République Algérienne Démocratique et Populaire
الجمهورية الجزائرية الديمقراطية الشعبية
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
وزارة التعليم العالي و البحث العلمي
École nationale Polytechnique

المدرسة الوطنية المتعددة التقنيات
Ecole Nationale Polytechnique

Département électronique

Mémoire de projet de fin d'études

Pour l'obtention du diplôme d'Ingénieur d'État en Électronique

Optimisation et planification de trajectoire pour les systèmes Multi-agents

Mohamed Nadjib RAHAL

Sous la direction de Pr .Cherif LARBES Prof. ENP, Alger

Présenté et soutenu publiquement le 01/07/2024 auprès des membres du jury :

Président	M.Rachid	ZERGUI	MAA.	ENP, Alger
Promoteur	M. Cherif	LARBES	Prof.	ENP, Alger
Examinateur	M .Mohamed Oussaid	TAGHI	MAA.	ENP, Alger

ENP 2024

10, Avenue des Frères Oudek, Hassen Badi, BP. 182, 16200 El Harrach, Alger, Algérie.
www.enp.edu.dz

République Algérienne Démocratique et Populaire
الجمهورية الجزائرية الديمقراطية الشعبية
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
وزارة التعليم العالي و البحث العلمي
École nationale Polytechnique

المدرسة الوطنية المتعددة التقنيات
Ecole Nationale Polytechnique

Département électronique

Mémoire de projet de fin d'études

Pour l'obtention du diplôme d'Ingénieur d'État en Électronique

Optimisation et planification de trajectoire pour les systèmes Multi-agents

Mohamed Nadjib RAHAL

Sous la direction de Pr .Cherif LARBES Prof. ENP, Alger

Présenté et soutenu publiquement le 01/07/2024 auprès des membres du jury :

Président	M.Rachid	ZERGUI	MAA.	ENP, Alger
Promoteur	M. Cherif	LARBES	Prof.	ENP, Alger
Examinateur	M .Mohamed Oussaid	TAGHI	MAA.	ENP, Alger

ENP 2024

10, Avenue des Frères Oudek, Hassen Badi, BP. 182, 16200 El Harrach, Alger, Algérie.
www.enp.edu.dz

ملخص

"تيسير المناورة الذاتية للروبوتات المتحركة أمر حيوي لتطبيقات متنوعة. تزداد هذه المهمة تعقيدًا في السياقات التي تتطلب هيكلية متعددة الوكلاء لإنجاز مهام معقدة للغاية. الهدف هو إنشاء مسارات مثلى خالية من الاصطدام تضمن تحقيق هدف مشترك. في هذا البحث، قدمتُ حلين: الاستكشاف اللامركزي المستند إلى خريطة الطريق الاحتمالية (PRM) والتعلم المركزي المستند إلى شبكة التعلم العميق .(DQN) هدف الدراسة هو تقييم أداء هذين النهجين لتحديد مدى قابليتهما للتطبيق في سياق أوسع."

كلمات مفتاحية : أنظمة متعددة الوكلاء، تخطيط المسارات، خريطة الطريق الاحتمالية ،(PRM) التعلم العميق بالتعزيز ، التحسين

Abstract

Facilitating the autonomous maneuvering of mobile robots is crucial for various applications. This task becomes more complex in contexts requiring a multi-agent framework to accomplish tasks too complex for a single agent. The goal is to create optimal collision-free paths that ensure the achievement of a common goal. In this research, two solutions are proposed : decentralized exploration based on the Probabilistic Roadmap (PRM) and centralized learning based on the Deep Q-Network (DQN). The study aims to evaluate the performance of these approaches to determine their applicability in a broader context.

Keywords : Multi-Agent Systems, Path planning, PRM ,RL, Deep Q-Learning, Optimization

---

Résumé

Faciliter la manœuvre autonome des robots mobiles est crucial pour diverses applications. Cette tâche devient plus complexe dans les contextes nécessitant une structure multi-agents pour accomplir des tâches trop complexes pour un seul agent. L'objectif est de créer des trajectoires optimales sans collision qui assurent l'atteinte d'un objectif commun. Dans cette recherche, deux solution sont proposées : l'exploration décentralisée basée sur la carte des routes probabilistes (PRM) et l'apprentissage centralisé basé sur le réseau de neurones profonds (DQN). L'étude vise à évaluer la performance de ces approches pour déterminer leur applicabilité dans un contexte plus large.

Mots clés : Systèmes multi-agents, planification de trajectoires, PRM, apprentissage profond par renforcement (Deep Q-Learning), optimisation.

Dédicace

"Cet ouvrage est dédié à mes merveilleux parents qui, par leur encouragement constant et leur soutien inébranlable, ont été les piliers de ma vie. Leurs conseils avisés et leur présence bienveillante ont façonné la personne que je suis aujourd'hui.

À mes chers frères et sœurs, vous êtes mes compagnons de route, mes confidents et mes complices. Vos rires résonnent comme des mélodies joyeuses dans mes souvenirs, et votre présence inconditionnelle illumine mes journées les plus sombres. Chacun de vous a apporté une lumière unique à ma vie, et je vous remercie du fond du cœur pour votre soutien sans faille.

À toute ma famille, je souhaite exprimer ma profonde gratitude pour votre soutien indéfectible. Votre amour et votre présence ont été une source inestimable de bonheur et de réconfort à chaque étape de ma vie. Chaque moment partagé avec vous a enrichi mon existence, et je suis reconnaissant pour les liens forts qui nous unissent.

Je souhaite également remercier chaleureusement ma promotion du département d'électronique. Vous avez été plus qu'une simple cohorte académique ; vous avez été des amis et des collègues exceptionnels. Votre camaraderie, vos échanges enrichissants et votre soutien mutuel ont rendu notre parcours ensemble inoubliable. Merci d'avoir été à mes côtés, tant dans les moments de réussite que face aux défis que nous avons surmontés ensemble.

À Foutia, Mohamed, Abdelbaki, Mehdi, Mouad, Ramzy, je vous adresse un immense merci. Votre soutien inconditionnel et vos conseils précieux ont été d'une valeur inestimable pour moi. Je ne saurais assez vous remercier pour tout ce que vous avez fait et pour la force que vous m'avez apportée.

À tous ceux qui me sont chers, à vous tous."

- Nadjib

Remerciements

Je souhaite exprimer ma profonde gratitude envers mon promoteur, le Prof. Cherif LARBES, pour son soutien inébranlable tout au long de mon parcours. Sa patience, sa motivation et son engagement sans faille ont été d'une valeur inestimable pour moi. Je suis extrêmement reconnaissant d'avoir eu la chance de bénéficier de ses conseils précieux.

Je tiens également à remercier chaleureusement le membre du jury, Monsieur Rachid ZERGUI, enseignant à l'École Nationale Polytechnique, et Monsieur Mohamed Oussaid TAGHI, enseignant chercheur à l'École Nationale Polytechnique, pour avoir consacré leur temps et leur expertise à l'évaluation de mon travail.

Mes sincères remerciements vont également à tous mes professeurs de l'École Nationale Polytechnique, dont les enseignements ont façonné ma formation académique et m'ont permis d'atteindre ce niveau de réussite.

Je ressens de la gratitude envers mes collègues de l'École Nationale Polytechnique, avec qui j'ai vécu des expériences enrichissantes et partagé des moments de collaboration précieux

Enfin, je tiens à exprimer ma gratitude envers toutes les personnes qui ont joué un rôle, direct ou indirect, dans mon parcours académique et dans la réalisation de ce projet.

Nadjib.

Table des matières

Liste des tableaux

Table des figures

Liste des abréviations

AGV *Automated Guided Vehicle.*

PRM *Probabilistic Roadmap.*

RL *Reinforcement Learning.*

RRT *Rapidly-exploring Random Tree.*

PT *Potential Field.*

CD *Cell Decomposition.*

GA *Genetic Algorithm.*

RP *Randomized Path Planning.*

PSO *Particle Swarm Optimization.*

ACO *Ant Colony Optimization.*

MCTS *Monte Carlo Tree Search.*

TD *Temporal Difference.*

REINFORCE *REward Increment Nonnegative Factorized Actor-Critic Elements.*

G(PO)MDP *Generalized (Partially Observable) Markov Decision Process.*

DQN *Deep Q-Network.*

SGD *Stochastic Gradient Descent.*

DDPG *Deep Deterministic Policy Gradient.*

GPRM *Generalized Probabilistic Road Map.*

AGPRM *Adaptive Generalized Probabilistic Roadmap.*

MTSP *Multi-Traveling Salesman Problem.*

MAGPRM *Multi-Agent Generalized Probabilistic Roadmap.*

MDP *Markov Decision Process.*

DP *Dynamic Programming.*

SARSA *State-Action-Reward-State-Action.*

MARL *Multi-Agent Reinforcement Learning.*

MGs *Markov Games.*

HAS *Hybrid Algorithm Search.*

Introduction générale

Introduction générale

Les systèmes multi agents sont devenus incontournables dans de nombreux domaines, notamment dans la logistique où ils jouent un rôle crucial. Ces systèmes, composés de robots et de véhicules autonomes guidés (AGV), offrent des solutions innovantes pour optimiser les processus logistiques et améliorer l'efficacité opérationnelle. Cependant, malgré leur utilité, l'optimisation de la planification de trajectoire reste un défi majeur pour garantir une productivité et une fluidité optimales entre les différents agents.

Le besoin d'améliorer les planificateurs de trajectoire multi agents découle directement de la nécessité d'optimiser la productivité et l'efficacité opérationnelle. En effet, pour que les opérations logistiques fonctionnent de manière fluide et rentable, il est essentiel que les agents se déplacent de manière coordonnée et efficace, tout en évitant les collisions et en respectant les contraintes de temps et d'espace.

Considérons, par exemple, un terminal à conteneurs tel que celui illustré dans la figure

Fig. 1 : Véhicules à guidage automatique dans un terminal à conteneurs

L'objectif principal de l'équipe d'agents autonomes, composée d'AGVs opérant dans le terminal à conteneurs, est d'assurer une gestion efficace du flux de conteneurs.

Toutefois, comme le montre la figure 1, les agents doivent aussi veiller à éviter les collisions avec les éléments de l'environnement ainsi qu'avec d'autres agents lors de l'accomplissement de ces tâches. Cette exigence est fondamentale pour la plupart des systèmes multi-agents : la capacité de naviguer en toute sécurité dans l'environnement où les tâches doivent être exécutées [1] .

Par conséquent, tout planificateur de trajectoire multi-agents pratique doit explicitement tenir compte des contraintes liées à son environnement. L'environnement dans un terminal à conteneurs, par exemple, diffère de celui trouvé dans un espace encombré tel qu'un entrepôt. Ainsi, la contrainte du rayon de sécurité minimal est plus significative dans un environnement encombré que dans un environnement ouvert comme celui d'un terminal à conteneurs.

À mesure que l'espace de navigation des agents s'étend, cela entraîne une augmentation du nombre d'équipes nécessaires pour couvrir l'ensemble de cet espace. Par conséquent, la complexité du problème croît considérablement avec le nombre croissant d'agents, dépassant rapidement les capacités d'un planificateur unique et centralisé pour gérer l'ensemble des agents.

La communication requise par un planificateur centralisé pour superviser l'ensemble des agents devient également trop complexe pour des problèmes à grande échelle.

Cette situation conduit au développement de planificateurs décentralisés, dans lesquels chaque agent prend des décisions à l'échelle locale. Toutefois, cette approche n'est pas dépourvue de difficultés, et tout planificateur décentralisé doit s'assurer d'éviter les conflits de décisions locales.

L'objectif de ce projet de fin d'études est d'explorer le problème de la planification de trajectoire à travers des environnements ($2D$) ouverts, impliquant un nombre d'agents autonomes, avec pour objectif commun l'accomplissement de tâches spécifiques.

-Positionnement du problème : :

La planification de trajectoire est l'un des domaines les plus étudiés en robotique, Pour un seul agent, le problème est relativement simple à résoudre. Dans ce contexte, le robot est généralement représenté comme un point dans un espace de configuration, où chaque point correspond à une configuration spécifique du robot. Les obstacles dans l'environnement de travail sont également représentés dans cet espace, permettant ainsi de définir l'ensemble des configurations valides que le robot peut prendre sans entrer en collision avec les obstacles.

Lorsque le robot se déplace d'une configuration à une autre, un coût est associé à ce mouvement, généralement mesuré en termes de distance euclidienne dans un espace de configuration simple à deux dimensions pour un robot planaire. Ce coût détermine la qualité du chemin dans l'espace de configuration : plus il est élevé, plus le robot effectue de mouvements inutiles pour atteindre son objectif.

Cependant, la planification de trajectoire devient considérablement plus complexe dans le cas de plusieurs agents. Dans ce scénario, il est essentiel de gérer efficacement les coûts de déplacement entre les différents agents pour assurer un fonctionnement harmonieux du système. La coordination des mouvements devient alors cruciale pour éviter les collisions entre les agents et garantir l'efficacité globale de la planification de trajectoire.

-Hypothèses :

L'objectif principal est de planifier des trajectoires sans collisions afin de minimiser le temps total de déplacement de tous les agents, représenté comme la somme des temps d'arrivée individuels (le temps de flux T_F). Pour simplifier le problème, tous les agents, qu'il s'agisse de robots ou de véhicules à guidage automatique (AGV), sont considérés comme des points matériels, et leur taille est représentée par un rayon de sécurité.

-Organisation :

Dans notre projet de fin d'études, nous avons entamé une exploration succincte des diverses approches en matière de planification de trajectoires, en se concentrant particulièrement sur les méthodes classiques et heuristiques. Cette revue a été présentée dans le chapitre 1, intitulé "État de l'Art".

Ensuite, nous avons choisi quelques algorithmes parmi les plus pertinents et les plus avancés dans chacune des approches identifiées, que nous avons approfondis dans les chapitres 2 et 3. Le chapitre 2 est consacré aux "Probabilistic Roadmap" (PRM), tandis que le chapitre 3 port sur l'apprentissage par renforcement (RL).

Par la suite, dans le chapitre 4 "Implémentation et résultats", nous avons évalué les performances des approches classique (PRM) et heuristique (RL) en termes d'efficacité de résolution. nous avons réalisé cette évaluation dans le cadre d'un agent unique et multi-agents, et établi une étude comparative dans différents cas de figure pour en tirer des conclusions.

Enfin, nous avons conclu notre étude en envisageant l'application des solutions proposées dans des scénarios réels impliquant des robots mobiles dans un environnement en deux dimensions.

Chapitre 1

Etat de l'art :Exploration des Avancées Récentes dans les Algorithmes de Planification de Trajectoire Multi agents

1.1 Classification des approches existante

1.1.1 Commande Centralisée Vs commande décentralisée

1.1.1.1 Commande Centralisée

L'approche de centralisation consiste à rassembler toutes les informations pertinentes sur les agents et leur environnement à un seul endroit, généralement un agent central ou une entité externe, qui utilise ces informations pour élaborer des plans de trajectoire pour chaque agent. Ces plans sont ensuite communiqués aux agents individuels, qui les suivent pour atteindre leurs objectifs.

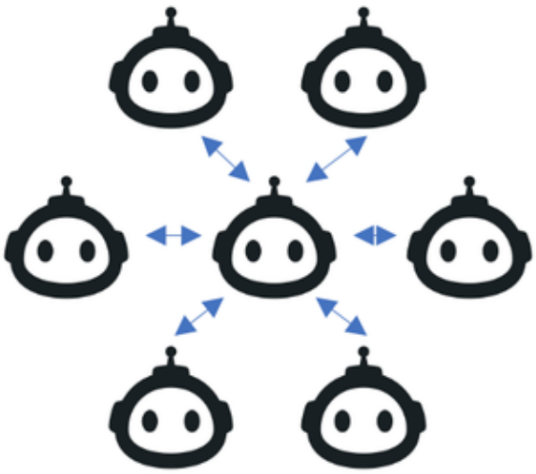


Fig. 1.1 : Commande Centralisée

Parmi les algorithmes centralisés existants, l'approche proposée par Parsons et Canny [2] se distingue par sa méthode de division de l'espace libre, c'est-à-dire l'espace où les agents peuvent se déplacer sans obstacle, en cellules. Ensuite, il utilise cette division pour créer un graphe d'adjacence qui représente les connexions entre les cellules. Ce graphe est ensuite utilisé pour trouver des chemins pour les agents à travers l'environnement.

Bien que les approches centralisées offrent l'avantage d'être complètes et optimales, elles sont confrontées à un défi majeur en termes de complexité computationnelle [3]. En effet, le temps de calcul de ces méthodes augmente de manière exponentielle à mesure que le nombre des agents dans le système augmente. Cela signifie que le traitement des données devient de plus en plus complexe et demande davantage de ressources computationnelles à mesure que la taille du système croît.

1.1.1.2 Commande Décentralisée

L'approche décentralisée dans la planification de trajectoire des systèmes multi-agents fait référence à une méthode où chaque agent dans un système distribué prend des décisions de manière autonome, en se basant sur des informations locales plutôt que sur une planification centralisée.

Dans cette approche, chaque agent est capable de percevoir son environnement et de prendre des décisions en fonction de ses propres objectifs et contraintes, sans nécessiter de coordination centralisée avec les autres agents. Les agents peuvent communiquer entre eux de manière limitée, mais la prise de décision reste principalement autonome.

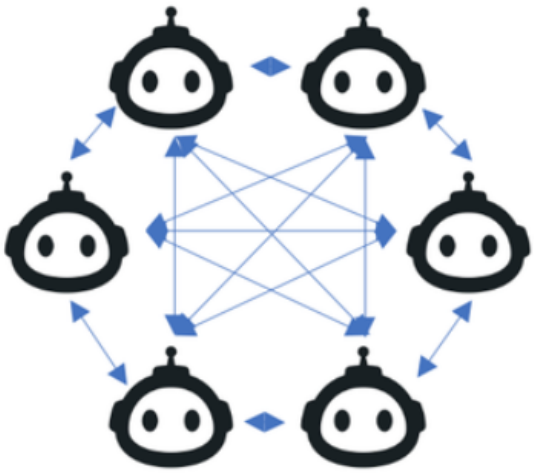


Fig. 1.2 : commande décentralisée

L'un des inconvénients majeurs des approches décentralisé est la perte potentielle de complétude et d'optimalité [4]. Lorsque chaque agent prend des décisions de manière autonome, basée uniquement sur ses informations locales et sans coordination centralisée, cela peut conduire à des résultats sous optimaux. En d'autres termes, chaque agent peut optimiser localement sa trajectoire sans tenir compte des conséquences globales, aboutissant parfois à des configurations où les objectifs globaux du système ne sont pas pleinement atteints. De plus, en l'absence d'une vision globale de l'environnement et des actions des autres agents, il existe un risque de conflits et de blocages pouvant compromettre la complétude de la solution.

L'avantage majeur de cette technique est l'autonomie et l'indépendance de l'agent, et donc la simplicité de résolution [5].

1.1.2 Approche étiquetée vs Approche non étiquetée

1.1.2.1 Planification de trajectoire étiquetée

Dans une approche étiquetée, chaque agent est associé à une étiquette ou un identifiant unique. Les trajectoires sont planifiées en tenant compte des positions et des mouvements de chaque agent, ainsi que de leurs identifiants respectifs. Cette méthode permet une distinction claire entre les différents agents dans l'environnement, ce qui peut faciliter la coordination et la prise de décision. Les informations sur les identifiants des agents peuvent être utilisées pour éviter les collisions, allouer des ressources et coordonner les actions de manière plus précise.

1.1.2.2 Planification de trajectoire non étiquetée

Dans une approche non étiquetée, les agents ne sont pas associés à des étiquettes ou des identifiants spécifiques. Les trajectoires sont planifiées en fonction des positions et des mouvements des agents, sans distinction entre eux. Cette approche est souvent utilisée dans des situations où les identifiants des agents ne sont pas disponibles ou ne sont pas nécessaires pour la planification. Cependant, cela peut rendre la coordination et la prise de décision plus complexes, car il n'y a pas de moyen direct de distinguer les agents les uns des autres.

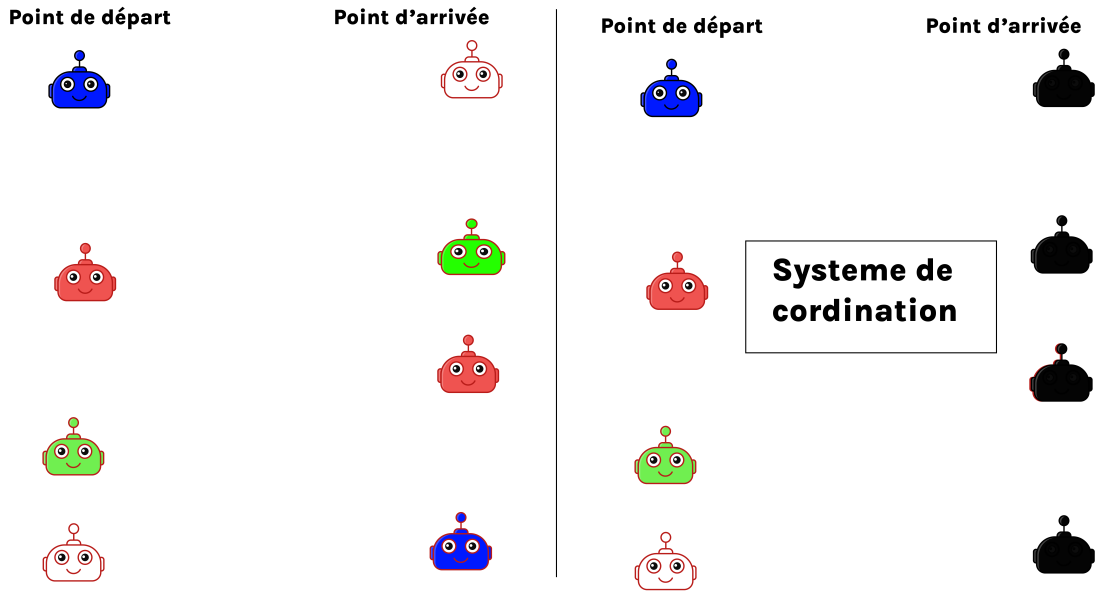
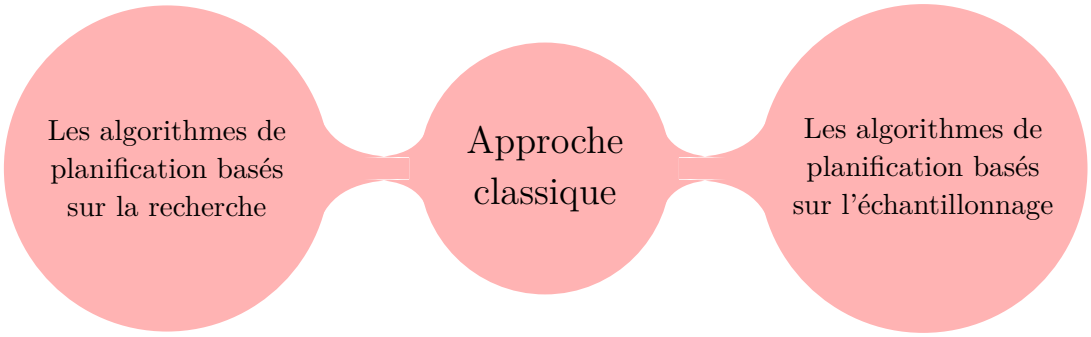


FIG. 1.3 : Approche étiquetée vs Approche non étiquetée

1.1.3 Selon l'exhaustivité

1.1.3.1 Approche Classique

L'approche classique repose sur des algorithmes et des techniques formelles pour résoudre les problèmes de planification de trajectoire. Elle utilise souvent des méthodes de résolution exacte, telles que la programmation linéaire, la recherche exhaustive, ou des algorithmes basés sur des modèles mathématiques précis. Cette approche est rigoureuse et garantit des solutions optimales ou proches de l'optimalité. On trouve deux grandes catégories d'algorithmes dans l'approche classique :



Voici un tableau qui résume les algorithmes basés sur l'échantillonnage Vs les algorithmes basés sur la recherche :

Basée sur l'échantillonnage	Basée sur la recherche
1. PRM (Probabilistic Roadmap) (1996)	1. Potential Field (1979)
2. Rapidly Exploring Random Tree (1998)	2. Cell Decomposition (1987)
3. Grid Based (1988)	

TAB. 1.1 : Les algorithmes de planification de l'approche Classique

1-Les algorithmes de planification basés sur l'échantillonnage :

Ces algorithmes de planification effectuent une recherche dans l'espace de configuration à l'aide d'un schéma d'échantillonnage. Essentiellement, ils sélectionnent un ensemble de points dans l'espace libre, qu'ils connectent pour former un arbre de trajectoires réalisables. Ces trajectoires sont ensuite exploitées pour déterminer la solution du problème de planification.

L'évitement des obstacles est géré grâce à un module de détection des collisions, qui élimine les trajectoires entrant en collision avec les obstacles présents dans l'environnement. Cette approche simplifie la construction de l'arbre en réduisant l'espace de recherche à explorer [6].

Ces algorithmes sont considérés comme probabilistiquement complets, ce qui implique que la probabilité de trouver une solution réalisable tend vers 1 lorsque le nombre d'échantillons augmente indéfiniment.

En outre, certaines versions de ces algorithmes peuvent également intégrer un critère de coût pour évaluer la qualité des différentes trajectoires réalisables, et sont également probabilistiquement optimales.

Cependant, elle peut être limitée par sa complexité computationnelle, ce qui rend son application pratique difficile pour des problèmes de grande taille ou dans des environnements dynamiques.

Les planificateurs basés sur l'échantillonnage les plus importants à ce jour sont Probabilistic Road-Maps (PRM) [7] et Rapidly-exploring Random Trees (RRT) [8] .

- **Feuilles de route (Probabilistic Road Map Method)**

  Dans la méthode PRM (Probabilistic Roadmap Method), l'espace libre (préarrangement des mouvements réalisables) est retiré, réduit à, ou représenté sur un système de lignes unidimensionnelles. La recherche d'une réponse est limitée au système, et la planification des mouvements devient un problème de recherche de graphe. Cette approche spécifique est également appelée approche squelette, retrait, ou autoroute. Dans cette approche, la locomotion du robot est réalisée en trois étapes :

  – le robot est déplacé de sa configuration de départ à un point sur la carte probabiliste des routes (PRM),en utilisant une stratégie de déplacement standard.

  – le robot est déplacé de la configuration cible à travers un point sur la carte routière de manière similaire.

  – enfin, les points de configuration de départ et de cible sont reliés pour obtenir le chemin requis.

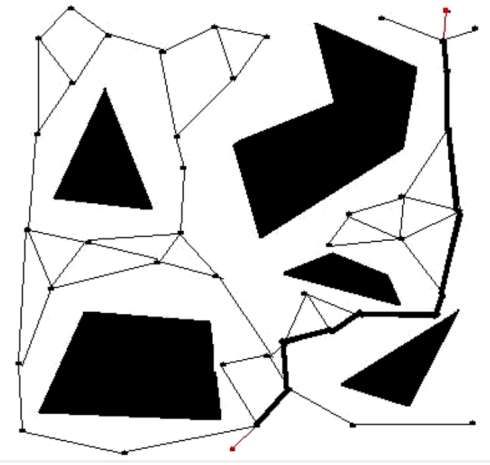


FIG. 1.4 : Road Maps [9]

La carte routière doit définir tous les chemins topologiquement distincts faisables dans l'espace libre. L'objectif du PRM paresseux est essentiellement de limiter le nombre de vérifications de collision tout en recherchant le chemin le plus court réalisable dans une carte routière par rapport à un planificateur PRM, ce qui se fait au détriment de la recherche de graphe régulière.

- **Rapidly-exploring Random Trees planners RRT**

L'algorithme RRT (Rapidly-exploring Random Tree) construit un arbre T dont le nœud racine est la configuration initiale de l'agent. À chaque étape, une nouvelle configuration n est échantillonnée aléatoirement dans l'espace sans obstacles. Le nœud n_{proche} le plus proche dans l'arbre T est trouvé, et une trajectoire réalisable reliant n_{proche} à n est calculée. Si cette trajectoire est sans collision, n est ajouté à T. L'arbre est complètement construit lorsqu'il atteint une longueur maximale ou lorsque la configuration cible est ajoutée. Si la configuration cible est dans l'arbre, la trajectoire de la racine au nœud cible est la solution ; sinon, la cible est connectée au nœud le plus proche par une trajectoire réalisable. L'algorithme est probabilistiquement complet et garantit une terminaison, mais ne garantit pas l'optimalité.

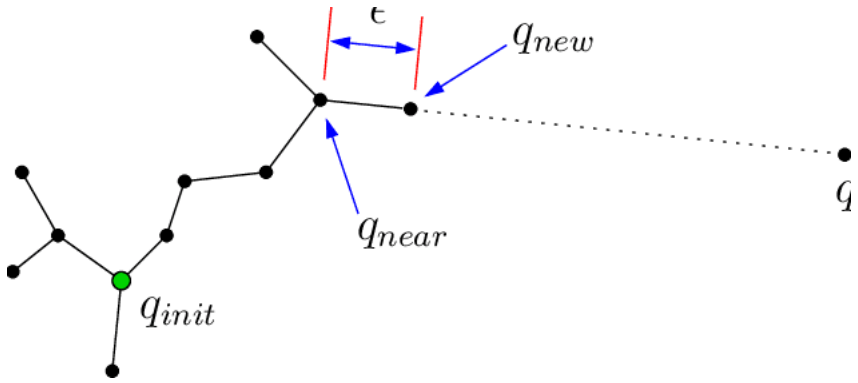


FIG. 1.5 : Procédure de l'algorithme RRT

2-Les algorithmes de planification basés sur la recherche :

Ces algorithmes ont pour objectif de déterminer la séquence optimale d'actions permettant à un agent de passer d'un état initial à un état cible, en construisant un graphe d'états et en recherchant une solution à travers ces graphes. Pour ce faire, ils requièrent la discrétisation de l'espace d'états continu en un ensemble fini d'états, ainsi que la définition des actions possibles entre ces états, avant d'appliquer des méthodes de recherche de graphes pour trouver la solution optimale[10].

La qualité de la solution trouvée dépend étroitement du schéma de discrétisation adopté. Par conséquent, des efforts considérables sont déployés pour diviser efficacement l'espace d'état afin de garantir des résultats optimaux. Parmi les algorithmes les plus couramment utilisés dans cette catégorie, on trouve la Décomposition Cellulaire et les Champs de Potentiel. Ces approches constituent des outils essentiels dans la résolution de problèmes de planification de trajectoire, offrant des solutions précises et efficaces dans un large éventail d'applications.

- **Champs de potentiel (Potential field) :**
  Cette méthode consiste à simuler les interactions entre un robot et son environnement en utilisant des champs attractifs et répulsifs. Le champ attractif est généré vers l'objectif, tandis que le champ répulsif est créé autour des obstacles. Le robot se déplace en suivant la direction de la force résultante induite par ces champs. En ajustant les forces d'attraction et de répulsion, le robot peut naviguer de manière efficace vers son objectif tout en évitant les obstacles[11].

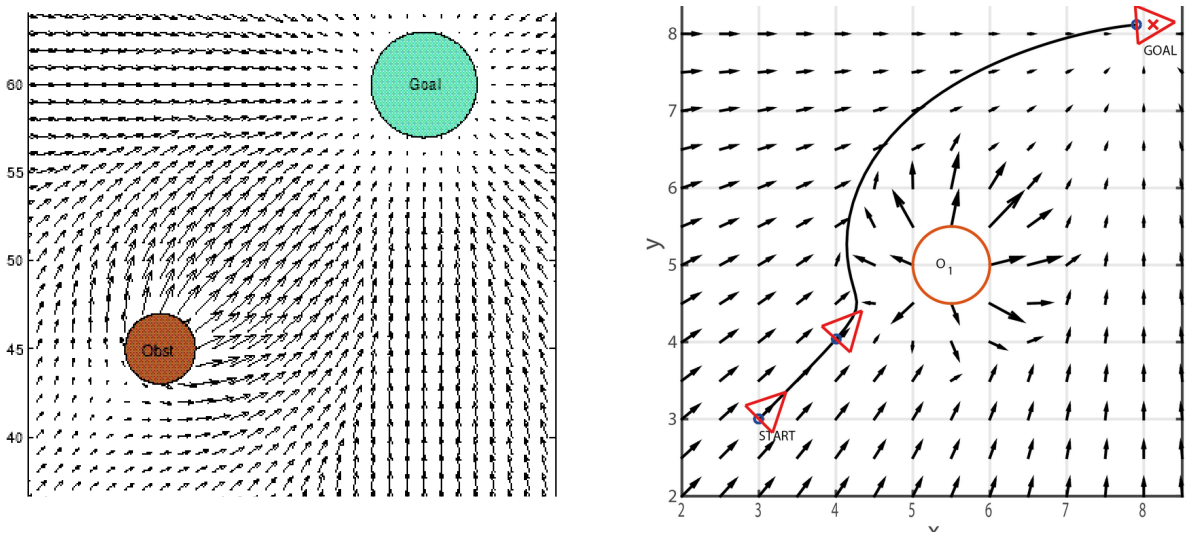


FIG. 1.6 : Potential Fields

Comme illustré dans la figure 1.6 , nous créons un champ attractif dirigé vers l'objectif. Ce champ de potentiel varie dans tout l'espace libre, et à chaque instant, à la position actuelle du robot, nous calculons ce champ, puis déterminons la force induite par celui-ci. De plus, le comportement du robot peut être configuré pour éviter les obstacles en générant un champ répulsif autour de chacun d'eux. Ce champ de répulsion agit pour éloigner le robot de l'obstacle à mesure qu'il s'en approche.

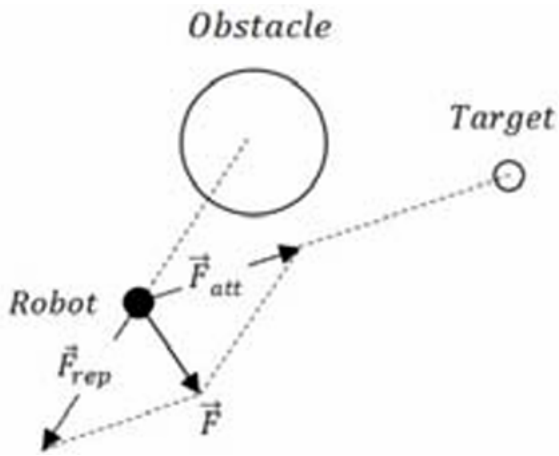


FIG. 1.7 : Attraction et répulsion par cible et obstacle

Le champ de potentiel attractif/répulsif est défini comme suit :

$$U(q) = U_{att}(q) + U_{rep}(q)$$

,Où $U_{att}(q)$ est le potentiel attractif qui est responsable de se rapprocher de la cible et $U_{\text{rep}}\,(q)$ est le potentiel répulsif pour éviter les obstacles.

$$U_{\text{att}}\,(q) = \frac{1}{2}\delta \cdot \rho^{m}\,(q, q_{\text{cible}}\,)$$

$$U_{\text{rep}}\,(q) = \begin{cases} \frac{1}{2}\alpha\left(\frac{1}{\rho(q,q_{\text{obs}}\,)} - \frac{1}{\rho_o}\right)^2 \rho^{n}\,(q, q_{\text{cible}}\,) & \text{si } \rho\,(q, q_{\text{obs}}\,) \leq \rho_o \\ 0 & \text{si } \rho\,(q, q_{\text{obs}}\,) > \rho_o \end{cases}$$

Où δ est un facteur d'échelle positif.

Les limites de la méthode du champ de potentiel incluent la capture des situations dues aux minima locaux, l'absence de passage entre des obstacles étroitement espacés, les mouvements en présence d'obstacles ou dans une entrée ou un passage étroit.

- **Décomposition cellulaire (Cell decomposition)**

  L'approche de décomposition en cellules propose une méthode pour distinguer les zones géométriques libres des zones occupées par des objets. L'algorithme de base de décomposition en cellules pour la planification de trajectoires peut être résumé comme suit :

  – Tout d'abord, diviser l'environnement en zones de base appelées "cellules".

  – Ensuite, identifier les cellules ouvertes contiguës et établir un "tableau de disponibilité".

  – Puis, localiser les cellules où se trouvent les configurations initiale et cible, et rechercher un chemin dans le tableau de disponibilité pour les relier.

  Enfin, à partir de la classification des cellules trouvées avec un algorithme de recherche approprié, déterminer un chemin à l'intérieur de chaque cellule, en suivant par exemple une séquence de mouvements le long des parois ou en se déplaçant le long des lignes droites des limites des cellules.

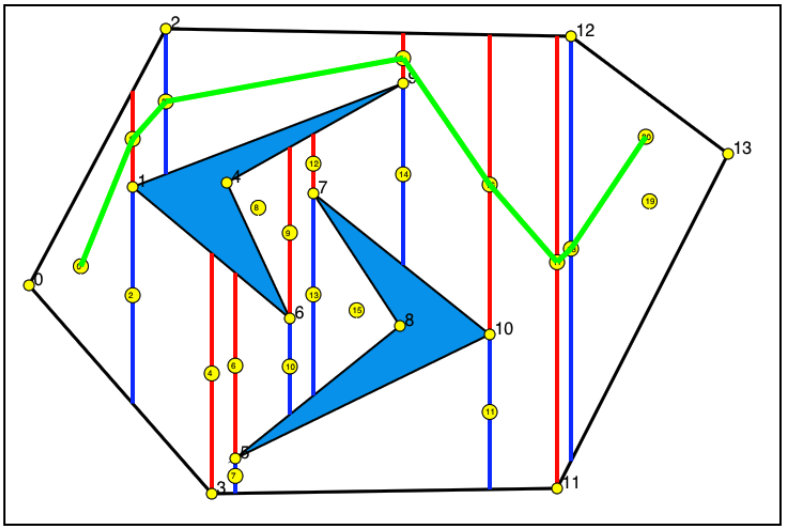


Fig. 1.8 : Cell Decomposition

Dans les méthodes de décomposition cellulaire, l'aspect le plus important est l'attribution des frontières entre les cellules. Si les frontières sont assignées en fonction de la structure de l'environnement, de sorte que la décomposition soit sans perte, alors la méthode est appelée décomposition cellulaire exacte. Si la décomposition résulte en une approximation de la carte réelle, le système est appelé décomposition cellulaire approximative, et ensuite vient la décomposition cellulaire probabiliste qui est similaire à la décomposition cellulaire approximative sauf que les frontières des cellules ne représentent aucune signification physique. La décomposition cellulaire est couramment utilisée dans les scénarios de planification de mouvement de robots. Fujii et al., Yang et Gu et Park et al [12] ont proposé de séparer de grands espaces d'apprentissage en plusieurs plus petits, ce qui facilite l'exploration. Les principaux avantages de cette stratégie sont que la détérioration de l'espace de travail du robot est naturelle et que l'état des cellules permet une planification robuste

et flexible. La décomposition cellulaire exacte et approximative a des avantages et des inconvénients. Les premières sont garanties d'être complètes, ce qui signifie que si une voie libre existe, la décomposition cellulaire exacte la trouvera ; cependant, le compromis pour cette précision est un processus mathématique plus difficile. La décomposition cellulaire approximative est moins complexe, mais peut donner des résultats similaires, voire identiques, à la décomposition cellulaire exacte.

Remarque :
Ces méthodes ont l'inconvénient d'être trop lentes pour être utilisées en pratique, surtout dans les problèmes à haute dimension, et de nécessiter une représentation explicite des obstacles, ce qui est très compliqué à obtenir dans la plupart des problèmes pratiques.

Conclusion :

Les algorithmes basés sur l'échantillonnage et ceux basés sur la recherche présentent plusieurs distinctions notables. Les algorithmes de la première catégorie sont des algorithmes probabilistes complets, tandis que ceux de la garantissent la terminaison et la complétude à condition que l'espace d'état soit bien défini. Cela signifie que si un résultat est obtenu, il constitue une solution valide ; sinon, l'algorithme signalera un échec.

Dans ce mémoire,un algorithme basé sur l'échantillonnage est choisi pour les raisons suivantes :

- Il permet de gérer facilement les obstacles grâce à un module de détection de collisions ;
- Il prend en compte certaines propriétés spécifiques de l'agent et les contraintes qui y sont associées ;
- Il est généralement plus rapide qu'un algorithme basé sur la recherche.

En conclusion, les approches classiques garantissent l'obtention de solution si elle existe, mais leurs principaux inconvénients qui les rendent inefficaces dans la pratique sont les suivants :

- Elles impliquent un coût de calcul élevé pour déterminer une trajectoire sans collision réalisable dans des dimensions élevées.
- Elles ont tendance à se coincer dans une solution optimale locale
- La solution est assez compliquée lorsque l'environnement est dynamique et complexe.

Ces inconvénients empêchent leur utilisation dans des environnements complexes en présence de plusieurs agents[13].

1.1.3.2 Approche Heuristique

Approche Heuristique	
1.	Algorithmes génétiques GA
2.	Apprentissage par renforcement RL
3.	Réseau neuronal Artificiel
4.	Optimisation de l'essaim de particules PSO
5.	Optimisation des colonies de fourmis ACO

Tab. 1.2 : Classification des approches heuristiques

Pour surmonter les inconvénients mentionnés des approches classiques, des approches heuristiques plus sophistiquées ont été développées [14].

Cheng et Zelinsky [10] ont préconisé l'utilisation d'approches heuristiques pour répondre aux exigences des environnements réels. Ces approches heuristiques reposent sur la conception d'un ensemble de comportements simples visant à résoudre des scénarios complexes.

Dans les approches heuristiques, la fiabilité est étroitement liée à la situation actuelle (état) et à l'ensemble de comportements conçus pour cet état. Par conséquent, elles se révèlent fiables dans des environnements dynamiques, car elles ne dépendent pas de la localisation et de la planification. Cependant, malgré leur fiabilité dans ces environnements dynamiques, elles peuvent rencontrer des difficultés liées à l'incertitude. Cette incertitude peut entraîner des défauts au niveau des états sélectionnés, ce qui conduit souvent à des blocages dans les états des robots.

Certaines des approches classées comme méthodes heuristiques sont également appelées méthodes basées sur la population et le comportement dans la littérature. Ceci est dû au fait que ces méthodes, telles que les algorithmes génétiques (GA) [15], l'optimisation par colonies de fourmis (ACO) [16], l'optimisation par essaim de particules (PSO) [17] et l'apprentissage par renforcement (RL), traitent un ensemble de solutions dans chaque itération. Dans ces méthodes, la stratégie est modifiée ou une nouvelle stratégie émerge à chaque itération par rapport aux précédentes. Bien que ces approches heuristiques ne garantissent pas l'obtention de la solution optimale, elles offrent généralement des temps de résolution plus courts par rapport aux méthodes classiques. Toutefois, il convient de noter qu'elles peuvent échouer à obtenir la solution ou trouver un optimum local.

1.2 Etat de l'Art du"Reinforcement Learning RL"

L'apprentissage par renforcement (RL) constitue un domaine dynamique de l'apprentissage automatique, caractérisé par une évolution rapide tant sur le plan théorique que pratique. Il appartient à la catégorie des méthodes d'apprentissage heuristique, visant à découvrir la solution heuristique optimale pour un environnement donné. Les algorithmes d'apprentissage par renforcement observent le comportement des agents dans des environnements et apprennent à optimiser leurs actions en fonction d'une stratégie de récompense et de punition.

On peut classer ces algorithmes principalement en deux catégories : l'apprentissage par la valeur (value-based) et l'apprentissage par la politique (policy-based).

1.2.1 Définitions et terminologies

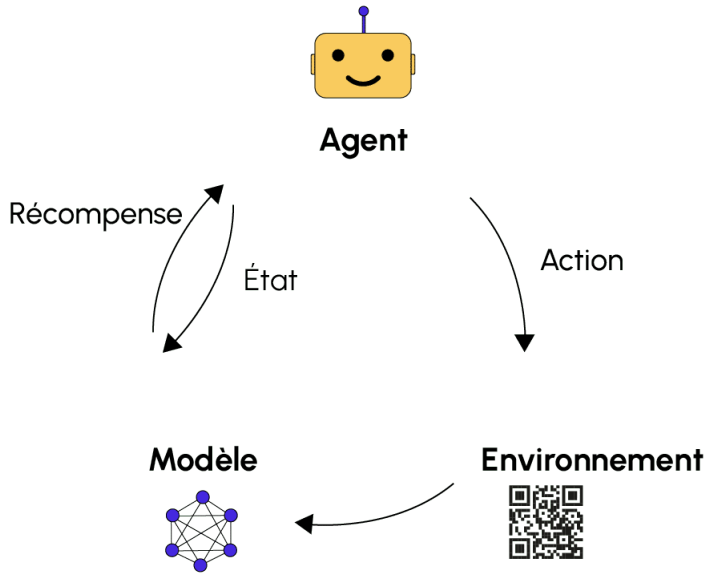


Fig. 1.9 : Interaction Agent-Environnement [18]

L'apprentissage par renforcement nécessite d'introduire un certain nombre de concepts et de métriques, dont les principaux sont les suivants :

-Agent : L'élément d'apprentissage qui effectue des actions dans un environnement et reçoit des retours évaluant ces actions.

-Environnement : Tout ce avec quoi l'agent peut interagir. L'environnement change lorsque l'agent effectue des actions, chaque changement étant considéré comme une transition d'état.

-État (Observation) : Chaque situation rencontrée par l'agent dans l'environnement est formellement appelée un état. L'agent fait passer l'environnement d'un état à un autre en effectuant des actions. Il convient de mentionner les états terminaux, qui marquent la fin d'un épisode. Après avoir atteint un état terminal, il n'y a plus d'états possibles, et un nouvel épisode commence.

-Actions : Les méthodes par lesquelles l'agent interagit et modifie son environnement, passant ainsi d'un état à un autre. Chaque action entraîne une récompense de la part de l'environnement. La stratégie guide la décision de l'action à choisir.

-Récompense (Reward) : Une valeur numérique reçue par l'agent de l'environnement en réponse directe à ses actions. L'objectif de l'agent est de maximiser la récompense totale qu'il reçoit au cours d'un épisode, motivant ainsi l'adoption de comportements souhaités.

-Épisode : Tous les états situés entre un état initial et un état terminal. Les différents épisodes sont complètement indépendants les uns des autres.

-Stratégie (Policy) : Une correspondance entre un état donné et les probabilités de sélection de chaque action possible. Par exemple, une stratégie peut choisir pour chaque état l'action ayant la plus grande valeur Q attendue.

-Valeurs état-action Q(s,a) :Évaluation de l'action a dans un état s, indiquant à quel point il est bon d'effectuer cette action dans cet état.

-Équation de Bellman : Une relation entre un état (ou une paire état-action) et ses successeurs, couramment utilisée pour définir la valeur Q.

-Facteur de remise : Détermine l'importance des récompenses dans un avenir lointain par rapport à celles dans un avenir immédiat.

1.2.2 Catégories du "Reinforcement Learning"

1.2.2.1 Méthodes basées sur la valeur "Value-Based"

Les méthodes de renforcement basées sur la valeur sont conçues pour estimer efficacement la fonction Q pour toutes les combinaisons état-action. En ayant une bonne estimation de Q, on peut extraire une stratégie optimale (ou approximative) en sélectionnant les actions avec les valeurs Q maximales.

Deux des algorithmes les plus populaires en apprentissage par renforcement basés sur la valeur sont le Q-learning et le SARSA (State-Action-Reward-State-Action). Dans ces algorithmes, l'agent maintient une estimation de la fonction Q optimale. Lorsqu'il effectue une transition de l'état actuel à l'état suivant en prenant une action, l'agent reçoit une récompense et met à jour la fonction Q(s, a) en fonction de cette transition, en utilisant l'équation de Bellman (1.1). Cependant, les stratégies de mise à jour diffèrent entre ces deux algorithmes.

$$Q(s,a) = \sum_{s',r} p(s', r \mid s, a)[r + \gamma \max_{a'} Q(s', a')] \tag{1.1}$$

Un autre algorithme populaire basé sur la valeur, mais suivant une approche différente, est la recherche arborescente de Monte-Carlo (MCTS) [19]. Contrairement à Q-learning et SARSA, qui opèrent sur une table de valeurs Q, MCTS simule des trajectoires d'actions potentielles à partir de l'état actuel pour estimer la valeur de chaque action.

Une tâche cruciale dans les méthodes basées sur la valeur est d'estimer la fonction de valeur associée à une stratégie donnée. Cette tâche, appelée évaluation de stratégie, est abordée par des algorithmes d'apprentissage par différence temporelle (TD). Les algorithmes TD sont conçus pour mettre à jour progressivement les estimations de la fonction de valeur en fonction des récompenses reçues et des estimations précédentes, permettant ainsi à l'agent d'apprendre efficacement sans avoir besoin de connaître le modèle complet de l'environnement.

1.2.2.2 Méthodes basées sur la stratégie "Policy-Based"

Un autre type d'algorithmes en apprentissage par renforcement fonctionne différemment des méthodes basées sur la valeur. Au lieu d'apprendre une fonction de valeur pour déterminer quelle action prendre dans chaque état, ces algorithmes recherchent directement la fonction de stratégie qui décrit la séquence d'actions à prendre, sans passer par une fonction de valeur intermédiaire.

Dans cette approche, l'objectif est d'optimiser directement la fonction de stratégie sans s'appuyer sur une fonction de valeur. Pour ce faire, on paramètre directement la stratégie, souvent estimée par des approximations de fonctions paramétrées comme les réseaux de neurones.

Une approche courante dans ce domaine est la méthode du gradient de stratégie (PG)[20], où l'on met à jour les paramètres de l'estimateur de la stratégie le long de la direction du gradient de la récompense à long terme maximale. Cela signifie que l'algorithme ajuste les paramètres de la stratégie pour maximiser les récompenses globales.

Diverses méthodes de gradient de stratégie ont été proposées, notamment REINFORCE [21], G(PO)MDP [22] et des algorithmes d'acteur-critique. Ces méthodes estiment le gradient de différentes manières, offrant ainsi une variété d'approches pour entraîner efficacement des politiques dans des environnements complexes.

1.2.3 Deep Q Learning

Dans cet algorithme, nous utilisons les DQN (Deep Q Networks), qui sont des réseaux de neurones profonds. Les DQN constituent un algorithme d'apprentissage par renforcement (RL) basé sur les valeurs. Contrairement à l'approche traditionnelle de Q-learning, qui utilise des itérations de valeur pour déterminer les valeurs Q et trouver la fonction Q optimale, les DQN utilisent un approximateur de fonction pour estimer la fonction Q optimale, en l'occurrence, des réseaux de neurones profonds [23].

L'objectif de ce réseau est d'approximer la fonction Q optimale, qui satisfait l'équation de Bellman. Pour ce faire, la perte du réseau est déterminée en comparant les valeurs Q prédites aux valeurs Q cibles obtenues du côté droit de l'équation de Bellman. Une fois la perte calculée, le réseau met à jour ses poids à l'aide d'une méthode d'optimisation, telle que la descente de gradient stochastique (SGD) combinée à la rétropropagation. Ce processus permet de minimiser la perte et d'améliorer progressivement l'estimation de la fonction Q [24].

1.2.3.1 Les méthodes de gradient de stratégie (Policy gradient)

Les méthodes de gradient de stratégie reposent sur l'optimisation directe d'une stratégie paramétrée en fonction du rendement attendu, c'est-à-dire la récompense cumulative à long terme, par descente de gradient. Cette approche vise donc à modéliser et optimiser la stratégie directement. Lorsque la stratégie est représentée par une fonction paramétrée et que la valeur de la fonction de récompense dépend de cette stratégie, l'objectif est d'ajuster les paramètres pour maximiser la récompense [25].

Une méthode récente de gradient de stratégie est le Deep Deterministic Policy Gradient (DDPG)[26]. Cette technique, de type Actor-Critic, apprend simultanément une fonction Q (en utilisant des techniques de type DQN) et une stratégie (en utilisant des techniques de Policy Gradient).

1.2.3.2 Les méthodes acteurs-critiques (Actor Critic)

Les méthodes Actor-Critic sont des techniques de renforcement basées sur la différence temporelle (TD), qui utilisent deux structures distinctes pour représenter la stratégie et la fonction de valeur de manière indépendante. La structure responsable de la stratégie est appelée l'acteur, car elle est utilisée pour sélectionner les actions à entreprendre. La fonction de valeur estimée est appelée la critique, car elle évalue les actions prises par l'acteur en fournissant des retours critiques sur ces actions.

Dans ce cadre, l'acteur est un modèle paramétré qui propose des actions à partir d'un ensemble d'états observés dans l'environnement, cherchant à maximiser la récompense cumulative. La critique, quant à elle, estime la valeur de l'état-action en calculant une fonction de valeur, souvent une fonction Q, qui prédit la récompense attendue de chaque action dans un état donné.

L'apprentissage se fait par l'ajustement des deux composantes : l'acteur modifie sa stratégie basée sur les retours reçus de la critique, tandis que la critique ajuste ses estimations de valeur en utilisant l'erreur de TD. Cette erreur de TD mesure la différence entre la récompense réelle reçue et la récompense estimée, permettant ainsi à la critique de fournir des retours plus précis à l'acteur pour l'amélioration continue de la stratégie.

1.2.3.3 Conclusion

Il existe plusieurs nouveaux algorithmes efficaces pour traiter le problème d'apprentissage par renforcement, souvent développés comme des extensions ou des combinaisons des méthodes précédentes. Par exemple, l'apprentissage Q-profond (Deep Q-Learning) est un algorithme basé sur la valeur qui utilise des réseaux de neurones profonds pour estimer les fonctions Q. Le gradient de stratégie, quant à lui, est un algorithme basé sur la politique, optimisant directement les paramètres de la politique pour maximiser la récompense cumulative. Enfin, l'approche acteur-critique combine ces deux perspectives : l'acteur ajuste la politique tandis que le critique évalue et guide cette politique par rapport à la valeur attendue des actions.

Ces méthodes illustrent comment différents aspects de l'apprentissage par renforcement peuvent être intégrés et améliorés pour résoudre des problèmes complexes d'apprentissage dynamique et de prise de décision.

Chapitre 2

Optimisation par l'algorithme "Probabilistic roadmap (PRM) "

2.1 Introduction :

L'algorithme Probabilistic roadmap (PRM) peut être décrit en termes généraux, sans se concentrer sur un type de robot spécifique. L'idée est que pendant la phase d'apprentissage de la Road Map(feuille de Route), une structure de données est construite de manière progressive de manière probabiliste, et que cette structure de données est ensuite utilisée lors de la phase de requête pour résoudre des problèmes individuels de planification de trajectoire pour plusieurs robots.

La structure de données construite pendant la phase d'apprentissage de la Road Map est [1]un graphe non orientée $G = (V, E)$, où les nœuds V sont des configurations libres générées de manière probabiliste pour chaque robot, et les arêtes E correspondent à des chemins réalisables (simples) entre ces configurations pour un ou plusieurs robots. Ces chemins simples, que nous appelons chemins locaux sont calculés par [2] un planificateur local adapté à la coordination multi-robots . . Un choix approprié du planificateur local garantit la complétude probabiliste du planificateur global pour la planification multi-robots.

Dans la phase de requête, étant donné une configuration de départ s et une configuration d'arrivée g pour chaque robot, nous essayons de connecter s et g à des nœuds appropriés s et g dans V. Ensuite, nous effectuons une recherche dans le graphe pour trouver une séquence des chemins locaux dans E connectant s à g pour chaque robot, tout en tenant compte de la coordination entre les robots pour éviter les collisions et optimiser les déplacements. Les chemins générés dans la phase de requête sont essentiellement des séquences de chemins locaux pour chaque robot, et par conséquent, les propriétés de ces chemins globaux sont induites par le planificateur local adapté à la planification multi-robots.

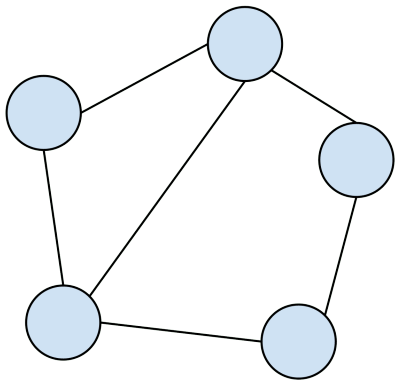


Fig. 2.1 : Exemple de graphe non orienté à 5 sommets.

[1]Un graphe non orientée est un type de structure de données utilisée en théorie des graphes pour représenter des relations entre des objets. Dans un graphe non orientée, les connexions entre les nœuds (ou sommets) sont bidirectionnelles, ce qui signifie que si un nœud A est connecté à un nœud B, alors B est également connecté à A.

[2]Un planificateur local L est simplement une fonction qui prend deux configurations en arguments pour chaque robot et renvoie un chemin les reliant, qui est réalisable en l'absence d'obstacles (c'est-à-dire que le chemin respecte les contraintes des robots et évite les collisions avec les autres robots)

2.2 La phase d'apprentissage

La phase d'apprentissage se compose de deux étapes successives, que nous appelons l'étape de construction et l'étape d'expansion. L'objectif de la première est d'obtenir un graphe raisonnablement connecté, avec suffisamment de sommets pour assurer une couverture plutôt uniforme de [3]l'espace libre C (free C-space [27]) et pour s'assurer que la plupart des régions "difficiles" dans cet espace contiennent au moins quelques nœuds. La deuxième étape vise à améliorer davantage la connectivité de ce graphe. Elle sélectionne des nœuds , selon une évaluation basée sur des critères spécifiques, se trouvent dans des régions difficiles de l'espace C et étend le graphe en générant des nœuds supplémentaires dans leurs voisinages. Ainsi, la couverture de C_f par la feuille de route (Road Map) finale n'est pas uniforme, mais dépend de la complexité locale de l'espace C.

2.2.1 L'étape de construction

Initialement, le graphe $G = (V, E)$ est vide. Ensuite, de manière répétée, une configuration libre aléatoire est générée et ajoutée à V. Pour chaque nouveau nœud c ainsi créé, nous sélectionnons un certain nombre de nœuds parmi les nœuds actuels de V et essayons de connecter c à chacun d'eux en utilisant le planificateur local L. Chaque fois que ce planificateur parvient à calculer un chemin réalisable entre c et un nœud sélectionné n, l'arête (c, n) est ajoutée à E. Le chemin local effectif n'est pas mémorisé.

La sélection des nœuds auxquels nous essayons de connecter c est effectuée comme suit. Tout d'abord, une ensemble V_c de voisins candidats est choisi parmi V. Cet ensemble est constitué de nœuds situés à une certaine distance de c selon une certaine métrique D. Ensuite, nous choisissons les nœuds de V_c dans l'ordre de la distance croissante par rapport à c. Nous essayons de connecter c à chacun des nœuds sélectionnés s'il n'est pas déjà connecté graphiquement à c. Par conséquent, aucun cycle ne peut être créé et le graphe résultant est une forêt, c'est-à-dire une ensemble de collection d'arbres. Étant donné qu'une requête ne réussit jamais grâce à une arête faisant partie d'un cycle, il est en effet raisonnable de ne pas consommer de temps et d'espace mémoire à calculer et stocker une telle arête. Cependant, dans certains cas, l'absence de cycles peut amener la phase de requête à construire des chemins inutilement longs. Cet inconvénient peut facilement être éliminé en appliquant des techniques de lissage soit à la Road Map pendant la phase d'apprentissage, soit aux chemins particuliers construits dans la phase de requête, soit aux deux. Même si la Road Map contenait des cycles, de telles opérations de lissage produiraient éventuellement de meilleurs chemins.

[3]L'espace libre C pour un robot est l'ensemble des configurations où il peut se déplacer sans rencontrer d'obstacles physiques, prenant en compte sa position et son orientation dans l'environnement. Cela inclut les restrictions liées à la géométrie de l'espace ainsi que les limitations de mouvement propres au robot, comme sa taille et la cinématique .

Chaque fois que le planificateur local parvient à trouver un chemin entre deux nœuds, les composantes connectées de R sont mises à jour dynamiquement. Par conséquent, aucune recherche dans le graphe n'est nécessaire pour décider si un nœud sélectionné dans V_c est déjà connecté à c ou non.

Pour préciser davantage la présentation de la solution , on considère :

- Δ comme une fonction symétrique $\mathcal{C}_f \times \mathcal{C}_f \to \{0, 1\}$, permettant de déterminer si le planificateur local peut calculer un chemin entre les deux configurations fournies en tant qu'arguments [28] ;
- D comme une fonction [4]$\mathcal{C} \times \mathcal{C} \to R^+ \cup \{0\}$, nommée fonction de distance, qui définit une pseudométrique dans $\{C\}$[28]. (on pose simplement que D soit symétrique .)

L'algorithme de construction peut être décrit comme suit :

Algorithm 1 Construction du Road Map [29]

$V \leftarrow \emptyset$
$E \leftarrow \emptyset$
Boucle
  $c \leftarrow$ une configuration libre choisie aléatoirement
  $V_c \leftarrow$ un ensemble de voisins candidats de c choisis parmi $N = V$
  $N \leftarrow N \cup c$
for all $n \in V_c$, dans l'ordre croissant de $D(c, n)$ **do**
  if $\neg$ même composante connectée $(c, n) \wedge \Delta(c, n)$ **then**
    $E \leftarrow E \cup (c, n)$
  end if
end for
Mettre à jour les composantes connectées de R.

voici une explication ligne par ligne de l'algorithme de construction :

- Initialisation de l'ensemble V comme étant vide : V est l'ensemble des configurations explorées jusqu'à présent.
- Initialisation de l'ensemble E comme étant vide : E est l'ensemble des arêtes de la roadmap, qui représente les connexions entre les configurations.
- Début de la boucle principale de l'algorithme.

[4]C'est une fonction qui prend deux éléments de l'espace $\{C\}$ (généralement des configurations ou des points dans un espace donné) en tant qu'entrée et renvoie un nombre réel positif ou nul. Cette fonction est souvent utilisée pour mesurer la distance ou la similarité entre deux éléments de l'espace $\{C\}$.

- Sélection d'une configuration libre c de manière aléatoire : une configuration libre est une position valide qui n'est pas occupée par un obstacle ou une configuration déjà explorée.

- Sélection d'un ensemble V_c de voisins candidats de c parmi les configurations déjà explorées N : cela permet de limiter la recherche aux configurations voisines de c.
- Ajout de la configuration c à l'ensemble V : V est mis à jour avec la nouvelle configuration explorée.
- Pour chaque voisin n dans V_c, trié par ordre croissant de la distance $D(c, n)$ entre c et n, effectuer les étapes suivantes :
- Vérification si les configurations c et n ne sont pas dans la même composante connectée et si un chemin peut être calculé entre elles $\Delta(c, n)$.
- Si les configurations ne sont pas dans la même composante connectée et qu'un chemin peut être calculé entre elles, ajouter l'arête (c, n) à l'ensemble E : cela signifie que les configurations c et n sont connectées dans la roadmap.
- Mettre à jour les composantes connectées de la roadmap R : cela permet de maintenir la connectivité de la roadmap.

Voici une figure qui illustre le résultat de l'algorithme de construction de la feuille de route (Road Map).

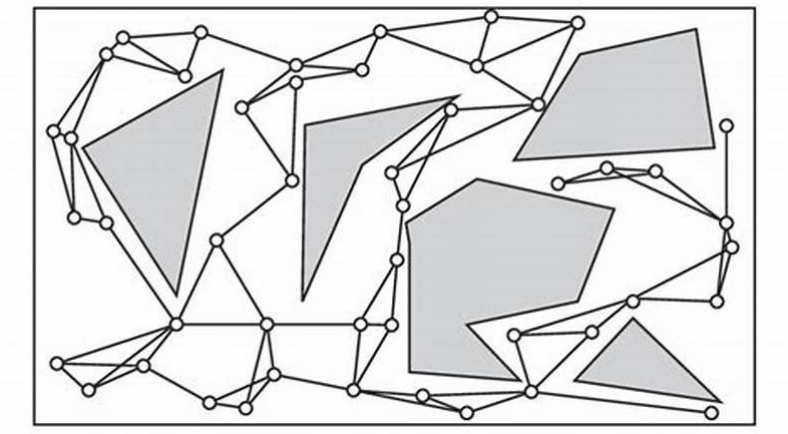


FIG. 2.2 : Construction de la Road Map

Il reste encore plusieurs aspects de l'algorithme ci-dessus à préciser. En particulier, il est nécessaire de définir le processus de création des configurations aléatoires en (4), de proposer un planificateur local pour l'étape (8), de clarifier la notion de voisin candidat en (5), et de choisir la fonction de distance D utilisée en (7).

2.2.1.1 Création d'une configuration aléatoire

Les nœuds de R doivent être obtenus à partir d'un échantillonnage aléatoire relativement uniforme de $\{C_f\}$. Chaque configuration est générée en sélectionnant aléatoirement chacune de ses coordonnées à partir de l'intervalle de valeurs correspondant au degré de liberté, en utilisant une distribution de probabilité uniforme sur cet intervalle. Ensuite, la configuration obtenue est vérifiée pour détecter les collisions. Si elle est libre de collision, elle est ajoutée à V ; autrement, elle est rejetée.

La vérification des collisions implique de tester si une partie quelconque du robot entre en collision avec un obstacle, ainsi que de vérifier si deux parties distinctes du robot se croisent mutuellement. Cette vérification peut être effectuée en utilisant diverses techniques générales existantes [30]. Dans le cadre de l'implémentation générale considérée dans le chapitre 5, le test est réalisé de manière analytique en utilisant un vérificateur de collision itératif.

2.2.1.2 Les planificateurs locaux

Les planificateurs locaux sont essentiels dans la construction et la manipulation des cartes de chemin, aussi bien lors de l'établissement des connexions entre les nœuds qu'entre les configurations initiale et finales (cible) (qinit et qgoal) et la carte de chemin lors du traitement des demandes de planification. On distingue généralement deux types de planificateurs locaux : les simples et les plus sophistiqués.

Les planificateurs locaux puissants ont souvent la capacité de trouver un chemin lorsqu'il existe, ce qui signifie qu'ils peuvent nécessiter peu de nœuds pour construire une carte de chemin robuste capable de répondre efficacement aux demandes de planification. Bien que ces planificateurs puissent être plus lourds en termes de calcul, leur utilisation est efficace lorsque la carte de chemin n'est pas très étendue, car l'algorithme de planification n'a pas besoin de les invoquer fréquemment.

En revanche, les planificateurs locaux simples sont rapides mais moins efficaces, ce qui signifie qu'ils peuvent nécessiter davantage de configurations pour construire une carte de chemin adéquate par conséquent, ils sont souvent sollicité à plusieurs reprises pour établir des connexions entre les nœuds.‘

L'algorithme suppose dans son fonctionnement que le planificateur local est symétrique et déterministe, mais il est également envisageable d'utiliser un planificateur local asymétrique et non déterministe. Dans ce cas, la connexion entre deux configurations ne

garantit pas nécessairement la possibilité d'effectuer le trajet inverse. Ainsi, la structure de la carte de chemin peut être orientée ou non (Graphe orientée), en fonction de la capacité du robot à suivre les chemins dans les deux sens.

Lorsqu'un planificateur local est déterministe, le chemin retourné est toujours le même entre deux configurations, ce qui signifie que la carte de chemin n'a pas besoin de stocker ce chemin spécifique, car il peut être recalculé au besoin. En revanche, avec un planificateur local non déterministe, chaque arête de la carte de chemin doit être associée au chemin spécifique calculé, ce qui peut augmenter considérablement les exigences de stockage de la carte de chemin.

Definition 1 [31] Soit Δ un planificateur local, et soit $q, q' \in \mathcal{C}_{\text{free}}$ et $\Delta(q, q') \neq NIL$: On dit que Δ est symétrique si le robot peut se déplacer de q à q, en exécutant le chemin calculé par Δ, et de q' à q en exécutant le même chemin en inverse.

Definition 2 [31] Soit Δ un planificateur local, et soit $q, q' \in \mathcal{C}_{\text{free}}$: On dit que Δ est déterministe s'il produit toujours le même chemin entre q et q'.

2.2.1.3 la notion de voisin candidat

Les voisins d'un nœud. Un autre choix important à faire concerne l'ensemble V_c, les voisins candidats de c. Le planificateur local sera appelé pour connecter c avec les nœuds dans V_c et le coût cumulatif de ces invocations domine le temps d'apprentissage.

Nous évitons les appels du planificateur local susceptibles de retourner un échec en ne soumet que des paires de configurations dont la distance relative (selon la fonction de distance D) est inférieure à un seuil constant **maxdist**. Ainsi, nous définissons :

$$V_c = \{\tilde{c} \in N \mid D(c, \tilde{c}) \leq \text{ maxdist } \}.$$

2.2.1.4 La fonction de distance

La fonction D est utilisée à la fois pour construire et trier l'ensemble V_c des voisins candidats de chaque nouveau nœud c. Elle devrait être définie de manière à ce que pour toute paire (c, n) de configurations, $D(c, n)$ reflète la probabilité que le planificateur local échoue à calculer un chemin réalisable entre ces configurations. Une possibilité est de définir $D(c, n)$ comme une mesure (aire/volume) de la région de l'espace de travail balayée par le robot lorsqu'il se déplace le long du chemin calculé par le planificateur local entre c et n en l'absence d'obstacles. Ainsi, chaque planificateur local induirait automatiquement sa propre fonction de distance spécifique. Comme le calcul exact des aires/volumes balayés tend à être assez chronophage ET Difficile , une mesure approximative mais peu coûteuse de la région balayée donne de meilleurs résultats pratiques., lorsque le planificateur local général décrit ci-dessus est utilisé pour connecter c et n, $D(c, n)$ peut être défini comme suit :

$$D(c,n) = \max_{x \in \text{robot}} \|x(n) - x(c)\|,$$

où x désigne un point sur le robot, $x(c)$ est la position de x dans l'espace de travail lorsque le robot est en configuration c, et $|x(n) - x(c)|$ est la distance euclidienne entre $x(c)$ et $x(n)$.

2.2.2 L'étape d'expansion

Si le nombre de nœuds générés lors de l'étape de construction est suffisamment élevé, l'ensemble V fournit une couverture assez uniforme de l'espace libre C. Dans les scènes simples, R est alors bien connecté. Mais dans des situations plus contraignantes où l'espace libre $\mathcal{C}_f$ est effectivement connecté, R se compose souvent de quelques grands composants et de plusieurs petits. Il ne capture donc pas efficacement la connectivité de $\mathcal{C}_f$.

L'étape d'expansion vise à améliorer la connectivité du graphe R généré par l'étape de construction. En général, si le graphe est déconnecté à un endroit où $\mathcal{C}_f$ ne l'est pas, cet endroit correspond à une région étroite, donc difficile, de l'espace libre C. L'idée sous-jacente à l'étape d'expansion est de sélectionner un certain nombre de nœuds dans N qui sont susceptibles de se trouver dans de telles régions et de les "étendre". En étendant une configuration c, nous entendons sélectionner une nouvelle configuration libre dans le voisinage de c, ajouter cette configuration à V et essayer de la connecter à d'autres nœuds de V de la même manière que dans l'étape de construction. Ainsi, l'étape d'expansion augmente la densité des configurations de la feuille de route dans les régions de $\mathcal{C}_f$ supposées difficiles. Étant donné que les "lacunes" entre les composants du graphe R se trouvent généralement dans ces régions, la connectivité de R est susceptible d'augmenter.

2.3 La phase de requête

Durant la phase de requête, des chemins doivent être trouvés entre des configurations de départ et d'arrivée arbitraires, en utilisant la feuille de route construite lors de la phase d'apprentissage. Supposons pour l'instant que l'espace libre C est connecté et que la feuille de route se compose d'un unique composant connecté R. Étant donné une configuration de départ s et une configuration d'arrivée g, nous essayons de connecter s et g à deux nœuds de R, respectivement $\tilde{s}$ et $\tilde{g}$, avec des chemins réalisables P_s et P_g. Si cela échoue, la requête échoue. Sinon, nous calculons un chemin P dans R connectant $\tilde{s}$ à $\tilde{g}$. Un chemin réalisable de s à g est finalement construit en concaténant P_s, le chemin recalculé correspondant à P, et P_g inversé . Si on le souhaite, ce chemin peut être amélioré en exécutant un algorithme de lissage dessus. Les techniques de lissage qui peuvent être utilisées ici [32], elle sélectionne des segments aléatoires du chemin global et essaie de les raccourcir en utilisant le planificateur local, et il existe la méthode [33] qui effectue de manière itérative des opérations géométriques locales (c'est-à-dire en coupant les coins des triangles).

La question principale est de savoir comment calculer les chemins P_s et P_g. Les requeetes devraient de préférence se terminer quasi-instantanément c'est à dire se terminer rapidement, donc aucun algorithme coûteux n'est souhaité ici. La stratégie pour connecter s à R est de considérer les nœuds dans R dans l'ordre croissant de la distance à partir de s (selon D) et d'essayer de connecter s à chacun d'eux avec le planificateur local, jusqu'à ce qu'une connexion réussisse. on ignorons les nœuds situés à une distance supérieure à maxdist de s, car la chance de succès du planificateur local est très faible.

La reconstruction d'un chemin de robot à partir de la séquence de nœuds dans P se réduit à la concaténation des chemins qui amènent le robot entre les nœuds adjacents dans P. Certains de ces chemins ont été produits par des marches aléatoires imprévisibles pendant la phase d'apprentissage et sont stockés dans les arêtes pertinentes de R. Les chemins correspondant aux connexions trouvées lors de l'apprentissage par le planificateur local sont recalculés. Le planificateur local est déterministe et produira le même chemin chaque fois qu'il est appelé avec les mêmes configurations d'entrée. Les collisions n'ont pas besoin d'être vérifiées le long des chemins locaux recalculé si le planificateur local a la propriété qu'il abandonne lorsqu'une collision est détectée : toutes les configurations intermédiaires le long du chemin ont été vérifiées pour une collision lorsque le chemin local a été calculé pour la première fois. Un exemple de planificateur ayant la propriété ci-dessus est le planificateur en ligne droite[34] . Si le planificateur local effectue une certaine action (déterministe) lorsqu'une collision est détectée, alors les collisions doivent être vérifiées le long du chemin recalculé afin que la même action puisse être répétée juste après qu'une collision soit détectée.

En général, cependant, la feuille de route peut se composer de plusieurs composants connectés R_i, $i = 1, 2, \ldots, p$. C'est généralement le cas lorsque l'espace libre C n'est pas lui-même connecté. Cela peut également se produire lorsque l'espace libre C est connecté, par exemple si la feuille de route n'est pas suffisamment dense. Si la feuille de route contient plusieurs composants, nous essayons de connecter à la fois s et g à deux nœuds dans le même composant, en commençant par le composant le plus proche de s et de g. Si la connexion de s et g à un certain composant R_i réussit, un chemin est construit comme dans le cas à composante unique. La méthode retourne un échec si elle ne peut pas connecter à la fois la configuration de départ et la configuration d'arrivée au même composant de la feuille de route. Comme dans la plupart des exemples, la feuille de route se compose de quelques composants seulement, l'échec est rapidement détecté.

Si les requêtes de planification de chemin échouent fréquemment, cela indique que la feuille de route ne capture peut-être pas adéquatement la connectivité de l'espace libre C. Par conséquent, plus de temps devrait être consacré à la phase d'apprentissage, c'est-à-dire que T_A (temps d'apprentissage)devrait être augmenté. Cependant, il n'est pas nécessaire de construire une nouvelle feuille de route à partir du début. Comme la phase d'apprentissage est progressive, nous pouvons simplement étendre la feuille de route actuelle en reprenant l'algorithme d'étape de construction et,ou l'algorithme d'étape d'expansion, en commençant par le graphe de feuille de route actuel, améliorant ainsi les phases d'apprentissage et de requête.

Algorithm 2 Algorithme de Résolution de Requête [29]

Entrées :
q_{init} : la configuration initiale
q_{goal} : la configuration cible
k : le nombre de voisins les plus proches à examiner pour chaque configuration
$G = (V, E)$: le graphe calculé par l'algorithme 1
Sortie :
Un chemin de q_{init} à q_{goal} ou un échec
$N_{q_{init}} \leftarrow$ les k voisins les plus proches de q_{init} dans V selon dist
$N_{q_{goal}} \leftarrow$ les k voisins les plus proches de q_{goal} dans V selon dist
$V \leftarrow \{q_{init}\} \cup \{q_{goal}\} \cup V$
Définir q' comme le voisin le plus proche de q_{init} dans $N_{q_{init}}$
repeat
    if $\Delta(q_{init}, q') \neq$ NIL **then**
        $E \leftarrow \{(q_{init}, q')\} \cup E$
    else
        Définir q' comme le prochain voisin le plus proche de q_{init} dans $N_{q_{init}}$
    end if
until une connexion a réussi ou l'ensemble $N_{q_{init}}$ est vide
Définir q' comme le voisin le plus proche de q_{goal} dans $N_{q_{goal}}$
repeat
    if $\Delta(q_{goal}, q') \neq$ NIL **then**
        $E \leftarrow \{(q_{goal}, q')\} \cup E$
    else
        Définir q' comme le prochain voisin le plus proche de q_{goal} dans $N_{q_{goal}}$
    end if
until une connexion a réussi ou l'ensemble $N_{q_{goal}}$ est vide
$P \leftarrow$ chemin le plus court(q_{init}, q_{goal}, G)
if P n'est pas vide **then**
    retourner P
else
    retourner échec
end if

2.4 Cas Multi agent

2.4.1 Introduction :

La généralisation de l'algorithme de la Road Map probabiliste (PRM) pour le cas multirobots est essentielle pour permettre à plusieurs robots de coopérer efficacement pour atteindre des objectifs communs. La planification de trajectoires pour plusieurs robots est une tâche considérablement plus complexe que pour un seul robot, car elle doit non seulement coordonner les mouvements individuels de chaque robot, mais aussi prendre en compte les interactions entre les robots et résoudre les conflits potentiels.

Dans ce contexte, il est nécessaire d'adapter les techniques de planification existantes pour répondre aux défis spécifiques posés par le mouvement simultané de plusieurs entités dans un environnement partagé. Cette adaptation implique l'extension de l'algorithme PRM, bien établi dans le domaine de la robotique pour la planification de trajectoires, afin de gérer efficacement les interactions et la coordination entre plusieurs robots[35].

2.4.2 Généralisation de l'algorithme PRM : GPRM

L'algorithme proposé résout le problème général de planification de mouvement multi-agents en présence d'incertitude de processus et de cartes stochastiques. L'algorithme construit une feuille de route en utilisant AGPRM entre chaque paire d'emplacements de départ et d'objectif. La boucle à la ligne 3 illustre cela, c'est-à-dire qu'avec chaque combinaison d'emplacements de départ et d'objectif, un nouvel AGPRM (qui est paramétré à l'emplacement objectif actuel) est résolu.

Dans le cas de plusieurs agents du même type, les lignes 4-5 garantissent que la feuille de route existante est soit étendue davantage, soit utilisée pour trouver une solution entre l'emplacement du ième agent et l'emplacement objectif du jème agent.

Dans le cas d'agents hétérogènes, différentes feuilles de route pour différents types d'agents doivent être construites, ce qui est donc plus intensif en calcul. Les points de repère peuvent encore être partagés, mais les calculs des coûts et des probabilités de transition impliqueront l'exécution des simulations et la construction d'une feuille de route différente chaque fois qu'un nouveau type d'agent entre dans le système. La ligne 7 met en évidence cette caractéristique de l'algorithme.

Une fois le coût des transitions entre chaque emplacement de départ et d'objectif calculé, le problème de routage est résolu en utilisant l'algorithme MTSP prospectif. La solution de MAGPRM est la séquence des emplacements d'objectif à visiter par chaque agent, qui prend en compte l'incertitude du processus dans la dynamique des agents et leur traversée le long d'une Road Map.

Algorithm 3 Multi-Agents GPRM (MAGPRM) [35]

Entrées :
A : Ensemble d'agents
x_0 : Emplacements de départ
x_g : Emplacements d'objectifs
$p_{\min}$: Probabilité minimale pour l'environnement
Sortie :
Un chemin pour chaque agent ou un échec
for chaque agent i à l'emplacement de départ $x_{0_i} \in x_0$ **do**
    for chaque emplacement d'objectif j, $x_{g_j} \in x_g$ **do**
        $p_s \leftarrow$ probabilité de succès $p_s(x_{0_i} \rightarrow x_{g_j})$
        repeat
            if le type de l'agent i est déjà évalué **then**
                Utiliser la feuille de route déjà existante pour construire et connecter davantage
            else
                Construire AGPRM, paramétré avec l'emplacement d'objectif x_{g_j} et le type d'agent de l'agent i
                Construire une matrice des coûts de transition pour chaque type d'agent (c'est-à-dire les coûts de transition entre les agents)
                Résoudre le problème de routage pour chaque agent(la phase de réquete)
            end if
        until $p_s \geq p_{\min}$
    end for
end for
retourner les chemins calculés pour chaque agent ou échec si aucun chemin ne satisfait $p_{\min}$

Chapitre 3

Optimisation par "Reinforcement Learning RL"

3.1 Introduction

Un agent d'apprentissage par renforcement (RL) apprend en interagissant avec un environnement dynamique. À chaque instant, l'agent observe l'état de l'environnement et prend une action qui le modifie vers un nouvel état. Une récompense scalaire évalue la qualité de chaque action, et l'objectif de l'agent est de maximiser la récompense cumulative au fil du temps. Contrairement à l'apprentissage supervisé, où les actions correctes sont fournies, le feedback RL est moins précis mais plus informatif. De plus, des algorithmes bien établis assurent la convergence de l'apprentissage pour un seul agent, ce qui rend cette approche encore plus prometteuse pour le contexte multi-agents [36].

3.2 Processus de Décision de Markov :

Le Processus de Décision de Markov (MDP) est un cadre couramment utilisé pour modéliser la prise de décision d'un agent dans un environnement où les états du système sont totalement observables [37].

Un MDP est défini par cinq éléments : l'ensemble des états possibles (S) et l'ensemble des actions disponibles (A), la fonction de transition de probabilité (P) qui indique la probabilité de passer d'un état à un autre en effectuant une action donnée, la fonction de récompense (R) qui détermine la récompense immédiate obtenue par l'agent pour une transition donnée, et le facteur de remise (γ) qui pondère les récompenses futures par rapport à celles immédiates.

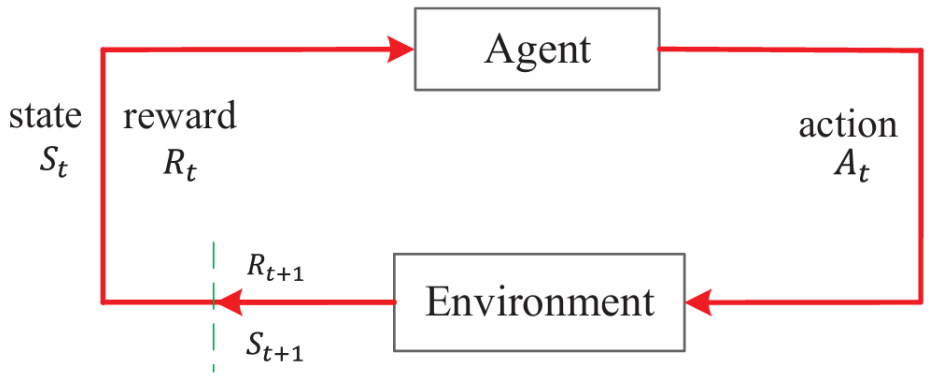


Fig. 3.1 : Interaction agent-environnement dans le processus de décision de Markov [38].

À chaque étape, l'agent choisit une action en fonction de l'état actuel du système. Cette action entraîne une transition vers un nouvel état et une récompense est reçue en conséquence. L'objectif est de trouver une stratégie, représentée par une fonction π, qui assigne à chaque état une distribution de probabilité sur les actions à entreprendre, maximisant ainsi la récompense cumulée au fil du temps [39].

En raison de la propriété markovienne, la recherche de la stratégie optimale dans un Processus de Décision de Markov (MDP) peut être réalisée soit par des approches de Programmation Dynamique (DP), soit par des méthodes d'Apprentissage par Renforcement (RL). Dans la DP, l'agent dispose de la connaissance complète de la fonction de récompense, tandis que dans le RL, l'agent apprend à partir de son expérience uniquement.

Lorsqu'on utilise le RL pour résoudre un MDP, on a généralement le choix entre deux approches principales : calculer les fonctions de valeur ou les valeurs Q pour chaque état, puis choisir les actions en fonction de ces valeurs ; ou bien calculer directement une stratégie qui spécifie les probabilités de choisir chaque action en fonction de l'état actuel, et agir en conséquence.

Chaque algorithme d'apprentissage par renforcement doit déterminer quelle action prendre à chaque état. Cependant, lors de l'apprentissage, l'algorithme n'a pas besoin de tenir compte de la stratégie suivie pour prendre ces décisions. Les algorithmes qui prennent en compte la stratégie utilisée pour les décisions précédentes sont appelés "on-policy", tandis que ceux qui l'ignorent sont qualifiés d'"off-policy".

Un exemple bien connu d'algorithme off-policy est le Q-Learning. Dans le Q-Learning, la mise à jour des valeurs Q utilise l'action qui maximise la valeur Q, indépendamment de la stratégie réellement suivie, ce qui signifie que la stratégie réelle pourrait restreindre cette action ou en choisir une autre. En revanche, la variante du Q-Learning qui tient compte de la stratégie suivie est appelée SARSA. Dans SARSA, la règle de mise à jour utilise l'action réellement choisie selon la stratégie en cours.

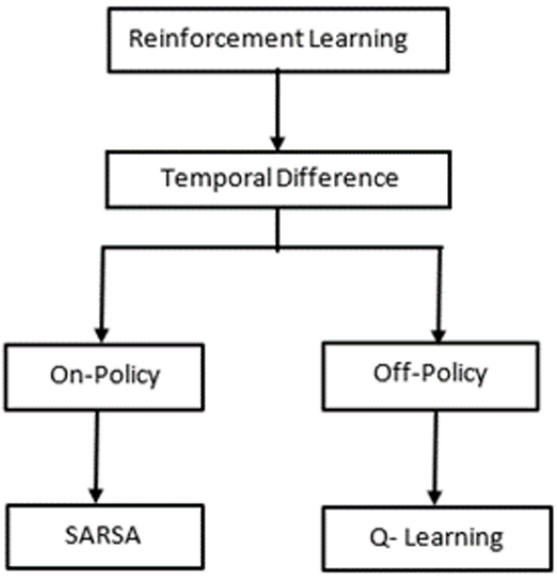


Fig. 3.2 : Schéma fonctionnel de la différence temporelle (TD)

3.3 Q-Learning

Q-Learning est un algorithme off-policy utilisé pour l'apprentissage par [1]différence temporelle TD. Il a été démontré que, avec un nombre suffisant d'itérations pour chaque stratégie [2]ϵ-greedy, l'algorithme converge vers une approximation de la fonction Q optimale avec une probabilité de 1.

3.3.1 Équation de Bellman Pour le Q-Learning :

L'équation de Bellman pour le Q-Learning est une équation fondamentale dans l'apprentissage par renforcement, notamment pour les méthodes basées sur les valeurs comme le Q-Learning. Cette équation exprime une relation importante entre la valeur d'une action dans un état donné (appelée la fonction Q) et les récompenses futures attendues.

L'équation de Bellman pour Q-Learning est donnée par :

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q\left(s', a'\right) - Q(s,a) \right]$$

où :

- $Q(s,a)$ est la valeur Q pour l'état s et l'action a,
- α est le taux d'apprentissage,
- r est la récompense immédiate,
- γ est le facteur de remise,
- s' est l'état suivant après avoir pris l'action a depuis l'état s
- a' est l'action choisie dans l'état s'.

3.3.2 Algorithme et explication

- Initialiser $Q(s,a)$ arbitrairement : Tout d'abord, l'algorithme initialise une fonction de valeur $Q(s,a)$ arbitrairement pour chaque paire état-action possible. Cette fonction de valeur estime la récompense attendue lorsque l'agent est dans un état donné s et effectue une action donnée a.
- Répéter : Les étapes suivantes sont répétées jusqu'à ce que l'état s soit terminal.

[1] La différence temporelle (TD) est une technique utilisée en apprentissage par renforcement (RL) pour estimer la valeur d'état d'un agent dans un environnement donné

[2]La politique greedy consiste à choisir constamment l'action avec la récompense attendue la plus élevée, sans exploration. En revanche, la politique $\epsilon - greedy introduit un taux d'exploration\ (0-1)$: $avec probabilit\ ,une action alatoire est choisie, sinon l'action optimale est slectionne.$

Algorithm 4 Q-Learning [40]

1: Initialiser $\mathbf{Q}(\mathbf{s}, \mathbf{a})$ arbitrairement
2: **repeat**
3: Choisir **a** depuis **s** en utilisant une politique dérivée de **Q** (par exemple, ϵ-glouton)
4: Prendre l'action **a**, observer **r**, $\mathbf{s}'$
5: $\mathbf{Q}(\mathbf{s}, \mathbf{a}) \leftarrow \mathbf{Q}(\mathbf{s}, \mathbf{a}) + \alpha \left[\mathbf{r} + \gamma \max_{\mathbf{a}'} \mathbf{Q}(\mathbf{s}', \mathbf{a}') - \mathbf{Q}(\mathbf{s}, \mathbf{a})\right]$
6: $\mathbf{s} \leftarrow \mathbf{s}'$
7: **until s** est terminal

- Choisir a depuis s en utilisant une politique dérivée de Q (par exemple, epsilon-glouton) : L'agent choisit une action a à partir de l'état actuel s en utilisant une politique basée sur la fonction de valeur Q. La politique epsilon -glouton est couramment utilisée, ce qui signifie que la plupart du temps, l'agent choisit l'action qui maximise la valeur Q pour l'état actuel, mais avec une petite probabilité , il explore de nouvelles actions de manière aléatoire.

- Prendre l'action a, observer r, s' : L'agent prend l'action a dans l'environnement et observe la récompense r résultante ainsi que le nouvel état s'.

- $Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma \max_{a'} Q\left(s', a'\right) - Q(s, a)\right]$: L'algorithme met à jour la fonction de valeur Q pour l'état-action actuel en utilisant la formule de mise à jour Q-learning. Cette formule calcule la différence entre la récompense actuelle et l'estimation de la récompense future, puis met à jour la valeur Q en conséquence. est le taux d'apprentissage qui contrôle la vitesse d'apprentissage, et est le facteur d'actualisation qui indique l'importance relative des récompenses immédiates par rapport aux récompenses futures.

- $s \leftarrow s'$: L'état actuel est mis à jour pour devenir le nouvel état observé s'.

- Jusqu'à ce que s soit terminal : Les étapes 3 à 6 sont répétées jusqu'à ce que l'état s devienne terminal, c'est-à-dire qu'aucune action supplémentaire ne puisse être prise.

3.4 SARSA

L'algorithme SARSA est une méthode On-Policy utilisée dans l'apprentissage par différence temporelle. Sa principale distinction par rapport au Q-Learning réside dans le fait que la récompense maximale pour l'état suivant n'est pas forcément utilisée pour mettre à jour les valeurs Q. Au lieu de cela, une nouvelle action et donc une nouvelle récompense sont sélectionnées en utilisant la même stratégie qui a déterminé l'action initiale. SARSA tire son nom de l'utilisation du quintuplet Q(s,a,r,s',a') pour les mises à jour,
où : s,a sont l'état et l'action actuels, r est la récompense observée dans l'état suivant et s',a' sont la nouvelle paire état-action

3.4.1 Équation de Bellman pour le SARSA

L'équation de Bellman pour SARSA définit comment les valeurs Q sont mises à jour itérativement dans l'algorithme SARSA (State-Action-Reward-State-Action). Elle exprime comment la valeur d'une action dans un état est mise à jour en fonction de la récompense observée après avoir pris cette action, ainsi que de la valeur de la prochaine action choisie dans le nouvel état. Cette mise à jour est effectuée en utilisant un taux d'apprentissage qui pondère l'importance de la nouvelle information par rapport aux valeurs Q existantes.

L'équation de Bellman pour SARSA est la suivante :

$$Q(s,a) = Q(s,a) + \alpha \left[r + \gamma Q\left(s', a'\right) - Q(s,a) \right]$$

Dans cette équation :

- s' est le prochain état après avoir pris l'action a dans l'état s,
- a' est la prochaine action choisie dans l'état s',

Les autres éléments de l'équation ont la même signification que dans Q Learning.

3.4.2 Algorithme et explication

Algorithm 5 SARSA [40]

Initialiser $\mathbf{Q}(\mathbf{s}, \mathbf{a})$ arbitrairement
repeat
    Initialiser $\mathbf{s}$
    Choisir $\mathbf{a}$ depuis $\mathbf{s}$ en utilisant une politique dérivée de $\mathbf{Q}$ (par exemple, ϵ-glouton)
    repeat
        Prendre l'action $\mathbf{a}$, observer $\mathbf{r}$, $\mathbf{s}'$
        Choisir $\mathbf{a}'$ depuis $\mathbf{s}'$ en utilisant une politique dérivée de $\mathbf{Q}$ (par exemple, ϵ-glouton)
        $\mathbf{Q}(\mathbf{s}, \mathbf{a}) \leftarrow \mathbf{Q}(\mathbf{s}, \mathbf{a}) + \alpha \left[\mathbf{r} + \gamma \mathbf{Q}(\mathbf{s}', \mathbf{a}') - \mathbf{Q}(\mathbf{s}, \mathbf{a})\right]$
        $\mathbf{s} \leftarrow \mathbf{s}'$, $\mathbf{a} \leftarrow \mathbf{a}'$
    until $\mathbf{s}$ est terminal
until convergence

Comme nous pouvons le voir, deux étapes de sélection d'action sont nécessaires pour déterminer simultanément la paire état-action suivante. SARSA utilise la stratégie de comportement de l'agent (souvent une stratégie gloutonne) pour sélectionner une action supplémentaire a_{t+1}, puis utilise $Q(s_{t+1}, a_{t+1})$ comme estimation des récompenses futures attendues. En revanche, Q-learning ne s'appuie pas sur la stratégie de comportement pour choisir une action supplémentaire. À la place, il estime les rendements futurs attendus en utilisant $\max Q(s_{t+1}, a)$.

3.5 Conclusion

Q-learning met à jour ses valeurs d'action en utilisant les actions optimales possibles, ce qui le rend plus agressif dans la recherche de la politique optimale globale. SARSA, en revanche, met à jour ses valeurs d'action en fonction des actions réellement prises, ce qui peut conduire à une politique plus sûre et stable dans des environnements stochastiques.

Q-learning est souvent préféré pour l'implémentation car :

- Il est plus agressif dans la recherche de la politique optimale globale.
- Il peut converger plus rapidement vers des solutions optimales.
- Il est plus flexible pour différentes stratégies d'exploration.

En conclusion, bien que SARSA puisse être plus sûr dans certains environnements, Q-learning est généralement considéré comme meilleur pour l'implémentation en raison de sa capacité à converger plus rapidement vers l'optimum global.

3.6 Multi-Agents Reinforcement learning MARL

Le Multi-Agents Reinforcement Learning (MARL) aborde les problèmes de prise de décision séquentielle dans un contexte où plusieurs agents interagissent. Dans cette configuration, l'état du système ainsi que les récompenses reçues par chaque agent sont influencés par les actions conjointes de tous les agents. De manière plus complexe, chaque agent cherche à optimiser sa propre récompense à long terme, qui dépend des stratégies adoptées par tous les autres agents.

Le MARL peut être vu comme une fusion entre l'apprentissage par renforcement, la théorie des jeux et d'autres techniques de recherche de stratégies. En conséquence, il peut être modélisé à travers différents cadres théoriques.

En général, on distingue deux cadres théoriques, en apparence distincts mais étroitement liés, pour aborder les problèmes de MARL : les jeux de Markov/stochastiques et les jeux de forme extensive.

3.6.1 Les jeux de Markov/stochastiques

Une extension naturelle des Processus de Décision Markoviens (MDP) pour représenter l'interaction entre plusieurs agents est donnée par les jeux de Markov (MGs), également appelés jeux stochastiques. Les MGs ont été largement utilisés dans la littérature pour concevoir des algorithmes de Multi-Agents Reinforcement Learning (MARL).

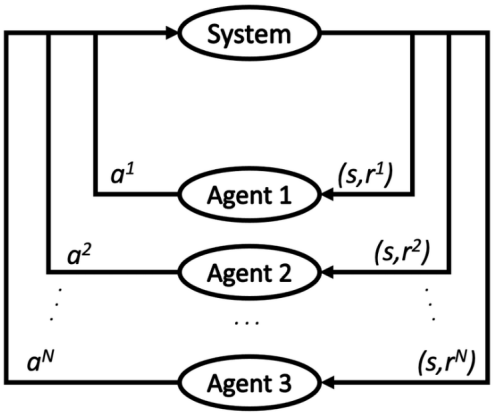


FIG. 3.3 : Extension du Décision de Markov MDP [41]

Définition :

Un jeu de Markov est défini par un tuple $\left(N, S, \{A_i\}_{i\in N}, P, \{R_i\}_{i\in N}, \gamma\right)$ où N désigne l'ensemble agents, S désigne l'espace d'état observé par tous les agents, A_i désigne l'espace d'action de l'agent i.

Soit $\mathrm{A} = A_1 \times \ldots \times A_N$, alors $\mathrm{P} : \mathrm{S}\times\mathrm{A} \to \Delta(\mathrm{S})$ désigne la probabilité de transition de tout état $s \in S$ vers tout état $s' \in \mathrm{S}$ pour toute action conjointe $\mathrm{a} \in \mathrm{A}$. $R_i : \mathrm{S}\times\mathrm{A}\times\mathrm{S} \to \mathbb{R}$

est la fonction de récompense qui détermine la récompense immédiate reçue par l'agent i pour une transition de s vers s', et $\gamma \in [0; 1)$ est le facteur de remise.

Au temps t, chaque agent i $\in$ N exécute une action a_i, selon l'état du système s_t. Le système passe ensuite à l'état s_{t+1}, et récompense chaque agent i par $R_i\left(s_t, a_t, s_{t+1}\right)$.

L'objectif de l'agent i est d'optimiser sa propre récompense à long terme, en trouvant la stratégie $\pi_i : \mathrm{S} \rightarrow \Delta\left(A_i\right)$.

Le concept de solution d'un MG diffère de celui d'un MDP, puisque la performance optimale de chaque agent est contrôlée non seulement par sa propre stratégie, mais aussi par les choix de tous les autres participants du jeu.

3.6.2 Les jeux de forme extensive

Dans le domaine de la théorie des jeux appliquée aux problèmes multi-agents, un jeu de forme extensive offre une spécification détaillée qui inclut la séquence des actions possibles pour chaque agent, les choix disponibles à chaque point de décision, les informations (potentiellement imparfaites) dont dispose chaque agent sur les actions des autres agents lors de la prise de décision, ainsi que les récompenses associées à chaque combinaison de décisions. Bien que ce cadre partage des similitudes avec les jeux de Markov, il est souvent privilégié pour analyser les interactions dynamiques entre les agents, tandis que les jeux de Markov sont davantage adaptés à la modélisation des jeux répétés.

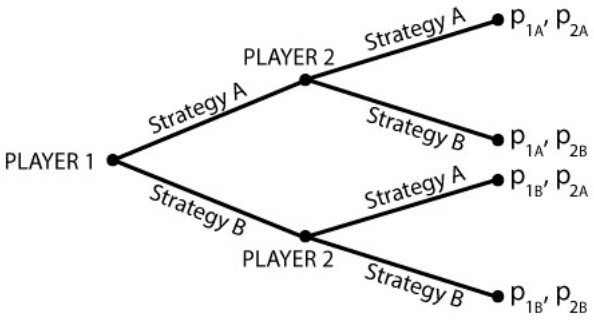


Fig. 3.4 : Les jeux de forme extensive

3.6.3 Solution proposée

3.6.3.1 Description du problème

Imaginons un scénario où N agents évoluent dans un environnement commun, avec pour objectif de réaliser une tâche coopérative, notamment se déplacer en formation tout en évitant les obstacles et les collisions. Chaque agent doit partager sa stratégie de déplacement avec les autres pour assurer une coordination efficace. Ainsi, la réussite de la mission dépend de la capacité des agents à collaborer et à communiquer pour coordonner leurs actions.

3.6.3.2 Motivation

Pour résoudre le problème de planification de trajectoire multi-agents, nous avons cherché une solution présentant des caractéristiques spécifiques. Notre objectif était de développer une approche centralisée où l'ensemble des agents de la formation serait considéré comme un seul agent global. Ainsi, nous pourrions calculer une seule solution pour l'ensemble, une seule fois. Chaque agent aurait alors accès à l'état des autres agents ainsi qu'à leurs stratégies, éliminant ainsi le besoin de tout autre échange d'informations ou communication.

Dans cette approche, tous les agents impliqués dans la tâche coopérative partageraient un objectif commun ainsi qu'une seule fonction de récompense. Dans la prochaine section, nous décrirons un schéma répondant à ces critères.

Les N agents commenceraient par construire un hyperespace d'action et d'état (HAS) de l'ensemble. Ce HAS inclurait l'espace de toutes les actions conjointes admissibles que les agents pourraient entreprendre, ainsi que l'espace global des états englobant les états des N agents. Ce modèle serait représenté par un processus de décision de Markov standard (MDP) avec une fonction de récompense commune.

Enfin, ce MDP serait résolu à l'aide d'un algorithme de planification ou d'apprentissage par renforcement adapté à la nature multi-agents du problème.

3.6.3.3 Centralized Deep Q-learning

Une première approche pour résoudre ce MDP est d'utiliser le Q-learning en tant qu'agent principal d'apprentissage. Toutefois, l'évaluation de cette méthode a mis en évidence des problèmes de performance. En effet, les performances du Q-learning se détériorent proportionnellement au nombre d'agents impliqués dans la tâche. De plus, il est limité aux espaces d'états et d'actions discrets et n'est pas adapté aux états et actions continus, car il repose sur le calcul explicite des valeurs Q pour chaque paire état-action.

Pour surmonter ces limitations, il est courant d'utiliser des approximations de fonctions. Par exemple, des réseaux neuronaux peuvent être employés pour estimer les valeurs Q, évitant ainsi le besoin de calculer une table Q complète pour toutes les paires état-action possibles.

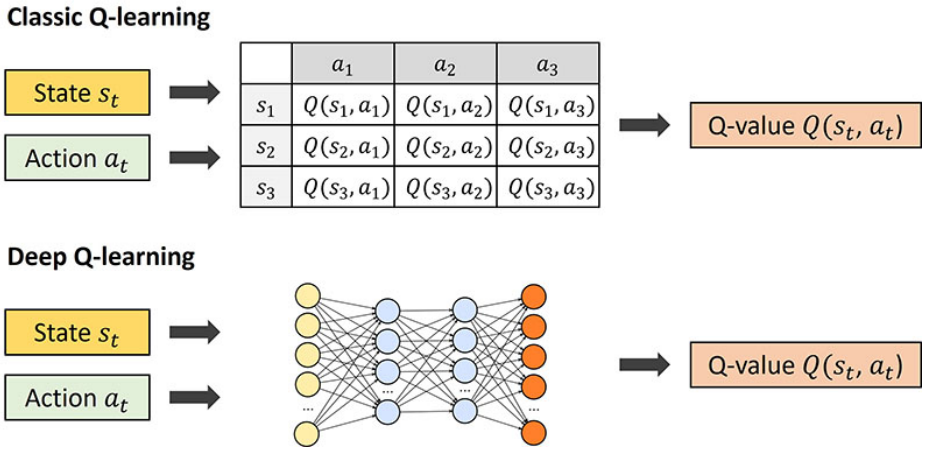


Fig. 3.5 : La différence entre le Q-Learning et le DQN (Deep Q-Network)

Les réseaux de neurones sont des approximateurs de fonctions particulièrement utiles en apprentissage par renforcement lorsque l'espace des états ou des actions est trop vaste pour être entièrement connu. Un réseau de neurones peut être employé pour approximer la fonction de valeur. Autrement dit, les réseaux neuronaux peuvent apprendre à associer des états à des valeurs Q. Plutôt que d'utiliser une table de recherche pour stocker, indexer et mettre à jour tous les états possibles et leurs valeurs correspondantes, ce qui est impraticable pour des problèmes de grande dimension, nous pouvons entraîner un réseau de neurones sur des échantillons de l'espace d'état afin d'apprendre à prédire leurs valeurs.

-L'apprentissage des réseaux de neurones :

Comme pour tous les réseaux de neurones, des coefficients sont utilisés pour approximer la fonction reliant les entrées aux sorties. L'apprentissage consiste à trouver les bons coefficients, ou poids, en ajustant ces poids de manière itérative en suivant le gradient (ou une autre méthode d'optimisation) dans le but de minimiser une fonction de coût, laquelle représente l'erreur d'apprentissage du réseau (Loss).

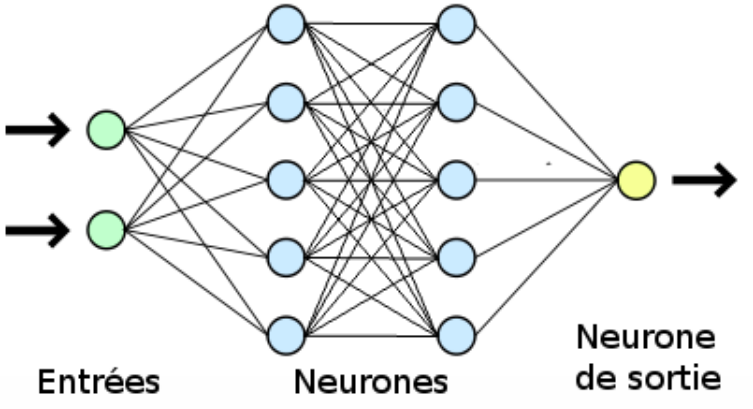


FIG. 3.6 : Deep neural network

- **Fonction objectif (Loss) :**Une fonction objectif sert de critère pour évaluer la qualité des solutions potentielles à un problème d'optimisation. Elle est utilisée pour guider le processus d'apprentissage en définissant un objectif clair : maximiser ou minimiser cette fonction. Par exemple, dans le contexte des réseaux de neurones, la fonction objectif est souvent une fonction de coût (ou de perte) qui mesure l'erreur entre les prédictions du modèle et les valeurs réelles. L'optimisation consiste alors à ajuster les paramètres du modèle de manière à réduire cette erreur, en utilisant des techniques telles que la descente de gradient ou d'autres méthodes d'optimisation, jusqu'à atteindre la meilleure solution possible selon le critère défini par la fonction objectif.

- **Backpropagation :**La rétropropagation du gradient est une méthode utilisée pour calculer le gradient de l'erreur par rapport aux poids de chaque neurone dans un réseau de neurones. Ce processus commence par l'évaluation de l'erreur entre la sortie prédite du réseau et la valeur attendue, puis il propage cette erreur en arrière à travers le réseau, de la dernière couche vers la première. À chaque couche, le gradient de l'erreur par rapport à la sortie de cette couche est calculé, puis ce gradient est utilisé pour ajuster les poids de chaque neurone dans la couche précédente. Ce processus se répète itérativement jusqu'à ce que les poids convergent vers des valeurs qui minimisent l'erreur globale du réseau. La rétropropagation du gradient est la base de l'apprentissage supervisé dans les réseaux de neurones et permet au réseau d'apprendre à partir des exemples d'entraînement en ajustant ses poids pour mieux correspondre aux données observées.

- **Algorithme du gradient :**La descente de gradient est un algorithme d'optimisation itératif pour trou ver un minimum d'une fonction coût. L'idée est de faire des pas répétés dans la direction opposée de la pente (ou pente approximative) de la fonction au point actuel, car c'est la direction de la descente la plus raide.

3.6.3.4 Algorithme de DQN et explication

Un réseau de neurones appelé réseau Q Network est conçu pour produire, pour un état donné s, un vecteur de valeurs d'action $Q(s, a; \theta_i)$, où θ_i représente les paramètres du réseau. Le but de ce réseau est d'apprendre à prédire la fonction Q-optimale.

Pour entraîner le Q Network, on utilise des échantillons d'expérience recueillis par l'agent au fil du temps en interagissant avec son environnement. Chaque expérience prend la forme suivante :

$$e = (\text{ état actuel } s, \text{ état suivant } s', \text{ récompense } r, \text{ action } a \ ).$$

Ensuite, l'optimisation de Q Network se fait en minimisant l'erreur de Bellman, en utilisant la récursion :

$$L_i(\theta_i) = \left(y_i^{DQN} - Q(s, a; \theta_i)\right)^2$$

Et y_i^{DQN}, la valeur cible de $Q(s, a; \theta_i)$, est obtenue par :

$$y_i^{DQN} = r + \gamma \max Q(s', a'; \theta_{i-1})$$

Cependant, on n'a toujours pas le terme $\max Q(s', a'; \theta_{i-1})$. Pour y remédier, Mnih et al. (2015) ont proposé l'utilisation d'un deuxième réseau dit Target Network pour la prédiction de cette valeur.

Le réseau Target Network, avec les paramètres θ_{i-1}, est le même que le réseau Q Network, (même architecture et paramètres initiaux) sauf que ses paramètres sont fixes, et pour chaque pas de temps T, ils sont mis à jour en prenant les paramètres de Q Network comme nouveaux paramètres de Target Network ($\theta_{i-1} = \theta_i$).

Algorithm 6 Entraînement d'un réseau Q Network avec Experience Replay et Target Network

Initialiser la capacité de Replay memory pour le stockage des échantillons d'expérience
Initialiser le réseau Q Network avec des poids aléatoires
Cloner le réseau Q Network et l'appeler le réseau Target Network
for chaque épisode **do**
    Initialiser l'état de départ s
    for chaque pas de temps **do**
        Sélectionner une action a (par exploration ou exploitation)
        Exécuter l'action a
        Observer la récompense r et l'état suivant s'
        Stocker l'expérience (s, a, r, s') dans Replay memory
        Transmettre un échantillon d'expérience au réseau Q Network
        Calculer la perte $L_i(\theta_i) = (y_i^{DQN} - Q(s, a; \theta_i))^2$
        où $y_i^{DQN} = r + \gamma \max Q(s', a'; \theta_{i-1})$
        La descente de gradient met à jour les poids dans le réseau Q Network pour minimiser l'erreur d'apprentissage
        if un certain nombre de pas de temps est écoulé **then**
            Mettre à jour les poids du réseau Target Network avec les poids du réseau Q Network
        end if
    end for
end for

Chapitre 4

Résultats et évaluation expérimentale

4.1 Introduction

Dans ce chapitre, les résultats obtenus aprés expérimentations. Mon L'objectif principal étant de mesurer l'efficacité des approches classique et heuristique présentées respectivement dans les chapitres 2 et 3. Tout d'abord, les performances des algorithmes PRM & MAGPRM et RL en utilisant un seul agent sont évaluées . Ensuite, les tests ont éte étendus à plusieurs agents. Toutes les expérimentations ont été réalisées dans l'environnement MATLAB et Python. Par la suite,nous avons procédé à l'implémentation sur ROS en utilisant les simulateurs Rviz et Gazebo pour simuler des environnements réalistes.

L'objectif de ce chapitre est également de tester théoriquement la faisabilité des deux approches afin d'évaluer leur applicabilité dans des scénarios concrets. Ainsi, en plus de comparer les performances des algorithmes, on va analyser leur robustesse, leur scalabilité et leur adaptabilité à des situations variées. Cette analyse approfondie permettra de mieux comprendre les avantages et les limites de chaque approche, et de déterminer dans quels contextes elles peuvent être les plus efficaces.

4.2 Implémentation de l'algorithme PRM & MAGPRM

Nous avons réalisé des simulations pour examiner les résultats produits par les algorithmes PRM et MAGPRM, tant pour un seul agent que pour un ensemble d'agents décentralisés. Par la suite, nous présenterons une discussion approfondie et tirerons des conclusions.

4.2.1 Cas d'un agent unique

Tout d'abord, nous avons mis en œuvre l'algorithme PRM pour la planification de trajectoire d'un seul agent dans un environnement 2D, en variant les degrés de complexité. L'objectif était de tester et d'évaluer l'efficacité des solutions obtenues dans des scénarios représentatifs d'un environnement réel. Ensuite, nous avons mené une étude comparative en variant le nombre de nœuds en entrée. Cette évaluation s'est focalisée sur le coût de la solution obtenue, évalué par la longueur de la trajectoire, ainsi que sur l'effort requis pour atteindre cette solution. Nous avons choisi de mesurer cet effort en termes de temps d'exécution nécessaire pour chaque algorithme.

4.2.1.1 L'efficacité de l'algorithme pour trouver une solution

Les paramètres d'entrée ont été définis comme suit : un rayon de robot de 0,5 m et un nombre de nœuds de 500.

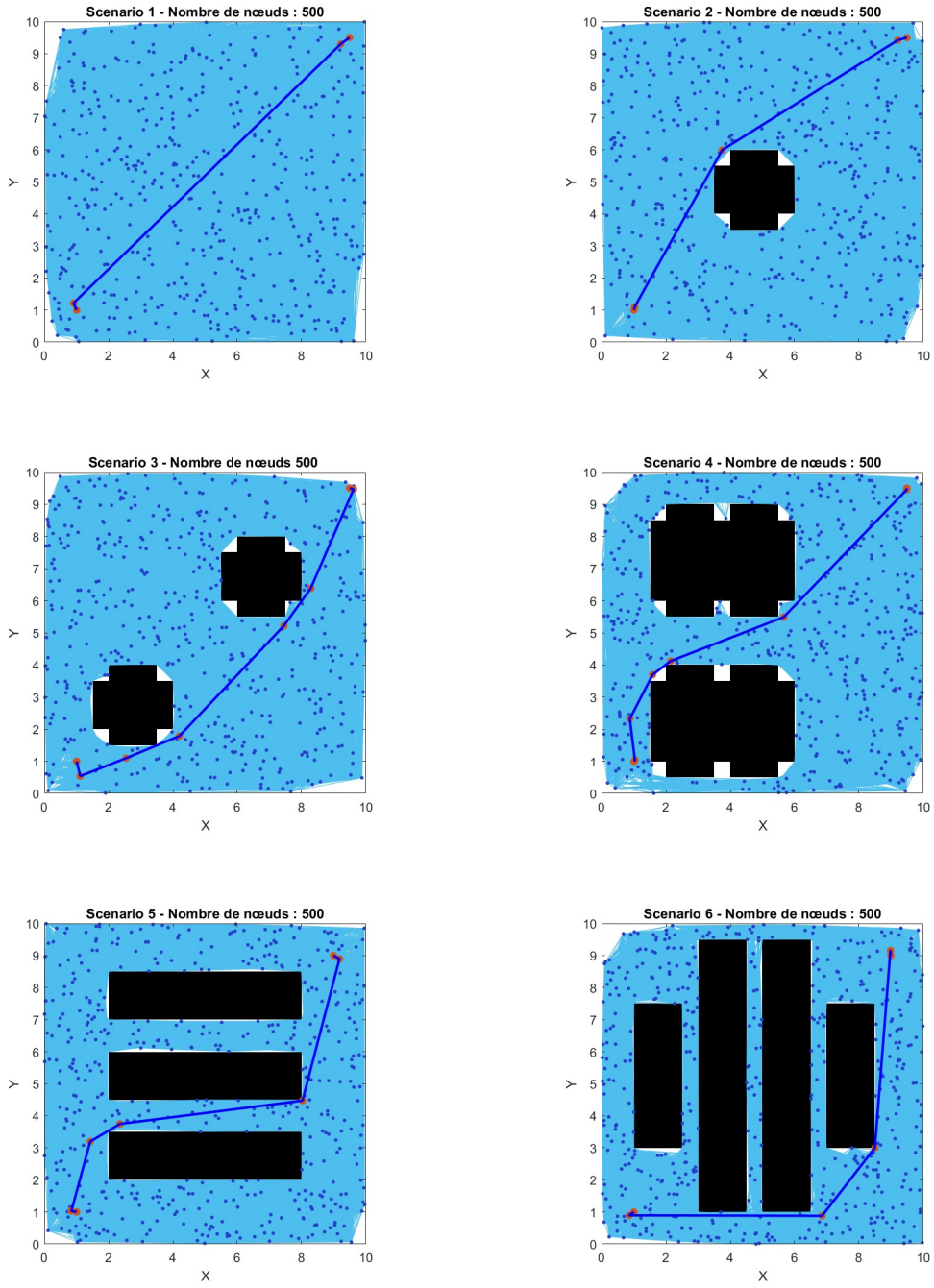


FIG. 4.1 : Trajectoires obtenues par l'algorithme PRM pour différentes Maps

Dans toutes les situations, quel que soit le degré de complexité de l'environnement, l'algorithme PRM parvient toujours à trouver une trajectoire réalisable. Cela met en évidence son fondement théorique en tant qu'algorithme conçu pour être probabilistiquement complet. En d'autres termes, à mesure que la densité de nœuds et de connexions dans l'espace libre augmente, la probabilité de trouver une trajectoire réalisable tend vers 1.

4.2.1.2 Étude comparative

Deux critères pour évaluer l'influence du nombre d'échantillons en entrée ont été utilisés : la longueur de la trajectoire et le temps d'exécution de chaque scénario. Ces critères me permettront d'analyser comment le nombre d'échantillons affecte ces deux aspects de manière comparative.

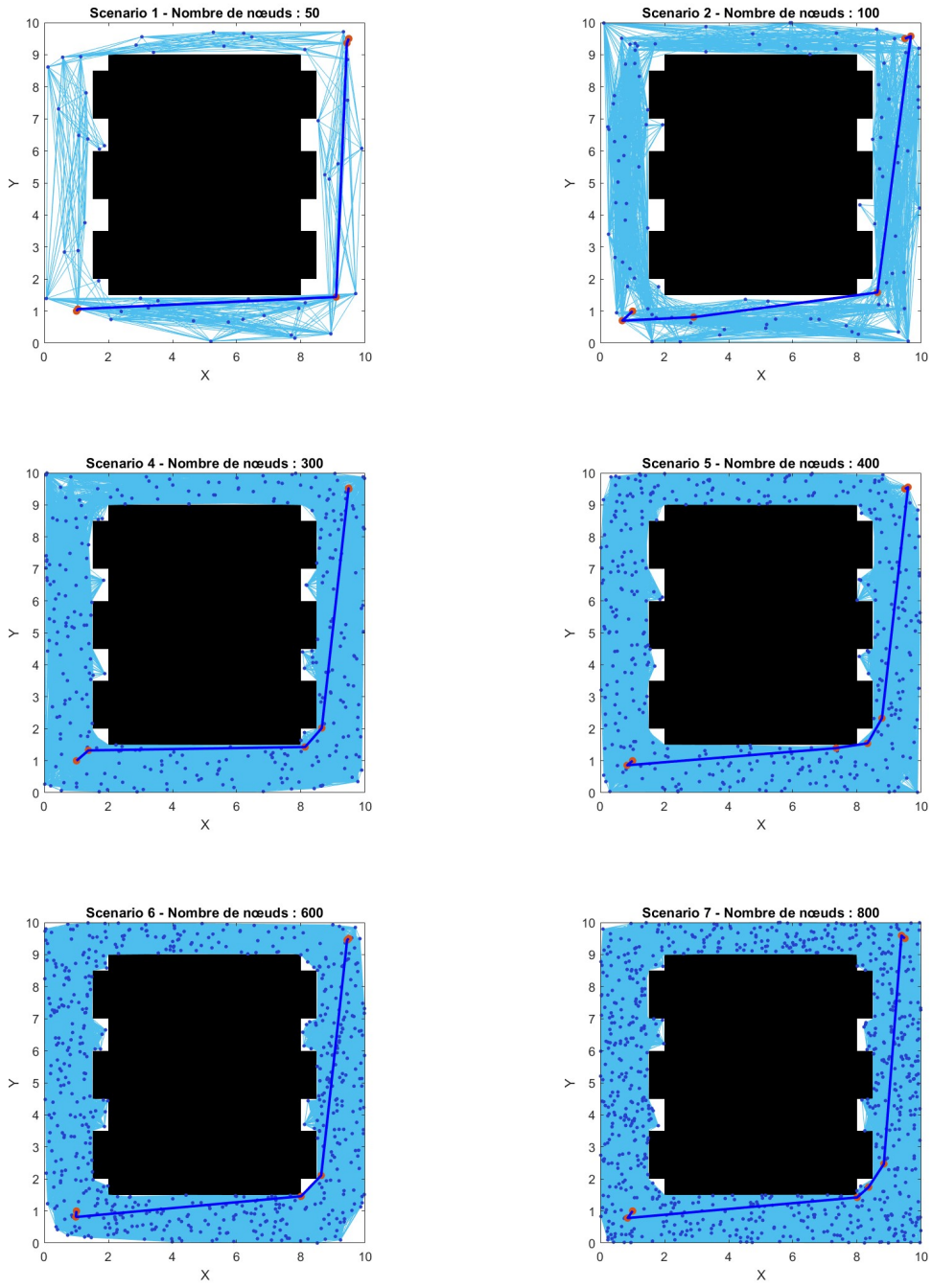


FIG. 4.2 : Trajectoires obtenues pour différentes nombre d'échantillonnage

-Par rapport au temps d'exécution :

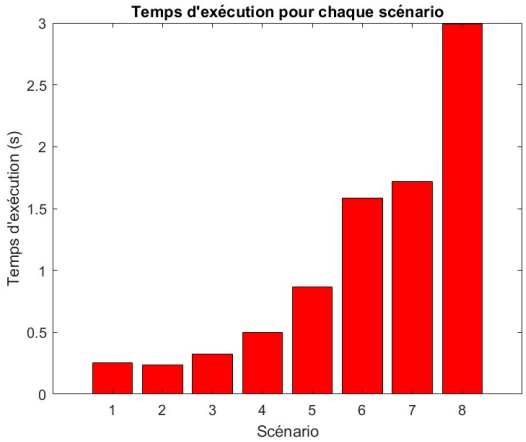


FIG. 4.3 : Comparaison entre le temps d'exécution pour chaque scénario

Le temps nécessaire à l'exécution de l'algorithme PRM semble suivre une croissance exponentielle en fonction du nombre de nœuds générés. Cette croissance rapide s'explique par le fait que chaque nouveau nœud introduit nécessite une évaluation des connexions potentielles avec les autres nœuds déjà présents dans l'espace d'état. À mesure que le nombre de nœuds augmente, le nombre de ces connexions à examiner augmente également, ce qui entraîne une augmentation significative du temps requis pour terminer l'algorithme

-Par rapport à la longueur de trajectoire :

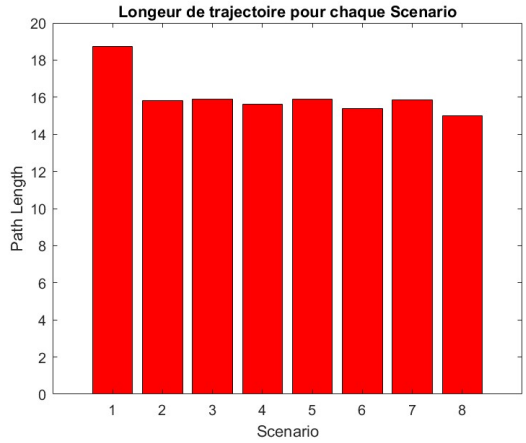


FIG. 4.4 : Comparaison entre la longueur de trajectoire pour chaque scénario

La longueur de la trajectoire de chaque scénario dans l'algorithme PRM varie en fonction du nombre de nœuds ajoutés. Initialement, avec un nombre minimal de nœuds, la trajectoire est la plus longue. Cependant, à mesure que l'on ajoute plus de nœuds, le temps d'exécution augmente, mais la longueur de la trajectoire diminue progressivement jusqu'à atteindre une trajectoire optimale avec un certain nombre de nœuds. En résumé, l'ajout progressif de nœuds dans le PRM conduit à une amélioration de la trajectoire en termes de longueur, jusqu'à un point optimal où l'équilibre entre précision et efficacité est atteint.

4.2.1.3 Simulation sur ROS

Robot Operating System ROS

ROS est un système d'exploitation spécialisé conçu pour les applications robotiques. Il offre la possibilité de tester diverses applications robotiques dans des environnements simulés, offrant ainsi une représentation proche de la réalité. Dans notre cas, nous allons utiliser le simulateur Gazebo, disponible sur ROS, pour tester l'algorithme PRM dans un environnement 2D.

-Résultats de simulation

Une fois que ROS est correctement installé sur la machine et que les packages nécessaires sont téléchargés et compilés, nous pouvons procéder à l'étape de test de l'algorithme PRM. Ce test vise à évaluer le fonctionnement de l'algorithme PRM sur un robot planaire dans un environnement à deux dimensions.

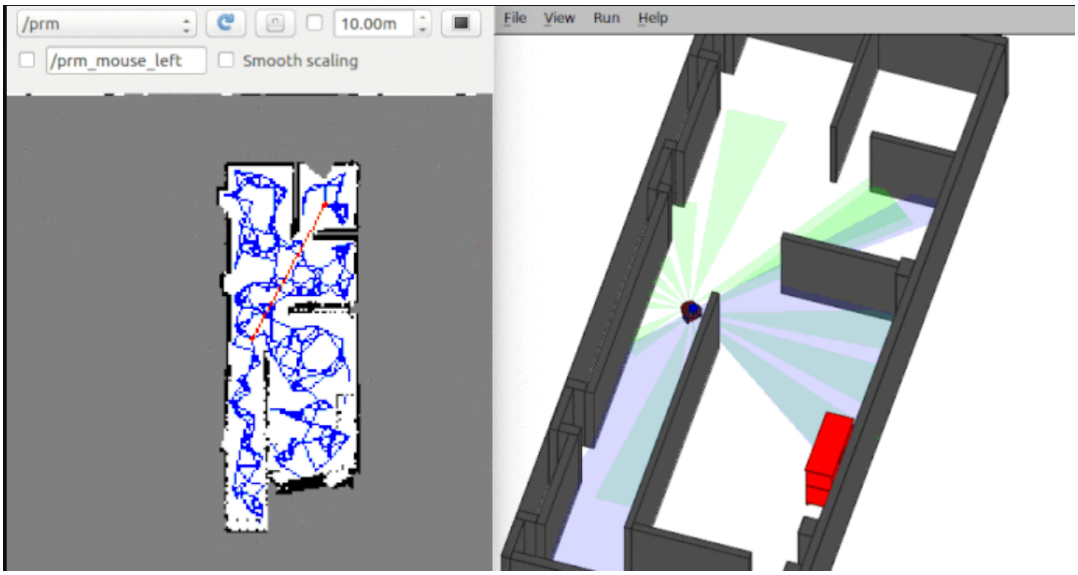


Fig. 4.5 : Simulation sur ROS

La mise en œuvre et le test de l'algorithme PRM sur un robot planaire dans un environnement 2D à l'aide de ROS et du simulateur Gazebo ont permis de vérifier son fonctionnement dans des conditions simulées. Cette expérience a démontré la capacité de l'algorithme PRM à trouver une solution optimale dans des environnements non encombrés. Les résultats obtenus sont prometteurs quant à l'efficacité de cet algorithme dans la planification de mouvement pour les robots dans des environnements réels, où la présence d'obstacles est limitée.

4.2.1.4 Conclusion

- L'algorithme PRM est généralement capable de trouver une solution pour la planification de trajectoire.
- La complétude de la solution dépend du nombre de nœuds générés en entrée.
- Le temps d'exécution de PRM augmente de manière exponentielle avec le nombre de nœuds générés.
- Cette augmentation du temps d'exécution peut rendre l'application de PRM dans des environnements de grande taille difficile à mettre en œuvre.
- L'utilisation de PRM nécessite une évaluation minutieuse des compromis entre la qualité de la solution, le temps d'exécution et les ressources disponibles.
- Des techniques d'optimisation et des stratégies de réduction de la complexité peuvent être nécessaires pour adapter PRM à des environnements de grande envergure de manière efficace.
- L'efficacité de PRM dépend de la capacité à équilibrer ces différents facteurs dans le contexte spécifique d'application.

4.2.2 Cas Multi agent

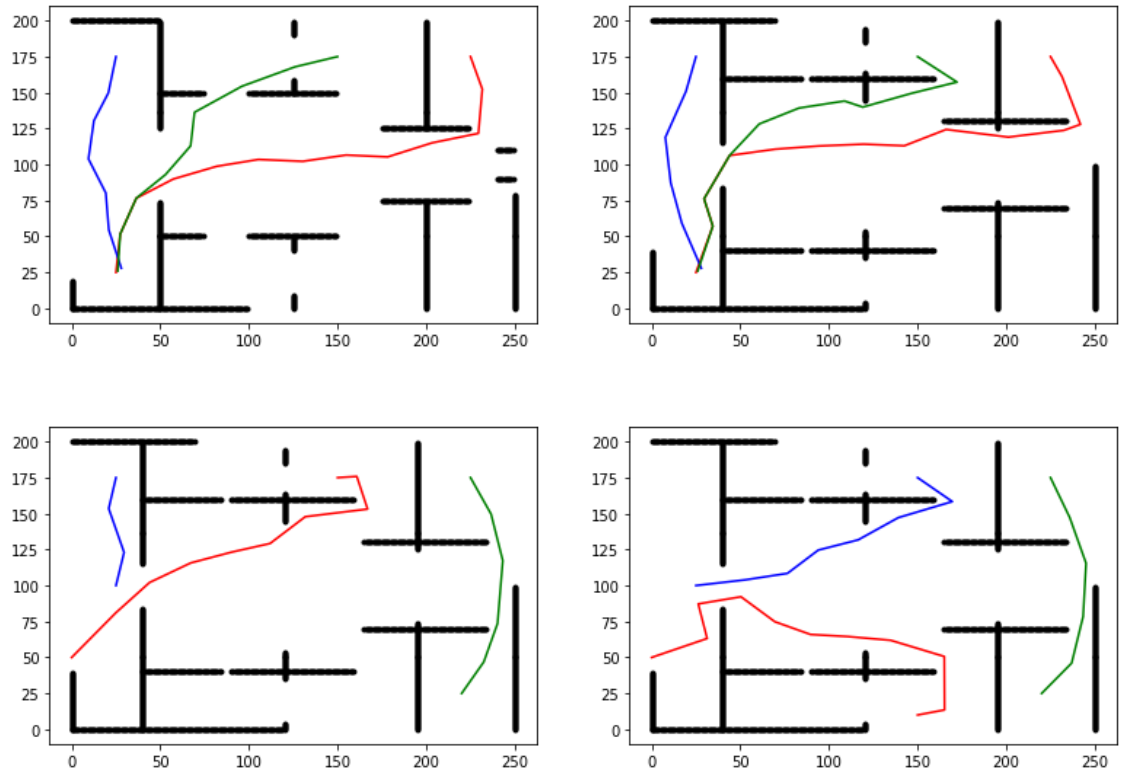


FIG. 4.6 : Trajectoires obtenues par l'algorithme MAGPRM pour différentes Maps

Dans ces expériences numériques, plusieurs agents homogènes, partant de différents endroits sur une Road Map, devaient couvrir un certain nombre de lieux cibles. Étant homogènes, le coût et la probabilité de transition d'un point de repère à un autre sont les mêmes pour tous les agents, car ils utilisent la même carte et les mêmes points de repère échantillonnés.

Les résultats montrent que MAGPRM peut être étendue pour gérer efficacement les systèmes multi-robots en planifiant des chemins individuels et en ajustant en temps réel pour éviter les collisions. Cette approche tire parti de la nature décentralisée, où chaque robot calcule indépendamment son chemin et s'ajuste en fonction d'informations locales.

t	Robot1(x)	Robot1(y)	Robot2(x)	Robot2(y)	Robot3(x)	Robot3(y)
0	0.0	50.0	25.0	100.0	220.0	25.0
1	31.30	63.18	52.86	103.94	236.93	46.08
2	26.27	87.20	76.63	108.46	243.41	78.48
3	50.39	92.17	93.97	124.61	245.06	115.47
4	69.50	74.82	116.49	131.73	235.84	147.26
5	89.61	65.97	139.16	147.18	225.0	175.0
6	110.57	64.54	169.59	158.35	225.0	175.0
7	134.80	61.93	150.0	175.0	225.0	175.0
8	165.08	50.67	150.0	175.0	225.0	175.0
9	165.22	13.55	150.0	175.0	225.0	175.0

Tab. 4.1 : Coordination des Trajectoires des Robots

Le tableau TAB. 5.1 fournit des données de position pour trois robots à différents instants. Ces informations sont cruciales pour la coordination des robots, car elles permettent de suivre leurs trajectoires et d'éviter les collisions. Par exemple, en connaissant les positions exactes des robots à tout moment, les systèmes de contrôle peuvent ajuster leurs trajectoires pour éviter tout croisement ou chevauchement, assurant ainsi un déplacement efficace et sécurisé dans leur environnement. Cette précision aide également à planifier les actions futures et à optimiser les itinéraires, garantissant une performance optimale des robots dans leurs tâches assignées, comme le déplacement d'articles dans un entrepôt automatisé.

4.2.3 Conclusion

En optant pour des algorithmes de l'approche classique basés sur des échantillons, nous avons choisi pour les simulations les algorithmes PRM (Probabilistic Roadmap) et GPRM (Generalized Probabilistic Roadmap). Les résultats des différentes simulations effectuées dans le cas d'un agent unique ont montré que l'algorithme PRM offre de meilleures solutions en termes de qualité. De plus, bien que l'algorithme PRM ait un temps d'exécution relativement raisonnable, l'ainous l'avons sélectionné pour une application dans une structure multi-agents décentralisée, détaillée dans le chapitre 3 .

Pour aborder la gestion des priorités dans une structure décentralisée et assurer une coordination efficace entre les différents robots lors de la planification de trajectoires,nous avons utilisé le MTSP (Multi-Tasking Shortest Path). Ce choix est crucial car il permet de déterminer quelle action chaque agent doit entreprendre pour éviter les collisions imminentes tout en optimisant la coordination globale du système multi-agents.

Le MTSP est particulièrement adapté car il permet de définir des priorités entre les différentes tâches que les robots doivent accomplir, tout en tenant compte des contraintes de temps et des interactions dynamiques entre les agents. En intégrant le MTSP à la planification de trajectoire, on peut assigner des priorités aux différentes missions ou aux zones à éviter en fonction de critères comme la proximité des obstacles, le temps restant avant la collision, ou l'importance stratégique des missions individuelles.

Cette approche permet ainsi de réduire les risques de collisions, d'améliorer l'efficacité des déplacements des robots et d'optimiser l'utilisation des ressources disponibles dans un environnement dynamique et décentralisé..

Pour aborder ce problème, plusieurs critères de priorité peuvent être envisagés, tels que la distance par rapport à l'obstacle, le temps prévu avant la collision, ou encore l'importance de la mission de chaque agent. Il est également envisageable d'implémenter des mécanismes de communication inter-agents pour faciliter une prise de décision plus coordonnée et réduire les risques de collision. Une évaluation approfondie et comparative de ces critères est essentielle pour déterminer la plus efficace dans un contexte donné.

De plus, il est important de prendre en compte la dynamique des agents et de s'assurer que les décisions prises en matière de priorité n'entraînent pas de recalculs excessifs de trajectoire, ce qui pourrait nuire à la performance en temps réel du système. Ainsi, la formulation de ces critères de priorité doit être non seulement juste, mais également optimisée pour minimiser les coûts computationnels et les délais de réaction des agents.

Enfin, l'implémentation de cette approche décentralisée avec l'algorithme PRM nécessite une phase de validation rigoureuse, incluant des tests dans des scénarios variés et complexes pour s'assurer que le système peut gérer efficacement les situations de collisions potentielles et maintenir une performance optimale dans des environnements réels.

4.3 Implementation de l'algorithme Q-learning

4.3.1 Agent unique :

Dans cette section , nous abordons le problème de l'apprentissage lié à la planification de trajectoire pour un agent unique évoluant dans un environnement quelconque. L'objectif principal est de déterminer une trajectoire optimale, définie comme celle maximisant la récompense totale cumulée, ce qui équivaut à minimiser le nombre d'actions inutiles (déplacements superflus). En d'autres termes, trouver la solution optimale revient à identifier la stratégie permettant à l'agent de rejoindre sa cible en empruntant le chemin le plus court possible.

Pour aborder ce problème d'apprentissage, l'algorithme de Q-learning est utilisé . Dans un premier temps, la convergence vers la stratégie optimale est évaluée . Ensuite, la rapidité de cette convergence ainsi que les caractéristiques de la stratégie obtenue grâce au Q-learning sont examinés. Ces analyses nous ont permis de mieux comprendre le comportement de l'algorithme et son efficacité..

Il est important de noter que chaque transition d'un état à un autre sera assortie d'une pénalité (ou d'une récompense négative) dans le but d'inciter l'agent à minimiser le nombre de déplacements. Cette approche vise à optimiser l'efficacité des mouvements de l'agent en réduisant les actions inutiles. En ce qui concerne l'entraînement des agents utilisant l'algorithme Q-learning, ils recourent tous deux à la stratégie ϵ-greedy.

Pour illustrer les 8 mouvements possibles d'un agent, on peut imaginer un agent se déplaçant dans une grille où chaque case représente une position possible. Les mouvements peuvent inclure les directions cardinales (nord, sud, est, ouest) ainsi que les directions diagonales (nord-est, nord-ouest, sud-est, sud-ouest).

Voici une illustration de ces mouvements sur une grille 3x3, où le point central représente la position actuelle de l'agent :

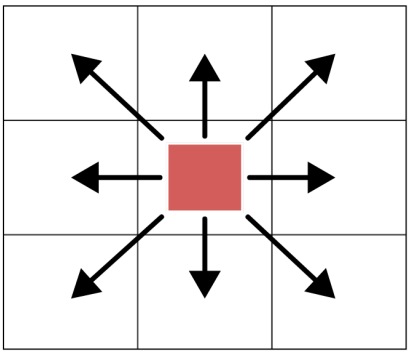


Fig. 4.7 : Action possible

Voici le résultat de l'implémentation pour les différentes complexités de l'environnement. Quatre cartes ont été créées : la MAP 1 est la plus difficile en termes de complexité, tandis que la MAP 4 est la plus facile. La comparaison se concentre donc sur ces deux cartes.

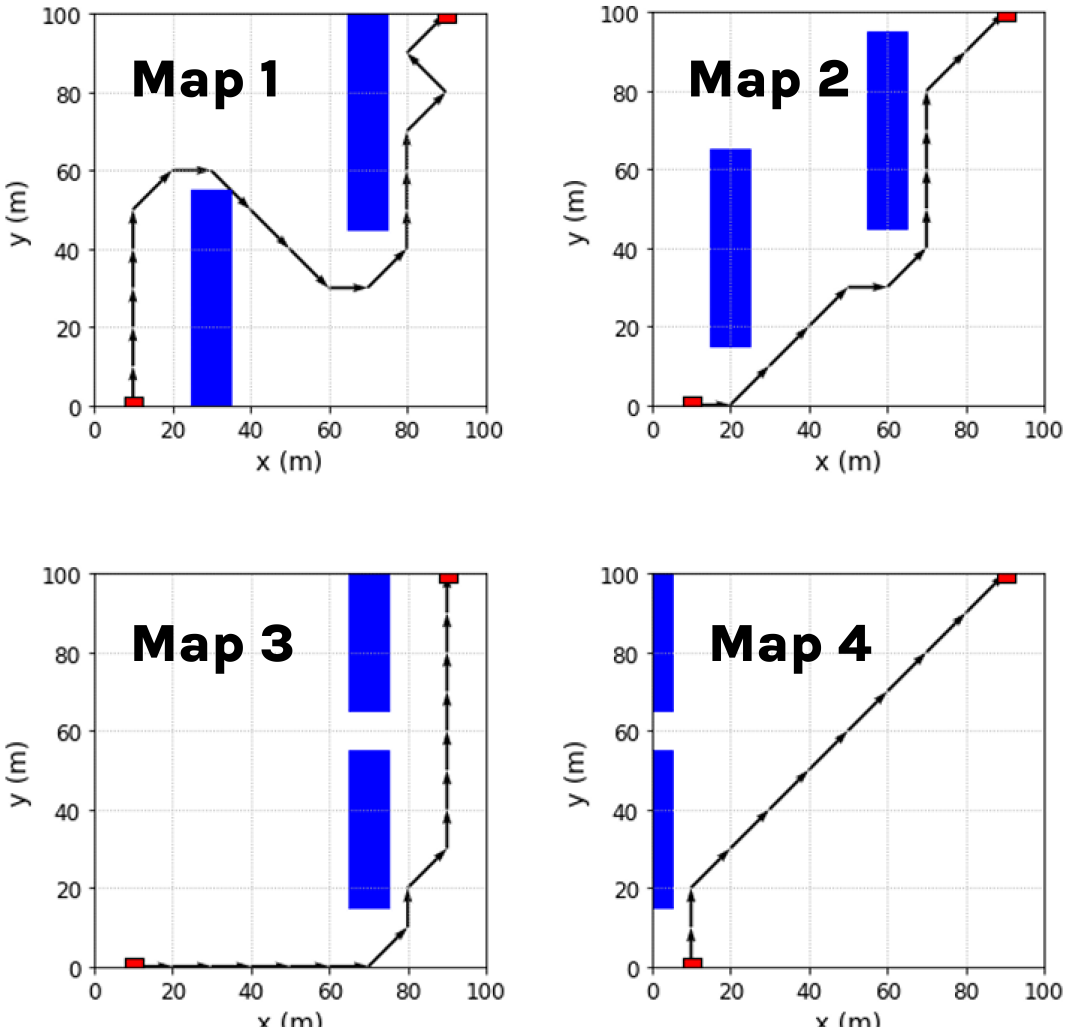


Fig. 4.8 : Trajectoires obtenues RL pour différentes Maps

4.3.1.1 Évaluation de la convergence vers l'optimum

Dans cette section, l'objectif principal est d'atteindre la cible avec la plus grande efficacité temporelle possible. L'environnement présenté (Map1) dépeint un scénario complexe

Le graphe ci-dessous illustre l'évolution de la récompense cumulée maximale de l'agent par épisode.

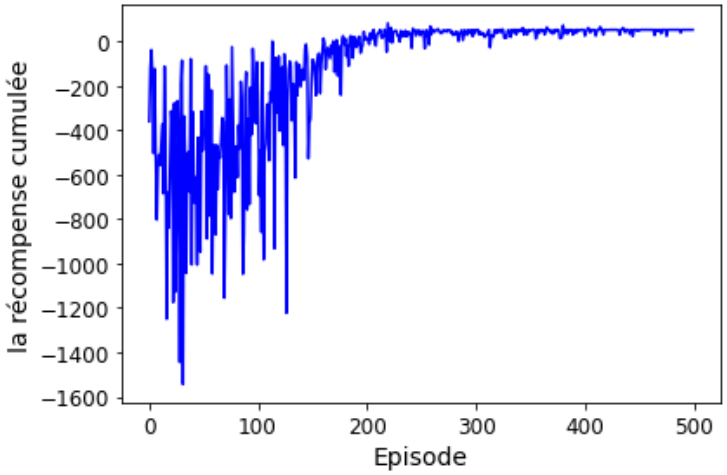


Fig. 4.9 : Évaluation de la convergence vers l'optimum (MAP 1)

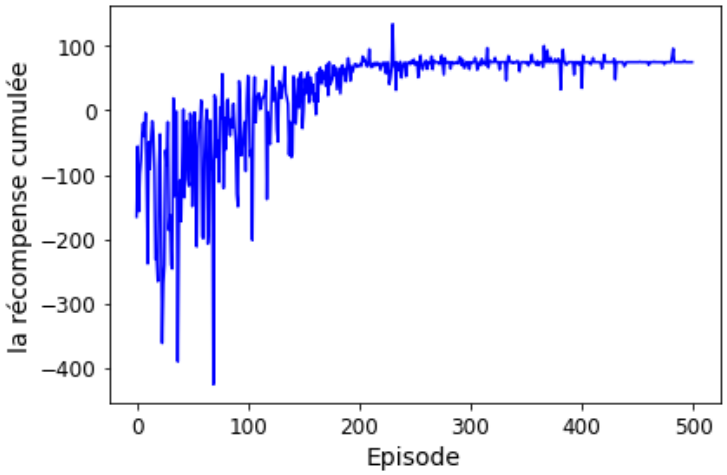


Fig. 4.10 : Évaluation de la convergence vers l'optimum (MAP 4)

La convergence se produit lorsque la stratégie adoptée par l'agent devient très proche de la stratégie optimale. À ce stade, la courbe d'apprentissage se stabilise et cesse d'augmenter, ce qui indique l'atteinte du score maximal (récompense) possible.

Au début de l'entraînement, le score total de l'épisode est généralement faible. Cette baisse s'explique par le fait que l'agent explore activement son environnement, étant nouvellement exposé à celui-ci. Pendant cette phase d'exploration, l'agent essaie différentes actions et stratégies pour comprendre comment maximiser sa récompense dans l'environnement donné. Cette exploration initiale est cruciale, car elle permet à l'agent de collecter des données sur les conséquences de ses actions.

Cependant, selon le degré de complexité de l'environnement, on observe une convergence plus ou moins rapide de la stratégie de l'agent vers l'optimalité. Pour la Map 4, qui est l'environnement le plus facile, la stratégie de l'agent converge vers l'optimalité après environ 100 épisodes. Cette convergence rapide signifie que l'agent a suffisamment exploré l'environnement et a découvert une stratégie efficace pour maximiser sa récompense.

En revanche, pour la Map 1, qui est l'environnement le plus difficile, la stratégie de l'agent converge vers l'optimalité après environ 200 épisodes. La complexité accrue de cet environnement nécessite une exploration plus longue et plus approfondie avant que l'agent ne parvienne à élaborer une stratégie optimale. Cette différence de temps de convergence entre les deux environnements reflète les défis variés auxquels l'agent est confronté lors de son apprentissage.

La convergence vers l'optimalité se traduit par une amélioration significative des performances de l'agent. Au fur et à mesure que l'agent apprend, il devient de plus en plus efficace dans ses actions, réduisant les erreurs et maximisant les récompenses. Ainsi, le score total des épisodes augmente progressivement, témoignant de la maîtrise croissante de l'agent sur son environnement.

En résumé, la phase initiale d'exploration est marquée par des scores faibles, mais elle est essentielle pour l'apprentissage de l'agent. La complexité de l'environnement influence directement le temps nécessaire pour que l'agent atteigne une stratégie optimale, avec des environnements plus simples nécessitant moins d'épisodes pour la convergence par rapport aux environnements plus complexes.

4.3.1.2 Conclusion sur la nature de stratégie :

Le Q-learning converge plus rapidement vers l'optimum. Cela s'explique par le fait que le Q-learning adopte une approche plus agressive, cherchant à apprendre directement la stratégie optimale. Il évalue les valeurs des actions en se basant sur une stratégie avide. Sous certaines conditions communes, il converge vers la fonction de valeur réelle.

En pratique, le Q-learning est particulièrement utile pour entraîner rapidement un robot en simulation. Toutefois, il convient de noter que cette approche peut comporter des risques, car elle peut prendre des mesures radicales pour atteindre l'optimum. Cela peut entraîner des erreurs coûteuses, surtout si elles surviennent de manière inattendue. Par conséquent, bien que le Q-learning soit efficace pour une formation rapide, il est essentiel de prendre en compte les potentielles implications des erreurs dans des environnements où elles peuvent avoir des conséquences significatives.

4.3.2 Deep Q learning

Pour optimiser le Q-learning, nous avons adopté une approche qui réduit la demande en ressources de calcul. Au lieu de calculer explicitement toutes les paires action-état, nous avons exploré une alternative basée sur l'apprentissage d'une approximation de la fonction Q à l'aide de réseaux de neurones. Les réseaux de neurones sont particulièrement efficaces dans ce rôle, grâce à leur capacité universelle à approximer diverses fonctions liant les paires action-état, indépendamment du nombre d'entrées et de sorties du réseau. Cette approche permet de contourner les limitations en ressources et en mémoire associées au Q-learning traditionnel.
Une étude comparative entre Q-learning et DQN est faite par la suite :

Par rapport à la vitesse de convergence optimale :

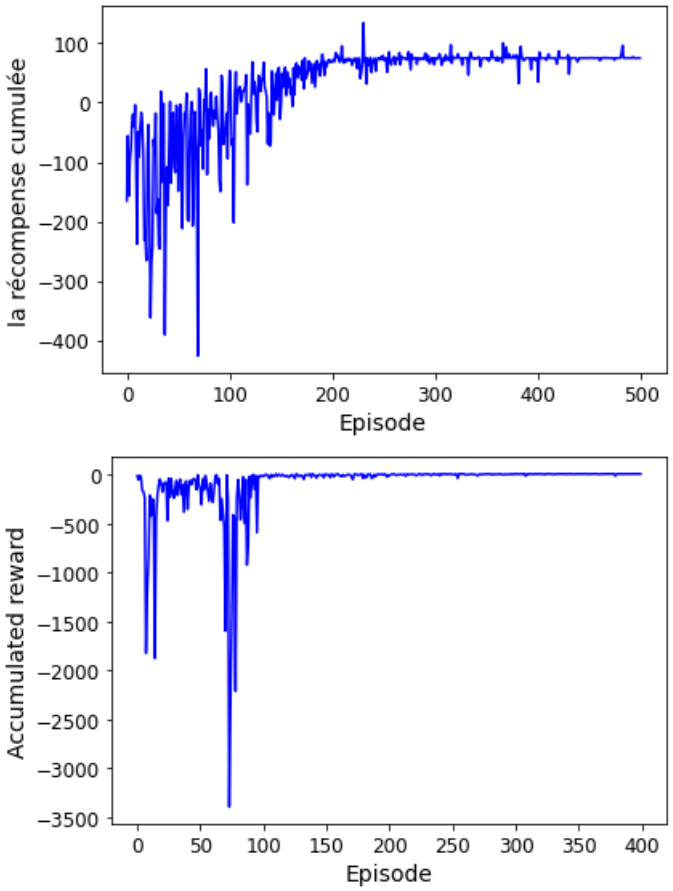


Fig. 4.11 : Comparaison de convergence entre QL et DQN pour la MAP 4

Après 200 épisodes, Q-Learning n'a toujours pas trouvé la solution optimale, tandis que DQN a convergé vers cette solution après seulement 100 épisodes. Cette évaluation a été réalisée dans le cadre d'un environnement avec un seul agent et une complexité moyenne. À mesure que l'environnement devient plus complexe, la différence de performance entre Q-Learning et DQN devient plus prononcée.

Par rapport au temps d'éxcution :

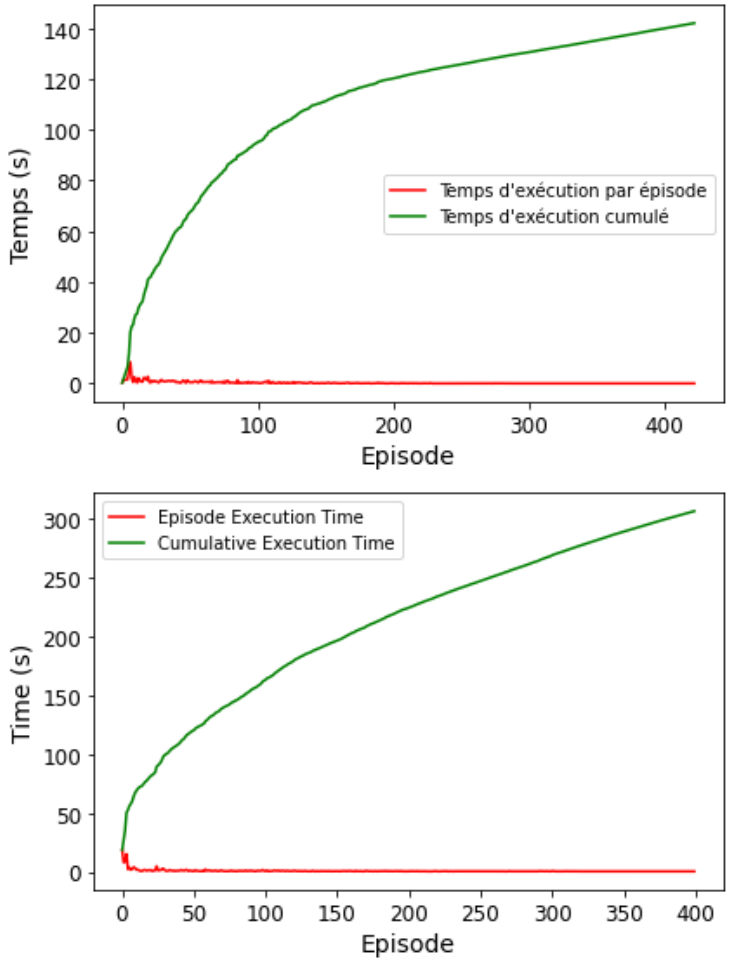


Fig. 4.12 : Comparaison de temps d'excution entre QL et DQN pour la MAP 4

Pour un entraînement de 500 épisodes, DQN a nécessité 300 secondes, soit une moyenne de 0,6 seconde par épisode. En comparaison, Q-Learning a achevé l'entraînement en 140 secondes, équivalent à une moyenne de 0,28 seconde par épisode. Cette différence de temps par épisode reflète généralement la complexité des calculs sous-jacents : Q-Learning effectue des mises à jour de sa table Q, tandis que DQN utilise un réseau de neurones pour approximer les valeurs d'action, ce qui peut être plus intensif en termes de calcul par épisode.

Conclusion

Les réseaux de neurones utilisés par DQN permettent d'explorer et d'apprendre efficacement dans des environnements complexes en adaptant leurs actions à des scénarios jamais rencontrés auparavant. Contrairement à Q-Learning, qui est limité à des combinaisons d'actions discrètes préalablement définies, les réseaux de neurones peuvent généraliser à partir de situations similaires, ce qui est crucial pour traiter des environnements complexes où les solutions ne sont pas directement prévisibles.

De plus, la mémoire requise par Q-Learning explose rapidement avec la complexité de l'environnement, tandis que DQN gère cette complexité de manière plus efficiente grâce à l'utilisation plus rationnelle de l'espace mémoire.

En conséquence, bien que DQN nécessite plus de temps pour s'entraîner que Q-Learning, il est capable de résoudre une variété beaucoup plus large d'environnements, y compris ceux avec des actions continues et une dynamique multi-agents complexe. En revanche, Q-Learning trouve sa limite dans des environnements simples et discrets, ce qui limite ses applications dans des scénarios réels et avancés où la complexité et la variabilité sont plus grandes. Ainsi, pour des applications nécessitant une adaptabilité et une généralisation élevées, DQN est généralement préféré malgré ses exigences en temps d'entraînement plus élevées.

4.3.3 Multi-agents

Pour répondre à la nécessité d'une solution centralisée, nous avons exploré une extension du Q-learning adaptée au contexte multi-agent. Cependant, cette approche a révélé plusieurs défis que nous examinerons en détail dans la section suivante. Nous discuterons également d'une proposition visant à remédier à ces problèmes.

Dans la méthode du Q-learning centralisé que nous avons développée, le MDP (Processus de Décision Markovien) est défini par un espace d'actions comprenant toutes les actions conjointes envisageables, et par un espace d'état global qui englobe les états de tous les agents impliqués. Pour évaluer la performance de cette solution, nous avons analysé la convergence du Q-learning en fonction du nombre d'agents.

Cette approche centralisée présente des implications significatives en termes de coordination et de gestion des états et des actions globaux, ce qui est crucial dans les environnements multi-agents où la collaboration et la synchronisation sont essentielles pour atteindre des objectifs communs.

À cause du calcul très intensif requis par cet algorithme dans un contexte multi-agents complexe, nous rencontrons des limitations significatives en termes de capacité de calcul pour obtenir des résultats optimaux.

4.3.4 Conclusion

Dans le contexte d'un apprentissage par renforcement pour un agent unique, Q-Learning est efficace et vérifie la convergence vers l'optimum. Cependant, lorsque l'on étend cette approche à un cadre multi-agents avec une structure centralisée, les limitations dues à la dimensionnalité de la Q-table deviennent contraignantes. Cette méthode exige des ressources considérables qui augmentent drastiquement avec le nombre d'agents impliqués.

Pour surmonter ces défis, une alternative efficace consiste à remplacer le calcul explicite des valeurs de la Q-table par des approximations de fonctions. Les réseaux de neurones, en particulier avec l'algorithme DQN, sont une solution prometteuse. DQN utilise des réseaux de neurones pour apprendre une fonction approximative, évitant ainsi les défis liés au calcul explicite.

Conclusion et perspectives

Conclusion et perspectives

Motivés par l'intérêt croissant pour les systèmes multi-robots, ce mémoire vise à contribuer à un domaine qui allie robotique et optimisation de la planification de trajectoires, pour des scénarios impliquant un agent unique et des systèmes multi-agents.

Dans un premier temps, j'ai exploré diverses approches de planification de trajectoires, en me concentrant sur les méthodes classiques et heuristiques. Pour les approches classiques, j'ai choisi d'examiner les méthodes basées sur l'échantillonnage. J'ai testé les algorithmes PRM dans différentes situations et réalisé une analyse comparative. Les résultats ont montré que l'algorithme PRM était particulièrement performant pour un agent unique. En conséquence, nous l'avons adopté pour les systèmes multi-agents, en mettant en place une structure décentralisée et en cherchant à améliorer ses performances.

Pour les approches heuristiques, j'ai opté pour le Reinforcement Learning. J'ai évalué les algorithmes Q-Learning et DQN dans diverses conditions pour un agent unique. Pour notre application, nous avons utilisé une structure centralisée pour l'entraînement. Les résultats ont montré que le DQN était la meilleure option, voire la seule réellement viable, grâce aux avantages offerts par les réseaux de neurones. Malgré les défis posés par le temps d'exécution, nous avons cherché à optimiser cette méthode en ajustant l'architecture des réseaux de neurones et les hyperparamètres de l'apprentissage.

En conclusion, ce mémoire présente une analyse approfondie et des solutions optimisées pour la planification de trajectoires dans des contextes de robotique, démontrant l'efficacité des algorithmes PRM et DQN pour des agents uniques et des systèmes multi-agents respectivement.

Dans la perspective de poursuivre cette recherche, plusieurs pistes d'amélioration et d'extension peuvent être envisagées :

- Optimisation des algorithmes existants : Continuer à affiner les paramètres des algorithmes PRM et DQN pour maximiser leur efficacité dans des environnements plus complexes et dynamiques.
- Exploration de méthodes hybrides : Investiguer la possibilité d'intégrer des approches hybrides combinant des techniques classiques et des méthodes d'apprentissage profond pour exploiter les avantages de chaque méthode tout en minimisant leurs inconvénients respectifs.
- Validation expérimentale approfondie : Étendre les expérimentations pour inclure des tests sur des plateformes matérielles réelles, afin de valider la performance des méthodes proposées dans des conditions réelles et potentiellement imprévisibles.

Ces perspectives visent à consolider les bases établies dans ce mémoire et à ouvrir de nouvelles voies pour l'application efficace des techniques de planification de trajectoires dans le domaine en constante évolution de la robotique multi-agents.

Bibliographie

1. LIAO, Yi-Lin ; SU, Kuo-Lan. Multi-robot-based intelligent security system. *Artificial Life and Robotics*. 2011, t. 16, n° 2, p. 137-141. ISSN 1614-7456. Disp. à l'adr. DOI : 10.1007/s10015-011-0888-x.
2. PARSONS, D. ; CANNY, J. A motion planner for multiple mobile robots. In : *Proceedings., IEEE International Conference on Robotics and Automation*. 1990, 8-13 vol.1. Disp. à l'adr. DOI : 10.1109/ROBOT.1990.125937.
3. AUTHORS. Path Planning and Trajectory Planning Algorithms : A General. *Springer*. 2023.
4. MAHDAVI, Atefeh ; CARVALHO, Marco. Distributed Coordination of Autonomous Guided Vehicles in Multi-agent Systems with Shared Resources. In : *2019 SoutheastCon*. 2019, p. 1-7. Disp. à l'adr. DOI : 10.1109/SoutheastCon42311.2019.9020456.
5. MASEHIAN, Ellips ; SEDIGHIZADEH, Davoud. Classic and heuristic approaches in robot motion planning-a chronological review. *World Academy of Science, Engineering and Technology*. 2007, t. 23, n° 5, p. 101-106.
6. LLOPIS-ALBERT, Carlos ; RUBIO, Francisco ; VALERO, Francisco. Optimization approaches for robot trajectory planning. *Multidisciplinary Journal for Education, Social and Technological Sciences*. 2018, t. 5, p. 1-16.
7. KAVRAKI, L.E. ; SVESTKA, P. ; LATOMBE, J.-C. ; OVERMARS, M.H. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*. 1996, t. 12, n° 4, p. 566-580. Disp. à l'adr. DOI : 10.1109/70.508439.
8. LAVALLE, Steven ; KUFFNER, James. Rapidly-Exploring Random Trees : Progress and Prospects. *Algorithmic and computational robotics : New directions*. 2000.
9. MASEHIAN, Ellips ; SEDIGHIZADEH, D. Multi-objective robot motion planning using a particle swarm optimization model. *Journal of Zhejiang University - Science C*. 2010, t. 11, p. 607-619. Disp. à l'adr. DOI : 10.1631/jzus.C0910525.
10. Planning Long Dynamically-Feasible Maneuvers for Autonomous Vehicles. In : *Robotics : Science and Systems IV*. 2009, p. 214-221.
11. WANG, Yunfeng ; CHIRIKJIAN, G.S. A new potential field method for robot path planning. In : *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*. 2000, t. 2, 977-982 vol.2. Disp. à l'adr. DOI : 10.1109/ROBOT.2000.844727.

12. JAN, Gene Eu ; LUO, Chaomin ; YANG, Chan-Yun ; HSIEH, Hui-Ching. A Survey of Cell Decomposition-based Path Planning. In : *2023 4th International Conference on Artificial Intelligence, Robotics and Control (AIRC)*. 2023, p. 83-87. Disp. à l'adr. DOI : 10.1109/AIRC57904.2023.10303086.

13. LLOPIS-ALBERT, Carlos ; RUBIO, Francisco ; VALERO, Francisco. Optimization approaches for robot trajectory planning. *Multidisciplinary Journal for Education Social and Technological Sciences*. 2018, t. 5, p. 1. Disp. à l'adr. DOI : 10.4995/muse.2018.9867.

14. LLOPIS-ALBERT, Carlos ; RUBIO, Francisco ; VALERO, Francisco. Optimization approaches for robot trajectory planning. *Multidisciplinary Journal for Education, Social and Technological Sciences*. 2018, t. 5, p. 1-16.

15. HU, Yanrong ; YANG, S.X. 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, t. 5, 4350-4355 Vol.5. Disp. à l'adr. DOI : 10.1109/ROBOT.2004.1302402.

16. MOHAMAD, Mohd Murtadha ; TAYLOR, Nicholas K. ; DUNNIGAN, Matthew W. Articulated Robot Motion Planning Using Ant Colony Optimisation. In : *2006 3rd International IEEE Conference Intelligent Systems*. 2006, p. 690-695. Disp. à l'adr. DOI : 10.1109/IS.2006.348503.

17. KENNEDY, J. ; EBERHART, R. Particle swarm optimization. In : *Proceedings of ICNN'95 - International Conference on Neural Networks*. 1995, t. 4, 1942-1948 vol.4. Disp. à l'adr. DOI : 10.1109/ICNN.1995.488968.

19. DAM, Tuan ; CHALVATZAKI, Georgia ; PETERS, Jan ; PAJARINEN, Joni. Monte-carlo robot path planning. *IEEE Robotics and Automation Letters*. 2022, t. 7, n° 4, p. 11213-11220.

20. CAMPANA, Mylène ; LAMIRAUX, Florent ; LAUMOND, Jean-Paul. A gradient-based path optimization method for motion planning. *Advanced Robotics*. 2016, t. 30, n° 17-18, p. 1126-1144.

21. ZHANG, Junzi ; KIM, Jongho ; O'DONOGHUE, Brendan ; BOYD, Stephen. Sample Efficient Reinforcement Learning with REINFORCE. *Proceedings of the AAAI Conference on Artificial Intelligence*. 2021, t. 35, n° 12, p. 10887-10895. Disp. à l'adr. DOI : 10.1609/aaai.v35i12.17300.

22. PAPINI, Matteo ; BINAGHI, Damiano ; CANONACO, Giuseppe ; PIROTTA, Matteo ; RESTELLI, Marcello. Stochastic Variance-Reduced Policy Gradient. In : DY, Jennifer ; KRAUSE, Andreas (éd.). *Proceedings of the 35th International Conference on Machine Learning*. PMLR, 2018, t. 80, p. 4026-4035. Proceedings of Machine Learning Research.

23. MEHTA, Deepanshu. State-of-the-Art Reinforcement Learning Algorithms. *International Journal of Engineering Research and*. 2020.

24. XIN, Jing ; ZHAO, Huan ; LIU, Ding ; LI, Minqi. Application of deep reinforcement learning in mobile robot path planning. In : *2017 Chinese Automation Congress (CAC)*. 2017, p. 7112-7116. Disp. à l'adr. DOI : 10.1109/CAC.2017.8244061.

25. SUTTON, Richard S ; MCALLESTER, David ; SINGH, Satinder ; MANSOUR, Yishay. Policy Gradient Methods for Reinforcement Learning with Function Approximation. In : SOLLA, S. ; LEEN, T. ; MÜLLER, K. (éd.). *Advances in Neural Information Processing Systems*. MIT Press, 1999, t. 12.

26. CASAS, Noe. Deep deterministic policy gradient for urban traffic light control. *arXiv preprint arXiv :1703.09035*. 2017.

27. LOZANO-PEREZ. Spatial Planning : A Configuration Space Approach. *IEEE Transactions on Computers*. 1983, t. C-32, nº 2, p. 108-120. Disp. à l'adr. DOI : 10.1109/TC.1983.1676196.

28. KAVRAKI, Lydia ; SVESTKA, Petr ; LATOMBE, J.C. ; OVERMARS, M.H. Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces. *Robotics and Automation, IEEE Transactions on*. 1996, t. 12, p. 566-580. Disp. à l'adr. DOI : 10.1109/70.508439.

29. ALARABI, Saleh ; LUO, Chaomin ; SANTORA, Michael. A PRM Approach to Path Planning with Obstacle Avoidance of an Autonomous Robot. In : *2022 8th International Conference on Automation, Robotics and Applications (ICARA)*. 2022, p. 76-80. Disp. à l'adr. DOI : 10.1109/ICARA55094.2022.9738559.

30. QUINLAN, S. Efficient distance computation between non-convex objects. In : *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*. 1994, 3324-3329 vol.4. Disp. à l'adr. DOI : 10.1109/ROBOT.1994.351059.

31. GE, Shuzhi Sam ; CUI, Youjing. Principles of Robot Motion : Theory, Algorithms, and Implementations. *Autonomous Robots*. 2002, t. 13, p. 207-222.

32. LAUMOND, J.-P. ; TAIX, M. ; JACOBS, P. A motion planner for car-like robots based on a mixed global/local approach. In : *EEE International Workshop on Intelligent Robots and Systems, Towards a New Frontier of Applications*. 1990, 765-773 vol.2. Disp. à l'adr. DOI : 10.1109/IROS.1990.262494.

33. BERCHTOLD, S. ; GLAVINA, B. A scalable optimizer for automatically generated manipulator motions. In : *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'94)*. 1994, t. 3, 1796-1802 vol.3. Disp. à l'adr. DOI : 10.1109/IROS.1994.407615.

34. TAYLOR, Russell H. Planning and Execution of Straight Line Manipulator Trajectories. *IBM Journal of Research and Development*. 1979, t. 23, nº 4, p. 424-436. Disp. à l'adr. DOI : 10.1147/rd.234.0424.

35. KUMAR, Sandip ; CHAKRAVORTY, Suman. Multi-agent Generalized Probabilistic RoadMaps : MAGPRM. In : *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2012, p. 3747-3753. Disp. à l'adr. DOI : 10.1109/IROS.2012.6385678.

36. BUSONIU, Lucian ; BABUSKA, Robert ; DE SCHUTTER, Bart. Multi-agent Reinforcement Learning : An Overview. In : 2010, t. 310, p. 183-221. ISBN 978-3-642-14434-9. Disp. à l'adr. DOI : 10.1007/978-3-642-14435-6_7.

38. DEBBAGH, Nadir ; BAGHDADI, Riyadh. *Graduation Thesis - Deep Learning Operators Optimization in Tiramisu (Sparse Neural Networks and Recurrent Neural Networks)*. 2020. Disp. à l'adr. DOI : 10.13140/RG.2.2.25062.24645. Thèse de doct.

39. GAO, Junli ; YE, Weijie ; GUO, Jing ; LI, Zhongjuan. Deep Reinforcement Learning for Indoor Mobile Robot Path Planning. *Sensors*. 2020, t. 20, n° 19. ISSN 1424-8220. Disp. à l'adr. DOI : 10.3390/s20195493.

40. BOUZIT, Zakaria ; MITICHE, Nour. Titre de l'article. *Nom de la revue ou de la conférence*. 2023.

41. D'AVELLA, Salvatore ; CAMACHO, Gerardo ; TRIPICCHIO, Paolo. On Multi-Agent Cognitive Cooperation : Can virtual agents behave like humans ? *Neurocomputing*. 2022, t. 480. Disp. à l'adr. DOI : 10.1016/j.neucom.2022.01.025.

Webographie

18. ROBERT, Jérémy. *Reinforcement Learning : Définition et application — datascientest.com* [https://datascientest.com/reinforcement-learning]. [s. d.]. [Accessed 07-06-2024].

37. ZHANG, Kaiqing ; YANG, Zhuoran ; BAŞAR, Tamer. *Multi-Agent Reinforcement Learning : A Selective Overview of Theories and Algorithms*. 2021. Disp. à l'adr. arXiv : 1911.10635 [cs.LG].