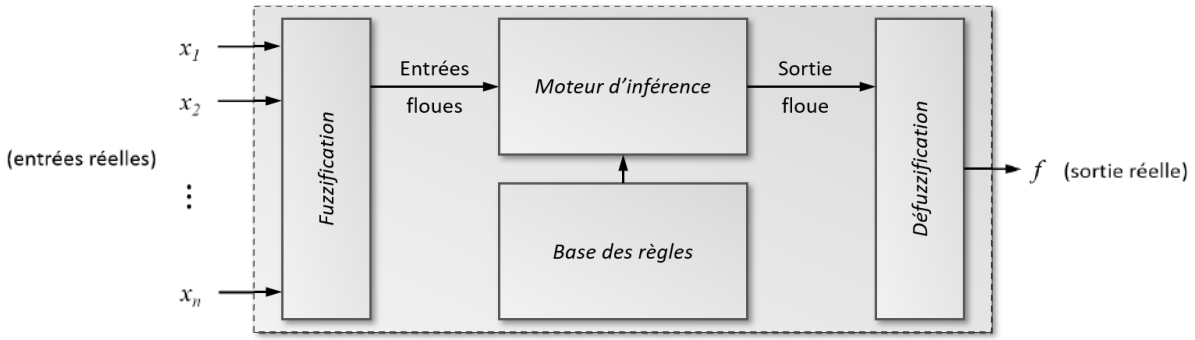


Figure 5.2 : *Structure d'un contrôleur flou.*

Dans le modèle d'inférence floue classique, la partie antécédente de chaque règle floue contient toutes les variables d'entrée, ce qui conduit à l'augmentation exponentielle du nombre de règles floues avec le nombre d'éléments d'entrée et rend difficile la mise en place de chaque règle [173]. Pour surmonter ces problèmes, le modèle d'inférence floue à SIRM connecté dynamiquement a été proposé [167].

La Figure (5.3) montre la structure d'un contrôleur basé sur les SIRM avec n variables d'entrée et un élément de sortie f. Pour chaque élément d'entrée x_i, deux modules flous sont définis : un *SIRM* et un *DID* qui utilisent la méthode *min-max* ou *somme-produit* avec la méthode du centre de gravité pour la défuzzification, ou la méthode d'inférence simplifiée pour produire leurs sorties.

Pour chaque variable d'entrée x_i, le *SIRM* correspondant définit le rôle individuel de la $i^{ème}$ entrée dans l'action de contrôle. Le *SIRM* utilise un certain nombre de fonctions d'appartenance m_i (A_i) pour la fuzzification d'entrée et les singletons C_i (*valeurs constantes*) pour le calcul de sortie. La base de règles (R_i) de ce module contient m_i règles d'inférence.

La sortie du module SIRM (f_i) est calculée à l'aide de l'équation suivante :

$$SIRM - i : \left\{ R_i^j : if\ x_i = A_i^j\ then\ f_i = C_i^j \right\}_{j=\overline{1,m_i},\, i=\overline{1,n}} \tag{75}$$

Avec i et j sont respectivement les indices de la variable d'entrée et de la règle d'inférence.

Il a été constaté que ce schéma ne fonctionne pas bien à cause des inégalités des rôles joués par chaque variable d'entrée dans la génération de l'action de contrôle. Certaines des variables d'entrée ont des contributions significatives à la performance du contrôleur, de sorte que leurs contributions doivent être renforcées. Alors que d'autres variables d'entrée ont de faibles contributions, leurs rôles n'ont donc pas besoin d'être renforcés [167]. C'est la raison qui justifie l'emploi des modules *DID*, qui sont utilisés pour régler le rôle et indiquer évidemment l'effet individuel de chaque entrée sur les performances du système en fonction de l'état de ce dernier.

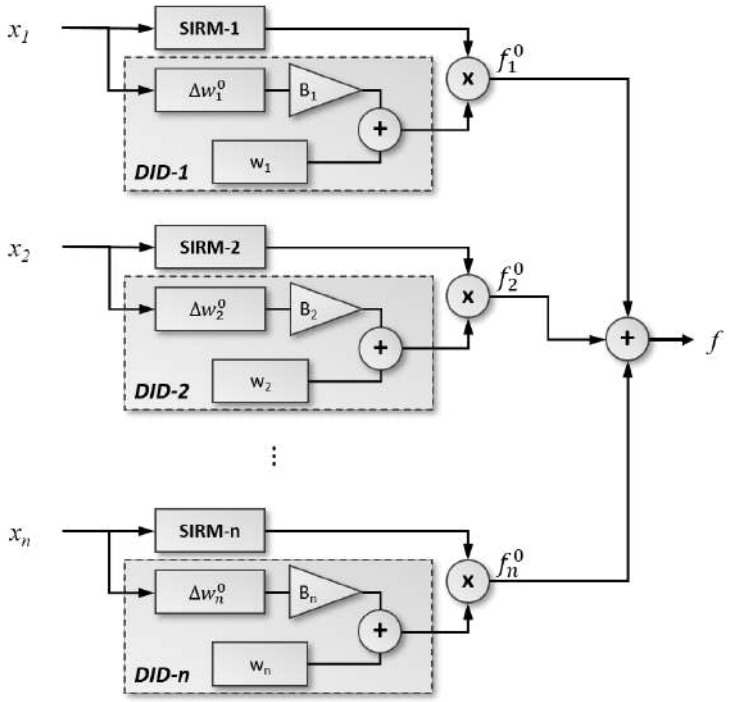


Figure 5.3 : *Structure d'un contrôleur flou à SIRM.*

Comme le montre la Figure (5.3), chaque *DID* contient un module flou qui génère le degré d'importance dynamique de l'élément d'entrée (Δw_i^0), multiplié par la largeur (B_i) pour augmenter ou diminuer son effet, et additionné à une valeur de base (w_i) qui représente la contribution minimale de la variable d'entrée [173].

La sortie d'un module *DID* individuel est définie comme :

$$w_i^D = w_i + B_i \Delta w_i^D \tag{76}$$

La sortie du *SIRM-i* est multipliée par la sortie du *DID-i* pour produire la sortie intermédiaire f_i^0. La somme de toutes les sorties intermédiaires donne la sortie du système (f), telle que définie par :

$$f = \sum_{i=1}^{n} w_i^D f_i^0 \tag{77}$$

5.4. La stratégie du suivi de chemin

Le chemin de référence est composé d'une série ordonnée de n points discrets N_i (*i=1, 2, ..., n*), par lesquels doit passer le robot mobile pour atteindre sa destination finale N_n. Les

coordonnées des points de la trajectoire sont données sous la forme $\left[x_d(i), y_d(i)\right]^T$. Le robot mobile est initialement situé au premier point du chemin avec une orientation arbitraire. Le contrôleur du robot est chargé de le conduire afin d'arriver à sa destination le plus vite possible avec des mouvements souples en ajustant ses vitesses linéaires et angulaires (v, ω) comme le ferait un conducteur humain.

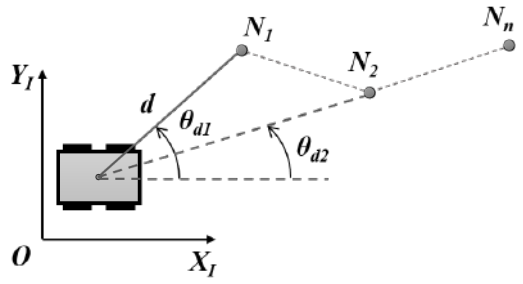


Figure 5.4 : *Paramètres du suivi de trajectoire.*

Lors de son déplacement vers le point actuel du chemin, le robot mobile compare la distance le séparant de ce point avec un certain seuil fixé en fonction de la géométrie du robot. Lorsque cette distance est dans la limite fixée, le robot mobile bascule vers le point suivant dans la série des points.

Le chemin désiré peut être vue comme un ensemble de lignes droites et de virages. Sur les parties rectilignes, le robot mobile peut augmenter sa vitesse (*et diminuer la vitesse angulaire*). A l'approche d'un virage, le robot mobile doit diminuer sa vitesse linéaire et ajuster sa vitesse angulaire en fonction de la courbure du chemin devant le robot (*Look-Ahead Curvature LAC*) pour éviter de se renverser et conserver la souplesse du mouvement.

L'estimation de la courbure du virage est obtenue par la valeur absolue de la différence des valeurs des orientations désirées par rapport au point cible actuel et au point suivant respectivement θ_{d1} et θ_{d2} (*Figure (5.4)*).

La courbure du virage est calculée à l'aide de l'expression suivante :

$$LAC = \left|\theta_{d1} - \theta_{d2}\right| \tag{78}$$

θ_{d1} et θ_{d2} sont définis comme suit :

$$\theta_{d1} = \operatorname{atan}2\left(y_d(i) - y, x_d(i) - x\right) \tag{79}$$

$$\theta_{d2} = \text{atan}\,2\left(y_d\left(i+1\right)-y, x_d\left(i+1\right)-x\right) \tag{80}$$

Avec *x* et *y* représentent la position du robot, $x_d\left(i\right)$ et $y_d\left(i\right)$ les coordonnées de la cible courante N_i et $x_d\left(i+1\right)$ et $y_d\left(i+1\right)$ les coordonnées du prochain point (N_{i+1}) du chemin.

Sur les parties rectilignes du chemin, le *LAC* aura de petites valeurs. Cependant, sa valeur augmentera lorsque le robot mobile s'approchera d'un virage ou d'un coin (*les parties courbées du chemin*).

5.5. Le contrôleur du robot mobile

L'architecture du contrôleur de suivi de chemin est illustrée sur la Figure (5.5). Le module de calcul a pour rôle de fournir les valeurs normalisées des entrées du contrôleur en effectuant des calculs sur les coordonnées du point actuel et du point suivant et la position actuelle du robot mobile. Les valeurs normalisées des entrées sont obtenues en les multipliant par des facteurs d'échelle. Les deux contrôleurs flous (*d'orientation et de vitesse*) génèrent les commandes de vitesses angulaire et linéaire pour conduire le robot mobile à suivre le chemin désiré et atteindre sa destination.

Pour éviter la tâche fastidieuse d'optimisation des paramètres du contrôleur et pour garder notre conception simple, toutes les valeurs de base sont considérées nulles et toutes les largeurs sont prises égales à 1 pour les modules *DID*. Cela signifie que tous les éléments d'entrée jouent des rôles égaux dans le contrôle du robot mobile.

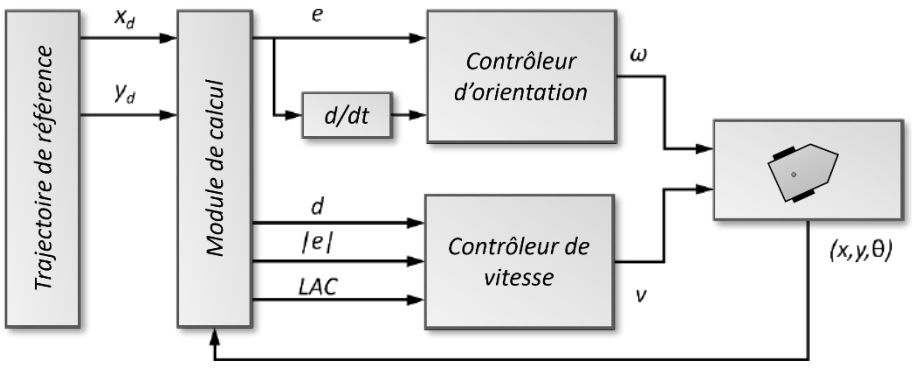


Figure 5.5 : *Architecture du contrôleur du robot mobile.*

5.5.1. Le contrôleur d'orientation

Ce contrôleur agit comme un contrôleur proportionnel dérivé (*PD*) flou conventionnel. Le contrôleur d'orientation prend en entrée deux variables : l'erreur d'orientation e et sa dérivée $\dot{e}$. La sortie de ce contrôleur est la commande de vitesse angulaire (ω) qui permet au robot mobile de corriger son orientation pour se diriger vers le point actuel du chemin. La commande en vitesse angulaire varie de -1 à 1 rad/s. L'erreur d'orientation est calculée par l'équation suivante :

$$e = \theta_{d1} - \theta \tag{81}$$

La structure du contrôleur d'orientation est représentée sur la Figure (5.6). Les *SIRM* des variables d'entrée e et $\dot{e}$ correspondant à l'erreur d'orientation et à sa dérivée peuvent être configurés comme dans le Tableau (5.1) qui représente les règles d'inférence pour ces modules. Ici, les variables linguistiques ***NE***, ***ZE***, ***PO*** représentent les fonctions d'appartenance de chaque variable d'entrée qui sont illustrées sur la Figure (5.7). Trois fonctions d'appartenance triangulaires uniformément réparties dans l'univers de discours normalisé [-1.0, 1.0] sont utilisées pour les entrées. Pour la sortie, trois singletons (*constantes*) sont utilisés comme représenté dans le Tableau (5.1).

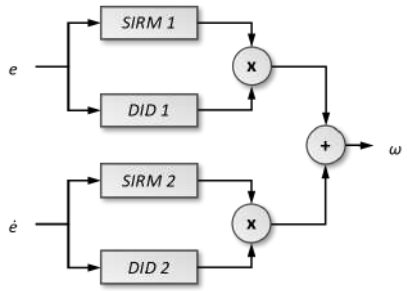


Figure 5.6 : *Structure du contrôleur d'orientation.*

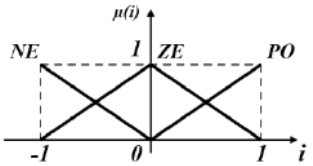


Figure 5.7 : *Fonctions d'appartenance des SIRM du contrôleur d'orientation.*

Tableau 5.1 : *Règles floues des SIRM du contrôleur d'orientation.*

Variable d'entrée e et $\dot{e}$	Variable de sortie f_i
NE	-1.0
ZE	0.0
PO	1.0

Pour le contrôleur d'orientation, les règles floues pour le *DID-1* et le *DID-2* des variables d'entrée e et $\dot{e}$ peuvent être établies comme dans le Tableau (5.2) en sélectionnant les valeurs absolues des éléments d'entrée comme variables antécédentes. Ici, les fonctions d'appartenance ***S***, ***M***, ***B*** sont définies dans l'intervalle [0.0, 1.0] comme représenté sur la Figure (5.8). Pour les sorties des DID, trois singletons sont utilisés comme l'indique le Tableau (5.2).

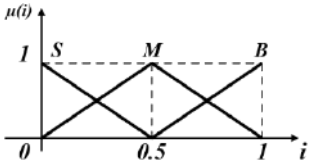


Figure 5.8 : *Fonctions d'appartenance des DID du contrôleur d'orientation.*

Tableau 5.2 : *Règles floues des SIRM du contrôleur d'orientation.*

Variable d'entrée |e| et |ė|	Variable de sortie $\Delta\omega_i$
S	0.0
M	0.5
B	1.0

5.5.2. Contrôleur de vitesse

Le contrôleur de vitesse génère le signal de commande de vitesse linéaire du robot (*variant dans l'intervalle [0.0, 1.0] m/s*). Il prend trois variables d'entrée : la distance au point actuel d, la *LAC* et la valeur absolue de l'erreur d'orientation e.

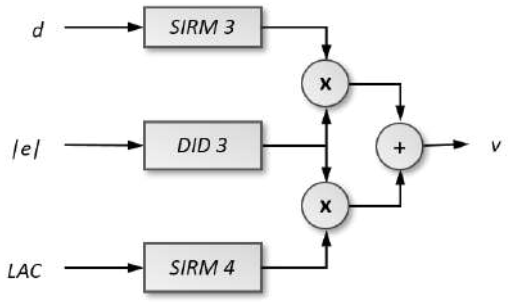


Figure 5.9 : *Structure du contrôleur de vitesse.*

La Figure (5.9) montre la structure du contrôleur de vitesse. Le contrôleur de vitesse utilise deux modules *SIRM* pour la distance *d* et la *LAC* et un module *DID* commun ayant comme entrée la valeur absolue de l'erreur d'orientation. Notez qu'un seul module *DID* est utilisé pour les deux modules *SIRM*.

Tableau 5.3 : *Règles floues du module SIRM 3.*

Variable d'entrée *d*	Variable de sortie f_i
S	0.0
M	0.5
B	1.0

Les *SIRM* des deux variables d'entrée *d* et *LAC* peuvent être configurés comme le montrent les Tableaux (5.3) et (5.4) respectivement. A partir de ces deux tableaux, on peut remarquer que la vitesse du robot mobile varie proportionnellement à la distance au point courant et varie inversement à la courbure au point considéré. D'après le Tableau (5.5), on peut en déduire que lorsque la valeur absolue de l'erreur d'orientation est modérée ou grande, le degré dynamique des éléments d'entrée est diminué de sorte que le robot peut corriger son orientation en premier. Toutes les entrées de ce module utilisent les fonctions d'appartenance de la Figure (5.8).

Tableau 5.4 : Règles floues du module SIRM 4.

Variable d'entrée *LAC*	Variable de sortie f_i
S	1.0
M	0.5
B	0.0

Tableau 5.5 : *Règles floues du module DID 3.*

Variable d'entrée |e|	Variable de sortie $\Delta\omega_i$
S	1.0
M	0.0
B	0.0

5.6. Résultats de simulation

Dans cette section, les résultats des simulations numériques du contrôleur proposé de suivi de trajectoire sont présentés et comparés aux résultats du contrôleur conventionnel à logique floue proposé dans [164]. Les simulations ont été réalisées à l'aide de la boîte à outils *robotics system toolbox* de Matlab.

Dans les exemples présentés, les deux contrôleurs ; le contrôleur basé sur les *SIRM* et le contrôleur flou utilisent les facteurs d'échelle indiqués dans le Tableau (5.6).

Tableau 5.6 : *Les paramètres des contrôleurs.*

Variable d'entrée	Facteur d'échelle
e	10.0
$\dot{e}$	0.01
d	1.0
LAC	1.0

La configuration initiale du robot mobile est : $\begin{bmatrix} x & y & \theta \end{bmatrix}^T = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T$.

La Figure (5.10) montre les résultats du suivi d'une trajectoire discrète en forme de huit. La trajectoire de référence est décrite à l'aide des équations suivantes :

$$\begin{cases} x_d = \dfrac{10\sin(t)}{\left(1+\cos^2(t)\right)} \\ y_d = \dfrac{10\sin(t)\cos(t)}{\left(1+\cos^2(t)\right)} \end{cases} \tag{82}$$

Ici, t est un nombre réel discret qui varie dans l'intervalle [-3.2, 3.2] avec un pas de 0.2.

Les trajectoires du robot mobile contrôlé par les deux méthodes de commande (*Figure (5.10)*) montrent une grande similitude en termes de suivi de chemin. Les deux contrôleurs peuvent piloter le robot mobile à suivre avec précision la trajectoire désirée.

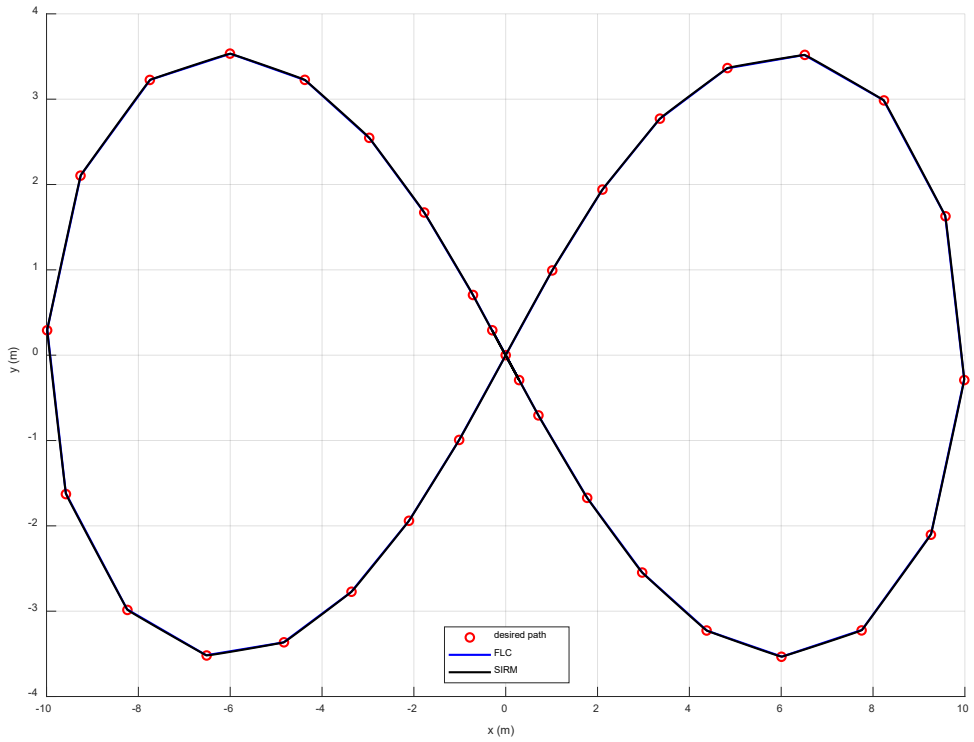
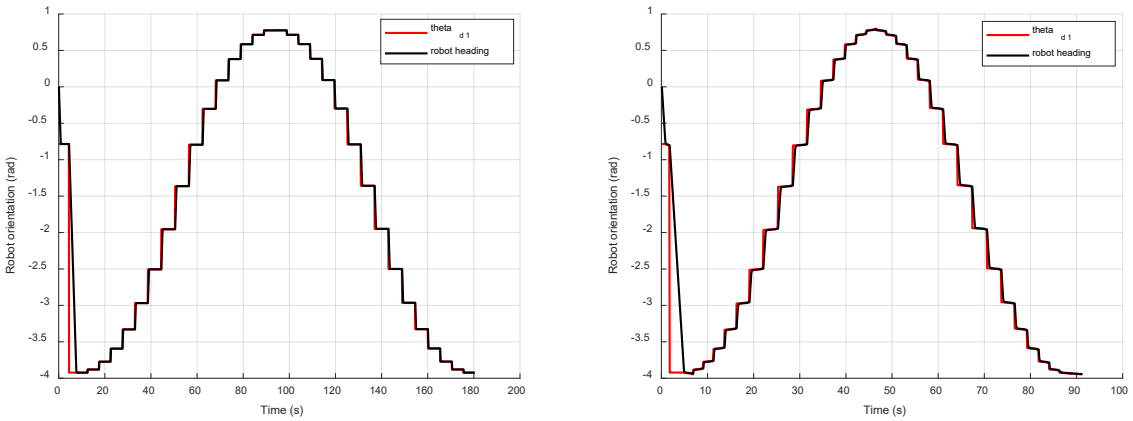


Figure 5.10 : *Suivi d'une trajectoire en forme de 8.*

La Figure (5.11) montre la commande de l'orientation du robot mobile ; l'orientation désirée est calculée en ligne à l'aide de l'Equation (79). L'orientation actuelle du robot est régulée par le contrôleur d'orientation qui génère le signal de commande de vitesse angulaire (*Figure (5.12)*). A partir de ces figures, on remarque que le robot mobile contrôlé par le contrôleur basé sur les *SIRM* est plus rapide dans la réalisation de sa tâche que celui contrôlé par le contrôleur flou conventionnel. Le contrôleur basé sur les *SIRM* atteint son objectif en 91.3 s tandis que le contrôleur flou prend 180.4 s pour atteindre l'objectif. Ceci peut être justifié par le fait que les valeurs moyennes des signaux de commandes issus du contrôleur à *SIRM* sont supérieures à celle générées par le contrôleur flou.

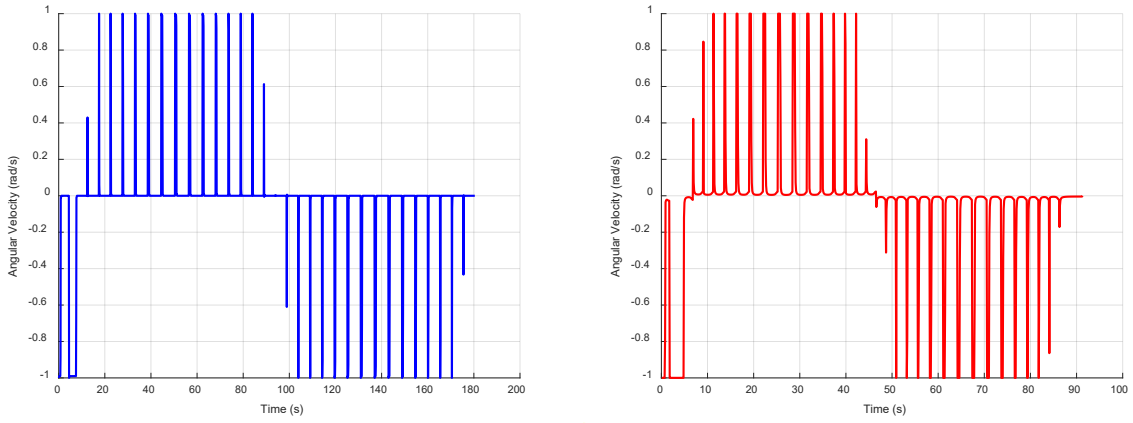


(***a***) (*b*)

Figure 5.11 : *Commande d'orientation (a : flou, b : SIRM).*

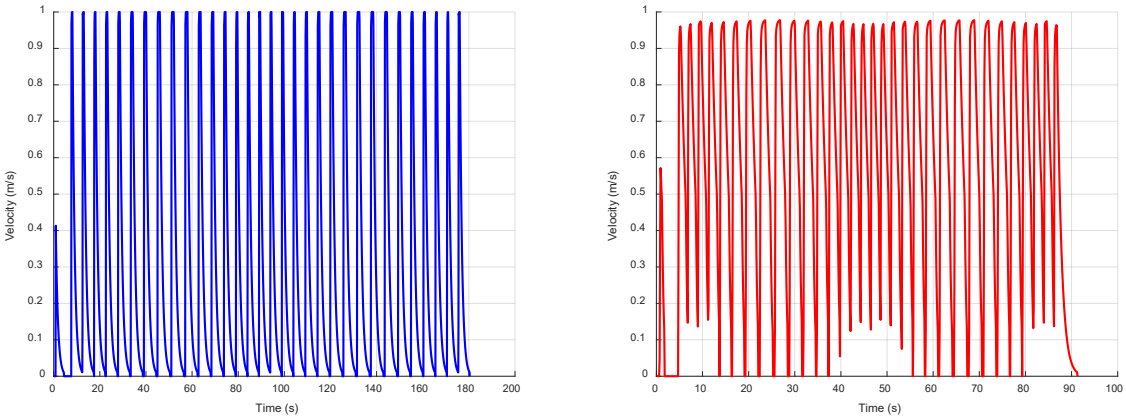
La Figure (5.13) montre les signaux de commande de vitesse linéaire générés respectivement par le contrôleur flou et le contrôleur de vitesse basés sur les *SIRM*. Les différences dans les formes des signaux sont dues aux différences des méthodes d'inférence entre le contrôleur flou qui utilise une méthode d'inférence floue conventionnelle et le contrôleur proposé qui est basé sur le modèle d'inférence floue à *SIRM* connecté dynamiquement. Ces résultats montrent également l'avantage du contrôleur basé sur les *SIRM* dans l'exécution de la trajectoire plus rapidement que le contrôleur flou classique.

La Figure (5.14) montre le résultat du suivi d'un trajectoire sinusoïdale discrète. On remarque la souplesse du mouvement du robot mobile sous le contrôle des deux contrôleurs.



(***a***) (***b***)

Figure 5.12 : *Signaux de la commande d'orientation (a : flou, b : SIRM).*



(*a*) (*b*)

Figure 5.13 : *Signaux de la commande de vitesse (a : flou, b : SIRM).*

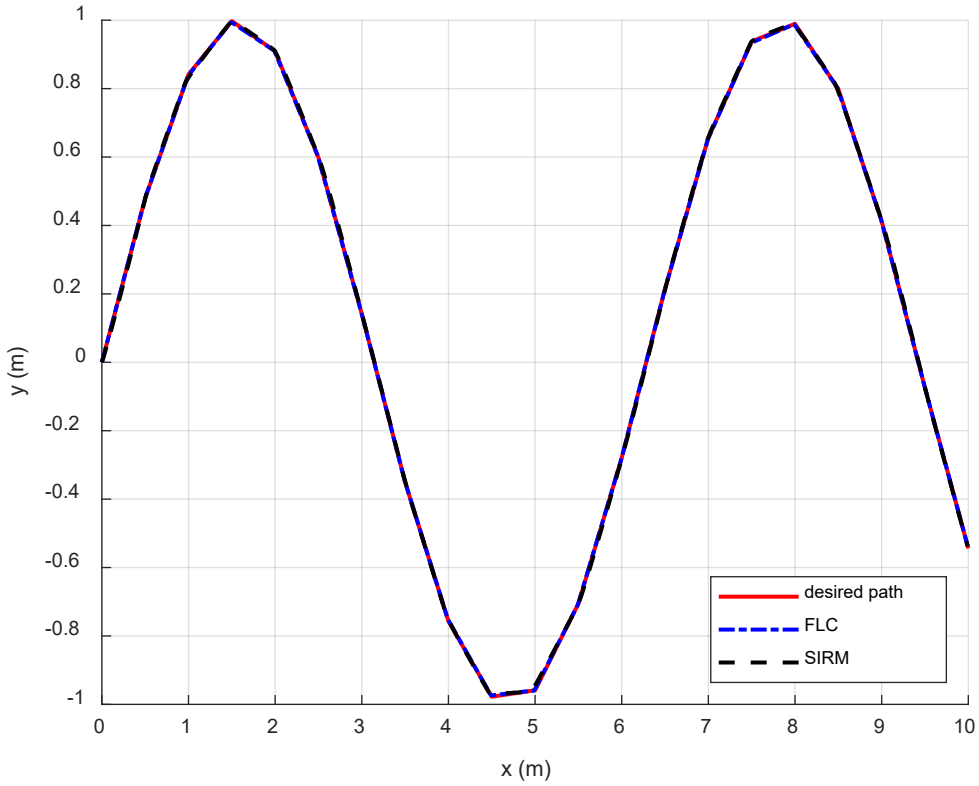


Figure 5.14 : *Suivi d'une trajectoire sinusoïdale.*

5.7. Conclusion

Ce chapitre propose un contrôleur de suivi de chemin pour les robots mobiles à roues à commande différentielle basé sur le modèle d'inférence floue à *SIRM* connecté dynamiquement. Ce type spécifique de modèle d'inférence est utilisé pour résoudre le problème de l'augmentation du nombre de règles d'inférence dans la commande floue

classique pour les systèmes de commande multi-entrées. La structure d'un système de contrôle basé sur les *SIRM* permet également de réduire le temps de traitement.

Le contrôleur proposé pilote un robot mobile pour suivre un chemin discret de référence composé d'une série ordonnée de points de passage distincts en ajustant les vitesses angulaire et linéaire du robot. Le contrôleur est composé de deux unités de contrôle indépendantes travaillant en parallèle ; le contrôleur d'orientation qui génère la commande de vitesse angulaire, et le contrôleur de vitesse qui est chargé de générer la commande de vitesse linéaire. Le contrôleur utilise un nombre total de 21 règles d'inférence pour imiter la manière de conduire d'un conducteur humain.

Le contrôleur d'orientation prend l'erreur d'orientation et sa dérivée comme entrées et prend la vitesse angulaire comme sortie. Chaque élément d'entrée est affecté d'un SIRM et d'un *DID*.

Le contrôleur de vitesse a trois variables d'entrée : la distance au point ciblé, l'estimation de la courbure devant le robot (*LAC*) et la valeur absolue de l'erreur d'orientation. Il génère en sortie le signal de commande de vitesse linéaire à appliquer au robot mobile. La structure du modèle d'inférence à *SIRM* a été modifiée pour ce contrôleur de sorte que la distance et le *LAC* sont attribués avec des *SIRM* et un *DID* commun qui prend la valeur absolue de l'erreur d'orientation comme entrée. Les *SIRM* et les *DID* du contrôleur de vitesse sont configurés de manière à ce que le contrôle de la vitesse angulaire du robot mobile ait la priorité sur le contrôle de la vitesse linéaire. L'ajustement des ordres de priorité est automatiquement effectué en ligne en fonction de la situation de navigation actuelle.

Aucune optimisation des paramètres du contrôleur n'est nécessaire puisque toutes les valeurs de base sont mises à 0 et toutes les largeurs sont mises à 1 pour les modules *DID*. Cela signifie que tous les éléments d'entrée jouent des rôles égaux dans le contrôle du robot mobile.

Les résultats de la simulation montrent que le contrôleur proposé surpasse le contrôleur flou conventionnel dans le suivi de trajectoire en termes de précision de suivi et de temps d'exécution de la trajectoire.

6. Evitement d'Obstacles

6.1. Introduction

Fournir une technologie permettant de reconnaître et d'éviter les obstacles lors de la navigation autonome de robots (*telle qu'appliquée actuellement dans divers domaines*) est un facteur important en matière de sécurité. Cependant, les exigences techniques pour une navigation sans collision ne sont que partiellement satisfaites par les fonctions que les méthodes de navigation actuelles peuvent exécuter. Avec la technologie actuelle, la sécurité n'est garantie que dans un environnement d'obstacles statiques ou dans un environnement spécifique prédéterminé. Un facteur difficile dans la navigation en raison de cette limitation technique est l'évitement d'obstacles dynamiques qui est indispensable pour que les robots se déplacent de manière autonome dans un environnement réel composé d'éléments dynamiques et incertains tels que des piétons et des robots en mouvement [183-185]. Cela nécessite des connaissances supplémentaires concernant des facteurs tels que la vitesse et la direction des obstacles, contrairement à l'évitement d'obstacles statique [186, 187]. En plus, les robots doivent être capables d'éviter les obstacles en prédisant leurs mouvements à l'aide de ces informations et en créant immédiatement un itinéraire d'évitement. Ces exigences sont compliquées par plusieurs défis, notamment le fait que l'évitement dynamique des obstacles doit être appliqué dans des environnements variés, que le coût de calcul pour le robot augmente avec la quantité d'informations requises, et que cela peut réduire la réactivité de l'algorithme.

Dans ce chapitre, nous avons supposé que les problèmes mentionnés précédemment peuvent être résolus si une assistance locale rapide et continuellement mise à jour de l'évitement d'obstacles dynamique par la méthode de la fenêtre dynamique (*DWA*) aide le robot dans certains mouvements. Pour vérifier cette hypothèse, nous avons proposé une méthode hybride qui combine un module *DWA* avec un module de contrôle de vitesse en utilisant le *DQN* [121]. Ce module utilise l'action appropriée de l'apprentissage *DQN* basé sur un évitement local continuellement mis à jour à partir de la *DWA* pour améliorer la fonction d'évitement du robot. Cette méthode permet d'améliorer la qualité des mouvements du robot dans des situations qui étaient problématiques dans les études antérieures utilisant le

RL, et permet de réduire la perte d'espace de dégagement pour les obstacles mobiles qui ne peuvent être résolus par la méthode *DWA* seule. Pour confirmer que le taux de réussite de l'évitement d'obstacles est augmenté grâce à un tel apprentissage, une série de tests a été menée dans le simulateur *CoppeliaSim*.

6.2. Principes de l'approche de la fenêtre dynamique

La fenêtre dynamique (*DWA*) est une méthode couramment utilisée en robotique mobile pour la planification locale de trajectoires en temps réel. Elle permet à un robot mobile de se déplacer de manière autonome dans un environnement en évitant les obstacles. La méthode *DWA* définit une fenêtre dynamique qui représente les limites de mouvement possibles pour le robot en fonction de ses capacités mécaniques et dynamiques. Cette fenêtre est générée à partir les limites minimales et maximales des vitesses et des accélérations possibles du robot.

Ensuite, les obstacles détectés sont projetés sur la fenêtre dynamique et une fonction de coût est définie pour chaque cellule de la fenêtre en fonction de la distance par rapport aux obstacles et de la distance jusqu'à la destination finale. La méthode *DWA* utilise alors un algorithme de recherche pour trouver la meilleure trajectoire possible qui minimise la fonction de coût.

Cette méthode comprend trois étapes pour chercher dans l'espace de vitesse en maximisant la fonction coût. Dans la première étape, les vitesses inaccessibles sont éliminées en fonction des contraintes dynamiques du robot qui proviennent de l'accélération du robot. De plus, le *DWA* ne considère que les vitesses atteignables qui sont plus sûres en ce qui concerne les obstacles [188]. La deuxième étape élimine toutes les paires de vitesse provenant des vitesses restantes du robot, qui sont incapables d'arrêter le robot avant une collision avec les obstacles. Dans la dernière étape, un ensemble de vitesse admissible est évalué en maximisant la fonction coût. Dans cette méthode, la trajectoire du robot est prédite en évaluant chaque paire de vitesses candidates par rapport à la vitesse linéaire, à l'orientation finale et à la distance de sécurité par rapport aux obstacles, comme présenté dans l'Equation (83) :

$$G(v,\omega) = \sigma\left(\alpha \cdot heading(v,\omega) + \beta \cdot dist(v,\omega) + \gamma vel(v,\omega)\right) \tag{83}$$

Où :

- (α, β, γ) sont les constantes de pondération et

- σ est l'opérateur de lissage de *DWA*.

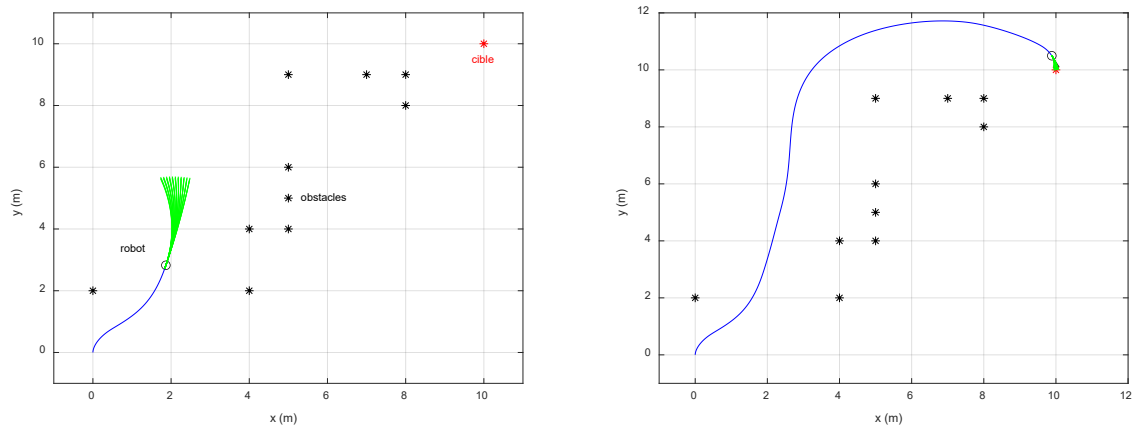


Figure 6.1 : *Exemple de la DWA.*

Dans la fonction coût, $heading(v,\omega)$ représente l'angle entre l'orientation du robot et les coordonnées de l'objectif, tandis que le but de $dist(v,\omega)$ est de fournir une navigation sûre. Il mesure et calcule la distance entre le robot et l'obstacle le plus proche sur le chemin à partir de la paire de vitesses échantillonnée (v,ω). Enfin, la commande de mouvement suivante est déterminée par la paire de vitesse (v,ω) qui maximise la valeur de la fonction coût de la vitesse linéaire. La méthode traditionnelle de *DWA* consiste à rechercher dans l'espace de vitesse les commandes pour contrôler les robots mobiles, et cet algorithme prend en compte la dynamique du robot [188]. Les trajectoires candidates sont représentées en vert dans la Figure (6.1).

L'avantage de la méthode *DWA* est qu'elle est rapide et efficace, car elle ne considère que les mouvements possibles du robot qui sont limités par ses capacités mécaniques et dynamiques. Cela permet de réduire le temps de calcul nécessaire pour trouver la trajectoire optimale. Cependant, la méthode *DWA* peut parfois ne pas produire la trajectoire optimale en raison de ses hypothèses simplificatrices et peut également ne pas être efficace dans les environnements avec des obstacles mobiles et/ou imprévisibles.

6.3. Configuration du système d'évitement d'obstacles

L'architecture du système de navigation locale est illustrée dans la Figure (6.2), il est composé des modules suivants :

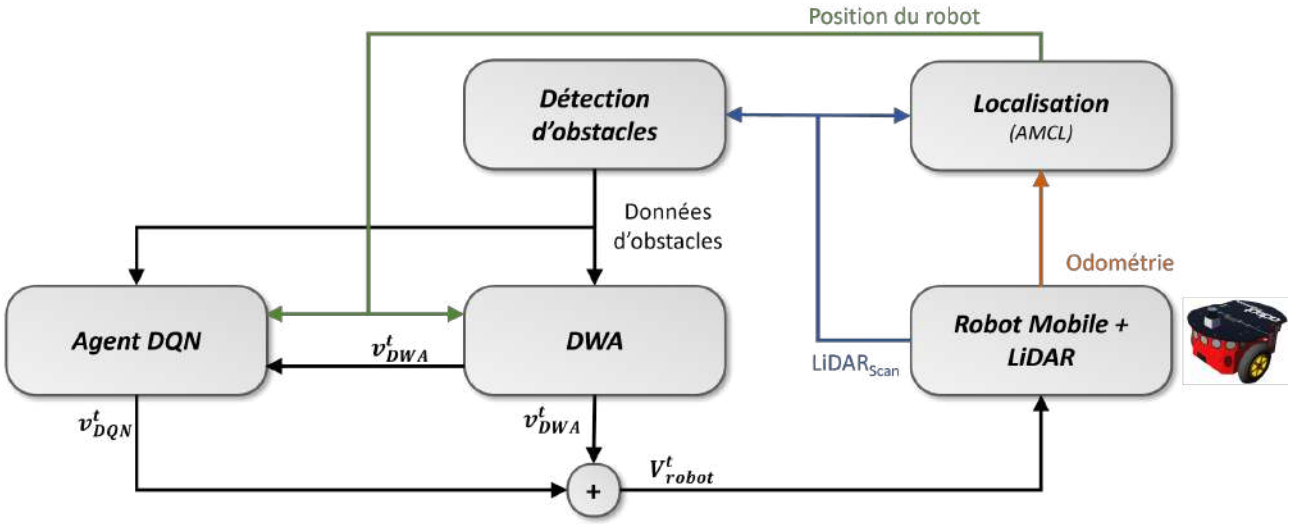


Figure 6.2 : *Architecture du système de navigation locale.*

- ***Le robot mobile et le LiDAR :*** recevant les consignes de vitesses de translation, et de rotation et fournissant les mesures odométriques et les données du *LiDAR*.
- ***Le module de localisation adaptative de Monte Carlo (AMCL) [155] :*** qui fusionne les données odométriques et celle du scan pour fournir la meilleure approximation possible de la position du robot dans son environnement (*supposant qu'une carte de l'environnement est disponible*).
- ***Le module de détection d'obstacles :*** ce module traite les données du balayage laser (*2D*) pour fournir la liste des obstacles détectés avec leurs distances au robot, leurs vitesses et leurs orientations.
- ***Le module DWA :*** utilise l'algorithme de la fenêtre dynamique pour planifier une trajectoire locale pour éviter la collision. La sortie de *DWA* v^t_{DWA} est composée de la vitesse de translation et de rotation (v, ω) du robot.
- ***Le module DQN :*** c'est l'agent d'apprentissage profond par renforcement qui apprend à améliorer la réponse du *DWA*. Le module apprend à sélectionner la meilleure action pour la politique actuelle (v^t_{DQN}) qui sera combinée avec la sortie du module (*DWA*) pour fournir la commande actuelle du robot (V^t_{robot}) qui calculée par l'Equation (84).

$$V^t_{robot} = v^t_{DWA} + v^t_{DQN} \tag{84}$$

6.4. Configuration de l'agent DQN

Le module *DQN* présenté dans la Figure (6.2) utilise un réseau de neurones profonds (*DNN*) pour améliorer l'algorithme d'apprentissage *Q* conventionnel. Cette méthode consiste à approximer la valeur de l'état-action à l'aide d'un *DNN*. Le modèle *DQN* met régulièrement à jour les poids du *DNN* en minimisant une fonction de coût dérivée du même réseau *Q* à partir des poids précédents. Le *DNN* utilisé dans cette approche est composé de 3 couches cachées entièrement connectées, comme présenté dans la Figure (6.3).

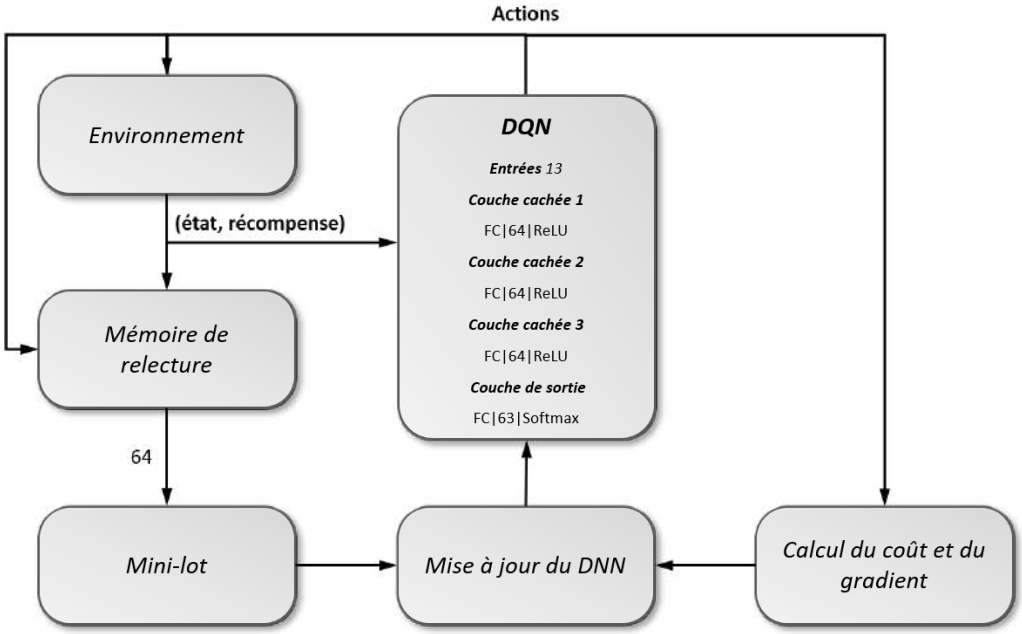


Figure 6.3 : *Le module DQN.*

La méthode de relecture d'expérience est utilisée pour stocker les valeurs de l'état actuel, de l'action, de la récompense et de l'état suivant à chaque pas de temps. Le module sélectionne au hasard un ensemble de 64 expériences de la mémoire, que l'on appelle un *mini-lot* d'expériences, lors de la relecture. Les poids du *DNN* sont mis à jour en utilisant l'optimisation *Root Mean Square Prop* (*RMSProp*). La politique utilisée dans la Figure (6.3) permet de trouver un compromis entre exploration et exploitation en utilisant l'exploration *ε-greedy*.

6.4.1. Définition de l'état

L'état S est l'état de l'environnement tel qu'il est reconnu par le robot. Des études antérieures ont montré qu'il est important de minimiser les paramètres de l'état pour améliorer la vitesse et les performances de l'apprentissage pour l'algorithme *DQN* [123]. Ainsi, nous

avons optimisé 13 paramètres de S constitués d'informations obtenues à partir des trois modules : modules de détection d'obstacles, *DWA* et *AMCL*. Tout d'abord, l'algorithme de détection d'obstacles dans l'Algorithme (6.1) (*basé sur le détecteur de point d'arrêt adaptatif* [189]) trouve les vitesses relatives v_{ob}^{n}, la distance l^{n} et les angles θ^{n} de jusqu'à trois obstacles dans l'ordre de l'obstacle le plus proche. Les distances (l^{n}) aux obstacles sont déterminées par les données *LiDAR* correspondant aux angles centraux (θ^{n}) des angles des points critiques obtenus en appliquant la méthode de l'Algorithme (6.1). Et les vitesses relatives (v_{ob}^{n}) des obstacles et du robot sont obtenues par la différence entre les distances aux obstacles du pas de temps précédent et les valeurs de distance du pas de temps courant. L'Algorithme (6.1) est une méthode utilisant un seul *LiDAR 2D*, de sorte que la précision, la distance de détection et la vitesse d'obstacle reconnaissable sont déterminées en fonction des performances du capteur utilisé. Les critères de sélection de ces informations sont les données d'entrée utilisées pour l'évitement dans [55]. La sélection de n est empiriquement basée sur le nombre maximal d'itérations autorisées dans le temps d'échantillonnage de notre système pour garantir de bonnes performances. A l'aide du module *AMCL*, la distance d et l'orientation relative θ_t entre le robot et la cible sont estimées, sur la base de la carte actuelle. Enfin, la vitesse linéaire actuelle v_r et la vitesse de rotation ω_r sont obtenues à partir de la sortie de l'étape précédente du module *DWA* (v_{DWA}^{t-1}). Ainsi, l'état S peut être représenté comme suit :

$$S = \left[v_{ob}^{n}, l^{n}, \theta^{n}, d, \theta_t, v_r, \omega_r \right] \in \mathbb{R}^{13}, n \in \{1,2,3\} \tag{85}$$

Les valeurs des paramètres dans l'Algorithme (6.1) sont les suivantes :

- $scan_{data}$ est un vecteur des valeurs des distances captées par le *LiDAR*,
- D_{max} est le seuil de classification des points de rupture,
- $\Delta\phi$ est la différence angulaire entre les particules du *LiDAR*,
- σ_r est la valeur de compensation du bruit du *LiDAR*,
- λ par rapport à la direction de balayage $scan_{data}[i-1]$ et vise à extrapoler le pire point de distance acceptable,

- $Left_{degree}$ et $Right_{degree}$ sont les valeurs d'angle du capteur correspondant au point de rupture d'obstacle trouvé.

Algorithme 6.1 : *Algorithme de détection d'obstacles*

1. *Initialiser toutes les données d'obstacles* $\left[v_{ob}^{n}, l^{n}, \theta^{n} \right], n \in \{1,2,3\}$ *;*

2. *Obtenir* $scan_{data}$ *en utilisant les données 2D du LiDAR ;*

3. ***Pour*** *chaque pas de temps* ***faire :***

4. ***Pour*** $i = 1,\ taille\left(scan_{data} - 1\right)$ ***faire :***

5. $D_{max} = scan_{data}\left[i\right] \times \dfrac{\sin\left(\Delta\phi\right)}{\sin\left(\lambda - \Delta\phi\right)} + 3\sigma_{r}$

6. ***Si*** $\left| scan_{data}\left[i+1\right] - scan_{data}\left[i\right] \right| < D_{max}$ ***alors*** *:*

7. *Obtenir* $Left_{degree}$ *et* $Right_{degree}$ *[1 ~ 3] ;*

8. ***Fin Si*** *;*

9. ***Fin Pour*** *;*

10. *Obtenir* $\left[\theta^{n} \right] = \dfrac{Left_{degree} + Right_{degree}}{2}$ ***;***

11. *Obtenir* $\left[l^{n} \right] = scan_{data}\left[\theta^{n} \right]$ ***;***

12. $\Delta time = current_{time} - old_{time}$ ***;***

13. *Obtenir* $\left[v_{ob}^{n} \right] = \dfrac{obstacle\ dist_{old} - l^{n}}{\Delta time}$ ***;***

14. ***Fin Pour*** *;*

15. ***Sortie :*** *données d'obstacles*

6.4.2. Définition des actions

Afin de fournir des vitesses continues applicables au robot avec un nombre limité d'actions, l'action est configurée de la manière suivante. L'action du robot différentiel englobe une mobilité de translation et de rotation, c'est-à-dire $v_{DQN}^{t} = \left[v^{t}, \omega^{t} \right]$. Dans cette étude, la vitesse linéaire du robot peut prendre une des valeurs suivantes : $v \in \{-1, -0.75, -0.5, -0.25, 0, 0.25, 0.5, 0.75, 1\}$ *m/s* et celle de la vitesse angulaire est de $\omega \in \{-3, -2, -1, 0, 1, 2, 3\}$ *rad/s* (*ce qui correspond à 63 combinaisons possibles à la sortie du module DQN*), en prenant en compte la fenêtre dynamique et la cinématique du robot. L'Algorithme (6.2) montre que les valeurs de cette action sélectionnée sont reflétées dans le robot. Dans cet algorithme, la vitesse désirée du robot V_{robot}^{t} est déterminée dans la plage de

la vitesse variable maximale V_{dwmax}^{t} à l'étape courante. Cette valeur est divisée en v_{dw} et ω_{dw} et est affectée par l'accélération maximale du robot $(v_{acc},\ \omega_{acc})$.

Algorithme 6.2 : *Algorithme de calcul de vitesse*

1. *Initialiser toutes les vitesses* à *0 ;*
2. ***Pour*** *chaque pas de temps* ***faire :***
  3. *Obtenir* $v_{DWA}^{t},\ v_{DQN}^{t},\ V_{dwmax}^{t} = [v_{dw},\ \omega_{dw}]$ *;*
  4. $V_{robot}^{t} = v_{DWA}^{t} + v_{DQN}^{t}$ *;*
  5. $\Delta time = current_{time} - old_{time}$ *;*
  6. $[v_{dw}, \omega_{dw}] = \Delta time \times [v_{acc}, \omega_{acc}]$
  7. ***Si*** $\left|V_{robot}^{t} - V_{robot}^{t-1}\right| < V_{dwmax}^{t}$ ***alors*** *:*
    8. ***Si*** $\left(V_{robot}^{t} - V_{robot}^{t-1}\right) < 0$ ***alors*** *:*
      9. $V_{robot}^{t} = V_{robot}^{t-1} + V_{dwmax}^{t}$ *;*
    10. Sinon
      11. $V_{robot}^{t} = V_{robot}^{t-1} - V_{dwmax}^{t}$ *;*
    12. Fin Si *;*
  13. Fin Si *;*
14. Fin Pour *;*

6.4.3. Définition de la fonction récompense

La récompense est un paramètre qui représente la valeur de l'état en fonction de l'action de la fonction *Q-value* [30]. Le but de ce changement est de créer une politique qui améliore la capacité à éviter les collisions entre le robot et les obstacles.

$$r^{t} = r_{g}^{t} + r_{c}^{t} + r_{m}^{t} \tag{86}$$

La récompense r reçue par le robot à l'instant t est la somme de trois termes, r_g, r_c et r_m comme indiqué dans l'Equation (86). Ces trois valeurs sont choisies de telle façon à maximiser le taux de réussite de notre agent à accomplir sa mission.

Tout d'abord, r_g^t est une récompense positive que reçoit le robot (*agent*) lorsqu'il atteint la position cible. Elle est définie par l'équation suivante :

$$r_{g}^{t} = \begin{cases} r_{arrivée} & si\ d < d_{goal} \\ 0 & si\ non \end{cases} \tag{87}$$

Dans l'Equation (87), le robot reçoit une récompense $r_{arrivée}$ lorsque la distance d le séparant de la position cible est inférieure au rayon d_{goal} centré sur la position cible, et si cette condition est satisfaite, l'épisode se termine. Le robot reçoit 0 lorsqu'il loin de la position cible.

Lorsque le robot entre en collision avec des obstacles dans l'environnement, il est pénalisé par r_c^t donné par :

$$r_c^t = \begin{cases} r_{collision} & si \;\; l^n < d_{robot\_Radius} \\ 0 & si\ non \end{cases} \tag{88}$$

La définition de d_{robot_Radius} correspond au rayon d'un cercle qui a la même taille que le modèle du robot et qui est centré sur celui-ci auquel est ajoutée une marge de sécurité. Si au moins une des distances l^n aux obstacles est inférieure à d_{robot_Radius}, la condition de collision (*Equation (88)*) est remplie et l'épisode se termine.

Afin d'éviter les minima locaux, une petite pénalité r_m^t est ajoutée à chaque étape temporelle.

$$r_m^t = r_{time} \tag{89}$$

Le terme r_{time} présentée dans l'Equation (89) sert à éviter les rotations sur place et les manœuvres inutiles du robot pour optimiser le temps de trajet.

Pour évaluer l'impact des trois paramètres de récompense r, nous avons mené des expériences d'apprentissage en modifiant les valeurs de ces récompenses. Nous avons constaté que les résultats d'apprentissage étaient médiocres lorsque la valeur de r_{time} n'était pas nulle, et que les valeurs de $r_{collision}$ et $r_{arrivée}$ étaient trop basses, ou lorsque la valeur de $r_{collision}$ était supérieure à celle de $r_{arrivée}$. En conséquence, nous avons catégorisé les paramètres de récompense comme suit en fonction de leur taille : $r_{arrivée} \in \{100,\ 200\}$, $r_{collision} \in \{-50, -100\}$ et $r_{time} \in \{0.0, -0.1\}$.

6.5. Configuration de l'environnement DRL

6.5.1. Environnement d'apprentissage

La configuration de l'environnement de simulation de la méthode d'évitement d'obstacles dynamiques dans un lieu incertain est la suivante. L'environnement de la simulation dynamique en *3D* du robot a été construit à l'aide du simulateur *CoppeliaSim* contenant un moteur physique pour le calcul des interactions mécaniques entre les différents objets dans la scène. De plus, les codes logiciels étaient basés sur *Matlab* pour intégrer l'apprentissage et le contrôle. Pour exécuter cet environnement, nous avons utilisé le même ordinateur et le même modèle de robot comme mentionné dans le chapitre 4.

Nous avons construit un environnement intérieur de 10×10 m^2 avec des murs (*Figure (6.4)*). Huit obstacles dynamiques de forme cylindrique ont été ajoutés, chacun avec un diamètre de 0.25 m et une hauteur de 0.6 m. Ils ont été mis en rotation à des vitesses variables dans le sens inverse des aiguilles d'une montre avec différents rayons. Pour réduire le temps d'apprentissage en condition d'achèvement de la tâche de navigation, d_{goal} en Equation (87) a été fixé à 0.5 m.

Pour détecter la collision entre les obstacles et le robot, le d_{robot_Radius} dans l'Equation (88) a été fixé à 0.6 m, ce qui est une valeur supérieure à la longueur du modèle de robot pour assurer la sureté de mouvement.

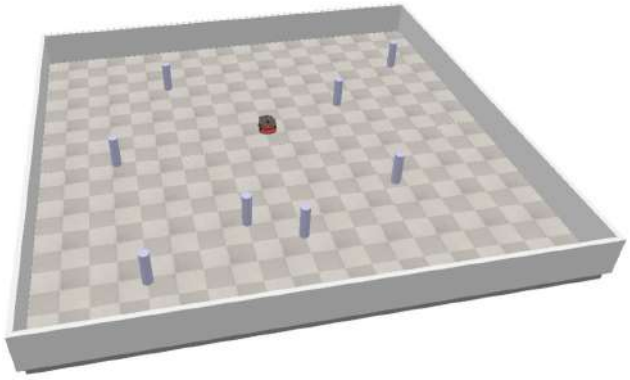


Figure 6.4 : *Environnements d'apprentissage.*

Nous avons utilisé les fonctions des boîtes à outils apprentissage par renforcement, *LiDAR*, navigation et systèmes robotiques de *Matlab* pour coder l'algorithme *DWA*. En

ajustant les paramètres du *DWA*, le système pourrait optimiser les performances d'évitement d'obstacles.

Les valeurs clés des hyperparamètres étaient les suivantes :

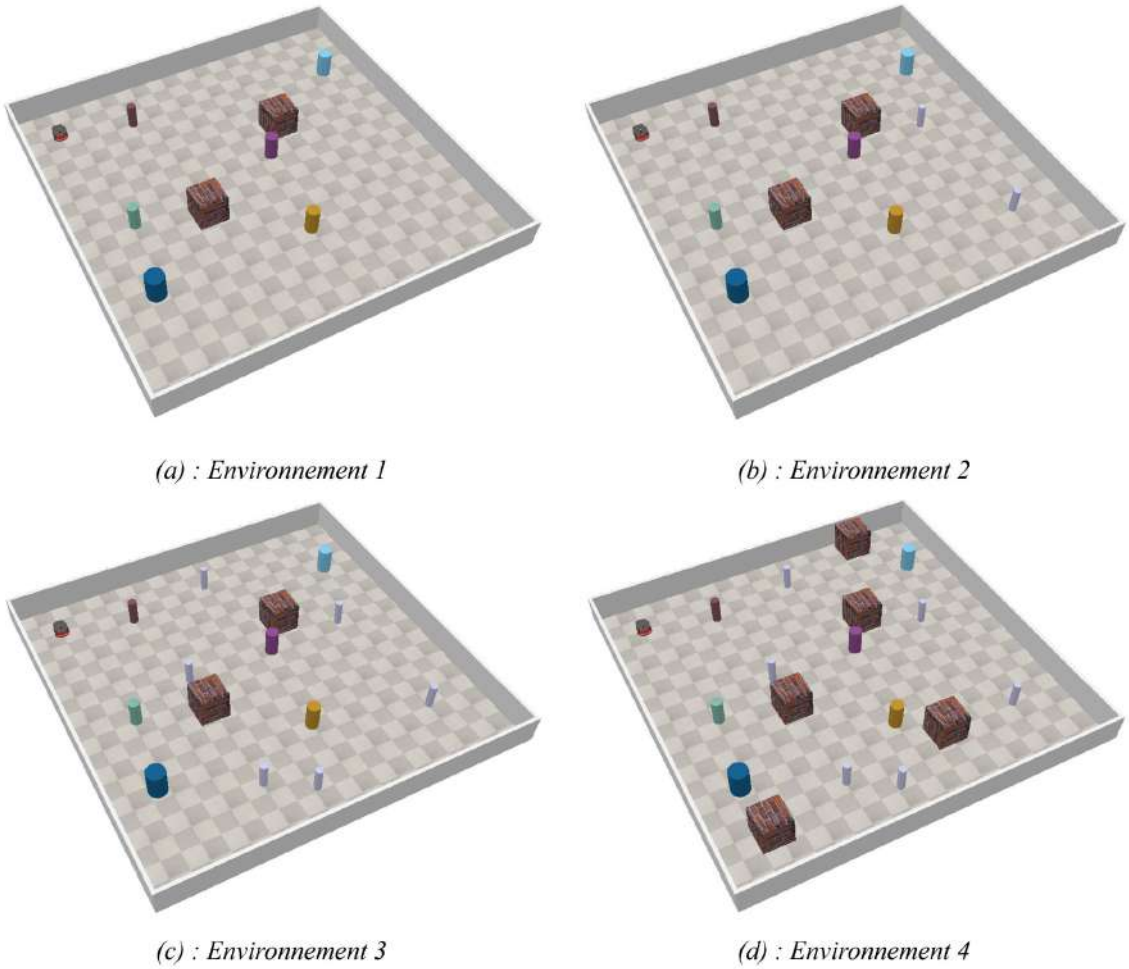
- Taux d'apprentissage, $L_r = 0.00025$,
- Décroissance $\varepsilon = 0.997$,
- Epsilon minimum $\varepsilon_{min} = 0.05$,
- Facteur de dévaluation $\gamma = 0.99$,
- Taille du lot d'apprentissage $b_s = 64$.

6.5.2. Environnement de test

Pour tester les performances de la méthode proposée dans d'autres environnements non appris, nous avons créé quatre environnements en variant la configuration de l'environnement d'apprentissage avec différents types d'obstacles dynamiques et statiques et différents niveaux de complexité :

- ***Environnement 1 (****Figure (6.5 a)****) :*** nous avons ajouté 2 obstacles statiques et 6 obstacles dynamiques avec différents diamètres tournant avec des vitesses ne dépassant pas 0.5 m/s.
- ***Environnement 2 (****Figure (6.5 b)****) :*** nous avons ajouté 8 obstacles dynamiques avec différents mouvements (*translation et rotation*) circulant à des vitesses ne dépassant pas 0.5 m/s en présence *de* 2 obstacles statiques.
- ***Environnement 3 (****Figure (6.5 c)****) :*** nous avons ajouté 12 obstacles dynamiques avec différents mouvements aléatoires circulant à des vitesses ne dépassant pas 0.5 m/s et 2 obstacles statiques.
- ***Environnement 4 (****Figure (6.5 d)****) :*** cet environnement est similaire au précédent sauf le nombre d'obstacles statiques insérés et les vitesses des obstacles dynamiques qui peuvent atteindre la valeur de 1 m/s.

Dans chaque expérience, 1000 épisodes ont été réalisés en utilisant la méthode *DWA* seule et la méthode proposée.



(a) : Environnement 1 (b) : Environnement 2

(c) : Environnement 3 (d) : Environnement 4

Figure 6.5 : *Environnements de test.*

6.6. Apprentissage et résultats de simulation

Pour démontrer l'efficacité de l'algorithme d'apprentissage utilisé et de l'effet des fonctions de récompenses, nous l'avons testé sur l'environnement d'apprentissage. Le déroulement de la phase d'apprentissage est le même que celui démontré dans la Figure (4.8), qui a été répétée 3000 fois dans cette simulation.

Les fonctions de récompenses testées sont données par le Tableau (6.1).

Tableau 6.1 : *Paramètres des fonctions de récompenses.*

	$r_{arrivée}$	$r_{collision}$	r_{time}
r_1	200	-50	-0.1
r_2	200	-50	0
r_3	100	-100	0
r_4	100	-100	-0.1

La Figure (6.6) montre les taux de réussite de la méthode *DWA+DQN* avec différentes combinaisons des constantes ($r_{arrivée}$, $r_{collision}$ et r_{time}) dans les fonctions de récompenses testées dans ce travail comparés au taux de réussite moyen sans collision de la méthode *DWA*. Dans cette figure, le taux de réussite moyen de la méthode *DWA* est *58.8 %*. Ce taux est obtenu après exécution de 3000 épisodes de l'algorithme *DWA* dans le même environnement avec les mêmes conditions. Pour la méthode *DWA+DQN*, le taux de réussite moyen est calculé chaque 200 épisodes.

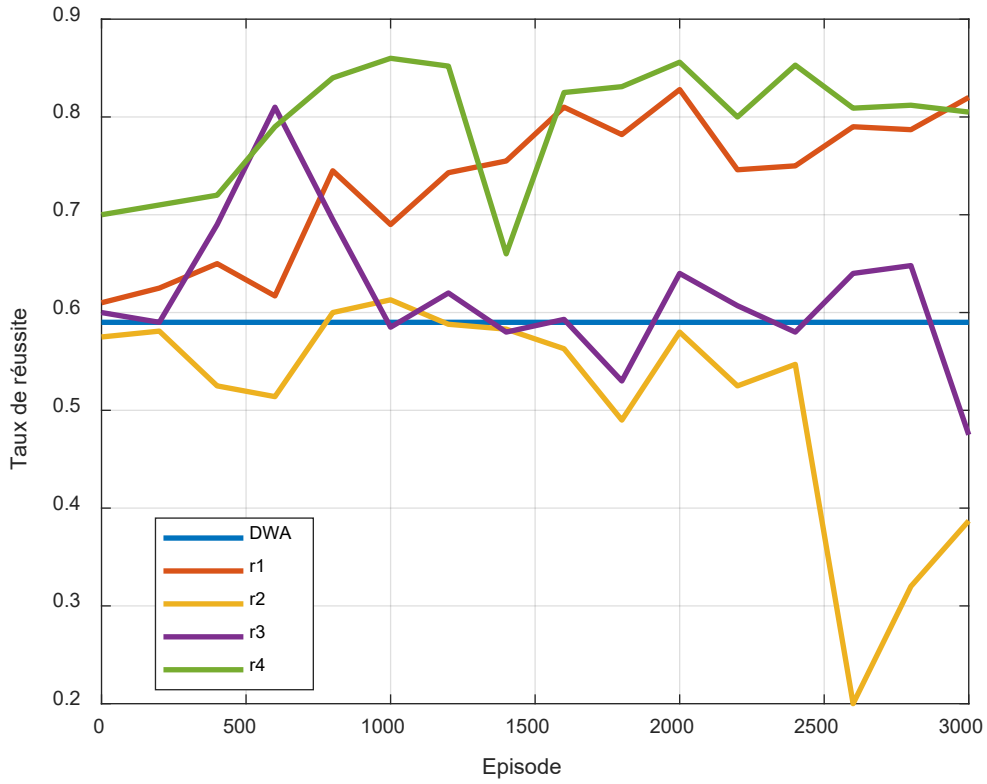


Figure 6.6 : *Taux de réussite en fonction des récompenses.*

Nous constatons que le facteur r_{time} a une grande influence sur le taux d'apprentissage et les performances de notre agent. Nous remarquons que les performances des fonctions de récompenses r_2 et r_3 sont inférieures à celles des fonctions r_1 et r_4 qui possèdent des taux de réussite supérieur à 80 %. De plus, par rapport au taux de réussite de la *DWA*, le taux de réussite de la récompense r_2 a progressivement diminué.

En divisant les fonctions de récompenses en deux groupes selon la valeur de r_{time}, nous remarquons que le choix des facteurs $r_{arrivée}$ et $r_{collision}$ affectent également les performances

d'évitement. Les récompenses r_3 et r_4, où la somme de $r_{arrivée}$ et $r_{collision}$ sont proches de *0* ont montré une meilleure amélioration des performances dans chaque groupe de fonctions. Tenant compte du taux de réussite d'évitement d'obstacles, la performance de la récompense r_4 appliquée avec deux facteurs positifs a montré le meilleur résultat sur les 3000 épisodes, la performance d'évitement d'obstacles a augmenté de 23,7 % par rapport à la méthode *DWA*.

Nous avons tenté de vérifier les résultats d'apprentissage par des expériences comparant la méthode de base *DWA* et la méthode *DWA+DQN*. Le robot utilisant les résultats d'apprentissage de la récompense r_4 a été testé dans des environnements avec plusieurs angles de départ, différentes vitesses de rotation d'obstacles et différents sens de rotation des obstacles. Les résultats ont prouvé que les performances du modèle entraîné étaient également valables dans d'autres environnements.

6.6.1. Test 1

Pour ce premier test, nous avons gardé le même environnement pour mener des expériences en modifiant les conditions telles que les vitesses et les angles de départ des obstacles. Grâce à ces tests préliminaires, il a été confirmé que la probabilité de collision augmentait lorsque les angles relatifs entre le robot et les obstacles les plus proches atteignaient certains degrés lors de l'opération initiale et lorsque les vitesses des obstacles sont augmentées. L'angle initial de chaque obstacle rotatif a été augmenté de 0° à 32° par incrément de 8°, et sa vitesse est variée de 0.2 à 0.9 m/s avec un incrément de 0.1 m/s. Chaque combinaison angle-vitesse a été testée 200 fois.

La *DWA* et la méthode *DWA+DQN* proposée ont été testées, et les résultats obtenus sont présentés sur la Figure (6.7). Lorsque la vitesse de l'obstacle a augmenté de 0.2 à 0.9 m/s, la capacité d'évitement globale a diminué. Par ce test, il a été validé que la méthode utilisant le module d'apprentissage avait un taux de réussite supérieur à la méthode *DWA* ce qui est justifié par le fait que la fonction d'évitement d'obstacles était entraînée pour éviter des obstacles en rotation avec différentes vitesses. Pour la plupart des vitesses d'obstacles, la différence entre la méthode proposée et la *DWA* dépasse 10 %.

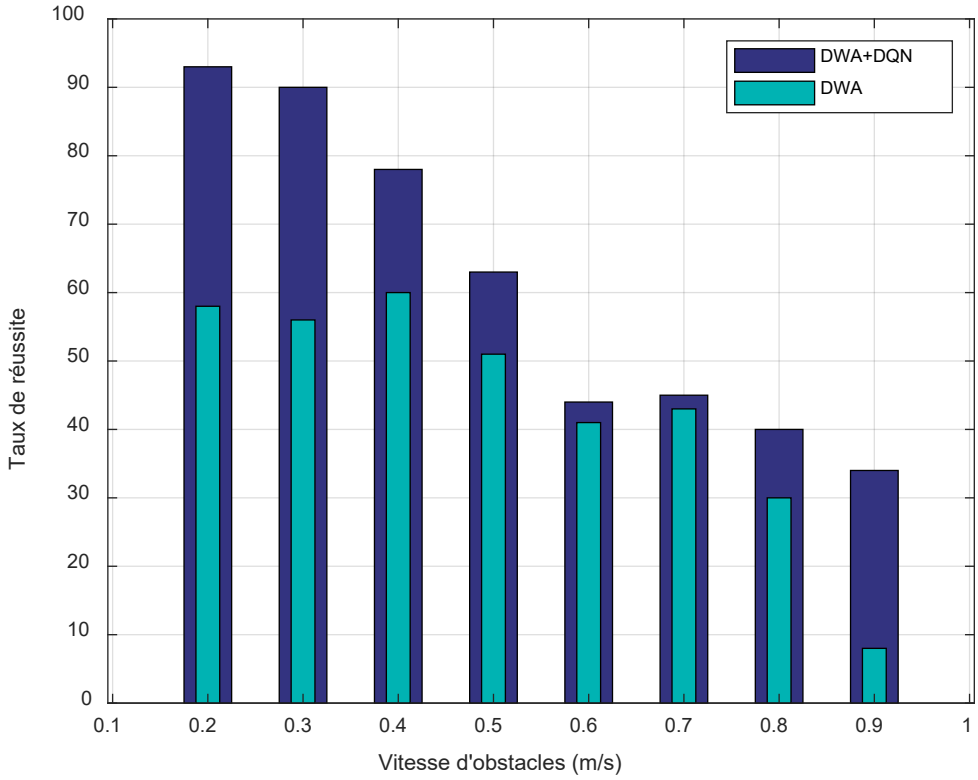


Figure 6.7 : *Taux moyens de réussite (Test 1).*

6.6.2. Test 2

Dans ce deuxième test, nous utiliserons les quatre environnements de test décrits précédemment (*voir Figure (6.5)*).

Les résultats de cette expérience ont confirmé que la performance de la méthode proposée était dominante dans tous les aspects, comme le montre la Figure (6.8). Nous remarquons que le taux de réussite diminue avec l'augmentation de la complexité de l'environnement.

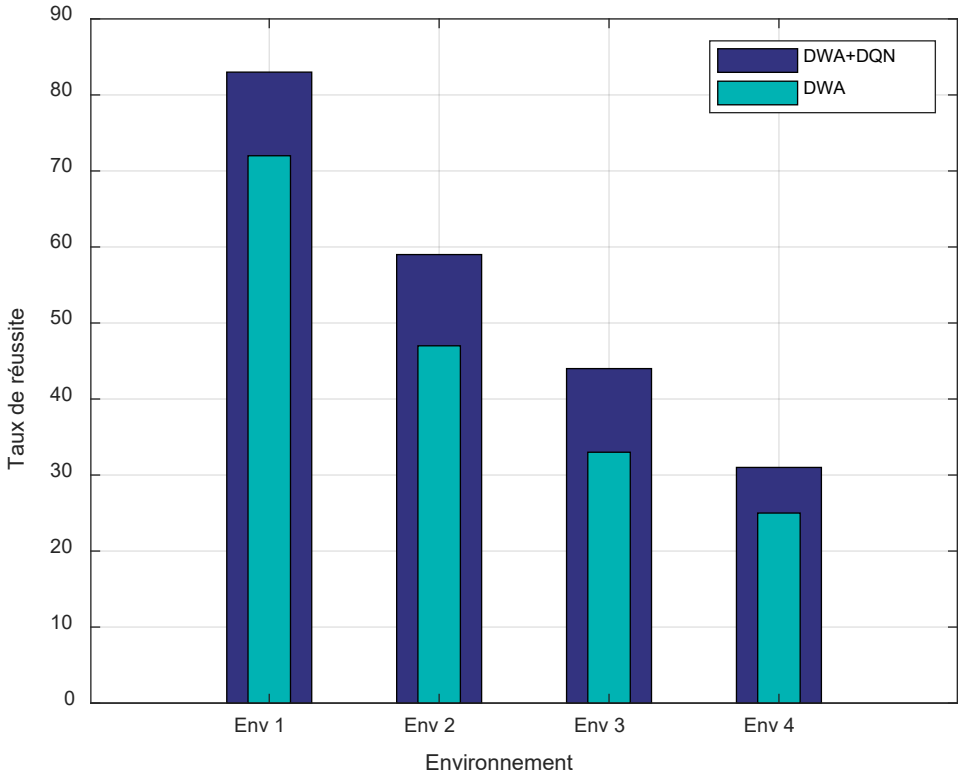


Figure 6.8 : *Taux moyens de réussite (Test 2).*

6.7. Conclusion

Dans ce chapitre, une méthode basée sur le *RL* pour éviter les obstacles dans un environnement dynamique et incertain a été présentée. En combinant les algorithmes *DWA* et *DQN*, un module de contrôle de vitesse a été développé pour éviter les obstacles dynamiques. Contrairement aux méthodes *RL* qui utilisent des actions discrètes ne prenant pas en compte la cinématique du robot, le module proposé sélectionne une valeur appropriée basée sur la vitesse calculée pour l'environnement proche, ce qui a amélioré les performances d'évitement d'obstacles. En outre, ce module est flexible et peut s'adapter aux changements dans l'environnement dynamique sans ajustement supplémentaire des paramètres. Les résultats des simulations ont confirmé que le taux de réussite en matière d'évitement d'obstacles étaient plus élevés pour le module utilisant la méthode *DWA+DQN* que pour celui utilisant *DWA* seule. L'efficacité du module dans des environnements incertains et non appris a été validée, et il a été confirmé que les performances de la méthode proposée étaient meilleures dans la plupart des cas.

Bien que l'utilisation d'un module de contrôle simple ait amélioré la méthode d'évitement classique, des limites existent pour la méthode proposée. En utilisant *DQN*, la vitesse de calcul est réduite en raison de la structure de la mémoire de relecture, ce qui rend difficile l'application d'actions continues lorsque la taille de l'agent augmente. Ainsi, nous prévoyons de poursuivre nos recherches pour améliorer les performances d'évitement d'obstacles en explorant d'autres méthodes, telles que l'optimisation de la politique proximale (*PPO*) ou un algorithme plus récent comme *A3C*.

7. Conclusion Générale

Dans cette section, nous conclurons la thèse en résumant les principaux résultats, puis nous présenterons quelques perspectives de recherche future pour compléter et améliorer ce travail.

Les robots mobiles jouent désormais un rôle important dans différents domaines, la capacité de navigation autonome dans des environnements dynamiques est l'une des compétences fondamentales. L'exigence selon laquelle le robot doit être capable de réagir aux changements des conditions environnementales a conduit au développement de contrôleurs robustes et fiables qui peuvent faire face à l'incertitude du monde. Etant donné qu'il est peu probable que nous puissions prévoir avec précision toutes les situations potentielles du monde réel, un robot préprogrammé ne peut pas répondre aux exigences futures, en particulier lorsqu'une interaction avec les humains est nécessaire. Le but de cette thèse a été de doter les robots mobiles d'une capacité d'apprentissage intelligent et adaptatif qui puisse répondre aux environnements dynamiques.

Dans cette thèse, nous nous sommes concentrés sur trois tâches élémentaires dans le cadre de la navigation intelligente de robot mobile à roues :

- ***Planification de chemins :*** un mode d'apprentissage progressif pour résoudre le problème de la planification de chemin basée sur l'apprentissage par renforcement profond (*DRL*) est proposé au *chapitre 4*. Nous avons conçu un algorithme basé sur le *DRL*, incluant les états d'observation, la fonction de récompense, la structure du réseau ainsi que l'optimisation des paramètres, dans un environnement *2D* pour éviter les tâches longues et fastidieuses liées à l'utilisation d'un environnement *3D*. Nous avons transféré l'algorithme conçu dans un environnement *3D* simple pour continuer l'apprentissage, afin d'obtenir les paramètres de réseau convergents, y compris les poids et les biais des réseaux de neurones profonds (*DNN*), etc. En utilisant ces paramètres comme valeurs initiales, l'apprentissage est poursuivi dans un environnement *3D* complexe. Pour améliorer la généralisation du modèle dans différents environnements, nous avons combiné l'algorithme *TD3* avec l'algorithme

PRM pour créer un planificateur de chemin plus efficace. L'efficacité du planificateur de chemin *PRM+TD3* et du mode d'apprentissage a été démontrée à travers des tests de simulation.

- ***Le suivi de chemin :*** nous avons opté pour l'utilisation d'un modèle d'inférence floue à connexion dynamique de modules de règles à entrées uniques (*SIRMs*) pour concevoir un contrôleur de suivi de chemin pour un robot mobile à roue. Le modèle *SIRMs* est utilisé pour réduire le nombre de règles d'inférence floue et réduire le temps de traitement des contrôleurs conventionnels à logique floue (*FLC*). La stratégie de suivi de chemin utilise deux unités de contrôle travaillant en parallèle pour conduire le robot mobile à suivre un chemin composé d'une succession de points discrets de passage. La structure du contrôleur est simple et les règles d'inférence sont intuitivement compréhensibles. Les simulations numériques ont montré que le contrôleur basé sur les *SIRMs* est plus performant qu'un contrôleur de suivi de chemin qui utilise le *FLC* conventionnel.
- ***L'évitement d'obstacles :*** nous avons combiné l'approche de la fenêtre dynamique (*DWA*) et l'apprentissage profond par renforcement pour construire un modèle de vitesse pour éviter les obstacles. Cette méthode supervise la *DWA* et essaye d'apporter des corrections aux vitesses linéaires et angulaires à appliquer au robot à l'aide de l'agent de renforcement conçu. Grâce à cette configuration, de nombreux mouvements robotiques peuvent être générés même avec des fonctions d'action limitées. Dans nos expériences, l'application de ce module d'apprentissage a montré un taux d'évitement d'obstacles de 23,7% plus élevé qu'avec la *DWA* seule. Les résultats de simulations ont confirmé que la méthode proposée améliore les performances de l'évitement d'obstacles pour de multiples environnements dynamiques sans aucun travail supplémentaire.

Perspectives

- ***Premièrement,*** cette thèse a établi les bases théoriques de l'apprentissage des robots mobiles et a obtenu des résultats fiables dans des simulateurs. Afin de faire progresser l'apprentissage du robot, il serait opportun de réaliser des tests sur des robots réels.

- ***Deuxièmement,*** un système de contrôle intelligent de robot repose sur les différents capteurs qui capturent les informations environnementales. Avec le développement des capteurs de robot, un robot devrait être capable d'apprendre une stratégie de contrôle à partir de la grande variété de données sensorielles, en particulier celles des entrées visuelles et auditives. La capacité d'apprendre à partir d'une complexité de données de capteur non seulement dotera le robot d'intelligence mais améliorera également l'interaction du robot avec les humains.
- ***Troisièmement,*** il est important de développer l'apprentissage des robots dans le cadre de systèmes multirobots. Etant donné que de plus en plus de tâches seront effectuées par un groupe de robots, il est essentiel de continuer à développer les techniques d'apprentissage afin de prédire le comportement des robots et de faciliter leur interaction.
- ***Enfin,*** en tant que membre de la famille de l'apprentissage profond, le *CNN* a montré son efficacité dans la reconnaissance d'images et la vision par ordinateur. Ainsi, les recherches sur l'apprentissage par renforcement profond sont susceptibles de susciter de nouvelles percées dans le domaine de la robotique.

Bibliographie

1. Alatise and Hancke *A Review on Challenges of Autonomous Mobile Robot and Sensor Fusion Methods*. IEEE Access, 2020. **8**, 39830-39846 DOI: 10.1109/ACCESS.2020.2975643.
2. Loganathan and Ahmad *A systematic review on recent advances in autonomous mobile robot navigation*. Engineering Science and Technology, an International Journal, 2023. **40**, 101343 DOI: https://doi.org/10.1016/j.jestch.2023.101343.
3. Li and Ding *Adaptive and intelligent robot task planning for home service: A review*. Engineering Applications of Artificial Intelligence, 2023. **117**, 105618 DOI: https://doi.org/10.1016/j.engappai.2022.105618.
4. Ntakolia, Moustakidis, and Siouras *Autonomous path planning with obstacle avoidance for smart assistive systems*. Expert Systems with Applications, 2023. **213**, 119049 DOI: https://doi.org/10.1016/j.eswa.2022.119049.
5. Cebollada*, et al. A state-of-the-art review on mobile robotics tasks using artificial intelligence and visual data*. Expert Systems with Applications, 2021. **167**, 114195 DOI: https://doi.org/10.1016/j.eswa.2020.114195.
6. Roy and Chowdhury *A Survey of Machine Learning Techniques for Indoor Localization and Navigation Systems*. Journal of Intelligent & Robotic Systems, 2021. **101**, 63 DOI: 10.1007/s10846-021-01327-z.
7. Zhao and Ning *Hybrid Navigation Method for Multiple Robots Facing Dynamic Obstacles*. Tsinghua Science and Technology, 2022. **27**, 894-901 DOI: 10.26599/TST.2021.9010073.
8. Zhou*, et al. Learn to Navigate: Cooperative Path Planning for Unmanned Surface Vehicles Using Deep Reinforcement Learning*. IEEE Access, 2019. **7**, 165262-165278 DOI: 10.1109/ACCESS.2019.2953326.
9. Zeng, Wang, and Ge *A Survey on Visual Navigation for Artificial Agents With Deep Reinforcement Learning*. IEEE Access, 2020. **8**, 135426-135442 DOI: 10.1109/ACCESS.2020.3011438.
10. Nahavandi*, et al.*, *A Comprehensive Review on Autonomous Navigation*. 2022.
11. Khan, Bahrami, and Anwar. *A Deep Learning Based Autonomous Mobile Robotic Assistive Care Giver*. in *2019 IEEE International Conference on E-health Networking, Application & Services (HealthCom)*. 2019.
12. Surmann*, et al. Deep Reinforcement learning for real autonomous mobile robot navigation in indoor environments*. ArXiv, 2020. **abs/2005.13857**.
13. Rafai, Adzhar, and Jaini *A Review on Path Planning and Obstacle Avoidance Algorithms for Autonomous Mobile Robots*. Journal of Robotics, 2022. **2022**, 2538220 DOI: 10.1155/2022/2538220.
14. Liu*, et al. Robot Navigation in Crowded Environments Using Deep Reinforcement Learning*. in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2020.
15. Sun, Zhang, Yu, and Zhang *Motion Planning for Mobile Robots—Focusing on Deep Reinforcement Learning: A Systematic Review*. IEEE Access, 2021. **9**, 69061-69081 DOI: 10.1109/ACCESS.2021.3076530.
16. Gao, Ye, Guo, and Li *Deep Reinforcement Learning for Indoor Mobile Robot Path Planning*. Sensors, 2020. **20**, DOI: 10.3390/s20195493.
17. Caruso*, et al. Robot Navigation in Crowded Environments: A Reinforcement Learning Approach*. Machines, 2023. **11**, DOI: 10.3390/machines11020268.
18. Patle*, et al. A review: On path planning strategies for navigation of mobile robot*. Defence Technology, 2019. **15**, 582-606 DOI: https://doi.org/10.1016/j.dt.2019.04.011.
19. Benmakhlouf, Meraoumia, and Louazene *Mobile Robot Path Following Controller Based On the Sirms Dynamically Connected Fuzzy Inference Model*. International Journal of Integrated Engineering, 2023. **15**, 233 - 247 DOI: https://doi.org/10.30880/ijie.2023.15.01.021.
20. Giralt, Sobek, and Chatila. *A Multi-level Planning and Navigation System for a Mobile Robot: A First Approach to HILARE*. in *International Joint Conference on Artificial Intelligence*. 1979.

21. Montello and Sas, *Human Factors of Wayfinding in Navigation*, in *International Encyclopedia of Ergonomics and Human Factors - 3 Volume Set*, W.K. Informa Healthcare, Editor. 2006, CRC Press: Boca Raton.
22. Knudson and Tumer *Adaptive navigation for autonomous robots*. Robotics and Autonomous Systems, 2011. **59**, 410-420 DOI: https://doi.org/10.1016/j.robot.2011.02.004.
23. Zhu and Zhang *Deep reinforcement learning based mobile robot navigation: A review*. Tsinghua Science and Technology, 2021. **26**, 674-691 DOI: 10.26599/TST.2021.9010012.
24. Ruan, Ren, Zhu, and Huang. *Mobile Robot Navigation based on Deep Reinforcement Learning*. in *2019 Chinese Control And Decision Conference (CCDC)*. 2019.
25. Tsai, Nisar, and Hu *Mapless LiDAR Navigation Control of Wheeled Mobile Robots Based on Deep Imitation Learning*. IEEE Access, 2021. **9**, 117527-117541 DOI: 10.1109/ACCESS.2021.3107041.
26. Zeng, *Learning Continuous Control through Proximal Policy Optimization for Mobile Robot Navigation*. 2018.
27. Tai, Paolo, and Liu. *Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation*. in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2017.
28. Cèsar-Tondreau*, et al. Improving Autonomous Robotic Navigation Using Imitation Learning*. Frontiers in Robotics and AI, 2021. **8**, DOI: 10.3389/frobt.2021.627730.
29. Dong, Wei, Liu, and Kan *A Novel Loop Closure Detection Method Using Line Features*. IEEE Access, 2019. **7**, 111245-111256 DOI: 10.1109/ACCESS.2019.2934521.
30. Sutton and Barto, *Reinforcement Learning: An Introduction*. 2018: A Bradford Book.
31. Schrittwieser*, et al. Mastering Atari, Go, chess and shogi by planning with a learned model*. Nature, 2020. **588**, 604-609 DOI: 10.1038/s41586-020-03051-4.
32. Sánchez-Ibáñez, Pérez-del-Pulgar, and García-Cerezo *Path Planning for Autonomous Mobile Robots: A Review*. Sensors, 2021. **21**, 7898.
33. Chen and Hwang *SANDROS: a dynamic graph search algorithm for motion planning*. IEEE Transactions on Robotics and Automation, 1998. **14**, 390-403 DOI: 10.1109/70.678449.
34. Karaman and Frazzoli *Sampling-based algorithms for optimal motion planning*. The International Journal of Robotics Research, 2011. **30**, 846-894 DOI: 10.1177/0278364911406761.
35. Chen. *A Simulative Bionic Intelligent Optimization Algorithm: Artificial Searching Swarm Algorithm and Its Performance Analysis*. in *2009 International Joint Conference on Computational Sciences and Optimization*. 2009.
36. Broumi*, et al. Applying Dijkstra algorithm for solving neutrosophic shortest path problem*. in *2016 International Conference on Advanced Mechatronic Systems (ICAMechS)*. 2016.
37. Duchoň*, et al. Path Planning with Modified a Star Algorithm for a Mobile Robot*. Procedia Engineering, 2014. **96**, 59-69 DOI: https://doi.org/10.1016/j.proeng.2014.12.098.
38. Meng, Johansson, and Dimarogonas. *Revising motion planning under Linear Temporal Logic specifications in partially known workspaces*. in *2013 IEEE International Conference on Robotics and Automation*. 2013.
39. Yu and LaValle. *Planning optimal paths for multiple robots on graphs*. in *2013 IEEE International Conference on Robotics and Automation*. 2013.
40. Quigley. *ROS: an open-source Robot Operating System*. in *IEEE International Conference on Robotics and Automation*. 2009.
41. Gammell, Srinivasa, and Barfoot. *Batch Informed Trees (BIT*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs*. in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. 2015.
42. Choudhury*, et al. Regionally accelerated batch informed trees (RABIT*): A framework to integrate local information into optimal path planning*. 2016 IEEE International Conference on Robotics and Automation (ICRA), 2016. 4207-4214.
43. Wang and Meng. *Variant step size RRT: An efficient path planner for UAV in complex environments*. in *2016 IEEE International Conference on Real-time Computing and Robotics (RCAR)*. 2016.

44. Chi, Wang, Wang, and Meng *Risk-DTRRT-Based Optimal Motion Planning Algorithm for Mobile Robots*. IEEE Transactions on Automation Science and Engineering, 2019. **16**, 1271-1288 DOI: 10.1109/TASE.2018.2877963.
45. Cai*, et al.*, *Mobile Robot Path Planning in Dynamic Environments: A Survey*. 2020.
46. Yanrong and Yang. *A knowledge based genetic algorithm for path planning of a mobile robot*. in *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*. 2004.
47. Xue*, et al. Robot path planning based on improved ant colony algorithm*. in *2021 Power System and Green Energy Conference (PSGEC)*. 2021.
48. Liu, Xu, Lu, and Zhang *A path planning approach for crowd evacuation in buildings based on improved artificial bee colony algorithm*. Applied Soft Computing, 2018. **68**, 360-376 DOI: https://doi.org/10.1016/j.asoc.2018.04.015.
49. Tharwat*, et al. Intelligent Bézier curve-based path planning model using Chaotic Particle Swarm Optimization algorithm*. Cluster Computing, 2019. **22**, 4745-4766 DOI: 10.1007/s10586-018-2360-3.
50. Le Gouguec, Kemeny, Berthoz, and Merienne. *Artificial Potential Field Simulation Framework for Semi-Autonomous Car Conception*. in *Driving Simulation Conference*. 2017. Stuttgart, Germany.
51. Bakdi*, et al. Optimal path planning and execution for mobile robots using genetic algorithm and adaptive fuzzy-logic control*. Robotics and Autonomous Systems, 2017. **89**, 95-109 DOI: https://doi.org/10.1016/j.robot.2016.12.008.
52. Turker, Sahingoz, and Yilmaz. *2D path planning for UAVs in radar threatening environment using simulated annealing algorithm*. in *2015 International Conference on Unmanned Aircraft Systems (ICUAS)*. 2015.
53. Petrović, Vuković, Mitić, and Miljković *Integration of process planning and scheduling using chaotic particle swarm optimization algorithm*. Expert Systems with Applications, 2016. **64**, 569-588 DOI: https://doi.org/10.1016/j.eswa.2016.08.019.
54. Paulo, Branco, de Brito, and Silva *BuildingsLife – The use of genetic algorithms for maintenance plan optimization*. Journal of Cleaner Production, 2016. **121**, 84-98 DOI: https://doi.org/10.1016/j.jclepro.2016.02.041.
55. Berg, Ming, and Manocha. *Reciprocal Velocity Obstacles for real-time multi-agent navigation*. in *2008 IEEE International Conference on Robotics and Automation*. 2008.
56. Fiorini and Shiller *Motion Planning in Dynamic Environments Using Velocity Obstacles*. The International Journal of Robotics Research, 1998. **17**, 760-772 DOI: 10.1177/027836499801700706.
57. Snape, Berg, Guy, and Manocha. *Independent navigation of multiple mobile robots with hybrid reciprocal velocity obstacles*. in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2009.
58. Bloembergen, Tuyls, Hennes, and Kaisers *Evolutionary dynamics of multi-agent learning: a survey*. J. Artif. Int. Res., 2015. **53**, 659–697.
59. Claes, Hennes, Tuyls, and Meeussen. *Collision avoidance under bounded localization uncertainty*. in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2012.
60. Alonso-Mora*, et al.*, *Optimal Reciprocal Collision Avoidance for Multiple Non-Holonomic Robots*, in *Distributed Autonomous Robotic Systems: The 10th International Symposium*, A. Martinoli, et al., Editors. 2013, Springer Berlin Heidelberg: Berlin, Heidelberg. p. 203-216.
61. Berg, Snape, Guy, and Manocha. *Reciprocal collision avoidance with acceleration-velocity obstacles*. in *2011 IEEE International Conference on Robotics and Automation*. 2011.
62. Guy*, et al.*, *ClearPath: highly parallel collision avoidance for multi-agent simulation*, in *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 2009, Association for Computing Machinery: New Orleans, Louisiana. p. 177–187.
63. Hennes, Claes, Meeussen, and Tuyls, *Multi-robot collision avoidance with localization uncertainty*, in *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems - Volume 1*. 2012, International Foundation for Autonomous Agents and Multiagent Systems: Valencia, Spain. p. 147–154.

64. Hasselt, Guez, and Silver, *Deep reinforcement learning with double Q-Learning*, in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*. 2016, AAAI Press: Phoenix, Arizona. p. 2094–2100.
65. Xu, Fang, Zhang, and Meng. *Path Planning Method Combining Depth Learning and Sarsa Algorithm*. in *2017 10th International Symposium on Computational Intelligence and Design (ISCID)*. 2017.
66. Peng and Williams *Incremental multi-step Q-learning*. Machine Learning, 1996. **22**, 283-290 DOI: 10.1007/BF00114731.
67. Kuderer, *Socially compliant mobile robot navigation*. 2015.
68. Vasquez, Okal, and Arras. *Inverse Reinforcement Learning algorithms and features for robot navigation in crowds: An experimental comparison*. in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2014.
69. Chen, Everett, Liu, and How *Socially aware motion planning with deep reinforcement learning*. 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2017. 1343-1350.
70. Chen, Liu, Everett, and How. *Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning*. in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. 2017.
71. Everett, Chen, and How. *Motion Planning Among Dynamic, Decision-Making Agents with Deep Reinforcement Learning*. in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2018.
72. Ciou*, et al. Composite Reinforcement Learning for Social Robot Navigation*. in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2018.
73. Long*, et al.*, *Towards Optimally Decentralized Multi-Robot Collision Avoidance via Deep Reinforcement Learning*, in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. 2018, IEEE Press: Brisbane, Australia. p. 6252–6259.
74. Soetanto, Lapierre, and Pascoal. *Adaptive, non-singular path-following control of dynamic wheeled robots*. in *42nd IEEE International Conference on Decision and Control (IEEE Cat. No.03CH37475)*. 2003.
75. Morin and Samson, *Motion Control of Wheeled Mobile Robots*, in *Springer Handbook of Robotics*, B. Siciliano and O. Khatib, Editors. 2008, Springer Berlin Heidelberg: Berlin, Heidelberg. p. 799-826.
76. Soueres, Hamel, and Cadenat. *A path following controller for wheeled robots which allows to avoid obstacles during transition phase*. in *Proceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. No.98CH36146)*. 1998.
77. Lapierre, Zapata, and Lépinay *Simulatneous Path Following and Obstacle Avoidance Control of a Unicycle-type Robot*. Proceedings 2007 IEEE International Conference on Robotics and Automation, 2007. 2617-2622.
78. Campani*, et al. A minimalist approach to path following among unknown obstacles*. in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2010.
79. Consolini, Morbidi, Prattichizzo, and Tosques, *Leader–follower formation control of nonholonomic mobile robots with input constraints.* Automatica, 2008. **44**(5): p. 1343-1349.
80. Ghommam, Mehrjerdi, Saad, and Mnif *Formation path following control of unicycle-type mobile robots*. Robotics and Autonomous Systems, 2010. **58**, 727-736 DOI: https://doi.org/10.1016/j.robot.2009.10.007.
81. Mazur *Hybrid adaptive control laws solving a path following problem for non-holonomic mobile manipulators*. International Journal of Control, 2004. **77**, 1297-1306 DOI: 10.1080/00207170420002971 62.
82. Samson *Time-varying Feedback Stabilization of Car-like Wheeled Mobile Robots*. The International Journal of Robotics Research, 1993. **12**, 55-64 DOI: 10.1177/027836499301200104.
83. Samson *Control of chained systems application to path following and time-varying point-stabilization of mobile robots*. IEEE Transactions on Automatic Control, 1995. **40**, 64-77 DOI: 10.1109/9.362899.

84. Krstic, Kokotovic, and Kanellakopoulos, *Nonlinear and Adaptive Control Design*. 1995: John Wiley & Sons, Inc.
85. Keighobadi, Sadeghi, and Fazeli *Dynamic Sliding Mode Controller for Trajectory Tracking Of Nonholonomic Mobile Robots*. IFAC Proceedings Volumes, 2011. **44**, 962-967 DOI: https://doi.org/10.3182/20110828-6-IT-1002.03048.
86. Salgado, Cruz-Ortiz, Camacho, and Chairez *Output feedback control of a skid-steered mobile robot based on the super-twisting algorithm*. Control Engineering Practice, 2017. **58**, 193-203 DOI: https://doi.org/10.1016/j.conengprac.2016.10.003.
87. youssef, Martins, De Pieri, and Moreno *PD-Super-Twisting Second Order Sliding Mode Tracking Control for a Nonholonomic Wheeled Mobile Robot*. IFAC Proceedings Volumes, 2014. **47**, 3827-3832 DOI: https://doi.org/10.3182/20140824-6-ZA-1003.02210.
88. Matraji*, et al. Trajectory tracking control of Skid-Steered Mobile Robot based on adaptive Second Order Sliding Mode Control*. Control Engineering Practice, 2018. **72**, 167-176 DOI: https://doi.org/10.1016/j.conengprac.2017.11.009.
89. Huang, Wen, Wang, and Jiang *Adaptive output feedback tracking control of a nonholonomic mobile robot*. Automatica, 2014. **50**, 821-831 DOI: https://doi.org/10.1016/j.automatica.2013.12.036.
90. Gonzalez*, et al. Adaptive Control for a Mobile Robot Under Slip Conditions Using an LMI-Based Approach*. European Journal of Control, 2010. **16**, 144-155 DOI: https://doi.org/10.3166/ejc.16.144-155.
91. Martins*, et al. An adaptive dynamic controller for autonomous mobile robot trajectory tracking*. Control Engineering Practice, 2008. **16**, 1354-1363 DOI: https://doi.org/10.1016/j.conengprac.2008.03.004.
92. Huang, Zhai, Ai, and Fei *Disturbance observer-based robust control for trajectory tracking of wheeled mobile robots*. Neurocomputing, 2016. **198**, 74-79 DOI: https://doi.org/10.1016/j.neucom.2015.11.099.
93. Merabti, Belarbi, and Bouchemal *Nonlinear predictive control of a mobile robot: a solution using metaheuristcs*. Journal of the Chinese Institute of Engineers, 2016. **39**, 282-290 DOI: 10.1080/02533839.2015.1091276.
94. Kim, Kim, Choi, and Jin Kim *Path Tracking for a Skid-steer Vehicle using Model Predictive Control with On-line Sparse Gaussian Process*. IFAC-PapersOnLine, 2017. **50**, 5755-5760 DOI: https://doi.org/10.1016/j.ifacol.2017.08.1140.
95. Škrjanc and Klančar *A comparison of continuous and discrete tracking-error model-based predictive control for mobile robots*. Robotics and Autonomous Systems, 2017. **87**, 177-187 DOI: https://doi.org/10.1016/j.robot.2016.09.016.
96. Ye, *Tracking control of two-wheel driven mobile robot using compound sine function neural networks*. Connection Science, 2013. **25**(2-3): p. 139-150.
97. Ye *Tracking control of a non-holonomic wheeled mobile robot using improved compound cosine function neural networks*. International Journal of Control, 2015. **88**, 364-373 DOI: 10.1080/00207179.2014.953590.
98. Li*, et al. Adaptive neural network tracking control-based reinforcement learning for wheeled mobile robots with skidding and slipping*. Neurocomputing, 2018. **283**, 20-30 DOI: https://doi.org/10.1016/j.neucom.2017.12.051.
99. Ahmed and Petrov *Trajectory Control of Mobile Robots using Type-2 Fuzzy-Neural PID Controller*. IFAC-PapersOnLine, 2015. **48**, 138-143 DOI: https://doi.org/10.1016/j.ifacol.2015.12.071.
100. Nazari and Naraghi. *Sliding mode fuzzy control of a skid steer mobile robot for path following*. in *2008 10th International Conference on Control, Automation, Robotics and Vision*. 2008.
101. Campa, Gordillo, and Soto *Speed and point-to-point control for trajectory tracking of a Skid-Steered Mobile Robot*. 11th IEEE International Conference on Control & Automation (ICCA), 2014. 637-642.
102. Lepej, Maurer, Uran, and Steinbauer *Dynamic Arc Fitting Path Follower for Skid-Steered Mobile Robots*. International Journal of Advanced Robotic Systems, 2015. **12**, 139 DOI: 10.5772/61199.

103. Moustris and Tzafestas *Switching fuzzy tracking control for mobile robots under curvature constraints*. Control Engineering Practice, 2011. **19**, 45-53 DOI: https://doi.org/10.1016/j.conengprac.2010.08.008.
104. Guzmán, Silva-Ortigoza, and Márquez-Sánchez *A PD path-tracking controller plus inner velocity loops for a wheeled mobile robot*. Advanced Robotics, 2015. **29**, 1015 - 1029.
105. Sordalen and Wit *Exponential control law for a mobile robot: extension to path following*. IEEE Transactions on Robotics and Automation, 1993. **9**, 837-842 DOI: 10.1109/70.265927.
106. Deng*, et al. ImageNet: A large-scale hierarchical image database*. in *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009.
107. Courville, *Deep Learning*. 2016: MIT Press.
108. A, *Neural Networks and Deep Learning*. 2018: Determination Press.
109. McCulloch and Pitts, *A logical calculus of the ideas immanent in nervous activity.* The bulletin of mathematical biophysics, 1943. **5**(4): p. 115-133.
110. Lecun, Bottou, Bengio, and Haffner *Gradient-based learning applied to document recognition*. Proceedings of the IEEE, 1998. **86**, 2278-2324 DOI: 10.1109/5.726791.
111. Simonyan and Zisserman, *Very deep convolutional networks for large-scale image recognition*. 2015, Computational and Biological Learning Society. p. 1-14.
112. He, Zhang, Ren, and Sun. *Deep Residual Learning for Image Recognition*. in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.
113. Szegedy*, et al. Going deeper with convolutions*. in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015.
114. Qian *On the momentum term in gradient descent learning algorithms*. Neural Networks, 1999. **12**, 145-151 DOI: https://doi.org/10.1016/S0893-6080(98)00116-6.
115. Kingma and Ba, *Adam: A Method for Stochastic Optimization*. 2015.
116. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley Series in Probability and Statistics. 1994.
117. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley Series in Probability and Statistics. 1994: Wiley. 1-649.
118. Sondik *The Optimal Control of Partially Observable Markov Process over the Infinite Horizon: Discounted Costs*. Operations Research, 1978. **26**, 282-304 DOI: 10.1287/opre.26.2.282.
119. Kaelbling, Littman, and Cassandra *Planning and acting in partially observable stochastic domains*. Artificial Intelligence, 1998. **101**, 99-134 DOI: https://doi.org/10.1016/S0004-3702(98)00023-X.
120. Bertsekas, *Dynamic Programming and Optimal Control*. 2000: Athena Scientific.
121. Mnih*, et al. Playing Atari with Deep Reinforcement Learning*. CoRR, 2013. **abs/1312.5602**.
122. Adam, Busoniu, and Babuska *Experience Replay for Real-Time Reinforcement Learning Control*. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), 2012. **42**, 201-212 DOI: 10.1109/TSMCC.2011.2106494.
123. Mnih*, et al. Human-level control through deep reinforcement learning*. Nature, 2015. **518**, 529-533 DOI: 10.1038/nature14236.
124. Van Hasselt, *Double Q-learning*. 2010. 2613-2621.
125. Van Hasselt, Guez, and Silver *Deep Reinforcement Learning with Double Q-Learning*. Proceedings of the AAAI Conference on Artificial Intelligence, 2015. **30**, DOI: 10.1609/aaai.v30i1.10295.
126. Schaul, Quan, Antonoglou, and Silver, *Prioritized Experience Replay*. 2016.
127. Andrychowicz*, et al.*, *Hindsight experience replay*, in *Proceedings of the 31st International Conference on Neural Information Processing Systems*. 2017, Curran Associates Inc.: Long Beach, California, USA. p. 5055–5065.
128. Todorov, Erez, and Tassa. *MuJoCo: A physics engine for model-based control*. in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2012.
129. Osband, Blundell, Pritzel, and Roy, *Deep exploration via bootstrapped DQN*, in *Proceedings of the 30th International Conference on Neural Information Processing Systems*. 2016, Curran Associates Inc.: Barcelona, Spain. p. 4033–4041.

130. Wang*, et al.*, *Dueling network architectures for deep reinforcement learning*, in *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*. 2016, JMLR.org: New York, NY, USA. p. 1995–2003.
131. Hessel*, et al.*, *Rainbow: combining improvements in deep reinforcement learning*, in *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*. 2018, AAAI Press: New Orleans, Louisiana, USA. p. Article 393.
132. Williams *Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning*. Mach. Learn., 1992. **8**, 229–256 DOI: 10.1007/bf00992696.
133. Schulman. *Policy Gradient Methods*. 2017; Available from: http://rll.berkeley.edu/deeprlcourse/docs/lec2.pdf.
134. Silver. *Lectures on Reinforcement Learning.* 2015; Available from: https://www.davidsilver.uk/teaching/.
135. Levine. *Deep Reinforcement Learning lecture*. 2017; Available from: http://rll.berkeley.edu/deeprlcourse/.
136. Silver*, et al.*, *Deterministic Policy Gradient Algorithms*, in *Proceedings of the 31st International Conference on Machine Learning*, P.X. Eric and J. Tony, Editors. 2014, PMLR: Proceedings of Machine Learning Research. p. 387--395.
137. Lillicrap*, et al.*, *Continuous control with deep reinforcement learning*. 2016.
138. Duan*, et al.*, *Benchmarking Deep Reinforcement Learning for Continuous Control.* CoRR, 2016. **abs/1604.06778**.
139. Henderson*, et al. Deep Reinforcement Learning That Matters*. Proceedings of the AAAI Conference on Artificial Intelligence, 2017. **32**, DOI: 10.1609/aaai.v32i1.11694.
140. Heess, Hunt, Lillicrap, and Silver *Memory-based control with recurrent neural networks*. ArXiv, 2015. **abs/1512.04455**.
141. Gabriel*, et al.*, *Distributed Distributional Deterministic Policy Gradients*, in *6th International Conference on Learning Representations*. 2018: Vancouver Convention Center, Vancouver, BC, Canada.
142. Jaderberg*, et al. Reinforcement Learning with Unsupervised Auxiliary Tasks*. 2016.
143. Wang*, et al. Sample Efficient Actor-Critic with Experience Replay*. CoRR, 2016. **abs/1611.01224**.
144. Espeholt*, et al.*, *IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures*. 2018. p. 1406-1415.
145. Mnih*, et al.*, *Asynchronous methods for deep reinforcement learning*, in *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*. 2016, JMLR.org: New York, NY, USA. p. 1928–1937.
146. John*, et al.*, *Trust Region Policy Optimization*. PMLR. p. 1889-1897.
147. Wu*, et al.*, *Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation*, in *Proceedings of the 31st International Conference on Neural Information Processing Systems*. 2017, Curran Associates Inc.: Long Beach, California, USA. p. 5285–5294.
148. Schulman*, et al.*, *Proximal Policy Optimization Algorithms.* CoRR, 2017. **abs/1707.06347**.
149. Haarnoja, Tang, Abbeel, and Levine, *Reinforcement learning with deep energy-based policies*, in *Proceedings of the 34th International Conference on Machine Learning - Volume 70*. 2017, JMLR.org: Sydney, NSW, Australia. p. 1352–1361.
150. Haarnoja, Zhou, Abbeel, and Levine, *Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor*. 2018. p. 1856-1865.
151. Ahmed, Le Roux, Norouzi, and Schuurmans. *Understanding the impact of entropy on policy optimization*. in *International Conference on Machine Learning*. 2018.
152. Wu*, et al.*, *Learning Improvement Heuristics for Solving Routing Problems.* IEEE Transactions on Neural Networks and Learning Systems, 2022. **33**(9): p. 5057-5069.
153. Injarapu and Gawre, *A survey of autonomous mobile robot path planning approaches*. 2017. 624-628.

154. Zhang, Zheng, and Cen *Present situation and future development of mobile robot path planning technology*. 2005. **17**, 439-443.
155. Dellaert, Fox, Burgard, and Thrun. *Monte Carlo localization for mobile robots*. in *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*. 1999.
156. Fujimoto, Hoof, and Meger *Addressing Function Approximation Error in Actor-Critic Methods*. 2018.
157. Konda and Gao *Actor-critic algorithms*. 2000.
158. Gao, Ye, Guo, and Li *Deep Reinforcement Learning for Indoor Mobile Robot Path Planning*. Sensors, 2020. **20**, 5493 DOI: 10.3390/s20195493.
159. Rubio, Valero, and Llopis-Albert *A review of mobile robots: Concepts, methods, theoretical framework, and applications*. International Journal of Advanced Robotic Systems, 2019. **16**, 1-21 DOI: https://doi.org/10.1177/1729881419839596.
160. De Luca, Oriolo, and Vendittelli, *Control of Wheeled Mobile Robots: An Experimental Overview*, in *Ramsete: Articulated and Mobile Robotics for Services and Technologies*, S. Nicosia, B. Siciliano, A. Bicchi, and P. Valigi, Editors. 2001, Springer Berlin Heidelberg: Berlin, Heidelberg. p. 181-226.
161. Nehmzow, *Mobile Robotics: A Practical Introduction*, Springer-Verlag, Editor. 2003.
162. Susnea, Filipescu, Vasiliu, and Filipescu. *Path following, real-time, embedded fuzzy control of a mobile platform wheeled mobile robot*. in *2008 IEEE International Conference on Automation and Logistics*. 2008.
163. Maalouf, Saad, and Saliah *A higher level path tracking controller for a four-wheel differentially steered mobile robot*. Robotics and Autonomous Systems, 2006. **54**, 23-33 DOI: https://doi.org/10.1016/j.robot.2005.10.001.
164. Benmakhlouf and Louchene *SIMPLE FUZZY LOGIC BASED PATH TRACKING CONTROLLER FOR A MOBILE ROBOT*. Journal of Electrical Engineering, 2016. **16**, 210-217.
165. Antonelli, Chiaverini, and Fusco *A Fuzzy-Logic-Based Approach for Mobile Robot Path Tracking*. IEEE Transactions on Fuzzy Systems, 2007. **15**, 211-221 DOI: 10.1109/TFUZZ.2006.879998.
166. Abatari and Tafti. *Using a fuzzy PID controller for the path following of a car-like mobile robot*. in *2013 First RSI/ISM International Conference on Robotics and Mechatronics (ICRoM)*. 2013.
167. Yi, Yubazaki, and Hirota *A proposal of SIRMs dynamically connected fuzzy inference model for plural input fuzzy control*. Fuzzy Sets and Systems, 2002. **125**, 79-92 DOI: https://doi.org/10.1016/S0165-0114(00)00135-4.
168. Seki *Nonlinear Identification Using Single Input Connected Fuzzy Inference Model*. Procedia Computer Science, 2013. **22**, 1121-1125 DOI: https://doi.org/10.1016/j.procs.2013.09.198.
169. Yubazaki*, et al. Trajectory tracking control of unconstrained objects based on the SIRMs dynamically connected fuzzy inference model*. in *Proceedings of 6th International Fuzzy Systems Conference*. 1997.
170. Jianqiang, Yubazaki, and Hirota. *Upswing and stabilization control of inverted pendulum and cart system by the SIRMs dynamically connected fuzzy inference model*. in *FUZZ-IEEE'99. 1999 IEEE International Fuzzy Systems. Conference Proceedings (Cat. No.99CH36315)*. 1999.
171. Yi, Yubazaki, and Hirota *Stabilization control of series-type double inverted pendulum systems using the SIRMs dynamically connected fuzzy inference model*. Artificial Intelligence in Engineering, 2001. **15**, 297-308 DOI: https://doi.org/10.1016/S0954-1810(01)00021-8.
172. Yi, Yubazaki, and Hirota. *Stabilization control of ball and beam systems*. in *Proceedings Joint 9th IFSA World Congress and 20th NAFIPS International Conference (Cat. No. 01TH8569)*. 2001.
173. Yi, Yubazaki, and Hirota *A new fuzzy controller for stabilization of parallel-type double inverted pendulum system*. Fuzzy Sets and Systems, 2002. **126**, 105-119 DOI: https://doi.org/10.1016/S0165-0114(01)00028-8.
174. Yi, Yubazaki, and Hirota *Anti-swing and positioning control of overhead traveling crane*. Information Sciences, 2003. **155**, 19-42 DOI: https://doi.org/10.1016/S0020-0255(03)00127-0.

175. Yubazaki, Yi, and Hirota. *SIRMs dynamically connected fuzzy inference model and PID controller*. in *1998 IEEE International Conference on Fuzzy Systems Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98CH36228)*. 1998.
176. Li*, et al. Data-Driven Optimization of SIRMs Connected Neural-Fuzzy System with Application to Cooling and Heating Loads Prediction*. in *Advances in Neural Networks – ISNN 2015*. 2015. Cham: Springer International Publishing.
177. Li, Yi, and Zhao. *Control of the TORA system using SIRMs based type-2 fuzzy logic*. in *2009 IEEE International Conference on Fuzzy Systems*. 2009.
178. Li, Gao, Yi, and Zhang, *Analysis and Design of Functionally Weighted Single-Input-Rule-Modules Connected Fuzzy Inference Systems.* IEEE Transactions on Fuzzy Systems, 2018. **26**(1): p. 56-71.
179. Li, Zhang, Yi, and Wang. *On the properties of SIRMs connected type-1 and type-2 fuzzy inference systems*. in *2011 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2011)*. 2011.
180. Jianqiang, Yubazaki, and Hirota. *Stability analysis of SIRMs dynamically connected fuzzy inference model*. in *10th IEEE International Conference on Fuzzy Systems. (Cat. No.01CH37297)*. 2001.
181. Li, Yi, and Wang. *Stability analysis of SIRMs based type-2 fuzzy logic control systems*. in *International Conference on Fuzzy Systems*. 2010.
182. Wu*, et al. Backstepping Trajectory Tracking Based on Fuzzy Sliding Mode Control for Differential Mobile Robots*. Journal of Intelligent & Robotic Systems, 2019. **96**, 109-121 DOI: https://doi.org/10.1007/s10846-019-00980-9.
183. Fan, Long, Liu, and Pan *Distributed multi-robot collision avoidance via deep reinforcement learning for navigation in complex scenarios*. The International Journal of Robotics Research, 2020. **39**, 856-892 DOI: 10.1177/0278364920916531.
184. Sathyamoorthy*, et al. DenseCAvoid: Real-time Navigation in Dense Crowds using Anticipatory Behaviors*. in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. 2020.
185. Velagić, Vuković, and Ibrahimović *Mobile Robot Motion Framework Based on Enhanced Robust Panel Method*. International Journal of Control, Automation and Systems, 2020. **18**, 1264-1276 DOI: 10.1007/s12555-019-0009-5.
186. Siegwart, Nourbakhsh, and Scaramuzza, *Introduction to Autonomous Mobile Robots*. 2011: The MIT Press.
187. Wong, Yang, Yan, and Gu. *Adaptive and intelligent navigation of autonomous planetary rovers — A survey*. in *2017 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*. 2017.
188. Fox, Burgard, and Thrun *The dynamic window approach to collision avoidance*. IEEE Robotics & Automation Magazine, 1997. **4**, 23-33 DOI: 10.1109/100.580977.
189. Zhang, Springenberg, Boedecker, and Burgard. *Deep reinforcement learning with successor features for navigation across similar environments*. in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2017.