RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE
Ministère de l'enseignement supérieur et de la recherche scientifique
École nationale polytechnique

Département d'Automatique

Laboratoire de commande des processus

Thèse de doctorat en

Automatique

Contribution à la commande des systèmes robotisés

Fadhila LACHEKHAB

Sous la direction de M. Mahamed TADJINE Professeur

Présentée et soutenue publiquement le : 19/03/2020

Composition du jury :

Président	M. Hachemi CHEKIREB,	Professeur	École Nationale Polytechnique
Rapporteur	M.Mohamed TADJINE,	Professeur	École Nationale Polytechnique
Examinateur	Mme Dalila ACHELI,	Professeur	Université de Boumerdes
Examinateur	M. Khelifa BENMANSOUR,	Professeur	École Supérieure de la Défense Aérienne du Territoire
Examinateur	M. Abdelkrim NEMRA,	MCA	École Militaire Polytechnique
Examinateur	M. Djamel BOUDANA,	MCA	École Nationale Polytechnique

ENP 2020

RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE
Ministère de l'enseignement supérieur et de la recherche scientifique
École nationale polytechnique

Département d'Automatique

Laboratoire de commande des processus

Thèse de doctorat en

Automatique

Contribution à la commande des systèmes robotisés

Fadhila LACHEKHAB

Sous la direction de M. Mahamed TADJINE Professeur

Présentée et soutenue publiquement le : 19/03/2020

Composition du jury :

Président	M. Hachemi CHEKIREB,	Professeur	École Nationale Polytechnique
Rapporteur	M.Mohamed TADJINE,	Professeur	École Nationale Polytechnique
Examinateur	Mme Dalila ACHELI,	Professeur	Université de Boumerdes
Examinateur	M. Khelifa BENMANSOUR,	Professeur	École Supérieure de la Défense Aérienne du Territoire
Examinateur	M. Abdelkrim NEMRA,	MCA	École Militaire Polytechnique
Examinateur	M. Djamel BOUDANA,	MCA	École Nationale Polytechnique

ENP 2020

بسم الله الرحمن الرحيم

Remerciements

Je tiens à remercier tout d'abord, le bon Dieu de m'avoir donné la force et la patience pour accomplir ce modeste travail.

Je remercie Mon directeur de thèse Monsieur M.Tadjine pour le soutien et l'aide qu'il n'a jamais manqué, et auprès de lequel j'ai trouvé les conseils et les instructions judicieuses pendant les moments difficiles, je tiens à le remercier pour sa patience ses encouragements.

Mes remerciements vont aussi à tous les membres de ma famille pour leurs soutien indéfectible.

Et merci à tous les membres du jury ainsi qu'à tous ceux qui, de près ou de loin, ont contribué afin que ce travail voie le jour.

J'adresse des remerciements très particuliers à Monsieur M.Aitouche pour son aide ses remarques…

ملخص: بخلاف التعلم الخاضع للإشراف، لا يتطلب تعزيز التعلم (RE) تعريف قاعدة تعليمية تمثيلية. تعتمد تقنيات AR هذه بشكل خاص في الروبوتات المحمولة. وبالتالي، فإنها تجعل من الممكن، من خلال عملية التجربة والخطأ، اتخاذ الإجراء الأمثل لكل موقف من أجل تعظيم المكافآت. حتى الآن، تم اقتراح خوارزميات AR المختلفة في الأدب. في العمل الحالي، يتم توجيه اهتمام خاص إلى الخوارزميات المكونتين: الممثل الناقد للتعلم وتعلم .Q تم تطبيق هذين النهجين، استنادًا إلى مبدأ البرمجة الديناميكية، من أجل تكييف استنتاجات وحدة التحكم الغامضة لملاحة الروبوت المتحرك. في وقت لاحق، تم إجراء دراسة مقارنة لمعايير خوارزميات (Fuzzy Actor Critic Learning) FACL وخوارزمية FQL (Q-Learning). يعتمد الجزء الثاني من هذا العمل على تنفيذ كاميرا RGB-Kinect لإعادة بناء خريطة البيئة ثلاثية الأبعاد 3D. باستخدام مرشحات SURF ، KNN ، RANSAC وICP، مكّن البرنامج المطوَّر من تقليل الخطأ المتوسط لدمج العديد من المشاهدات المتتالية وتطوير خريطة ثلاثية الأبعاد لبيئة تطور الروبوت المتحرك.

الكلمات الدالة: روبوت محمول، تعزيز التعلم، FQL، FACL، RGB-Kinect، خريطة ثلاثية الأبعاد.

ABSTRACT: Unlike supervised learning, reinforcement learning (AR) does not require the definition the training set. The AR techniques are particularly adopted in mobile robotics. They enable finding by a trial and error process, the optimal action to be performed in each situation in order to maximize the rewards. To date, various AR algorithms have been proposed in the literature. In the present work, a particular interest is brought to the two algorithms denominated: Actor Critic Learning and Q-Learning. These two approaches, based on the principle of dynamic programming were applied for the adaptation of the conclusions of the fuzzy controller of the mobile robot navigation. Thereafter, a comparative study of the parameters of the two algorithms - FACL (Fuzzy Actor Critic Learning) and FQL (Fuzzy Q-Learning) - was performed. The main aim of the second part of this work is the implementation of the RGB-Kinect camera for the 3D environment map reconstruction. Using the SURF, KNN, RANSAC and ICP filters, the developed program minimizes the average error of the fusion of several successive views and develop the 3D map of the mobile robot environment.

Key words: Mobile Robot, Reinforcement Learning, FQL, FACL, RGB-Kinect, 3D Map.

Résumé : Contrairement à l'apprentissage supervisé, l'apprentissage par renforcement (AR) ne nécessite pas la définition d'une base d'apprentissage représentative. Ces techniques d'AR sont particulièrement adoptées à la robotique mobile. Elles permettent ainsi de trouver par un processus d'essais-erreurs, l'action optimale à effectuer pour chaque situation afin d'en maximiser les récompenses. À ce jour, divers algorithmes d'AR ont été proposés dans la littérature. Dans le présent travail, un intérêt particulier est porté aux deux algorithmes dénommés : Actor Critic Learning et Q-Learning. Ces deux approches, basées sur le principe de la programmation dynamique ont été appliquées pour l'adaptation des conclusions du contrôleur flou de la navigation du robot mobile. Par la suite une étude comparative des paramètres des deux algorithmes FACL (Fuzzy Actor Critic Learning) et FQL (Fuzzy Q-Learning) a été effectuée. La seconde partie du présent travail, repose sur la mise œuvre de la caméra RGB- Kinect pour la reconstruction de la carte de l'environnement 3D. Par le biais des filtres SURF, KNN, RANSAC et ICP, le programme développé a permis de minimiser l'erreur moyenne de la fusion de plusieurs vues successives et l'élaboration de la carte 3D de l'environnement d'évolution du robot mobile.

Mots clés : Robot mobile, Apprentissage par renforcement, FQL, FACL, RGB- Kinect, Carte 3D.

Table des matières

Chapitre I : *Robotique et navigation autonome*

Liste des tableaux

Chapitre I : *Robotique et navigation autonome*

Chapitre III : *Programmations et simulations*

Chapitre IV : *Construction de carte 3D de l'environnement à l'aide de KINECT*

Liste des figures

Chapitre I : *Robotique et navigation autonome*

Chapitre II : *Apprentissage par renforcement*

Chapitre III : *Programmations et simulations*

Chapitre IV : *Construction de carte 3D de l'environnement à l'aide de KINECT*

Introduction générale

La robotique et plus précisément la robotique mobile autonome est un axe de recherche qui vise à donner à une machine la capacité de se mouvoir dans un environnement sans assistance ou intervention humaine [4] [9] [13]. Le mot machine désigne ici à la fois la machine traditionnelle, qui agit de façon tangible dans un environnement donné et l'ordinateur, qui apporte un support fondamental pour la décision.

Une première génération des robots a consisté en des machines capables d'évoluer dans des environnements parfaitement connus [7][8] pour la réalisation des missions planifiées à partir d'une modélisation complète de l'environnement (le laboratoire par exemple) en suivant une trajectoire par un mécanisme de filo-guidage [8][9] (robots de manutention). Le point commun de ces robots est qu'ils évoluent dans un environnement qui leur est totalement dédié.

Cependant, lorsque l'environnement devient plus complexe (i.e. partiellement connu, dynamique, . . .), il apparaît indispensable que le robot soit doté de capacités décisionnelles aptes à le faire réagir aux aléas qui peuvent contrarier ses mouvements (pannes partielles, obstacles imprévus). On peut citer à cet effet, le cas où le robot mobile évolue dans des environnements hostiles à l'homme [27] [28] (milieu radioactif par exemple) ou trop éloignés [10] [34] (exploration spatiale). Pour cela, le robot doit suivre le schéma correspondant au paradigme *Percevoir-Décider-Agir* au sein d'une architecture de contrôle.

Les recherches récentes en robotique mobile s'orientent vers des architectures dotées de moyens de perception, d'action et de décision permettant une certaine autonomie [12] [28]. Cette autonomie est liée aux capacités de reconstruction de la carte de l'environnement dans des milieux contraints et non structurés [44], aux moyens de prise de décision à partir d'informations issues des capteurs pouvant être imprécises et /ou incertaines et l'apprentissage à partir de l'interaction robot/environnement.

Position du problème

Plusieurs méthodes de contrôle de systèmes ont été développées en robotique mobile. On peut les classer en deux catégories : celles basées sur la navigation globale [7] et celles utilisant une navigation locale [11] [15]. Les méthodes globales basées sur une connaissance précise du robot et de l'environnement, ont l'avantage d'une part de prouver l'existence d'une solution optimale permettant au robot d'atteindre son but et d'autre part de construire une cartographie d'espace libre. Mais elles ont comme inconvénient d'exiger de disposer d'un modèle exact de l'environnement et sans tenir compte d'une quelconque variation du modèle de l'environnement. Les méthodes locales peuvent être assimilées à des méthodes réactives ou d'actions réflexes [16] [19] [20] qui sont basées seulement sur l'interaction robot-environnement. Comme le robot n'a qu'une vue très réduite de l'environnement, ces méthodes ne peuvent garantir le succès de la mission en toute circonstance. L'une ou l'autre de ces stratégies (globale ou locale) ne peut pas être efficace à elle seule pour le contrôle du mouvement d'un robot mobile dans un environnement complexe réel qui, généralement, ne

peut être que partiellement modélisé. Donc un contrôle efficace du mouvement d'un robot mobile dans un environnement complexe doit intégrer des capacités de prise de décision tout en exploitant les informations de la perception environnementale.

Par ailleurs, il est évident que pour mener à bien ce contrôle, il est nécessaire que le robot connaisse sa position par rapport à un repère pris comme référence dans son environnement.

Dans ce contexte, le souci majeur est de mettre au point des techniques de navigation efficaces, de sorte que la sécurité soit prioritaire par rapport à l'optimalité. La stratégie de navigation doit intégrer le fait que, généralement, on dispose d'un modèle d'environnement dans lequel sont représentés les principaux éléments fixes : murs, portes, meubles, etc.… Cependant, en présence d'un environnement plus complexe (*i.e.* partiellement connu, dynamique, …), il apparaît indispensable que le robot soit doté de capacités décisionnelles aptes à contraindre à réagir aux aléas qui peuvent contrarier ses mouvements. Dans ces conditions, deux cas extrêmes peuvent se produire :

- Si l'environnement perçu par le robot correspond à la carte mémorisée, il pourra suivre à une vitesse maximale une trajectoire planifiée en utilisant la méthode globale.
- Si au contraire l'environnement n'est pas reconnu, l'inattendu peut mener le robot à des échecs de navigation, ce qui nécessite de s'adapter à des nouvelles situations à travers des méthodes d'apprentissage qui approchent au mieux le comportement humain.

Entre ces deux situations limites, une évolution progressive se fera en fusionnant les commandes issues des deux modules en fonction du degré de reconnaissance de la scène mémorisée sous forme d'une carte 2D ou 3D.

L'absence de structure géométrique, et les caractéristiques des environnements naturels, rendent les algorithmes de modélisation de l'environnement difficiles. Ainsi le problème de la segmentation, c'est à dire la subdivision d'une scène en objets ou en éléments simples, reste un problème très ouvert pour ce contexte. L'utilisation de la stéréovision (deux caméras) a été pendant plusieurs années le meilleur moyen de navigation visuelle [59] [60] des robots mobiles, car elle est capable de reconstruire des scènes 3D à partir des images 2D. Cependant ces méthodes nécessitent des algorithmes robustes d'extraction, de mise en correspondance et de triangulation des points d'intérêt. Ajoutons à cela le fait qu'ils requièrent des temps de calculs plus longs. La minimisation du temps de calcul demeure l'objectif de tout système robotisé. Elle permet au système d'accomplir ses différentes tâches en temps réel. Ainsi la recherche est orientée au développement d'un nouveau capteur de vision qui permet à la fois la minimisation du temps de calcul, et la fourniture de la valeur de profondeur de chaque pixel afin de faciliter la construction 3D [79][80].

La Kinect représente une véritable révolution dans le domaine de la robotique. Elle dispose d'un capteur qui fournit à la fois une image couleur et une image de profondeur, à partir desquelles la construction 3D se fait directement sans étape de triangulation [58].

C'est dans cette perspective que s'inscrit l'objectif du travail présenté dans ce mémoire, au cours duquel nous nous sommes intéressés au problème d'autonomie du robot en appliquant des algorithmes d'apprentissage par renforcement. Par la suite, le problème de la

modélisation de l'environnement est abordé à l'aide d'une carte élaborée à partir des informations perceptuelles de la Kinect.

La résolution des problèmes de la navigation d'un robot mobile est sujette en grande partie aux recherches et aux résultats obtenus en intelligence artificielle. Nous avons choisi d'utiliser un raisonnement basé sur l'utilisation de la logique floue où des informations plus ou moins précises sont traitées d'une façon similaire à celle que pourrait adopter l'être humain. Elle permet de s'affranchir de toute phase complexe de la modélisation mathématique des systèmes à commander. Bien plus, elle assure une prise de décision et un traitement des informations en des temps de calcul très réduits.

Les premiers travaux présentés dans notre travail portent essentiellement sur le développement d'un système de contrôle réactif pour un robot mobile basé sur la logique floue. Le but principal est de faire « apprendre » à un robot mobile un certain nombre de comportements simples tels que la convergence vers un but, l'évitement d'obstacles. Nous utiliserons des méthodes d'apprentissage par renforcement qui sont basées sur l'exploration active de l'environnement afin d'appréhender les états conduisant à l'émission de récompenses et de punitions. Dans ce cadre, nous avons utilisé deux méthodes d'apprentissage par renforcement, le Fuzzy Actor-Critic Learning (FACL) et le Fuzzy Q-Learning (FQL) qui reposent sur la méthode de prédiction des différences temporelles (TD). Ils permettent la sélection de la meilleure action parmi un ensemble d'actions discrètes disponibles dans chaque règle floue. Par la suite nous effectuons une étude comparative entre ces deux algorithmes afin de définir les paramètres qui peuvent causer la divergence de ces derniers.

En seconde partie de cette thèse, nous présentons une approche de construction d'une carte 3D de l'environnement à partir d'une séquence d'images acquises par une Kinect dans le repère local et global. Ceci après avoir détaillé les techniques d'extraction et d'association des points d'intérêt et les algorithmes de filtrage (SURF, RANSAC), et le classifieur des points KNN pour éliminer les mauvaises associations et l'algorithme ICP pour la détermination des transformations entre les nuages des points.

Cette thèse est organisée de la manière suivante :

Le premier chapitre est consacré à la robotique mobile et la navigation autonome. Un aperçu général sur cette dernière sera abordé pour examiner en bref les principaux stratégies et approches utilisées afin d'assurer le déplacement du robot dans l'environnement d'évolution. Par la suite nous introduisons quelques types des robots et leurs parties constitutives ainsi que leurs domaines d'applications. L'architecture matérielle et logicielle du robot Pioneer II sur lequel nous envisageons d'appliquer les deux algorithmes FACL et FQL est également introduite dans ce chapitre.

Dans le deuxième chapitre, une description générale de l'apprentissage par renforcement est effectuée en introduisant les préliminaires mathématiques présents dans les processus de décision Markoviens PDM, lesquels constituent la base théorique des méthodes que nous adoptons. Les méthodes classiques de la programmation dynamique PD sont décrites également. À la fin, nous abordons les méthodes de l'apprentissage par renforcement floues, les plus populaires : Q-Learning (FQL) et Actor-Critic Learning (FACL) ce qui permet de manipuler des systèmes de grandes dimensions et/ou dont l'espace d'état est continu.

Dans le troisième chapitre, les deux méthodes FQL et FACL sont appliquées pour l'apprentissage en ligne des comportements élémentaires de navigation réactive d'un robot mobile : « Convergence vers un but », « Évitement d'obstacles ». Une étude comparative de l'influence de chaque paramètre constituant ces méthodes est également présentée.

Le dernier chapitre est consacré à l'élaboration d'une carte 3D de l'environnement en utilisant la caméra Kinect. L'interface développée permet de collecter les données RGB-D de la Kinect pour en faire une reconstruction 3D automatique et précise de l'environnement du robot en temps réel, c'est-à-dire pendant son déplacement.

Pour cela, nous utilisons le filtre SURF pour déterminer les points d'intérêts entre les deux images acquises par la Kinect. Le classifieur KNN permet de lier les points correspondants dans ces deux images. Cette liaison peut engendrer des fausses correspondances et peut causer la divergence de l'algorithme de cartographie. Pour éliminer ces derniers (fausses correspondances) nous utilisons le RANSAC. Enfin, et pour obtenir une transformation entre les deux vues pour la fusion de nuages de points 3D nous utilisons l'algorithme ICP.

Nous terminerons ce manuscrit par une conclusion générale récapitulant ce qui a été fait et exposons les perspectives de ce travail.

Chapitre I

Robotique et navigation autonome

Prendre les bonnes décisions c'est prendre le bon chemin qui est devant nous.

Clémence G.

Chapitre I

Robotique et navigation autonome

I.1 Introduction

Le monde de la robotique est l'un des domaines les plus passionnants de l'innovation et dont l'évolution est constante. La robotique est interdisciplinaire et fait de plus en plus partie de notre quotidien. Elle ne s'agit plus une vision du futur mais d'une réalité du présent. Dans notre quotidien, les robots remplissent de nombreuses fonctions importantes : construire nos voitures, prodiguer de l'aide aux personnes à mobilité réduite, intervenir dans des endroits hostiles à l'être humain…etc. Il est à noter qu'ils sont présents presque partout et au fur et à mesure que le temps passe, ils seront de plus en plus présents.

En robotique, une attention particulière est accordée aux robots mobiles, car ces derniers ont la capacité de naviguer dans leur environnement et ne sont pas fixés à un seul emplacement physique. Les tâches qu'ils peuvent accomplir sont très nombreuses et leur possibilité de contribuer à notre vie sont innombrables (véhicules aériens sans pilote, véhicules sous-marins autonomes, par exemple).

Dans ce chapitre nous allons présenter un état de l'art sur la robotique, à savoir la robotique mobile, les différents types des robots existants ainsi les systèmes de vision en robotiques mobile.

I.2 Introduction à la robotique

La robotique est un domaine de recherche important qui fait appel aux connaissances croisées de plusieurs disciplines. L'objectif est l'automatisation des systèmes mécaniques, en les dotant de capacités de perception, de décision et d'action permettant au robot d'interagir rationnellement avec son environnement sans intervention humaine.

La robotique mobile est un domaine dans lequel l'expérimentation est primordiale. L'intérêt indéniable de cette discipline est d'avoir permis d'augmenter considérablement nos connaissances sur la localisation et la navigation des systèmes autonomes. Cet axe de recherche prolonge le domaine d'application de la navigation autonome à toutes sortes d'environnements non structurés, dû principalement à la nécessité d'aider ou de remplacer l'intervention humaine [1].

Ces dernières années, nous avons assisté à une augmentation importante du nombre de robots mobiles. De nos jours, ils font l'objet de grands projets de recherche dans les secteurs de l'industrie, dans le domaine militaire, de la sécurité et même domestique, en tant que produits de consommation destinés au divertissement et à l'aide à domicile.

Le domaine de la robotique mobile présente de nombreux défis à relever. Navigation (cartographie, localisation, planification de trajectoire), perception de l'environnement (capteurs), comportement autonome (actionneurs, règles de comportement, mécanismes de contrôle), équilibre des mouvements), capacité de travailler pendant une période prolongée sans intervention humaine (assistance humaine), sont quelques exemples de possibilités de recherche, parmi beaucoup d'autres. Certains de ces défis sont abordés dans cette thèse.

I.2.1 Historique de la robotique

La robotique est passée par plusieurs générations :

- 1947 : Premier manipulateur électrique télé-opéré.
- 1954 : Premier robot programmable.
- 1961 : Utilisation d'un robot industriel, commercialisé par la société UNIMATION (USA), sur une chaîne de montage de General Motors.
- 1961 : Premier robot avec contrôle en effort.
- 1963 : Utilisation de la vision pour commander un robot.
- 1978 : Le robot ARGOS. Développé à l'Université Paul Sabatier de Toulouse (France). Le robot ARGOS simule la navigation d'un robot mobile équipé d'un système de vision au fur et à mesure de ses déplacements.
- 1979 : Le robot HILARE. Les chercheurs du L.A.A.S. de Toulouse (France) étudièrent la planification des trajectoires d'un robot mobile ponctuel, dans un environnement totalement connu.
- 1981 : Le robot VESA. Ce robot, construit à l'I.N.S.A (France) de Rennes, est équipé d'un arceau de sécurité pour réaliser la détection d'obstacles dans un environnement totalement inconnu.
- 1984 : Le robot FLAKEY. Ce robot, conçu et construit au Stanford Research Institute est le reflet des améliorations apportées par 14 années de développement. Le robot FLAKEY est équipé de deux roues motrices avec encodeurs, mais sa vitesse maximale est de 66 cm/s au lieu de quelques centimètres par seconde. Ce robot est capable de naviguer dans des environnements réels.
- 1993 : Les robots ERRATIC et PIONNER. Le robot ERRATIC a été conçu par Kurt Konolige, au Stanford Research Institue, comme un robot mobile de faible coût pour ses cours de robotique.
- Les robots mobiles actuels : A présent la plupart des travaux de recherche portent sur les problèmes de perception, de planification de trajectoires, l'analyse et la modélisation de l'environnement de robot. La recherche actuelle porte sur la conception mécanique des robots mobiles pour des applications hautement spécialisées, comme l'exploration sous-marine, les robots volants et le micro robots [3].

I.3 Les stratégies de navigation

La navigation d'un robot mobile est un des problèmes clés pour la communauté de la robotique. Le problème consiste à donner aux systèmes mobiles la capacité d'aller d'une position initiale à une position finale de manière autonome, en utilisant l'information perçue par ses capteurs.

Historiquement, la navigation de ces systèmes mobiles a tout d'abord été considérée comme un problème de planification de trajectoires. Le modèle de la scène est supposé connu et des stratégies globales ou locales [72] permettent de définir la séquence de configurations que doit prendre le système mobile pour atteindre son objectif.

Depuis le début des années 80, les avancées technologiques et algorithmiques permettent de contrôler directement les mouvements d'un système robotique mobile à partir des informations perçues par ses capteurs. Ces informations sont utilisées pour définir une loi de commande du système mobile sans pour autant nécessiter le modèle de la scène.

Plus récemment, les recherches en robotique mobile ont considéré des environnements de navigation très vastes, à tel point que les capteurs embarqués (caméra ou même laser) ne peuvent les décrire que localement à un instant donné. Une représentation interne de l'environnement de navigation devient alors nécessaire. Elle permet dans ce cas d'étendre le champ d'action des capteurs du système robotique, mais aussi de définir un ensemble de stratégies de navigation adaptées à l'environnement traversé [4] [5].

Les stratégies de navigation permettent à un robot mobile de se déplacer pour atteindre un but, réaliser une mission ou une tâche. Nous pouvons distinguer plusieurs types de stratégies, cette classification permet de distinguer les stratégies sans modèle interne et les stratégies avec modèle interne [6].

I.3.1 Approche d'un objet

Cette capacité de base permet de se diriger vers un objet visible depuis la position courante du robot. Cette stratégie est locale ou elle est appliquée uniquement quand le but est visible. Elle utilise des actions réflexes dans lesquelles chaque perception est directement associée à une action.

I.3.2 Guidage

Cette stratégie permet d'atteindre un but qui n'est pas directement visible, mais qui est caractérisé par un ensemble d'objets remarquables, ou amers, qui l'entourent. Elle utilise également des actions réflexes et réalise une navigation locale qui requiert que les amers caractérisant le but soient visibles. Cette stratégie est inspirée de certains insectes, comme les abeilles, et a été expérimentée sur divers robots.

I.3.3 Actions associées à un lieu

Cette stratégie est la première à réaliser une navigation globale, atteindre un but qui n'est pas visible et même les amers qui caractérisent son emplacement sont invisibles. Elle requiert une représentation interne de l'environnement qui consiste à définir des lieux comme des zones de l'espace dans lesquelles les perceptions restent similaires, et à associer une action à effectuer en chacun de ces lieux. L'enchaînement de ces actions associées définit un chemin vers le but (figure I.1). Ces modèles sont limités à un but fixé : Un chemin qui permet d'atteindre un but ne pourra en effet pas être utilisée pour rejoindre un but différent.

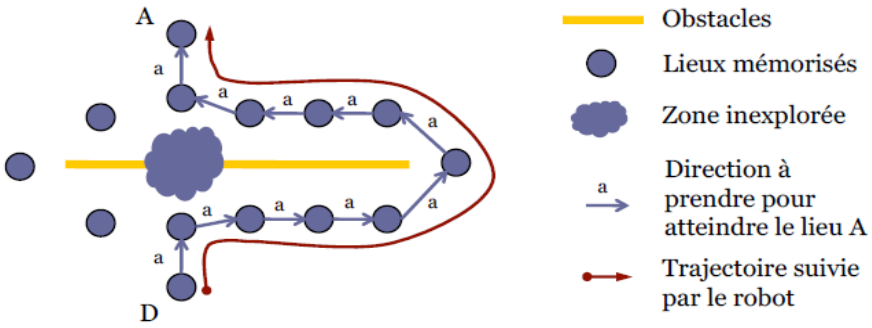


Figure I.1 : Stratégie de navigation : Actions associés à un lieu [7].

I.3.4 Navigation topologique

Cette capacité est une extension de la précédente qui mémorise dans le modèle interne les relations spatiales entre les différents lieux. Ces relations indiquent la possibilité de se déplacer d'un lieu à un autre, mais ne sont plus associées à un but particulier. Le modèle interne est un graphe qui permet de déterminer différents chemins entre deux lieux arbitraires (figure I.2). Ce modèle ne permet toutefois que la planification de déplacements parmi les lieux les plus connus et suivant les chemins connus.

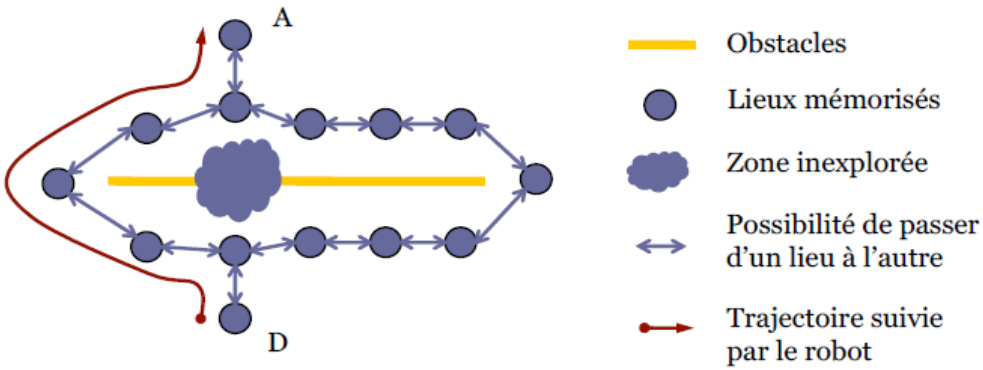


Figure I.2 : Stratégie de navigation : Navigation topologique [7].

I.3.5 Navigation métrique

Cette stratégie est une extension de la précédente car elle permet au robot de planifier des chemins au sein de zones inexplorées. Elle mémorise pour cela les positions métriques relatives des différents lieux, en plus de la possibilité de passer de l'un à l'autre. Ces positions relatives permettent, par simple composition de vecteurs, de calculer une trajectoire allant d'un lieu à un autre, même si la possibilité de ce déplacement n'a pas été mémorisée sous forme d'un lien.

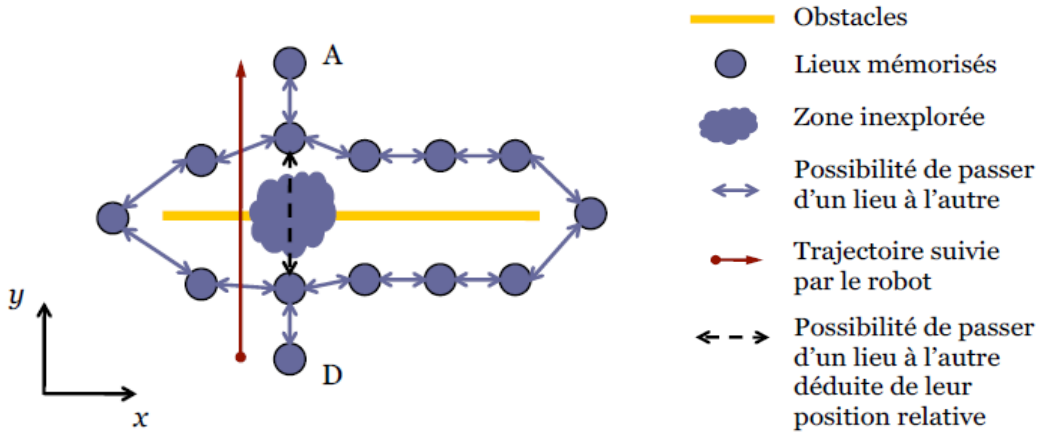


Figure I.3 : Stratégie de navigation : Navigation métrique [7].

I.3.6 Navigation réactive

Par définition, les approches réactives n'utilisent que les valeurs courantes des capteurs, et non des données provenant d'un modèle interne, pour décider de l'action à effectuer. Elles sont généralement exécutées très rapidement et elles permettent de réaliser des tâches de bas niveau, comme l'évitement d'obstacles imprévus, essentiel à la sécurité d'un système mobile. Dans ce type d'approche, la perception est indispensable car on considère que le système mobile évolue dans un environnement qui lui est partiellement connu.

I.4 Différents types de robot

On peut dire qu'il y a deux catégories des robots selon le critère de mobilité :

- Les robots manipulateurs.
- Les robots mobiles.

I.4.1 Les robots manipulateurs

Ce type de robots est conçu pour réaliser une tâche unique et indéfiniment répétée telle que l'assemblage des pièces dans une chaine de production dans une usine. Ce type n'est pas mobile, le robot ne doit accomplir que la tâche pour laquelle il a été conçu. La définition donnée aux robots manipulateurs industriels par la Fédération Internationale de la Robotique sous le standard ISO/TR 8373 est la suivante :

Un robot industriel manipulateur est dédié à un contrôle automatique, reprogrammable. C'est un manipulateur multifonction programmable en trois axes ou plus, qui peut être soit immobile dans un endroit, soit mobile. Il est destiné à des applications d'automatisation industrielles.

La définition qu'on vient de citer nous informe que ces robots étaient conçus essentiellement pour répondre aux besoins de l'industrie ; ils sont intelligents programmables et peuvent assurer un mouvement en trois dimensions (figure I.4).

Figure I.4 : Exemple de robots manipulateurs actuels.

- **Domaine d'utilisation des robots manipulateurs**

Il existe de multiples raisons et situations dont il est préférable de se servir des robots manipulateurs : ils peuvent fonctionner sans arrêt, si nécessaire 24 h sur 24, et nécessitent un entretien minimal. Ils sont plus rapides que l'être humain et exécutent le travail en continu, d'une manière précise et constante.

Cependant les robots ne sont pas encore prêts à remplacer totalement l'être humain parce que de nombreux obstacles techniques et sécuritaires se dressent encore ; par conséquent la présence de l'homme est indispensable pour en assurer le contrôle et la maintenance.

Ci-dessous nous citons quelques raisons pour utiliser les robots manipulateurs :

- Environnement dangereux qui présente beaucoup de risque pour la vie de l'être humain.
- Un travail répétitif et cyclique.
- Travail difficile pour l'Homme ou tâches qui nécessitant une haute précision (nano technologie).

I.4.2 Les robots mobiles

Ces robots exécutent des tâches qui nécessitent un déplacement dans un environnement donné.

Ils sont en effet le plus souvent désignés par leur type de locomotion, qu'ils soient marcheurs, sous-marin ou aérien. Contrairement aux robots manipulateurs prévus pour travailler exclusivement dans des espaces connus et de manière répétitive, les robots mobiles sont destinés à évoluer de manière autonome dans des environnements peu ou non structurés.

Néanmoins, le concepteur doit augmenter considérablement ses connaissances sur la localisation et la navigation de systèmes autonomes et prendre en considération l'environnement du mouvement lors de la conception de la machine. On peut citer le robot « Helpmate » comme exemple de ce type (figure I.5).

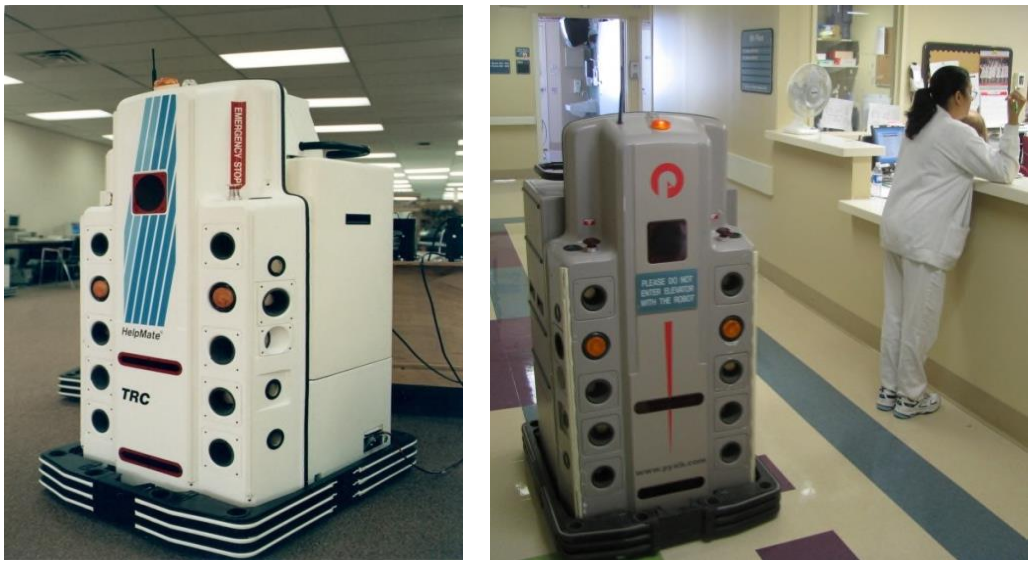


Figure I.5 : « Helpmate » Robot utilisé dans les hôpitaux [10].

I.5 Composantes d'un robot mobile

À la base, un robot mobile est constitué de composantes matérielles et logicielles. Parmi les composantes matérielles, on retrouve une plateforme mobile à laquelle sont rattachées toutes les autres composantes comme les capteurs, les actionneurs et une source d'énergie (batteries) [9][12].

I.5.1 Capteurs

Les capteurs ont pour fonction d'acquérir des données provenant de l'environnement. Les capteurs typiquement installés sur un robot mobile (figure I.6) sont des sonars à ultrasons, un capteur laser de proximité, des encodeurs de roues (odomètres), une ou deux caméras optiques et des microphones. Les types d'informations perçues ainsi que leur précision varient beaucoup d'un capteur à l'autre. Par exemple la figure I.7 montre qu'un capteur laser de proximité (c) permet de mieux percevoir les contours de l'environnement (a) que les sonars (b) puisque le capteur offre une meilleure résolution angulaire et une meilleure précision en termes de distance [9] [11][12].

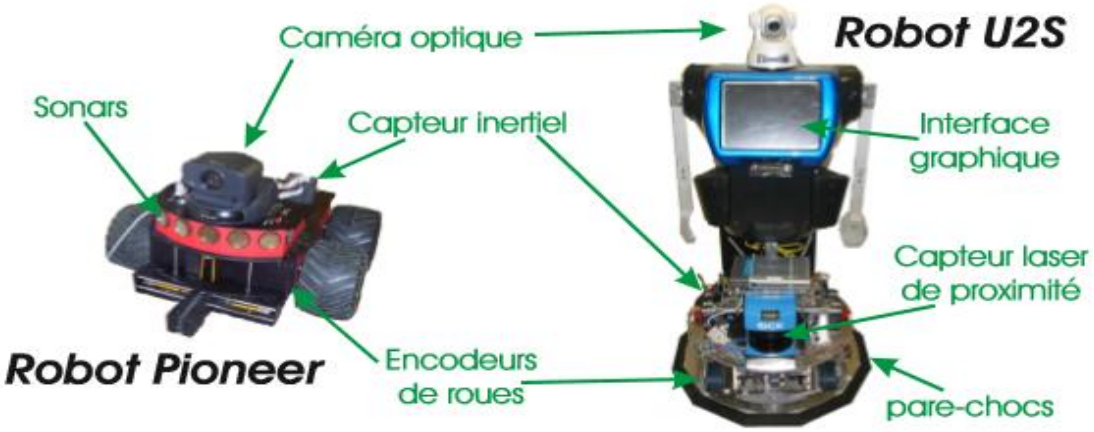


Figure I.6 : Capteurs typiques d'un robot [11].

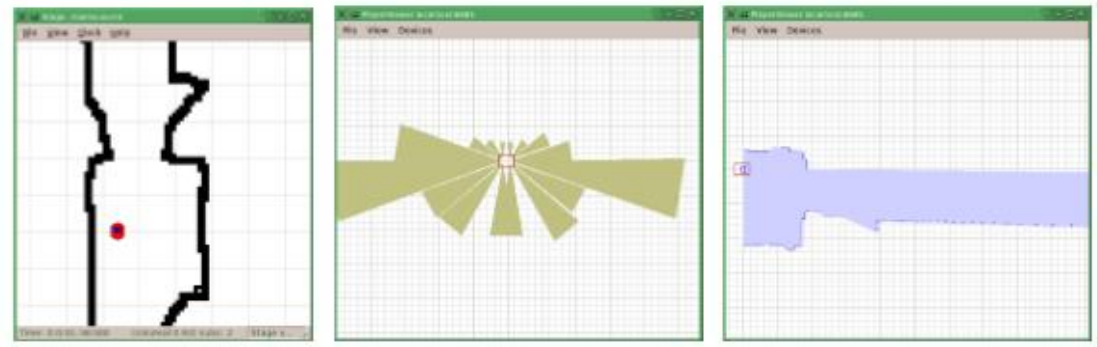


Figure I.7 : Perceptions des quelques capteurs [12].

I.5.2 Actionneurs

Pour naviguer à l'intérieur de son environnement et interagir avec celui-ci, un robot est équipé d'actionneurs. Par exemple, un robot est muni d'un ou de plusieurs moteurs pouvant faire tourner ses roues afin d'effectuer des déplacements. Généralement, les roues du robot sont contrôlées par deux commandes motrices, soit une vitesse d'avancement et un taux de rotation. Habituellement, ces commandes s'expriment en mètres par seconde (m/s) et en degrés de rotation par seconde (deg/s) [13] [14].

I.5.3 Modules logiciels

Afin de faire fonctionner un robot mobile, plusieurs modules logiciels sont mis à contribution [21] [22]. Ces modules peuvent servir à interpréter les données perçues par les capteurs afin d'en extraire des informations, ou à traiter des commandes de haut niveau pour générer d'autres commandes à un niveau inférieur. Les modules les plus fréquemment utilisés sont les modules de localisation, de navigation, de vision, audio et de séquencement d'activités du robot [16] [19].

I.6 Caractéristiques d'un robot mobile

Un robot doit être choisi en fonction de l'application qu'on lui réserve. Voici quelques paramètres à prendre, éventuellement, en compte :

- **La charge maximale transportable**

De quelques kilos à quelques tonnes, à déterminer dans les conditions les plus défavorables (maximale).

- **L'espace de travail**

Il est défini comme l'ensemble des points qu'on peut atteindre par l'organe terminal (dextrous- workspace). Autrement dit est l'espace dont le robot peut atteindre avec toutes les orientations possibles de l'organe terminal [48]. Dans la plupart des cas, tous les mouvements de ce derniers (l'organe terminal) ne sont pas possibles en tout point de ce volume. On appelle espace de travail maximal, l'espace dont le robot peut atteindre via au moins une orientation (reachable workspace).

- **Le positionnement absolu**

Correspondant à l'erreur entre un point souhaité (réel), défini par une position et une orientation dans l'espace cartésien et le point atteint. Il est calculé via le modèle géométrique inverse du robot. Cette erreur est due au modèle utilisé, à la quantification de la mesure de position et à la flexibilité du système mécanique [49].

En général, l'erreur de positionnement absolu, également appelée précision, est de l'ordre de 1 mm.

- **La répétabilité**

Ce paramètre caractérise la capacité que le robot à retourner vers un point (position, orientation) donné. La répétabilité correspond à l'erreur maximum de positionnement en un point prédéfini dans le cas de trajectoires répétitives.

- **La vitesse de déplacement**

Elle est définie comme la vitesse maximale atteinte en accélération maximale.

Il existe d'autres caractéristiques comme : La masse du robot, son coût du robot, en sa maintenance, …

I.7 Les différents types d'environnements

On rencontre principalement 3 types d'espaces de navigation : les terrains plats, les terrains accidentés et les espaces 3D. Les terrains plats sont généralement utilisés pour modéliser les milieux urbains et les intérieurs de bâtiments. Le robot évolue sur un plan 2D considéré sans pentes, et tout objet qui sort de cet espace 2D est considéré comme un obstacle. Cette représentation est la plus simple à étudier et la plus répandue pour les robots mobiles à roues. En première approche, elle permet de se concentrer sur les problèmes de contrôle et de navigation autonome du robot.

Figure I.8 : Robot mobile ATRV2 pour des terrains plats [22].

Les terrains accidentés correspondent généralement aux milieux en extérieur, comme les forêts, les champs agricoles, ou encore les terrains rocheux (Figure I.17). La différence avec les terrains plats est la présence de pentes, de bosses et de creux sur le terrain d'évolution du robot. Cela interdit l'utilisation d'une métrique standard 2D et complique la détection d'obstacles et la modélisation des déplacements du robot.

Figure I.9 : Robot dans la planète mars [23].

Les espaces d'évolution 3D sont par exemple utilisés pour modéliser la navigation des drones volants (Figure I.11) et des robots sous-marins (Figure I.10). Les problèmes rencontrés sont spécifiques à l'application visée. Chaque type de terrain correspond à des problématiques bien spécifiques. Le type de robot étudié dans ce mémoire est destiné à circuler en environnement urbain, la modélisation terrain plat sera utilisée. Cela signifie que l'on considère que tous les mouvements sont contenus dans un plan de navigation, parallèle au sol.

Figure I.10 : Robot sous-marin [51].

Figure I.11: Robot drone [50].

La tâche de navigation consiste à donner au robot la possibilité d'obtenir les informations dont il a besoin pour raisonner et le doter de capacité de locomotion adaptée à son environnement [48]. Cependant, celle-ci implique des systèmes complexes dans la réalisation, où leurs maîtrises posent d'importants problèmes non seulement technologiques mais aussi scientifiques. L'intérêt de la navigation est d'atteindre un but désiré, tout en s'adaptant à certaines variations des conditions de fonctionnement sans intervention humaine [47] [49] [39] [41].

I.8 Acquisition de données de l'environnement

I.8.1 Capteurs intéroceptifs

Ils fournissent des données sur l'état interne du robot (vitesse, position, orientation,…). Ces informations renseignent le robot en cas de mouvement, sur son déplacement dans l'espace (la localisation). Ce sont des capteurs que l'on peut utiliser directement, mais ils souffrent d'une dérive au cours du temps qui rend leur utilisation seule inefficace ou avec limitation. Nous citons par exemple : l'odomètre, radar doppler, systèmes inertiels,… [48], [52].

(a)

(b)

Figure I.12 : Capteurs intéroceptifs ; (a) Capteur de position télémécanique, (b) Capteur de vitesse [48].

I.8.2 Capteurs extéroceptifs

Ils ont pour objectif d'acquérir des informations sur l'environnement proche du véhicule. Ils fournissent des mesures caractéristiques de la position que le robot peut acquérir dans son environnement par la détection des objets. Ces informations peuvent être de natures très variées. Nous citons comme exemple les télémètres à ultrason, infrarouge, laser, les caméras,…etc.

De même pour acquérir des données à partir d'un environnement quelconque deux types de capteurs sont utilisés :

- Les capteurs actifs.
- Les capteurs passifs.

Pour une application donnée, le choix du capteur adéquat dépend du type même de la tâche à réaliser (navigation, inspection, etc.), de la précision requise, et bien évidemment du type de l'environnement à explorer (scènes d'intérieur, environnement hostile, environnement naturel, etc.).

I.8.2.1 Les capteurs actifs

Le principe de ce type de capteur est le calcul du temps de propagation des ondes qu'il émet. Ce temps mis par une onde pour parcourir la distance aller-retour entre la source du capteur et un point de l'objet de la scène permet de calculer la position de ce point dans un plan. Ils sont appelés capteurs actifs car ils fournissent de l'énergie pour interagir avec l'environnement. Les exemples les plus répandus de tels systèmes sont le télémètre laser et la sonde à ultrasons, qui sont utilisés souvent pour la navigation et la localisation des robots mobiles.

I.8.2.2 Les capteurs passifs

À la différence des capteurs précédents, ces capteurs n'ont aucune influence sur l'environnement étudié (ne transmettent aucune énergie). Leur principe est simple. Ils reçoivent les signaux émis ou réfléchis par l'environnement. Nous distinguons principalement les caméras qui sont utilisées dans de nombreuses applications. Elles permettent de remonter à une représentation 3D de la scène perçue en utilisant le plus souvent des algorithmes de stéréovision [52].

I.9 Les systèmes de vision

Tant que capteurs passifs, les caméras sont des éléments qui permettent de capturer une partie d'une scène réelle, sous forme d'une image digitalisée ou d'un tableau de pixels. Donc, elles nous donnent des images sensorielles de l'environnement apparu sur son champ de vision, en mesurant la luminosité des points correspondent aux pixels de l'image. De plus, les caméras donnent une grande quantité d'informations exploitables avec une longue portée, rapidité et grande précision.

En revanche, l'inconvénient majeur de tels capteurs se situe d'abord au niveau de la gestion du flux important de données exploitables (traiter une image demeure une opération

délicate et surtout coûteuse en temps de calcul), ensuite à leur sensibilité aux conditions d'éclairage.

Les systèmes de vision en robotique sont basés sur l'utilisation d'une caméra CCD. L'arrivée des capteurs CCD (Charge Coupled Device), en 1975, a été déterminante dans l'évolution de la vision : la rapidité d'acquisition, la robustesse et la miniaturisation sont autant d'avantages qui ont facilité leur intégration. Les systèmes de vision sont très performants en termes de portée, précision et quantité d'informations exploitables. Ils sont de plus les seuls capables de restituer une image sensorielle de l'environnement la plus proche de celle perçue par l'être humain. En revanche, l'inconvénient majeur de tels systèmes de perception se situe au niveau de la gestion du flux important de données exploitables : traiter une image demeure une opération délicate et surtout coûteuse en temps de calcul. Utilisé seule, une caméra CCD ne peut fournir qu'une information 2D. Les techniques qui permettent d'obtenir des informations 3D à partir d'un tel capteur sont généralement liées à l'adjonction d'un autre capteur. Dans ce cadre nous pouvons identifier les techniques suivantes :

I.9.1 La stéréovision

Elle consiste à observer une même scène avec deux caméras qui sont éloignées l'une de l'autre et dont on connaît la distance qui les séparent. Connaissant la géométrie exacte du système stéréoscopique la première étape de reconstruction 3D consiste à mettre en correspondance les deux images. Cette phase réside dans la détermination de couples de points observés dans les deux images, ou dans l'apparition de points d'intérêt. L'information 3D pourra alors être fournie par triangulation. La stéréovision est basée sur le même principe de reconstitution de la profondeur que la vision chez l'être humain.

Le principe de stéréovision a par exemple été utilisé par Olson pour le très médiatique robot Rocky7 destiné à être envoyé sur la planète Mars [53]. Le robot ne pouvant posséder une carte préétablie de son environnement extra-terrestre d'évolution, celle-ci est construite en utilisant la stéréovision. Rocky7 se sert ensuite de cette carte globale pour apparier la carte locale, obtenue elle aussi par stéréovision, en utilisant une mesure de similarité basée sur la distance de Hausdorff. La stéréovision est utilisée dans de nombreuses autres applications robotiques [54] [55] [56]. Plus récemment Gluckman et Nayar dans [57] ont placé deux miroirs plans pour calculer la profondeur d'une scène. L'utilisation d'une caméra et de deux miroirs plans est équivalente à la stéréoscopie avec deux caméras. Elle présente en outre l'avantage que les données stéréoscopiques proviennent de la même caméra.

La stéréovision apparaît comme un des moyens de perception les plus performants en robotique. Toute la problématique de la stéréovision réside dans la robustesse de la phase de mise en correspondance des informations : elle est souvent liée à de nombreuses ambiguïtés mais aussi à des temps de calcul très importants (aspect fortement combinatoire de l'appariement).

I.9.2 Caméra de profondeur

Les problèmes de la vision 3D sont résolus avec l'arrivée des caméras des caméras de profondeur (RGB-D). Elles permettent de repérer des objets beaucoup plus facilement et

rapidement et même de répondre aux gestes des humains grâce à l'extraction de squelettes. La Kinect représente une véritable révolution dans le domaine de la robotique. Le capteur Kinect que nous allons exploiter sur le robot nous permet d'obtenir des informations 3D d'une manière simple, rapide et moins chère.

I.10 Caméra Kinect

I.10.1 Historique sur le capteur Kinect

En juin 2009, a été annoncée la Kinect sous le nom « Project Natal ». Le nom a été pris en compte, d'après la ville brésilienne « Natal », qui est la ville natale du directeur de Microsoft « Alex Kipman ». Le 25 mars 2010, Microsoft annonce officiellement que le système est appelé Kinect. Ce nom fusionne deux mots « Kinetic » et « Connect », qui décrivent des aspects clés de l'initiative. Les premières utilisations de la Kinect eurent lieu en novembre en Amérique du Nord.

Lancée en 2010, Kinect a définitivement fait basculer la vision en 3D dans le 21ème siècle tout en révolutionnant les machines. L'idée de la technologie de détection de profondeur a été inventée en 2005 par l'institut BINA (Barilan Institute of Nanotechnology and Advanced materials).

En juin 2011, Microsoft a annoncé la sortie officielle de son pilote de développement logiciel pour un usage non commercial. En octobre 2011 Microsoft a annoncé le lancement de la version commerciale du programme de Kinect pour Windows avec la version du SDK (Soft Development Kit).

Enfin, en février 2012, Microsoft a révélé la version commerciale de Kinect pour Windows SDK [58]. Donc la Kinect se présente comme une véritable innovation pour la robotique. Cela refigure sans doute l'avènement d'une nouvelle classe de capteurs intelligents dotés de capacité de prétraitement de l'information et permettent l'accès à une information beaucoup plus riche que les capteurs actuels.

La Kinect de Microsoft a été utilisée dans plusieurs projets. Parmi les premiers projets qu'ont utilisé la Kinect en robotique mobile, il y a celui de Peter Henry et son groupe lancé en décembre 2010. Ils ont utilisé la Kinect pour la modélisation 3D d'un environnement intérieur (cartographie 3D).

Après une année ils ont travaillé sur l'OVK (Odométrie Visuelle Kinect), ils ont embarqué une caméra RGB-D sur un robot de type quadrotor. D'autres idées ont été réalisées comme celles de l'université de RYERSONA Toronto Canada, lors duquel ils ont utilisé une entrée vidéo 3D pour estimer la position du robot [59].

La figure I.13 montre une vue de face d'un capteur Kinect qui a été déballé de son boîtier noir.

Figure I.13 : La Kinect sans cache[60].

I.10.2 Principe de fonctionnement de la Kinect

Le principe de fonctionnement de la Kinect consiste à projeter un flux laser sur la scène, qui appartient à la gamme infrarouge, et à capturer simultanément une image IR (dans toutes les conditions de lumière ambiante) avec un capteur CMOS monochrome équipé d'un filtre IR, en décodant la déformation des flux émis [60].

Ainsi les valeurs de profondeur réelle sont les estimations de la distance entre l'objet et le plan de la caméra IR (Figure I.14). De plus, le capteur de profondeur est considéré comme un dispositif qui peut retrouver les coordonnées cartésiennes (X, Y, Z) des objets 3D.

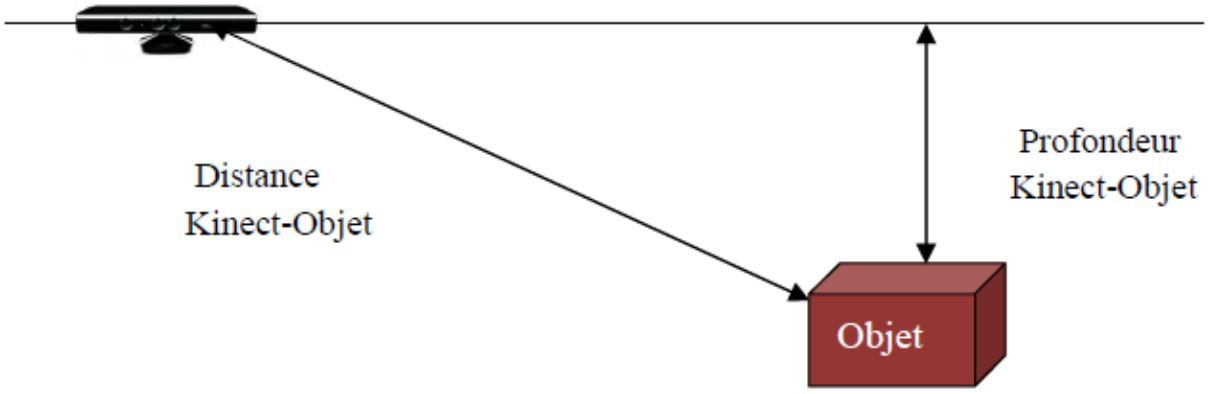


Figure I.14 : La technique de mesure de la profondeur [60].

I.11 Calibrage de la Kinect

Pour que la Kinect soit prête à l'utilisation, elle doit être étalonnée (calibrée). Ceci se fait en deux phases : la première consiste à calibrer les deux caméras toutes seules (RGB et IR) pour obtenir les paramètres intrinsèques et la deuxième phase consiste à déterminer les paramètres extrinsèques, en considérant les deux caméras comme un banc stéréoscopique dans lequel nous nous intéressons à la distance entre les deux caméras.

La procédure consiste à déterminer les différentes erreurs fournis par les défauts des lentilles. Dans ce cas, nous devons aligner le tableau des pixels de chaque caméra avec son axe optique.

On prend des différentes vues d'une mire dans lesquelles les images IR sont prises avec un couvert du projecteur IR.

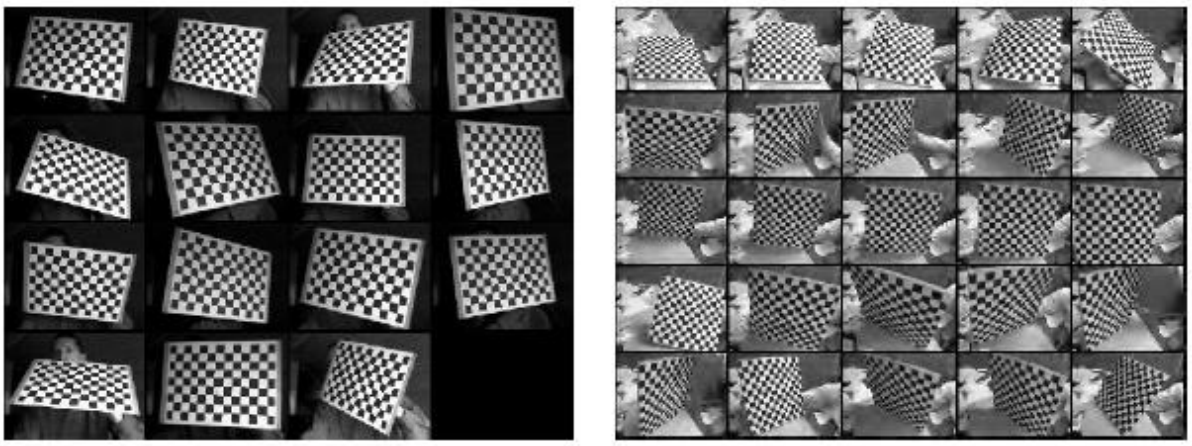


Figure I.15 : Différentes vues d'une mire de calibrage[60]

I.11.1 Calibrage des deux caméras IR et RGB

Le calibrage permet de déterminer les paramètres intrinsèques et extrinsèques d'une ou plusieurs caméras en utilisant une mire dont le modèle est connu à l'avance. Pour réaliser le calibrage des deux caméras, les paramètres à estimer sont :

- Les paramètres intrinsèques de chaque caméra.
- Les paramètres extrinsèques.
- Les coefficients de distorsion.

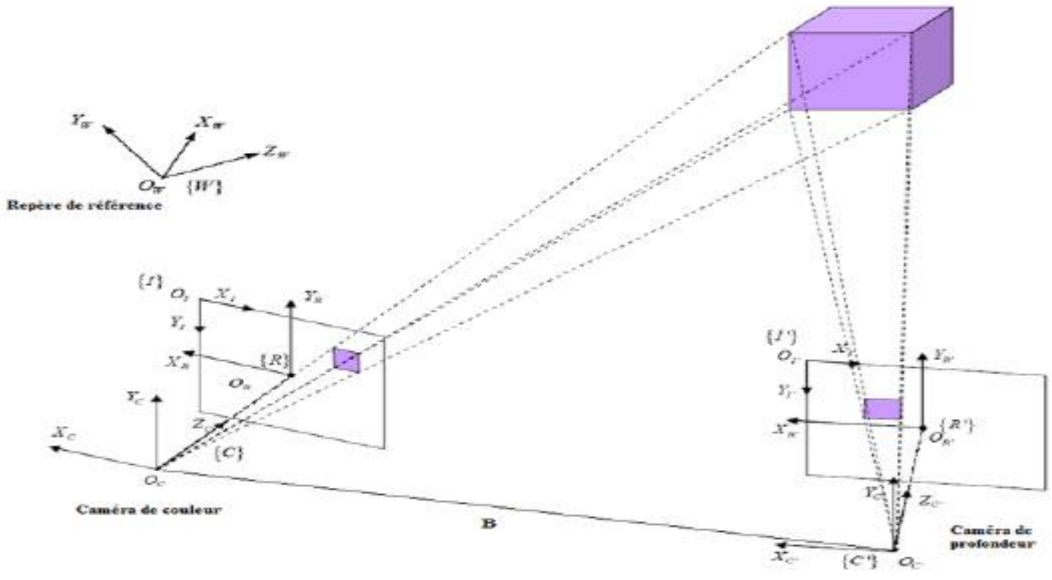


Figure I.16 : Principe du calibrage stéréoscopique [60].

Pour déterminer les paramètres du modèle de la caméra, nous la placerons devant une mire (objet étalon) de calibrage. C'est un ensemble de points dont les coordonnées sont parfaitement connues dans un repère de la mire qui est différent du repère caméra. Chaque point de la mire se projette dans l'image, puis nous mesurons ses coordonnées dans le repère

image. Chaque point est défini par ses coordonnées (Xw, Yw, Zw) dans le repère de référence. Il est également précisé par ses coordonnées (Xc, Yc, Zc) dans le repère caméra.

I.11.2 Les paramètres intrinsèques

Le changement de repère qui lie les coordonnées du point Q dans le repère image (Figure I.17) avec le repère lié à la caméra est défini par les paramètres de la caméra.

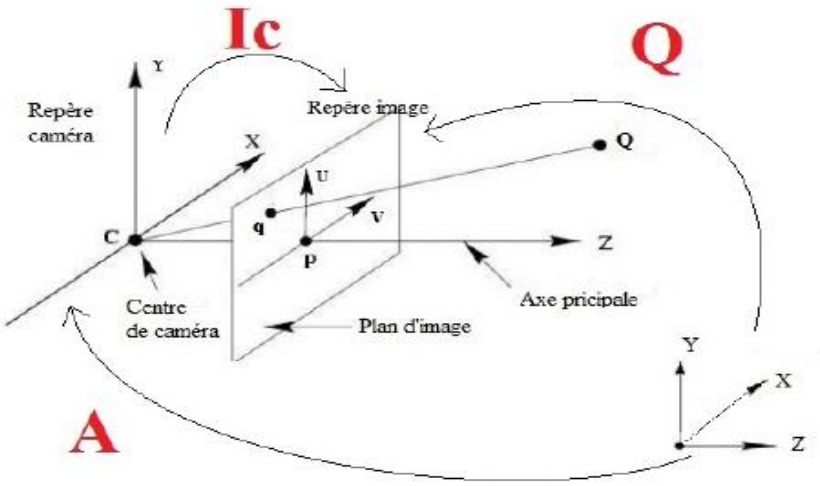


Figure I.17 : Projection perspective.

En projection perspective, un point Q est transformé en son homologue q dans le plan image par :

$$\begin{cases} u = u_0 - k_u f \dfrac{x_c}{z_c} \\ v = v_0 - k_v f \dfrac{y_c}{z_c} \end{cases}$$

Avec : Q = Ic * A.

(u, v) : sont les coordonnées dans le repère image,

(u_0, v_0) : est le centre d'image.

f : la distance focale.

(k_u, k_v) : les facteurs d'échelle.

(x_c, y_c) : sont les coordonnées du point dans le repère caméra.

Soit une notation matricielle :

$$\begin{bmatrix} U \\ V \\ W \end{bmatrix} = \begin{bmatrix} -k_u & 0 & u_0 \\ 0 & k_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix}$$

On pose :

$$I_c = \begin{bmatrix} a_u & 0 & u_0 \\ 0 & a_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} avec\ a_u = -k_u f\ et\ a_v = -k_v f$$

C'est une application linéaire de l'espace projectif vers le plan projectif exprimant la transformation perspective :

$$\begin{bmatrix} su \\ sv \\ s \end{bmatrix} = I_c \begin{bmatrix} x \\ y \\ z \\ s \end{bmatrix}$$

Dans ce modèle figure quatre paramètres a_u, a_v, u_0, v_0 ce sont les paramètres d'estimation du calibrage. Notons que la distance focale ne peut être calculée explicitement. Nous introduisons des coordonnées caméras normalisées par rapport à Z telles que :

$$\begin{cases} x_c = \dfrac{x}{z} \\ y_c = \dfrac{y}{z} \\ z_c = 1 \end{cases}$$

À partir de cela la relation entre les coordonnées image et les coordonnées caméra est:

$$\begin{cases} u = a_u x_c + u_0 \\ u = a_v y_c + v_0 \end{cases}$$

I.11.3 Les paramètres extrinsèques

Les paramètres extrinsèques permettent de définir la transformation géométrique entre le repère caméra et la scène (repère objet), ils sont représentés par la matrice A :

$$A = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Dans ce cas, nous devons extraire la matrice de transformation entre les deux repères des caméras à partir le toolboox calib-gui qui nous permet d'obtenir la matrice de rotation et de translation ainsi que les positions de la mire par rapport à la caméra pendant le calibrage.

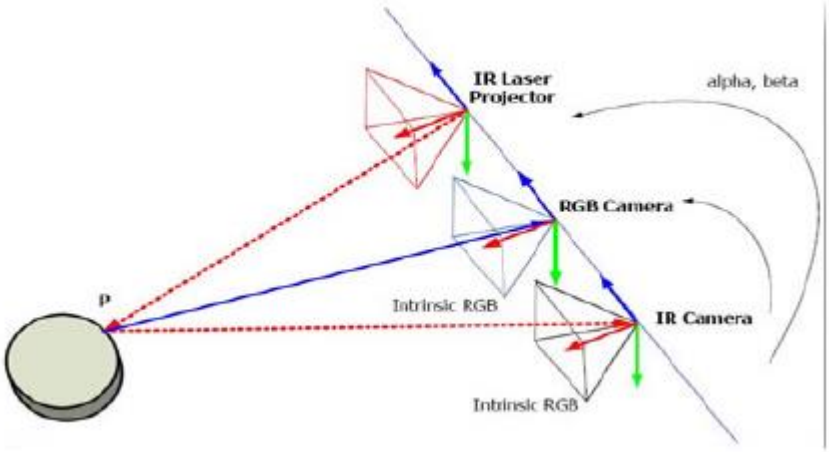


Figure I.18 : La position des deux caméras (RGB, IR) par rapport à la mire.

Les résultats de calibrage obtenus sont représentés dans le tableau, ils montrent que la matrice de rotation entre les deux caméras est égale à l'identité. La première composante du vecteur de translation (selon x) est égale à 26,67705 mm, dont on sait a priori (mesure prise manuellement) qu'elle est égale à 26mm, donc nous pouvons dire que notre estimation des paramètres est juste avec une petite erreur de 0,67705mm.

Figure I.19 : Distance entre caméra RGB et caméra IR.

Caméra droite IR	Le focal	a_u = 526.97494 a_v = 528.35283
	Le point central	u_0 = 324.17104 v_0 = 254.74485
Caméra gauche RGB	Le focal	a_u = 582.52835 a_v = 584.54913
	Le point central	u_0 = 322.26189 v_0 = 240.73137

Transformation entre les deux caméras	Matrice de rotation	r_{11}=0.99984628826577793 r_{12}=0.0012635359098409581 r_{13}=-0.017487233004436643 r_{21}=-0.0014779096108364480 r_{22}=0.99992385683542895 r_{23}=-0.012251380107679535 r_{31}=0.017470421412464927 r_{32}=0.012275341476520762 r_{33}=0.99977202419716948
	Vecteur de translation	t_x = 0.019985242312092553 t_y = -0.00074423738761617583 t_z = -0.010916736334336222

Table I.1 : Résultats de calibrage stéréoscopique de la Kinect.

I.12 Plateforme expérimentale

I.12.1 Architecture matérielle

Dans la première partie de cette étude nous utilisons un type particulier de robot mobile holonome à roues. : Le « Pioneer II » (Figure I.20). Il se prête à la navigation dans les coins serrés et espaces encombrés telle que les salles de classes, les laboratoires et petits bureaux. Destiné à l'utilisation d'intérieur sur les surfaces dures et plates, le Pioneer II possède des pneus en caoutchouc pleins et deux roues différentielles avec un système réversible d'entraînement à courant continu avec une roulette arrière pour l'équilibre.

Chaque système d'entraînement du moteur intègre un capteur incrémental avec une résolution de 19 pulses/mm pour déterminer précisément la position et la vitesse.

Pour réaliser la détection d'obstacles il dispose d'une rangée de huit capteurs ultrasonores à travers huit transducteurs placés en son avant S_i (avec i=1……8). Les champs de mesures de ces capteurs sont compris entre 10 cm de portée minimale et 5 m de portée maximale et un champ de vision de 20° (Figure I.21).

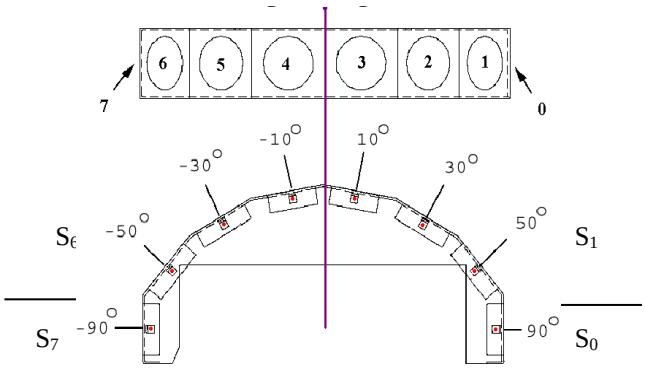


Figure I.20 : Le robot Pionner II. **Figure I.21 :** Le groupe des capteurs ultrasons de Pionner II.

I.12.2 Architecture software

Le robot Pioneer II est contrôlé par le logiciel Saphira développé par Kurt Konolige du Laboratoire International d'intelligence Artificielle (SRI). À partir de la version 8.x, plusieurs fonctions de noyau de Saphira ont été déplacées complètement au logiciel ARIA « Activmedia Robot Interface for Application ».

Saphira et Aria sont écrits en langage C++, et leurs API se servent des propriétés des objets de C++pour fournir une interface efficace à la programmation des robots.

Pour des facilités la programmation, nous avons utilisé un schéma bloc, intégré dans Simulink et utilisant les API (S-Function), qui regroupe les principales fonctions de Saphira de perception, de localisation et de commande (figure I.22).

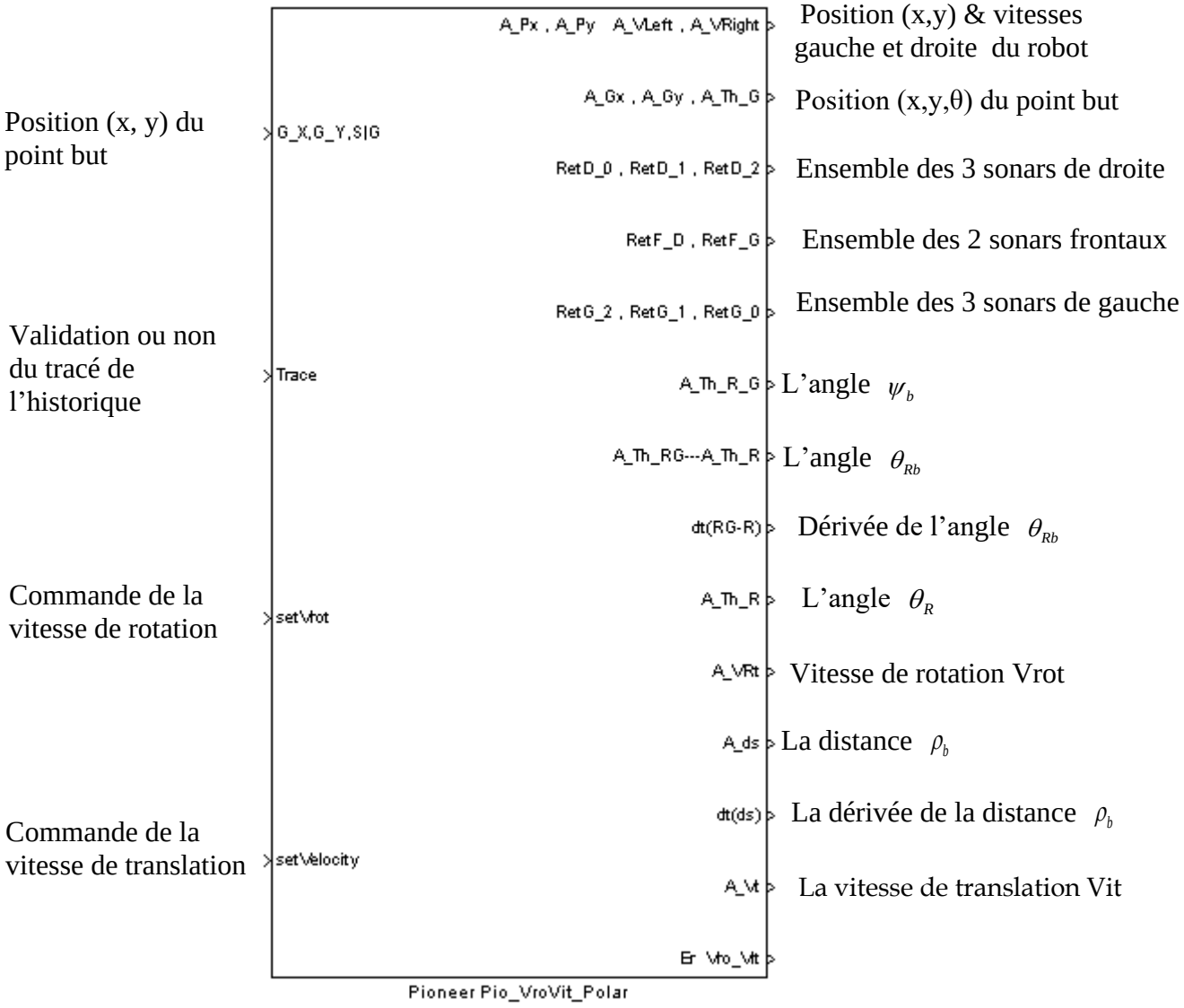


Figure I.22: Schéma bloc réalisant les fonctions de Saphira.

I.13 Conclusion

La robotique est la branche de l'intelligence artificielle (IA) concernée par l'étude des systèmes automatiques capables d'interagir directement avec le monde physique. C'est une automatisation des dispositifs, où l'objectif est d'augmenter les capacités de localisation et de navigation dans son espace de travail.

Ce chapitre nous a permis de présenter un exposé exhaustif sur le domaine de la robotique mobile, en définissant les robots d'une façon générale, leurs différents types, la navigation autonome des robots mobiles en particulier.

On a présenté aussi les différentes applications des robots présentant des structures mécaniques assez simple, ainsi leurs systèmes de commandes.

La commande d'un robot mobile se divise généralement en trois étapes principales : perception, décision et action. La première étape est très importante, elle concerne l'acquisition des informations de l'environnement extérieur. C'est une étape qui doit être maîtrisée efficacement pour accomplir les missions qu'on lui attribue avec succès. D'après l'étude effectuée, on peut dire qu'un robot est un système combiné de hardware et de software capable de collecter, traiter les données et d'effectuer des tâches intelligentes dans un environnement physique.

Après cet aperçu, le chapitre suivant, est destiné à présenter une technique d'intelligence artificielle qui permet de faire apprendre au robot comment accomplir la tâche demandée en appliquant un système de récompense-punition, à savoir l'apprentissage par renforcement.

Chapitre II

Apprentissage par renforcement

Essaye encore, échoue encore : échec profitable.

Samuel Beckett.

Chapitre II

Apprentissage par renforcement

II.1 Introduction

L'apprentissage est considéré comme un moyen de construction de nouvelles connaissances ou d'amélioration des connaissances présentés. Le but est d'améliorer les performances d'un système en tenant compte des ressources et des compétences dont il dispose. Il existe plusieurs formes d'apprentissage et de modification de comportement. Selon le type d'informations disponibles, on peut citer : l'apprentissage supervisé, non supervisé et l'apprentissage par renforcement. Chacun d'eux possède des propriétés pertinentes.

Dans le chapitre, nous allons introduire en premier lieu quelques définitions et notions de base. Par la suite nous développerons la technique d'apprentissage par renforcement. Grâces à plusieurs caractéristiques, l'utilisation de cette approche pour l'amélioration des comportements et la commande des processus devient un domaine très important dans ces dernières décennies [61][62].

Le formalisme mathématique d'apprentissage par renforcement étudié dans ce travail est donné à travers les deux algorithmes : actor-critic learning ACL et Q-learning. Ces techniques d'apprentissage automatique basé sur un signal critique d'évaluation adoptés utilisent un apprenti de type système d'inférence flou SIF.

II.2 Définitions

Avant de développer les notions de l'approche d'apprentissage étudiée, nous retenons les notions utiles suivantes :

II.2.1 Agent

D'après Ferber [63]; un agent est une entité autonome réelle ou abstraite qui est capable d'agir sur elle-même et sur son environnement, et peut communiquer avec d'autre agents, dont le comportement est la conséquence de ses observations, ses connaissances et de ces interactions avec son environnement. Il doit être (autonome, interactif, adaptatif, rationnel, coopératif et intelligent…). La notion d'agent se diffère selon l'utilisation.

II.2.2 Apprentissage

L'apprentissage automatique fait référence au développement, à l'analyse et à l'implémentation de méthodes qui permettent à une machine d'évoluer grâce à un processus d'apprentissage, et ainsi de remplir des tâches qu'il est difficile ou impossible de remplir par des moyens algorithmiques plus classiques. L'objectif est d'extraire et d'exploiter automatiquement l'information présentée dans un jeu de données. On peut schématiser le processus d'apprentissage comme par la figure II.1, l'environnement agit sur le module d'apprentissage de l'agent. Cela a pour effet de consulter et de modifier la base de connaissances pour arriver à une exécution adéquate. De cette exécution, un retour est attendu pour évaluer le résultat obtenu.

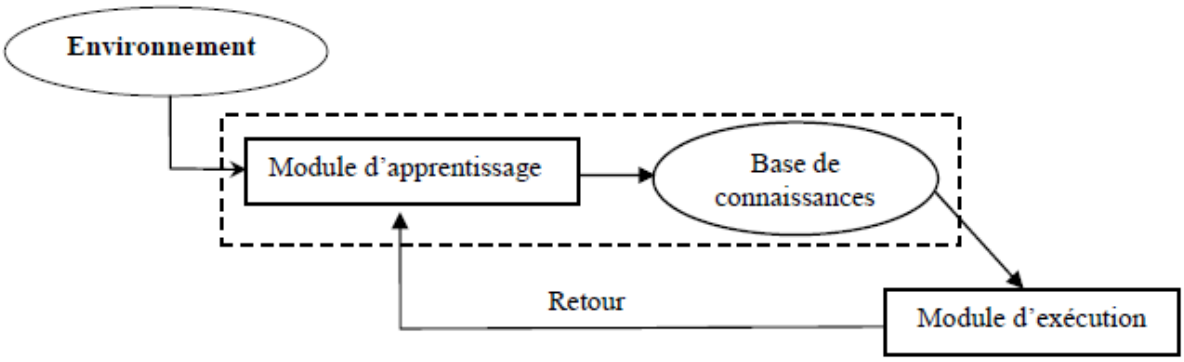


Figure II.1 : Représentation de l'apprentissage automatique [64].

Les algorithmes d'apprentissage peuvent se catégoriser selon le type d'apprentissage, qu'ils emploient [65] :

- **Apprentissage supervisé :** le contrôleur (ou le maître) fournit l'action qui doit être exécutée. L'utilisation nécessite un expert capable de fournir un ensemble d'exemples formés de situations et d'actions correctes associées (figure II.2) [65]. Ces exemples doivent être représentatifs de la tâche à accomplir.

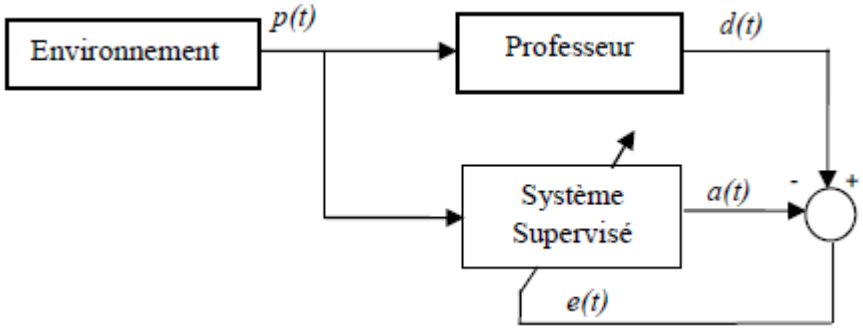


Figure II.2 Représentation de l'apprentissage supervisé.

- **Apprentissage non supervisé :** l'apprenant doit identifier par lui-même la meilleure réponse possible, il n'y a pas de réponse désirée. La tâche peut être par exemple de créer des regroupements de données selon des propriétés communes (catégorisation) [66].

- **Apprentissage par renforcement (AR) :** le contrôleur a un rôle d'évaluateur et non pas d'instructeur. L'information disponible est un signal de renforcement ; généralement appelé critique (figure II.3). Son rôle est de fournir une mesure indiquant si l'action générée est appropriée ou non. Le contrôleur doit déterminer et modifier ses actions de manière à obtenir une meilleure évaluation dans le futur [61].

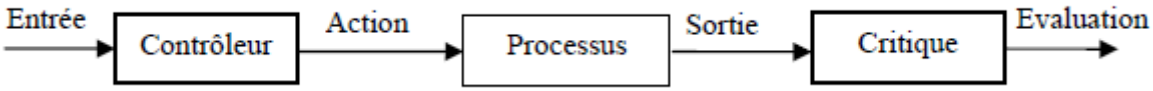


Figure II.3 Représentation de l'apprentissage par renforcement.

II.3 Principe de l'apprentissage par renforcement

L'apprentissage par renforcement concerne une famille de problèmes dans lesquels un agent évolue en analysant les conséquences de ses actions, grâce à un simple signal scalaire (le renforcement) émis par l'environnement.

Cette définition générale met en évidence deux caractéristiques importantes :

- L'agent interagit avec son environnement et la duo « agent + environnement » constitue un système dynamique ;
- Le signal de renforcement, qui est généralement perçu en termes de récompense ou de sanction, permet à l'agent de modifier son comportement.

Dans l'apprentissage supervisé, appelé également "apprentissage avec maître", le système d'apprentissage connaît l'erreur qu'il commet à tout moment : Pour chaque vecteur d'entrée, la sortie souhaitée correspondante est connue. Cette différence entre la sortie réelle et la sortie de référence peut être utilisée pour modifier les paramètres.

Dans l'apprentissage par renforcement (ou avec un critique), le signal reçu est la sanction (positive, négative ou neutre) d'un comportement : ce signal indique l'action à entreprendre sans préciser comment le faire. L'agent utilise ce signal pour déterminer une politique permettant d'atteindre un objectif à long terme.

Une autre différence entre ces deux approches est que l'apprentissage par renforcement est fondamentalement en ligne, car les actions de l'agent modifient l'environnement i-e pour accomplir sa tâche, l'agent doit associer plusieurs actions en suivant une politique afin de maximiser les récompenses futures.

L'apprentissage par renforcement est une approche de l'intelligence artificielle qui permet l'apprentissage d'un agent par l'interaction avec son environnement ; afin de trouver, par un processus essais-erreurs, l'action optimale à effectuer pour chacune des situations que l'agent va percevoir pour maximiser ses récompenses [61][67]. C'est une méthode de programmation ne nécessitant que de spécifier les moments pour lesquels on doit punir ou récompenser l'agent. Il n'est nul besoin de lui indiquer quoi faire dans telle ou telle situation, l'agent se charge d'apprendre par lui-même en renforçant les actions qui s'avèrent les

meilleures [68]. On considère un agent situé dans un environnement, avec lequel il peut interagir. L'agent à l'instant t perçoit d'une part cet environnement par la perception S_t, et peut agir en exécutant l'action a_t, et d'autre part; reçoit une récompense r_t. L'interaction de l'agent avec son environnement est représentée sur la figure II.4, où S_{t+1} *et* r_{t+1} sont la situation et la récompense relative à l'instant $t+1$. L'agent ne connaît la situation dans laquelle il se trouve que par l'intermédiaire de l'historique de ses perceptions. Son but dans le cadre de l'apprentissage par renforcement est de trouver le comportement le plus efficace dans une application donnée, c'est à dire savoir, dans chaque situation perçue, quelle action à accomplir pour maximiser l'espérance des récompenses ? [61][62][69].

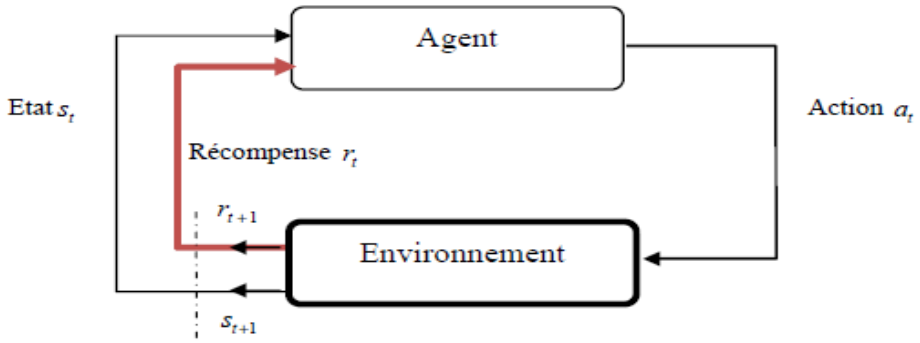


Figure II.4 Système d'apprentissage par renforcement.

Le processus général, est le suivant :

1- Au pas de temps t, l'agent est dans l'état S_t,

2- L'agent choisit l'une des actions possibles dans cet état a_t, ,

3- Il applique l'action, ce qui provoque :

- le passage à un nouvel état, S_{t+1},

- la réception de renforcement, r_t ;

4- $t \leftarrow t+1$,

5- Aller à 2 ou arrêter si le nouvel état est le point d'arrivé.

D'une manière générale, la formalisation du modèle standard d'apprentissage par renforcement tel que nous l'avons décrit précédemment est abordée en considérant la tâche comme un processus de décision Markovien.

II.4 Processus de Décisions Markoviens (PDM)

Un processus de décisions Markovien (PDM) [70] [71] est un système dynamique à temps discret où le temps est représenté par une séquence de pas de temps $t = 1, 2...$, et qui est entièrement défini par :

Un ensemble fini d'états, $S = \left\{ s^1, s^2, ..., s^{|s|} \right\}$, où $|s|$ représente le cardinal de l'ensemble d'états S.

- Un ensemble d'actions disponibles dans chaque état, U_{s_t} ; par soucis de clarté, on ne considérera dans la suite de ce chapitre qu'un unique ensemble d'actions disponibles U, i.e. les actions disponibles sont identiques pour tous les états.

- Une fonction donnant les renforcements primaires espérés relativement aux états et aux actions, $R : S \times U \to R$.

- Des probabilités de transitions d'états relatives aux états et aux actions, $P : S \times U \times S \to [0.1]$; ces probabilités sont indépendantes des états et des actions passées. Le système dynamique est donc un processus sans mémoire (propriété de Markov).

À chaque pas de temps t l'apprenti observe l'état courant S_t et sélectionne une action U_t parmi l'ensemble des actions disponibles U. Lorsque l'action choisie, U_t, est appliquée dans l'état S_t, le nouvel état du système au pas de temps suivant est S_{t+1} avec la probabilité de transition $P_{s_t s_{t+1}}(U_t)$. De plus, suite à cette transition, le système émet un signal de renforcement r_{t+1} de manière aléatoire, $R(S_t, U_t) = E(r_{t+1})$ représente la valeur espérée de ce renforcement primaire.

Le choix des actions dans chaque état, i.e. le comportement C, est déterminé par la politique de l'apprenti. Une politique est donc une fonction associant à chaque état une action (nous ne considérons ici que des politiques stationnaires, i.e. n'évoluant pas en fonction de l'historique du système). Elles sont notées $\pi = (\pi(s^1), ..., \pi(s^{|s|}))$.

À chaque politique π est associée une fonction d'évaluation, V^π, qualifiant le comportement de l'apprenti en termes de renforcements. Dans le cas d'horizons finis, i.e. avec une séquence de pas de temps de longueur déterminée, cette fonction d'évaluation peut représenter la « somme des renforcements » reçus durant la séquence. Toutefois, la connaissance de la longueur de la séquence représente une très forte contrainte, raison pour laquelle les horizons infinis sont plus usités. Deux types de fonctions d'évaluation sont alors possibles. Les fonctions d'évaluation calculant la « moyenne des renforcements » par pas de temps, et celles représentant la « somme escomptée des renforcements ». Le principal défaut des fonctions d'évaluation reposant sur la moyenne réside dans la perte de la notion temporelle. Il est en effet impossible de différencier les politiques recevant beaucoup de récompenses dans les phases initiales de celles en recevant dans les phases finales. Cette distinction étant possible lors de l'utilisation de la somme escomptée, c'est naturellement ce modèle qui est le plus utilisé. Les fonctions d'évaluation sont alors définies pour chaque état $s \in S$ par :

$$V^\pi = E_\pi \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, \pi(s_t)) \middle| s_0 = s \right] \qquad \text{II.1}$$

Où E_π est la valeur espérée en posant l'hypothèse que la politique π est toujours utilisée et que s est l'état initial. Le facteur d'escompte $\gamma \in [0,1]$ est utilisé d'une part pour

que la somme ait une valeur finie, mais également pour pondérer les renforcements relativement au temps.

L'objectif d'un PDM tel que celui que nous l'avons décrit est de trouver une politique optimisant la fonction d'évaluation décrite dans l'équation (II.1), pour tous les états $s \in S$. Une telle politique est appelée politique optimale et notée π^*. Une politique n'est optimale que lorsque aucune autre politique lui est supérieure pour au moins un état. En outre, bien que la fonction d'évaluation optimale V^π soit unique, elle peut toutefois correspondre à plusieurs politiques optimales.

Le facteur d'escompte γ, utilisé dans l'équation (II.1), pondère les renforcements considérés dans la fonction d'évaluation en fonction du temps. Il détermine ainsi l'influence des renforcements futurs espérés dans les politiques optimales. Lorsque $\gamma = 0$, la politique optimale consiste à choisir les actions optimisant le renforcement primaire, i.e. le renforcement directement associé à la transition. Au fur et à mesure que γ tend vers 1, les renforcements futurs (i.e. les conséquences des actions à long terme) deviennent de plus en plus significatifs dans la détermination des actions optimales.

Dans le cas non escompté où $\gamma = 1$, il est impératif, pour que la fonction d'évaluation ait une valeur finie, que le PDM possède un ensemble d'états absorbants. Un ensemble d'état est absorbant lorsqu'il est impossible d'en sortir et que le renforcement émis après chaque application d'action est nul. Lorsque le renforcement associé à toutes les autres transitions (i.e. hors ensemble d'états absorbants) est identique et négatif, le problème est alors un problème de plus court chemin stochastique [61].

II.5 Programmation dynamique (PD)

La programmation dynamique [71] est un ensemble de méthodes permettant de calculer une politique optimale dans un PDM connu, en utilisant les propriétés des équations de Bellman, pour lesquelles le modèle de l'environnement est connu. L'objectif est d'estimer la fonction d'évaluation optimale V^π afin d'en déduire une politique optimale π. On trouve deux méthodes : *Value Iteration* (VI) et *Policy Iteration* (PI). Cette dernière a deux fonctions, qui sont l'évaluation et l'amélioration de la politique. Les valeurs de chaque état sont les entrées du processus d'amélioration de politique. Le but de l'amélioration de la politique est d'ajuster la politique selon les nouvelles valeurs d'état. Le résultat de cette amélioration est une nouvelle politique optimale [68] [69].

Si l'agent ne dispose pas d'un modèle de la dynamique de son environnement, il doit effectuer son apprentissage de la politique optimale en interagissant avec son environnement. À partir des conséquences de ses interactions, il déduira une estimation de la fonction valeur (ou de la fonction qualité) qu'il raffinera petit à petit, d'où il déduira une politique. C'est un apprentissage (en-ligne) alors que les algorithmes de la programmation dynamique que l'on a vus précédemment font un apprentissage (hors-ligne) sans interagir avec leur environnement.

Avant de décrire ces méthodes classiques de PD, il nous faut spécifier le mode de calcul de la fonction d'évaluation pour une politique donnée.

II.5.1 Calcul de la fonction d'évaluation d'une politique donnée

D'après l'équation (II.1), l'évaluation de l'état s pour la politique π, donnée par $V^{\pi}(s)$ représente le retour espéré (en termes de renforcement) si s est l'état initial et si π est utilisée sur un horizon infini. Une idée maîtresse de la PD consiste à approximer cette valeur par une évaluation sur un horizon fini de n pas de temps, l'évaluation n pas de l'état s, notée $V_n^{\pi}(s)$ représente la valeur espérée de la somme $(r_1 + \gamma r_2 + \gamma^{n-1} r_n)$. Lorsque $n \to \infty$, l'approximation devient de plus en plus précise. La méthode repose donc sur des approximations successives, partant de $n = 1$ et faisant croître l'horizon progressivement.

Cette approximation de l'état s, notée $V_1^{\pi}(s)$ est simplement égale à la valeur du renforcement primaire espéré pour l'application de l'action $\pi(s)$:

$$V_1^{\pi}(s) = R(s, \pi(s)),$$

L'application de cette équation à tous les états $s \in S$ détermine entièrement la fonction d'évaluation de la politique π. Le calcul de V_2^{π} peut alors être réalisé par :

$$V_2^{\pi}(s) = R(s, \pi(s)) + , \gamma \sum_{s' \in S} P_{ss'}(\pi(s)) V_1^{\pi}(s'), \forall s \in S,$$

i.e. le renforcement primaire espéré pour l'application de $\pi(s)$, plus la somme escomptée des évaluations des états successeurs de s lorsque l'action appliquée est pondérée par les probabilités de transitions associées. En généralisant cette équation, on obtient alors une des équations constituant la base de la PD :

$$V_n^{\pi}(s) = R(s, \pi(s)) + \gamma \sum_{s' \in S} P_{ss'}(\pi(s)) V_{n-1}^{\pi}(s'), \forall s \in S, \qquad \text{II.2}$$

À partir de l'équation II.2 on peut déduire, dans le cas des horizons infinis, que la fonction d'évaluation d'une politique est l'unique fonction satisfaisant l'équation de Bellman :

$$V^{\pi}(s) = R(s, \pi(s)) + \gamma \sum_{s' \in S} P_{ss'}(\pi(s)) V^{\pi}(s'), \forall s \in S. \qquad \text{II.3}$$

Cette équation définit en fait un système d'équations linéaires (une pour chaque état) qu'il est possible de résoudre itérativement par l'application successive de l'équation (II.2).

II.5.2 Value Itération (VI)

Les algorithmes de la VI déterminent une politique optimale en approximant la fonction d'évaluation optimale V^*. Il est en effet aisé de constituer une politique optimale quand cette fonction d'évaluation est connue [61] [62]. Lorsque le système se trouve dans l'état s l'action optimale U_s^* est celle satisfaisant :

$$U_s^* = \arg \max_{u \in U} (R(s,u) + \gamma \sum_{s' \in S} P_{ss'}(u) V^*(u')) \qquad \text{II.4}$$

C'est donc l'action qui maximise la somme du renforcement primaire espéré et de la valeur espérée (escomptée par γ) de tous les états futurs possibles, en supposant qu'une politique optimale sera utilisée par la suite.

L'approximation de la fonction d'évaluation optimale s'effectue par approximations successives similaires à celles décrites dans le paragraphe précédent. Cependant, il est nécessaire d'y intégrer le caractère d'optimalité de la fonction d'évaluation. L'approximation est alors définie par :

$$V_1^*(s) = \max_{u \in U} R(s,u), \ \forall s \in S,$$

L'équation générale devenant donc :

$$V_n^{\pi}(s) = \max_{u \in U}(R(s, \pi(s)) + \gamma \sum_{s' \in S} P_{ss'}(\pi(s)) V_{n-1}^{\pi}(s')), \forall s \in S, \qquad \text{II.5}$$

La fonction d'évaluation optimale est alors définie comme étant l'unique fonction satisfaisant l'équation d'optimalité de Bellman :

$$V^{\pi}(s) = \max_{u \in U}(R(s, \pi(s)) + \gamma \sum_{s' \in S} P_{ss'}(\pi(s)) V^{\pi}(s')), \forall s \in S. \qquad \text{II.6}$$

C'est ce processus de résolution itérative, réalisé par l'application successive de l'équation (II.5), qui explique le nom de VI attribué à ces méthodes. L'initialisation de $V^*(s)$ peut se faire avec des valeurs arbitraires, i.e. pas obligatoirement avec V_1^* [61].

II.5.3 Policy Iteration (PI)

Dans la seconde catégorie de méthodes, baptisée PI, la recherche d'une politique optimale est basée non plus sur l'approximation de la fonction d'évaluation optimale mais sur la génération de séquences de politiques. Dans cette génération, chaque politique nouvellement crée est supérieure à la politique précédente. Si tel n'est pas le cas, les politiques sont toutes deux optimales.

Après avoir choisi arbitrairement une politique initiale π_0 et calculé sa fonction d'évaluation V^{π_0} (c.à.d. par application successive de l'équation II.2), il faut définir une politique π_1 lui étant supérieure. Cela peut être simplement réalisé en affectant à π_1 les actions satisfaisant :

$$\pi_1(s) = \arg \max_{u \in U}(R(s,u) + \gamma \sum_{s' \in S} P_{ss'}(s) V^{\pi_0}(s')), \forall s \in S$$

La politique π_1 est alors soit supérieure à π_0 soit elles sont toutes les deux optimales. Si π_1 est supérieure, il est alors nécessaire de calculer sa fonction d'évaluation et de renouveler le processus avec une politique π_2

Du fait du calcul de la fonction d'évaluation de la politique courante à chaque itération, la PI à une complexité plus élevée par itération que la VI, mais d'un autre côté, elle nécessite en pratique moins d'itérations pour converger. Cependant, la théorie de la PI n'étant, pour le moment, pas aussi avancée que celle associée à la VI, il est impossible de spécifier quel type de méthodes est préférable [72].

II.6 Différences temporelles (TD)

Si l'on devait identifier une idée centrale et nouvelle de l'AR, c'est sans doute la méthode des Différences Temporelles (DT). L'apprentissage TD apprend à partir de l'expérience directe de l'agent, sans modèle de l'environnement d'une part. et d'autre part, TD met à jour les états en fonction de leurs voisins, sans dépendre des états finaux. Comme dans les paragraphes précédents, nous commençons par le cas de l'évaluation d'une politique, ce que nous appelons la prédiction TD, puis nous décrirons deux méthodes de contrôle, acteur critique learning (ACL) et Q-learning, recherchant à optimiser des fonctions de valeurs d'action.

L'évaluation de politique va donc se faire à partir d'expériences comme en utilisant une moyenne pour estimer la valeur d'un état. Cette moyenne mise sous forme itérative conduit à l'équation suivante :

$$V(s_t) \leftarrow V(s_t) + \alpha[R_t - V(s_t)] \qquad \text{II.7}$$

avec

$$R_t = r_{t+1} + \gamma r_{t+2} .... + \gamma^p r_{t+p+2}$$

L'idée dans cette équation est d'utiliser l'estimation du revenu à partir de la récompense suivante et de la valeur de l'état suivant, au lieu d'utiliser les récompenses de la suite de l'expérience. La méthode des différences temporelles utilise donc la mise à jour suivante :

$$V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)] \qquad \text{II.8}$$

Cette mise à jour se fait naturellement de manière incrémentale, au fur et à mesure des expériences de l'apprenti.

L'équivalent de cette mise à jour pour la fonction Q est donné par :

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1},, a_{t+1}) - Q(s_t,, a_t)] \qquad \text{II.9}$$

La méthode la plus importante de l'apprentissage par renforcement est probablement le Q-Learning [61] qui utilise le maximum de la fonction sur les actions suivantes au lieu de l'action effectivement réalisée :

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1},, a_{t+1}) - Q(s_t,, a_t)] \qquad \text{II.10}$$

Cet algorithme fait converger Q vers Q^* indépendamment de la politique suivie, tant que cette politique garantit une exploration exhaustive (*c-à-d* une probabilité non nulle pour toutes les actions dans tous les états).

Comme pour les autres méthodes, la politique optimale se déduit simplement de la fonction valeur optimale.

II.7 Les systèmes d'inférence floue et L'AR

En robotique mobile, les espaces d'entrée et de sortie des capteurs et des actionneurs sont rarement discrets, ou, s'ils le sont, le nombre d'état est très grand. Or l'apprentissage par renforcement tel que nous l'avons décrit utilise des espaces d'états et d'actions discrets qui doivent être de taille raisonnable pour permettre aux algorithmes de converger en un temps réalisable en pratique.

Au cours des dernières années, les caractéristiques offertes par la logique floue sont de plus en plus utilisées dans beaucoup de domaines d'applications, comme la commande de systèmes et particulièrement le contrôle des robots mobiles.

L'idée principale des algorithmes basés sur la logique floue, appelés généralement Systèmes d'Inférence Floue (SIF), est d'imiter le processus du raisonnement humain pour commander des systèmes mal définis ou difficilement modélisables. Les systèmes d'inférence floue permettent d'exprimer la connaissance humaine sous forme de règles linguistiques de type « Si *Prémisses* Alors *Conclusions* ». La partie prémisse « Si... » est la description de l'état du système qui doit déclencher la règle et les conclusions « Alors... » sont les actions qui doivent être effectuées sur le système.

Afin de profiter des nombreux développements théoriques associés aux méthodes d'apprentissage par renforcement précédemment évoqués, il est nécessaire d'y apporter les modifications pour permettre le traitement d'apprentis de type SIF. Il faut en effet les adapter de telle manière qu'elles prennent en considération les caractéristiques spécifiques des SIF, avec notamment l'utilisation d'états et d'actions continus

L'utilisation des systèmes d'inférence floue apporte plusieurs avantages [72] :

- La possibilité de représenter la valeur d'une politique, en raison des propriétés d'approximation universelle de la plupart des systèmes d'inférence floue ;

- La possibilité de traiter simplement des espaces d'état et d'action continus ;

- L'intégration de connaissances *a priori* et l'interprétation des connaissances acquises à l'issue de l'apprentissage.

Avant de présenter les deux méthodes d'apprentissage utilisées et implémentées, nous consacrons le paragraphe suivant à une description plus précise de l'apprenti.

II.7.1 Choix de l'apprenti

De par la simplicité des SIF de type *Takagi-Sugeno* d'ordre zéro, *i.e.* absence d'étape de défuzzification, leur propriété d'approximateur universel, et de par leur large application dans la commande de systèmes réels c'est naturellement ce type de SIF que nous avons choisi pour implémenter l'apprenti.

De plus, pour favoriser l'interprétation et la compréhension des différentes approches, nous allons de plus, imposer les conditions suivantes :

- La structure est fixée *a priori* par l'utilisateur, qui doit indiquer les sous-ensembles flous qu'il veut manipuler sur chaque entrée ;

- Les partitions floues sur chaque domaine d'entrée sont des partitions floues fortes triangulaires ;
- La T-norme utilisée pour modéliser le *Et* dans la prémisse des règles, est le produit

Enfin pour les besoins des méthodes d'apprentissage, l'apprenti possède une sortie supplémentaire pour l'algorithme FACL : celle du *critique*. Elle est indépendante de la sortie du contrôleur et ne participe pas dans le contrôle du système.

Le SIF est alors constitué de N règles de la forme :

$$R_i : si\ S_1\ est\ L_1^i\ et.....et\ S_{N_1}\ est\ L_{N_1}^i\ alors\ Y_1\ est\ O_1^i\ et\ Y_2\ est\ O_2^i$$

Comme, indiqué, les fonctions d'appartenance des variables d'entrée (la perception de l'apprenti de son environnement) sont définies par extraction naturelle de connaissance. Quant au nombre de règles, il est automatiquement déterminé par le nombre de termes linguistiques de chaque variable. Les caractéristiques modifiables de l'apprenti sont donc uniquement constituées des *paramètres concernant les conclusions* O_j^i

Les deux méthodes d'apprentissage que nous développerons par la suite consistent à choisir la conclusion O_j^i (*i.e.* la conclusion concernant la commande) de chaque règle R_i parmi un ensemble de conclusions disponibles pour cette même règle et ce uniquement en fonction des renforcements primaires reçus en cours d'interaction avec l'environnement. Ces renforcements pouvant être retardés et rares, le choix des conclusions est en fait guidé par la seconde sortie de l'apprenti, Y_2 *le* critique interne pour le cas de l'algorithme FACL. Il représente la fonction d'évaluation des états d'approximation des renforcements que l'apprenti espère obtenir dans le futur, et constitue de ce fait un critère plus riche que les renforcements primaires pour guider le choix des actions.

II.7.2 Les méthodes d'apprentissage par renforcement appliquées

Les deux algorithmes que nous proposons, le *Fuzzy Actor-Critic Learning* (FACL) et le *Fuzzy Q-Learning* (FQL), sont assimilées à des méthodes de programmation dynamique adaptative directes basées sur un apprenti de type SIF. La perception des états est de ce fait continue. Outre cet avantage de traiter les états continus, ces méthodes offrent la possibilité de travailler avec des actions discrètes ou continues, ce qui est rarement le cas dans les méthodes utilisant des approximateurs de fonctions globaux.

II.7.2.1 L'algorithme Fuzzy Actor Critic Learning (FACL)

Les deux algorithmes FACL et FQL permettent l'apprentissage en ligne des deux sorties de l'apprenti. Dans le cas du FACL auquel nous nous intéressons dans un premier temps, la première sortie Y_1, indiquant la commande à appliquer sur le robot (vitesse de rotation et vitesse de translation), représente *l'Acteur.* Quant à la seconde sortie, Y_2, elle a pour rôle l'évaluation des états et constitue le *Critique***.** L'algorithme FACL met alors en jeu deux types d'apprentissage, les différences temporelles pour l'apprentissage des prédictions de renforcements du critique, et une méthode de comparaison de renforcements pour l'apprentissage de l'acteur [73].

II.7.2.2 Structure de l'apprenti

À la différence du critique dont la valeur est directement inférée à partir d'un vecteur de conclusions v . L'inférence de l'action est indirecte. En théorie l'acteur doit en effet choisir une action parmi un vocabulaire limité. Il implémente alors un processus de compétition entre actions.

La gestion de ce processus est classiquement réalisée en associant à chaque action une qualité dépendant de l'état. Cette qualité détermine alors la probabilité du choix de l'action, cette action étant obligatoirement discrète. Pour permettre également la gestion des actions continues nous proposons de ne plus considérer la qualité des actions en fonction des états mais en fonction des règles. Pour ce faire, les actions correspondant à chaque règle R_i possèdent une qualité q^i . À partir de cette qualité, l'acteur choisit, pour chaque règle activée (i.e. celles décrivant l'état courant), une action parmi l'ensemble des actions discrètes disponibles dans ces règles. L'action finale discrète ou continue est alors le résultat de l'inférence obtenue avec les diverses actions localement élues.

En résumé, chaque règle Ri de l'apprenti possède donc :

- Une conclusion v_i utilisée pour l'approximation de la fonction d'évaluation de la politique courante V^{π_t} ,
- Un ensemble d'actions discrètes U_i ,
- Un vecteur de paramètres q^i indiquant la qualité des différentes actions discrètes disponibles et intervenant dans la définition de la politique courante.

II.7.2.3 Le critique

Le rôle du critique est de fournir à l'acteur, à chaque pas de temps, un critère plus riche pour le choix des actions que celui représenté par les renforcements primaires. La fonction d'évaluation constitue un très bon critère. En effet, elle prend en compte non seulement les renforcements immédiats mais également ceux devant être reçus dans le futur.

Au pas de temps t, la valeur de notre critique pour l'état S_t est inférée à partir du vecteur de conclusions v_t de la manière suivante :

$$V_t(S_t) = \sum_{R_i \in A_t} v_t^i . \alpha_{R_i}(S_t) = v_t . \phi_t^T \qquad \text{II.11}$$

où ϕ_t^T représente la perception continue de l'apprenti au pas de temps t (*i.e.* contient la valeur de vérité de toutes les règles pour l'état S_t) et A_t l'ensemble des règles activées dans l'état S_t . En se basant sur la condition que doit satisfaire le critique pour approximer la fonction d'évaluation de la politique courante et en utilisant uniquement les valeurs disponibles au pas de temps t+1, l'erreur d'approximation est estimée par :

$$\tilde{\varepsilon}_{t+1} = r_{t+1} + \gamma . V_t(S_{t+1}) - V_t(S_t), \qquad \text{II.12}$$

Cette erreur correspond exactement à l'erreur des différences temporelles (TD), lorsque l'apprenti désire prédire la somme *escomptée* des renforcements primaires :

$$V_t(S_t) = \sum_{k=0}^{\infty} \gamma^k . r_{t+k+1} \qquad \text{II.13}$$

La règle d'apprentissage du critique utilise alors cette erreur TD pour mettre au point le vecteur de conclusions v par une descente de gradient stochastique classique :

$$\begin{aligned} v_{t+1} &= v_t + \beta . \tilde{\varepsilon}_{t+1} . \nabla_v V_t(S_t), \\ &= v_t + \beta . \tilde{\varepsilon}_{t+1} . \phi_t \end{aligned} \qquad \text{II.14}$$

où β représente le taux d'apprentissage du critique et $\nabla_v V_t(S_t)$ le vecteur de dérivées partielles de $V_t(S_t)$ par rapport à chaque composante du vecteur de conclusions v_t. Dans le cas de notre apprenti, ce vecteur est simplement égal aux valeurs de vérités des règles.

II.7.2.4 L'acteur

Alors que le critique approxime la fonction d'évaluation de la politique courante, l'acteur a pour rôle l'amélioration de cette politique afin d'obtenir une politique optimale.

Pour définir la politique courante, nous proposons d'élaborer des politiques locales aux règles en utilisant un vecteur de paramètres q^i pour chaque règle R_i. Ce vecteur associe à chaque action du vocabulaire limitée de la règle une qualité par rapport à la tâche à réaliser.

Les actions locales sont alors *élues* pour chaque règle activée, sur la base de ces qualités, mais également en fonction d'une stratégie d'exploration que nous verrons plus loin. L'action globale pour l'état S_t est alors définie par l'inférence de ces actions localement élues :

$$\begin{aligned} U_t(S_t) &= \sum_{R_i \in A_t} Election_{U_i}(q_t^i) . \alpha_{R_i}(S_t) \\ &= Election(q_t) . \phi_t^T \end{aligned} \qquad \text{II.15}$$

Où Election est une fonction retournant l'action élue pour chaque règle.

L'apprentissage de la politique optimale consiste donc à ajuster le vecteur de paramètres q de façon à ce que la politique induite soit améliorée.

De nouveau, le critique fournit la mesure d'utilité de l'action *t-optimale*. En effet, si le critique constitue une bonne approximation de la fonction d'évaluation de la politique *t-optimale*, une erreur TD positive implique que l'action venant d'être appliquée est préférable à l'action *t-optimale*. Il faut donc dans ce cas augmenter la qualité de l'action appliquée pour l'état dans lequel elle l'a été. Réciproquement, si l'erreur TD est négative, il est nécessaire de baisser la qualité de l'action appliquée car elle a conduit le système dans un état dont l'évaluation est inférieure à celle attendue si l'action *t-optimale* avait été appliquée.

L'action appliquée étant inférée à partir des diverses actions locales élues, nous proposons de répercuter l'erreur TD sur chacune de ces actions locales. On obtient alors une règle d'apprentissage similaire à celle du critique (équation II.14), l'unique différence étant que la règle d'apprentissage de l'acteur dépend des actions :

$$q_{t+1}^{i}(U_t^i) = q_t^i(U_t^i) + \tilde{\varepsilon}_{t+1}.\alpha_{R_i}(S_t), \forall R_i \in A_t \qquad \text{II.16}$$

Les paramètres q servent uniquement au processus de compétition et non à l'approximation d'une fonction, ainsi il n'est pas nécessaire de recourir à un taux d'apprentissage.

II.7.2.5 Méthode d'exploration proposée

Afin de découvrir les actions dignes d'intérêt il est nécessaire de mettre en place un processus d'exploration dans la fonction d'élection afin que la politique ne soit pas une politique gloutonne, mais essaie de temps en temps des actions autres que l'action *t-optimale*. Le problème est alors double [73] :

- Le choix des actions à tester, *i.e.* éviter de tester des actions provoquant l'émission de punitions,
- Trouver un compromis entre l'exploration pure et l'exploration pure (politique gloutonne).

On recense deux types de techniques d'exploration :

- *L'exploration non dirigée* : parmi les méthodes existantes, la méthode la plus populaire est la distribution de Boltzmann

$$P(U) = \frac{e^{q(U)/\theta}}{\sum_{U' \in U} e^{q(U')/\theta}} \qquad \text{II.17}$$

où θ, *la température*, détermine l'importance des facteurs aléatoires et peut être réduit en cours d'apprentissage pour réduire l'exploration [63] [72] ;

- *L'exploration dirigée* utilise la connaissance issue de l'exploration afin de choisir les actions dont le gain de connaissance espéré est maximal [71][72].

La stratégie d'élection des actions que nous avons élaborée est une combinaison d'exploration dirigée et non dirigée. Le choix des actions est en effet guidé par un terme d'exploitation représenté par la qualité de l'action dans la règle, cas continu, ou dans l'état (cas discret) et par un terme d'exploration constitué d'une partie aléatoire et d'une partie basée sur un compteur.

Alors que dans le cas continu, la qualité des actions pour chaque règle est directement disponible, il est nécessaire d'inférer la qualité des actions discrètes pour l'état courant :

$$q_t^{S_t}(U) = \sum_{R_i \in A_t} q_t^i(U).\alpha_{R_i}(S_t), \forall U \in U,$$
$$= q_t(U).\phi_t^T, \forall U \in U, \quad \text{II.18}$$

où U représente l'ensemble des actions discrètes disponibles pour la tâche.

La fonction globale d'élection, s'appliquant soit au niveau des règles (cas continu) soit au niveau des états (cas discret), est alors définie par :

$$Election_U(q) = ArgMax_{U \in U}(q(U) + \eta(U) + \rho(U)), \quad \text{II.19}$$

q est le vecteur de qualité associé, $\eta(U)$ est le terme d'exploration non dirigée et $\rho(U)$ est le terme d'exploration dirigée.

- Le terme d'exploration non dirigée η est en fait un vecteur de valeurs aléatoires.

Il correspond à un vecteur ψ de valeurs tirées selon une loi exponentielle, normalisé afin de prendre en compte l'échelle de grandeur des qualités q :

$$s_f(U) = \begin{cases} 1 \quad si \quad Max(q(U)) = Min(q(U)) \\ \dfrac{s_p(Max(q(U)) - Min(q(U)))}{Max(\psi)}, \quad \text{sinon}, \end{cases} \quad \text{II.20}$$
$$\eta(U) = s_f(U).\psi,$$

où s_p représente la taille maximale du bruit relativement à l'amplitude des qualités, s_f étant le facteur de normalisation correspondant. Utilisé seul, ce terme d'exploration permet un choix aléatoire des actions lorsque toutes les qualités sont identiques et permet dans le cas contraire, de tester uniquement les actions dont la qualité satisfait :

$$q(U) \geq q(U^*) - s_p(Max(q(U)) - Min(q(U)))$$

Ce terme permet alors d'éviter le choix des actions néfastes (satisfaction du premier point décrit en début du paragraphe).

- Le terme d'exploration dirigée ρ permet quant à lui de favoriser les actions n'ayant pas été appliquées souvent. Il est donc nécessaire de mémoriser le nombre de fois ou l'action a été élue. Ce terme est défini de la manière suivante :

$$\rho(U) = \frac{\theta}{e^{n_t(U)}} \quad \text{II.21}$$

où θ représente un facteur positif permettant de doser l'exploration dirigée et $n_t(U)$ le nombre d'applications de l'action U au pas de temps t.

La définition de cette dernière valeur dépend du type des actions :

- *Actions continues* : $n_t(U)$ correspond au nombre d'applications de l'action discrète U dans la règle considérée. Soit U_t^i l'action discrète élue au pas de temps t dans la règle R_i ; la mise à jour de cette variable est réalisée par l'équation suivante :

$$n_t(U^i) = n_{t-1}(U^i) + 1, \quad \forall R_i \in A_t, U^i = U_t^i \qquad \text{II.22}$$

- *Actions discrètes* : $n_t(U)$ représente le nombre d'applications de l'action discrète U dans l'état S_t, $n_t^{S_t}(U)$. De la même manière que pour la qualité, ce terme n'est pas directement disponible. Il est nécessaire de l'inférer à partir du nombre d'applications de l'action dans chaque règle décrivant l'état S_t :

$$n_t^{S_t}(U) = n_t(U).\phi_t^T, \forall U \in U.$$

La mise à jour des $n_t(U)$ reste quant à elle identique à celle décrite dans l'équation (II.22) ; les actions localement élues dans les règles activées étant bien évidemment celles correspondant à l'action discrète élue U_t *i.e.* :

$$U_t^i = U_i, \forall R_i \in A_t$$

La gestion de la notion de « *temps en temps* » évoquée dans le début du paragraphe est réalisée dans un premier temps par le biais des paramètres s_p (pour l'exploration non dirigée) et θ pour l'exploration dirigée. Par la suite, l'exploration s'éteint naturellement au fur et à mesure que l'apprentissage avance, suite à l'augmentation des qualités des actions *t-optimale* et du nombre d'application des actions [71] [74].

II.7.2.6 Traces d'éligibilités

Les méthodes utilisant les différences temporelles que nous avons vues précédemment ne prennent en compte que l'étape qui suit l'évolution. De plus seul l'état courant est concerné. Sutton [Sutton 90] a étendu l'évaluation à tous les états, en fonction de leur éligibilité, c'est à dire du degré de visite d'un état dans le passé récent. L'éligibilité (ou la trace d'éligibilité) peut être définie de plusieurs façons : éligibilité accumulatrice ou éligibilité remplaçante (figure II.5).

Les règles d'apprentissage de l'acteur et du critique ne modifient que les paramètres correspondant aux actions et/ou aux règles activées au pas de temps t (équations II.14 et II.16). Les traces d'éligibilité vont permettre de modifier également les paramètres des règles/actions activées au pas de temps précédents.

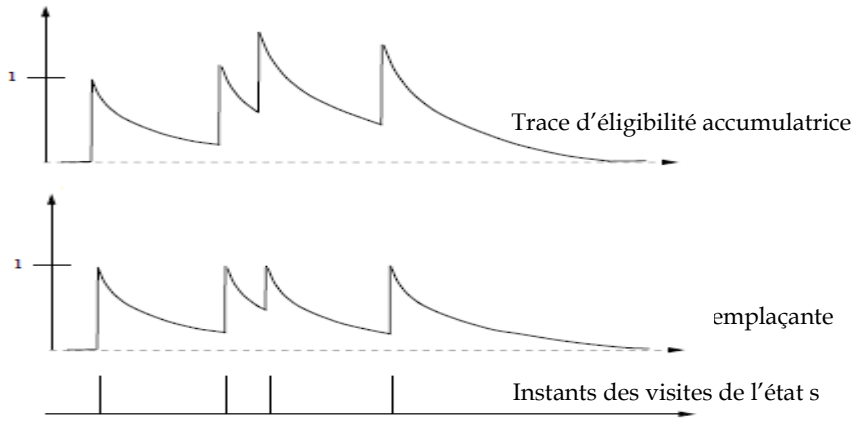


Figure II.5 : Traces d'éligibilité accumulatrice et remplaçante

Ces traces d'éligibilité ont alors pour vocation la mémorisation des règles précédemment activées $(A_k, k = (1,t))$ et des actions locales élues à ces différents pas de temps $(U_k^i, (k = 1,t))$ pondérées par leur proximité du pas de temps t. Ces traces fonctionnent donc comme un processus de mémorisation court terme, activé par l'occurrence de la règle ou du couple règle/action, et qui décroît graduellement au fur et à mesure que le temps passe.

La valeur de la trace indique alors comment la règle ou la règle/action sont *éligibles* pour l'apprentissage. Les valeurs de vérité sont dans la majeure partie des cas inférieures à 1. C'est la raison pour laquelle nous proposons d'utiliser des traces accumulatrices plutôt que des traces remplaçantes. Elles permettent en effet d'obtenir des traces significatives pour les règles et règles/actions activées plusieurs pas de temps consécutifs, comme on peut le déduire aisément de la figure (II.5).

Soit $\overline{\phi}_t$, une telle trace pour le critique au pas de temps t, mémoire court terme des états visités par l'apprenti. Cette mémoire est basée sur la perception de l'apprenti, *i.e.* les valeurs de vérité des différentes règles :

$$\begin{aligned} \overline{\phi}_t &= \sum_{k=0}^{t} (\gamma\lambda)^{t-k} . \phi_k , \\ &= \phi_t + \gamma\lambda \sum_{k=0}^{t-1} (\gamma\lambda)^{t-1-k} . \phi_k , \\ &= \phi_t + \gamma\lambda . \overline{\phi}_{t-1,} \end{aligned} \qquad \text{II.23}$$

Où le *facteur de proximité* λ est utilisé pour réaliser la *fonction d'oubli*.

Une trace équivalente pour l'acteur consiste à mémoriser les actions appliquées en fonction des états. Sa gestion est donc moins directe que celle correspondant au critique car cette mémoire est relative aux parties sensorielles et motrice. En raison du principe adopté pour la génération d'actions (continues ou discrètes), il est nécessaire de tenir compte à la fois des règles activées et des actions localement élues (*i.e.* celles ayant permis la génération de l'action globale). Nous proposons donc d'utiliser une mémoire court terme des valeurs de vérité de chaque règle en fonction de chaque action disponible dans ces règles.

Soit $e_t^i(U^i)$ la valeur de la trace de l'action U^i dans la règle R_i au pas de temps t :

$$e_t^i(U^i) = \begin{cases} \gamma\lambda' e_{t-1}^i(U^i) + \phi_t^i, (U^i = U_t^i), \\ \gamma\lambda' e_{t-1}^i(U^i), \quad \text{sinon} \end{cases} \qquad \text{II.24}$$

où λ' représente le *facteur de proximité* propre aux paramètres de l'acteur.

Les mises à jour des paramètres définies par les équations (II.22) et (II.24) se généralisent donc pour toutes les règles et actions :

$$v_{t+1} = v_t + \beta\tilde{\varepsilon}_{t+1}\bar{\phi}_t, \qquad \text{II.25}$$

$$q_{t+1} = q_t + \tilde{\varepsilon}_{t+1}e_t, \qquad \text{II.26}$$

II.7.2.7 Taux d'apprentissage adaptatifs

Le principe sur lequel la règle d'apprentissage de l'acteur est basée pose l'hypothèse que le critique représente une mesure fiable préconisant l'application de l'action t-optimale. Or, le critique étant mis au point parallèlement, sa valeur en début d'apprentissage risque fort de ne pas être fiable. Il est donc nécessaire que son élaboration soit rapide et surtout exempte de toutes oscillations.

Afin d'accélérer sa vitesse d'apprentissage et éviter d'éventuelles oscillations, nous utilisons les principes mis en évidence dans les travaux de Kesten sur l'approximation stochastique [75] :

Soit X_t l'approximation de θ au pas de temps t. De très fréquentes fluctuations du signe de $(X_t - \theta) - (X_{t-1} - \theta) = X_t - X_{t-1}$) indiquent que l'erreur d'approximation $|X_t - \theta|$ est faible, alors que de rares changements de signe indique une erreur importante.

Ainsi, au pas de temps t, si le nombre de changements de signe avant cet instant est important, le processus correctif doit être faible alors que dans le cas contraire il doit être fort.

Il est alors possible d'en déduire les quatre heuristiques suivantes pour une détermination en ligne du processus correctif [75] :

1. Chaque paramètre caractérisant l'approximateur possède son propre taux d'apprentissage afin de tenir compte des particularités de chaque dimension ;
2. Chaque taux d'apprentissage peut évoluer au cours du temps ;
3. Si la dérivée d'un paramètre a plusieurs fois consécutivement le même signe, son taux d'apprentissage est augmenté ;
4. Si la dérivée d'un paramètre change de signe plusieurs pas de temps consécutifs, son taux d'apprentissage est réduit.

L'implémentation de ces heuristiques permet la mise à jour des taux d'apprentissage relatif à chaque paramètre :

$$\Delta\beta_t^i = \begin{cases} k & si \quad \bar{\delta}_{t-1}^i.\delta_t^i > 0, \\ -\psi\beta_t^i & si \quad \bar{\delta}_{t-1}^i.\delta_t^i < 0, \\ 0 & \text{sinon.} \end{cases} \qquad \text{II.27}$$

Avec :

- β_t^i est le taux d'apprentissage du critique pour la règle R_i au pas de temps t ;

- $\delta_t^i = \tilde{\varepsilon}_{t+1}.\bar{\phi}_t^i$ représente la variation apportée au paramètre du critique v_t^i et dans laquelle nous proposons d'intégrer directement la trace d'éligibilité et non pas simplement la dérivée partielle ;
- $\bar{\delta}_t^i = (1-\varphi)\delta_t^i + \varphi\bar{\delta}_{t-1}^i$ représente la moyenne exponentielle

Cette règle augmente alors les taux d'apprentissage de manière linéaire afin de prévenir qu'ils ne deviennent trop grands trop vite, et les décrémente de manière exponentielle pour assurer qu'ils soient en permanence positifs et qu'ils décroissent rapidement.

La mise à jour des paramètres du critique (équation II.14) devient donc :

$$v_{t+1} = v_t + \tilde{\varepsilon}_{t+1}.\beta_{t+1}.\bar{\phi}_t^T \qquad \text{II.28}$$

II.7.2.8 Mise à jour résiduelle

À l'origine, les méthodes de programmation dynamique adaptative ont été étudiées dans le cas d'un codage des états par des tables. Le fait d'utiliser des approximateurs de fonctions généraux pour la perception des états, comme les SIF, peut éventuellement être source d'instabilité. Afin de pallier à ce problème, Baird a mis au point une nouvelle classe d'algorithmes qu'il a appelée *algorithmes résiduels* [73]. L'idée générale est d'effectuer une descente de gradient non pas sur la résiduelle de Bellman définie par ε dans l'équation (II.22) mais sur *la moyenne quadratique résiduelle de Bellman* :

$$E = \frac{1}{n}\sum_S (R(S,U) + \gamma \sum_{S' \in S} P_{SS'}(U)V(S') - V(S))^2 \qquad \text{II.29}$$

où *n* est le nombre d'états, S' représentant les différents états successeurs possible de S .

La modification à apporter aux paramètres de l'approximateur appelée modification *à gradient résiduel*, est alors définie par [70] :

$$v_{t+1} = v_t + \beta\tilde{\varepsilon}_{t+1}(\nabla_v V_t(S_t) - \gamma\nabla_v V_t(S_{t+1})). \qquad \text{II.30}$$

Alors que les algorithmes basés sur cette équation peuvent être stables mais lents ceux effectuant une descente de gradient sur la résiduelle de Bellman peuvent être rapides mais instables. Cette descente de gradient étant un cas particulier du gradient résiduel

(l'unique différence réside dans la présence ou non du terme $\gamma\nabla_v V_t(S_{t+1})$, équation II.14, il est possible de profiter des avantages des deux approches en utilisant un facteur $\rho \in [0,1]$ pour pondérer l'influence de chacune d'elles [69] :

$$v_{t+1} = v_t + \beta\tilde{\varepsilon}_{t+1}(\nabla_v V_t(S_t) - \gamma\rho\nabla_v V_t(S_{t+1})). \qquad \text{II.31}$$

Du fait de l'utilisation d'une trace d'éligibilité par le critique, il nous est impossible d'utiliser cette équation directement. Nous proposons donc d'intégrer cette variation directement dans la trace d'éligibilité. Cette trace est alors remise à jour par :

$$\bar{\phi}_t \leftarrow \bar{\phi}_t - \gamma\rho\phi_{t+1} \qquad \text{II.32}$$

Pendant le calcul de la fonction d'évaluation de l'état nouvellement atteint S_{t+1}.

II.7.2.9 Description de la procédure d'exécution

Afin de décrire le fonctionnement global de l'algorithme FACL, nous décrivons dans ce paragraphe son exécution sur un pas de temps. Cette exécution peut se diviser en six étapes principales. Soit t +1 le pas de temps courant; l'apprenti a alors appliqué l'action U_t élue au pas de temps précédent et reçu en contrepartie le renforcement primaire r_{t+1} pour la transition de l'état S_t à S_{t+1}. Après le calcul de la perception continue de l'état courant ϕ_{t+1}, *i.e.* le calcul des valeurs de vérité des règles les six étapes sont les suivantes :

1. Calcul de la fonction d'évaluation t-optimale de l'état courant par le critique :

$$V_t(S_{t+1}) = v_t.\phi_{t+1}^T \qquad \text{II.33}$$

   et mise à jour des traces d'éligibilité correspondant aux règles activées afin de tenir compte des modifications résiduelles :

$$\bar{\phi}_t \leftarrow \bar{\phi}_t - \gamma\rho\phi_{t+1} \qquad \text{II.34}$$

2. Calcul de l'erreur TD :

$$\tilde{\varepsilon}_{t+1} = r_{t+1} + \gamma.V_t(S_{t+1}) - V_t(S_t), \qquad \text{II.35}$$

3. Mise à jour des taux d'apprentissage (taux adaptatifs) correspondant aux paramètres du critique pour toutes les règles en trois étapes :

   - $\delta_t^i = \tilde{\varepsilon}_{t+1}.\bar{\phi}_t^i$

$$\beta_{t+1}^i = \begin{cases} \beta_t^i + k & si \quad \bar{\delta}_{t-1}^i . \delta_t^i > 0, \\ \beta_t^i (1-\psi) & si \quad \bar{\delta}_{t-1}^i . \delta_t^i < 0, . \\ \beta_t^i & \text{sin } on. \end{cases} \quad \text{II.36}$$

- $\bar{\delta}_t^i = (1-\varphi)\delta_t^i + \varphi \bar{\delta}_{t-1}^i$

4. Apprentissage du critique et de l'acteur par la mise à jour du vecteur v et de la matrice q :

$$v_{t+1} = v_t + \tilde{\varepsilon}_{t+1} . \beta_{t+1} . \bar{\phi}_t^T \quad \text{II.37}$$

$$q_{t+1} = q_t + \tilde{\varepsilon}_{t+1} e_t \quad \text{II.38}$$

5. Nouveau calcul de la fonction d'évaluation t-optimale de l'état courant par le critique mais cette fois avec les paramètres nouvellement mis à jour :

$$V_{t+1}(S_{t+1}) = v_{t+1} . \phi_{t+1}^T \quad \text{II.39}$$

Cette valeur sera utilisée pour le calcul de l'erreur TD au prochain pas de temps.

6. Election de l'action à appliquer sur l'état S_{t+1} :

- *Actions discrètes* : inférence de la qualité globale de chaque action discrète à partir de leurs qualités locales :

$$q_{t+1}^{S_{t+1}}(U) = q_{t+1}(U) . \phi_{t+1}^T, \forall U \in U, \quad \text{II.40}$$

et élection d'une de ces actions suivant la politique d'exploration/exploitation :

$$Election_U(q_{t+1}^{S_{t+1}}) = ArgMax_{U \in U}(q_{t+1}^{S_{t+1}}(U) + \eta^{S_{t+1}}(U) + \rho^{S_{t+1}}(U)), \quad \text{II.41}$$

- *Actions continues* : pour chaque règle activée élection d'une action locale et l'inférence de l'action continue à appliquer :

$$U_{t+1}(S_{t+1}) = \sum_{R_i \in A_{t+1}} Election_U(q_{t+1}^i) . \alpha_{R_i}(S_{t+1}), \forall U \in U, \quad \text{II.42}$$

où Election est défini par :

$$Election_U(q_{t+1}^i) = ArgMax_{U \in U}(q_{t+1}^i(U) + \eta^i(U) + \rho^i(U)), \quad \text{II.43}$$

Et mise à jour des traces d'éligibilité pour le critique et l'acteur :

$$\bar{\phi}_{t+1} = \phi_t + \gamma\lambda.\bar{\phi}_t, \qquad \text{II.44}$$

$$e^i_{t+1}(U^i) = \begin{cases} \gamma\lambda'.e^i_t(U^i) + \phi^i_{t+1}, (U^i = U^i_{t+1}), \\ \gamma\lambda'.e^i_t(U^i), \quad \text{sin } on \end{cases} \qquad \text{II.45}$$

II.7.3 Fuzzy Q-Learning (FQL)

L'élaboration de cette seconde méthode d'apprentissage découle de l'observation et l'équation de mise à jour de l'acteur dans le FACL est sensiblement identique à celle du critique. En effet, la seule différence réside dans le fait que l'apprentissage de l'acteur dépend de l'action. L'idée consiste alors à réutiliser les mêmes principes que ceux que nous avons proposés dans le FACL, *i.e.* utiliser la matrice q pour implémenter non seulement les politiques locales aux règles, mais également pour représenter la fonction d'évaluation de la politique t-optimale globale. On obtient alors le FQL, adaptation du Q-Learning de Watkins [76] pour un apprenti de type SIF.

Nous considérons que cette version est une version compacte du FACL. Elle permet donc elle aussi le traitement des états et des actions de façon continue. De même que pour le FACL nous proposons d'utiliser des traces d'éligibilité pour mémoriser les couples règles/actions et également des taux d'apprentissage adaptatifs, qui, dès lors que l'apprentissage de la fonction d'évaluation repose sur le vecteur q, sont relatifs aux règles/actions et non plus uniquement aux règles comme dans le cas de l'algorithme FACL.

Cet apprenti, infère la valeur Q à partir de la matrice q présentée dans le FACL (l'acteur) :

$$Q_t(S_t, U_t) = \sum_{R_i \in A_t} q^i_t(U^i_t)\alpha_{R_i}(S_t), \qquad \text{II.46}$$

où U^i_t représente l'action locale élue dans la règle *Ri* au pas de temps *t*, comme dans l'équation (II.35) et U_t l'action globale discrète ou continue.

La démarche est alors identique à celle de FACL et consiste à approximer cette expression uniquement avec les valeurs disponibles au pas de temps t+1 :

$$Q_t(S_t, U_t) = r_{t+1} + \gamma Q^*_t(S_{t+1}), \qquad \text{II.47}$$

L'erreur d'approximation de l'apprenti est alors définie simplement par :

$$\tilde{\varepsilon}_{t+1} = r_{t+1} + \gamma Q^*_t(S_{t+1}) - Q_t(S_t, U_t) \qquad \text{II.48}$$

Où $\tilde{\varepsilon}_{t+1}$ représente l'erreur TD.

La mise à jour des paramètres q modélisant la fonction Q est alors strictement identique à celle définie pour l'acteur (équation II.26), aux taux d'apprentissage prés, car ici ces paramètres ne sont pas uniquement utilisés pour implémenter la compétition entre actions mais approximent également la fonction Q. On obtient alors :

$$q_{t+1}^{i} = q_t^{i} + \vartheta_t^{i} \tilde{\varepsilon}_{t+1} e_t^{i^T} \quad , \forall R_i \qquad \text{II.49}$$

ou e_t^i représente la trace d'éligibilité des différentes actions disponibles dans la règle R_i identique à celle de l'acteur (équation II.45) et ϑ_t^i les taux d'apprentissage adaptatifs de chaque action dans la règle R_i comme ceux utilisés par le critique.

II.7.3.1 Description de la procédure d'exécution

D'une manière analogue à celle utilisée pour la description globale de l'algorithme FACL, nous décrivons ici l'exécution de l'algorithme FQL sur un pas de temps. Elle peut se subdiviser en six étapes. Soit t+1 le pas de temps courant, l'apprenti a appliqué l'action U_t (discrète ou continue) en utilisant strictement la même fonction d'élection que celle définie dans le FACL. Il a alors reçu le renforcement primaire r_{t+1} dû au passage de l'état S_t à l'état S_{t+1}.

Il perçoit alors cet état par le biais des valeurs de vérité de ses règles et effectue les six étapes suivantes :

1. Calcul de la fonction d'évaluation t-optimale de l'état courant :

$$Q_t^*(S_{t+1}) = \sum_{R_i \in A_t} (\underset{U \in U}{Max}(q_t^i(U)) \alpha_{R_i}(S_{t+1}), \qquad \text{II.50}$$

2. Calcul de l'erreur TD :

$$\tilde{\varepsilon}_{t+1} = r_{t+1} + \gamma Q_t^*(S_{t+1}) - Q_t(S_t, U_t) \qquad \text{II.51}$$

3. Mise à jour des taux d'apprentissage pour tous les couples règles actions en trois étapes :

- $\delta_t^i(U) = \tilde{\varepsilon}_{t+1} . e_t^i(U)$
- $$\vartheta_{t+1}^i(U) = \begin{cases} \vartheta_t^i(U) + k & si \quad \bar{\delta}_{t-1}^i . \delta_t^i > 0, \\ \vartheta_t^i(U)(1-\psi) & si \quad \bar{\delta}_{t-1}^i . \delta_t^i < 0, \\ \vartheta_t^i(U) & \text{sin } on. \end{cases}$$

$$\bar{\delta}_t^i(U) = (1-\varphi)\delta_t^i(U) + \varphi \bar{\delta}_{t-1}^i(u)$$

4. Mise à jour de la matrice q

$$q_{t+1}^{i} = q_{t}^{i} + \vartheta_{t}^{i}\tilde{\varepsilon}_{t+1}e_{t}^{i^{T}} \ , \forall R_i \qquad \text{II.51}$$

5. Election de l'action discrète ou continue à appliquer sur l'état S_{t+1} (cette étape est strictement identique au pas 6 de l'exécution du FACL et mise à jour des traces d'éligibilité :

$$e_{t+1}^{i}(U^{i}) = \begin{cases} \gamma\lambda.e_{t}^{i}(U^{i}) + \phi_{t+1}^{i}, (U^{i} = U_{t+1}^{i}), \\ \gamma\lambda.e_{t}^{i}(U^{i}), \qquad \text{sin } on \end{cases} \qquad \text{II.52}$$

6. Nouveau calcul de la fonction d'évaluation correspondant aux actions élues et à l'état courant avec les paramètres nouvellement mis à jour :

$$Q_{t+1}(S_{t+1}, U_{t+1}) = \sum_{R_i \in A_{t+1}} q_{t+1}^{i}(U_{t+1}^{i})\alpha_{R_i}(S_{t+1}), \qquad \text{II.53}$$

Cette valeur sera utilisée pour le calcul de l'erreur TD au prochain pas de temps.

II.8 Conclusion

Nous nous sommes intéressés au cours de cette première partie de ce chapitre à la définition des notions nécessaires pour la compréhension de la technique d'apprentissage par renforcement. Nous avons présenté deux algorithmes d'apprentissage par renforcement FACL, FQL qui utilisent sur un apprenti de type SIF le formalisme des processus de décision Markovien et reposent sur le principe de la programmation dynamique.

Toutefois, contrairement aux méthodes de programmation dynamique adaptative classiques (i.e. où les états et les actions sont de type discret), qui possède des preuves de convergence sous certaines conditions (c-à-d un nombre de visites suffisant tous les couples état/action), les méthodes introduisant la généralisation par le biais d'approximateur de fonctions, comme les SIF, perdent ces preuves de convergence.

Les deux algorithmes FACL et FQL vont être utilisés dans le chapitre suivant pour la sélection de la meilleure action parmi un ensemble d'actions discrètes disponibles d'apprentis de type SIF en réponse aux récompenses et punitions qu'ils reçoivent en cours d'interaction avec le système.

Chapitre III

Programmations et simulations

La persévérance est sans doute le plus sûr moyen pour atteindre ses objectifs.

Mouctar Keïta

Chapitre III

Programmation et simulations

III.1 Introduction

La résolution des problèmes de la navigation d 'un robot mobile est sujette en grande partie aux recherches et aux résultats obtenus en intelligence artificielle. Nous avons choisi d'utiliser un raisonnement basé sur l'utilisation de la logique floue où des informations plus ou moins précises peuvent être traitées d'une façon similaire à celles que pourrait adopter l'être humain. Elle permet de s'affranchir de toute une phase complexe de modélisation mathématique des systèmes à commander. Elle assure de plus une prise de décision et un traitement des informations en des temps de calcul très réduits. De par ses performances, nous avons donc choisi d'utiliser cet outil dans un certain nombre de modules de l'approche proposée.

Les premiers travaux présentés dans ce chapitre portent essentiellement sur le développement d'un système de contrôle pour le robot mobile Pioneer II à base de la logique floue. Le but général est de faire « *apprendre* » au robot mobile un certain nombre de comportements simples (convergence vers un but, évitement d'obstacles). Pour cela nous utilisons des méthodes d'apprentissage par renforcement qui sont basées sur l'exploration active de l'environnement afin de découvrir les états provoquant l'émission de récompenses et de punitions. Dans ce cadre, nous avons utilisé deux méthodes d'apprentissage par renforcement, le *Fuzzy Actor-Critic Learning* (FACL) et le *Fuzzy Q-Learning* ((FQL) qui reposent sur la méthode de prédiction des différences temporelles (TD). Eles permettent la sélection de la meilleure action parmi un ensemble d'actions discrètes disponibles dans chaque règle floue.

En seconde partie, une étude comparative sera menée afin de définir des règles quant à eux la détermination des paramètres caractérisant les deux algorithmes précédant devient plus simple.

III.2. Programmations des comportements élémentaires de navigation utilisant FACL et FQL

Comme on a vu précédemment, l'apprentissage par renforcement est une méthode qui se base sur le principe de punition-récompense pour résoudre un problème de décision séquentielle. Elle est aussi une approche adaptative parce qu'on part d'une solution inefficace que l'on améliore progressivement en fonction de l'expérience acquise. On peut la considérer aussi comme une modification automatique du comportement où l'action optimale à effectuer pour chacune des situations est celle qui permet de maximiser l'espérance de gain [61]. À chaque étape, le robot doit définir l'état dans lequel il se trouve. À partir de cet état, il doit prendre une décision sur l'action à exécuter. En fonction du résultat obtenu lors de l'exécution de cette action, il est soit puni, pour diminuer la probabilité d'exécution de la même action dans le futur, soit récompensé, pour favoriser ce comportement dans les situations identiques.

On a mentionné précédemment que deux approches sont possibles pour l'utilisation de la méthode d'apprentissage par renforcement. La première consiste à discrétiser le problème afin de fournir des espaces d'états qui pourront être utilisés directement par des algorithmes utilisant des tableaux de qualités *Q*. L'utilisation est difficile et exige le stockage des valeurs de la fonction qualité pour tous les couples (état, action) [76] [77]. Dans les problèmes discrets de faible dimension, on peut utiliser des tableaux dans lesquels une ligne correspond aux qualités des différentes actions pour un état donné. Mais dans le cas des espaces d'états et d'actions continus, le nombre de situation est infini et la représentation de la fonction *Q* par des tableaux est impossible. La deuxième approche va permettre de travailler directement dans les espaces d'états et d'actions continus en utilisant des méthodes d'approximation des fonctions comme les réseaux de neurones et les systèmes d'inférences floues qui offrent des solutions prometteuses pour ce problème et limitent les effets parasites qui pourraient apparaitre suite à un mouvais choix des espaces état-action pour la discrétisation

III.2.1. Première série d'expérimentation- comportement aller au but-

Ce premier comportement permet de réaliser l'action « *convergence vers le but* » par le robot à partir de la connaissance de sa position courante et la définition d'une position relative à atteindre spécifiée par le contrôle. Le but est de gérer les déplacements du robot pour qu'il puisse atteindre le point d'arrivée. Il est bien entendu que ce comportement ne peut opérer que dans des environnements peu encombrés où aucun obstacle ne gênent la progression du robot.

Dans ce cas, le robot doit mesurer la distance robot-but (ρ_b) et l'erreur angulaire (θ_{Rb}). En utilisant ces deux valeurs, le contrôleur flou optimisé par le renforcement génère les deux actions de déplacement vers le but (vitesse de rotation et vitesse de déplacement).

La définition du point à atteindre est effectuée par l'intermédiaire de deux variables d'entrées "ρ_b" et "θ_{Rb}", les coordonnées polaires de ce point exprimées dans le repère principal du robot (figure III.1). La variable représente l'angle entre le vecteur vitesse du robot et le vecteur Robot-But et la variable ρ_b représente la distance Robot-But. Les deux variables de sorties sont la vitesse de rotation du robot *Vrot* et la vitesse de translation linéaire *Vit*.

L'apprenti dans ce cas est un contrôleur flou de type Sugeno d'ordre zéro (figure III.2) où les conclusions de la vitesse de translation et la vitesse de rotation et les conclusions du critique sont initialement mises à zéro. Les entrées du contrôleur flou "ρ_b" et "θ_{Rb}", sont définies respectivement par deux et trois fonctions d'appartenance (figure III.3).

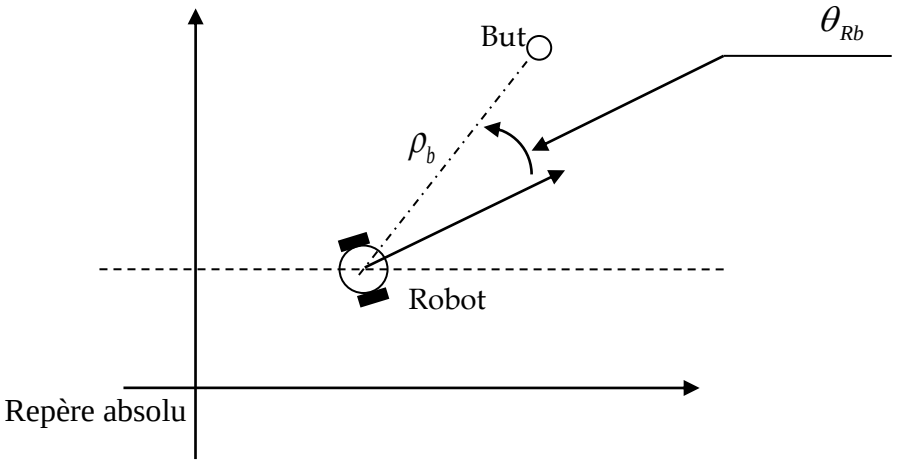


Figure III.1 : Représentation des variables d'entrées de l'apprenti.

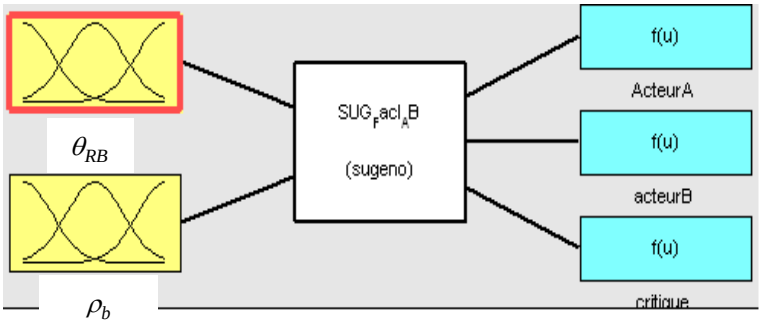


Figure III.2 : Structure de l'apprenti.

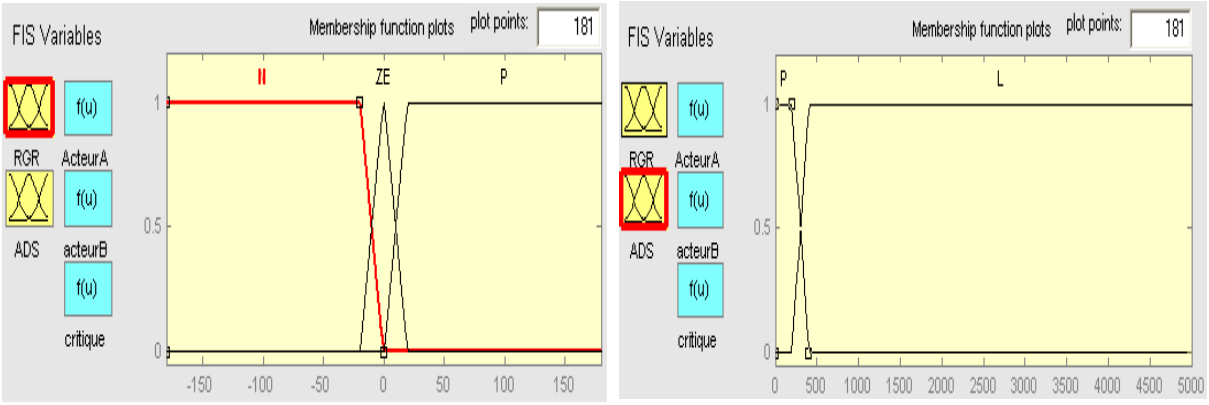


Figure III.3 : Fonctions d'appartenance des entrées.

III.2.1.1. L'ensemble d'actions

L'acteur dans ce cas est constitué de la paire d'actions : vitesse de rotation et vitesse de translation.

On considère un ensemble d'actions U commun à toutes les règles floues, constitué de sept valeurs pour la vitesse de rotation (a_1= -20(°/s), a_2 = -10(°/s), a_3 = -5 (°/s), a_4 = 0 (°/s), a_5 = 5 (°/s), a_6= 10(°/s), a_7 = 20(°/s),) et de trois valeurs pour la vitesse de translation (a'_1 = 0(mm/s), a'_2 = 200 (mm/s), a'_2 = 150 (mm/s)) . Le contrôleur flou peut donc générer des actions de type continu comprises entre -20°/s et 20 °/s pour la vitesse de rotation, et entre 0 et 200 mm/s pour la vitesse de translation, par interpolation de ces actions discrètes.

La base des règles est alors comme suit :

1) If (θ_{Rb} is N) and (ρ_b is P) then (*Vrot* is A_1) (*Vit* is B_1) (Critique is v_1)
2) If (θ_{Rb} is N) and (ρ_b is L) then (*Vrot* is A_2) (*Vit* is B_2) (Critique is v_2)
3) If (θ_{Rb} is ZE) and (ρ_b is P) then (*Vrot* is A_3) (*Vit* is B_3)(Critique is v_3)
4) If (θ_{Rb} is ZE) and (ρ_b is L) then (*Vrot* is A_4) (*Vit* is B_4) (Critique is v_4)
5) If (θ_{Rb} is P) and (ρ_b is P) then (*Vrot* is A_5) (*Vit* is B_5)(Critique is v_5)
6) If (θ_{Rb} is P) and (ρ_b is L) then (*Vrot* is A_6) (*Vit* is B_6)(Critique is v_6)

III.2.1.2. La fonction de renforcement

Pendant l'apprentissage le robot reçoit des renforcements qui peuvent être positifs ou négatifs. Dans ce cas la fonction de renforcement est définie selon la position du point but, autrement dit le robot effectue une rotation dans la direction du point but puis convergence directement avec une vitesse maximale.

$$\begin{cases} r = 10 \; if \; (\theta_{RB}\dot{\theta}_{RB} < 0 \;\&\; Vit = 0 \;\&\; \rho_b < 100) \\ r = -1 \; if \; (\theta_{RB}\dot{\theta}_{RB} > 0 \;\&\; Vit \neq 0 \;\&\; \rho_b < 100) \\ r = 1 \; if \; (\theta_{RB}\dot{\theta}_{RB} < 0 \;\&\; \rho_b > 100) \\ r = 1 \; if \; (-1 < \theta_{RB} < +1 \;\&\; \rho_b > 100) \\ r = 0 \; if \; (\theta_{RB}\dot{\theta}_{RB} = 0 \;\&\; \rho_b > 100) \\ r = -1 \; sinon \end{cases} \qquad \text{III.1}$$

III.2.1.3. Conditions de l'expérimentation

Dans cette simulation, et après plusieurs tests, les valeurs des paramètres de l'algorithme FQL sont représentées dans la table suivante :

γ	λ	ψ	φ	θ	Sp
0.6	0.7	0.8	0.6	10	0. 01

Table III.1 : Table des paramètres utilisés.

Avec :

- β : taux d'apprentissage adaptatif.
- γ : taux d'escompte (pour évaluer l'influence du long terme dans la prédiction de la fonction d'évaluation),
- S_P, θ : terme d'exploration et l'exploitation respectivement,
- λ **:** facteur de proximité du critique prise en compte des traces d'éligibilités,

III.2.1.4. Résultats de l'expérimentation

La figure III.4 représente les trajectoires obtenues avant et après apprentissage pour un point but (6000,-5200). Durant la phase d'apprentissage, le robot effectue une exploration de l'environnement en testant l'ensemble des actions disponibles. Dans cette phase, le processus d'élection des actions renforce les actions qui permettent au robot de se diriger directement vers le point but. Après cette phase, les actions qui ont les qualités les plus élevées sont sélectionnées. La figure de droite (phase après apprentissage) montre, qu'au départ le robot s'oriente correctement vers le point but ensuite il se diriger directement vers ce dernier.

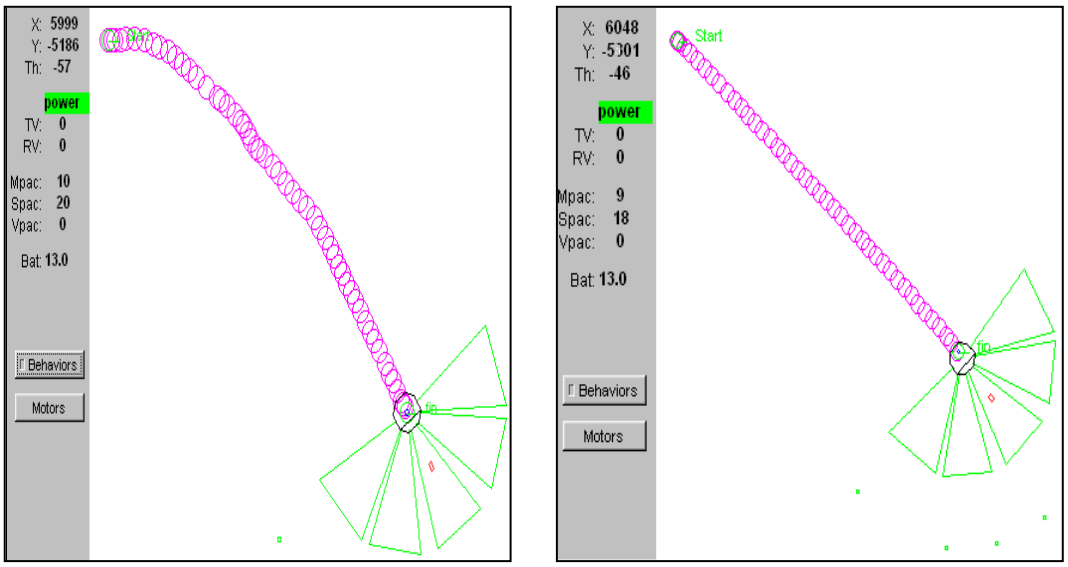


Figure III.4: Trajectoire du robot durant et après apprentissage.

Pour permettre au robot de visiter les différents états possibles et de tester l'ensemble des actions de chaque règle floue, nous procédons à une simulation où le robot est amené à atteindre plusieurs points but. Cette procédure permet d'activer l'ensemble des règles floues.

Pour cela, nous reprenons les mêmes conditions du test d'expérience précédent tout en augmentant le paramètre du taux d'exploration dirigée à une valeur (θ =60).

Le tableau III.2 représente les valeurs de qualité des actions obtenues après apprentissage et montre que toutes les actions pour chaque règle floues ont été testées.

Règle/q	q1	q2	q3	q4	q5	q6	q7
1	-0.096622	-0.12813	-0.12819	-0.24515	-0.12558	-0.089278	-0.068118
2	-0.016057	-0.20722	-0.34726	-0.060279	-0.12747	-0.29313	-0.16429
3	-2.5846	-2.5842	-2.6223	-2.4939	-2.7075	-2.3358	-2.9283
4	-0.24087	-0.13086	-0.86621	-0.028632	-0.3909	-0.041876	-0.25556
5	-2.1779	-2.2109	-1.6111	-2.2276	-1.6434	-1.7296	-2.307
6	-0.038601	0.17881	0.047188	-0.083261	0.11393	0.18947	-0.20892

Table III.2 : Table des valeurs de qualités des actions par règles.

La figure III.5 montre que durant la phase d'apprentissage, une fois la phase d'exploration est achevée, le robot est alors capable de se diriger directement vers les points buts B3 et B4.

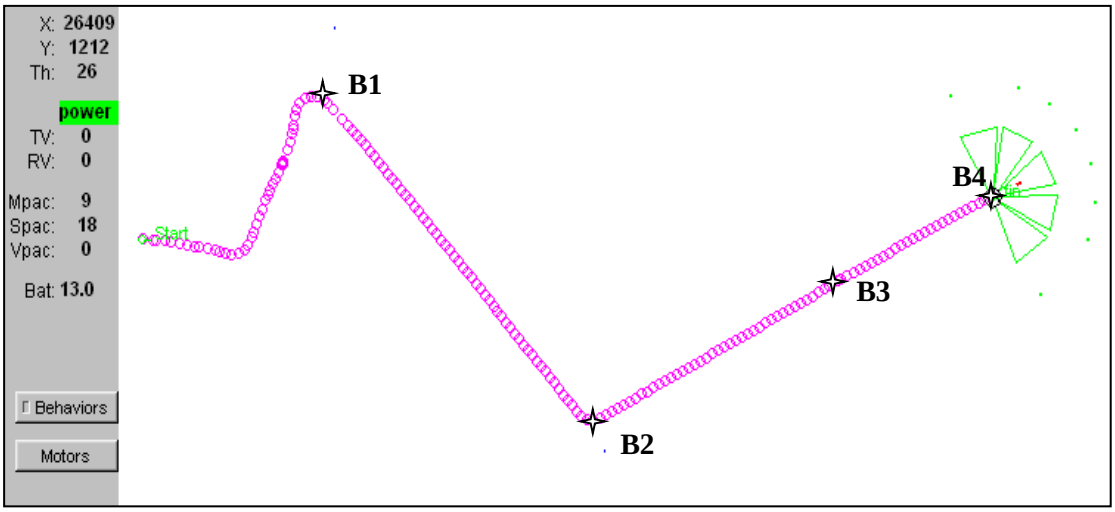


Figure III.5: Trajectoire du robot durant l'apprentissage.

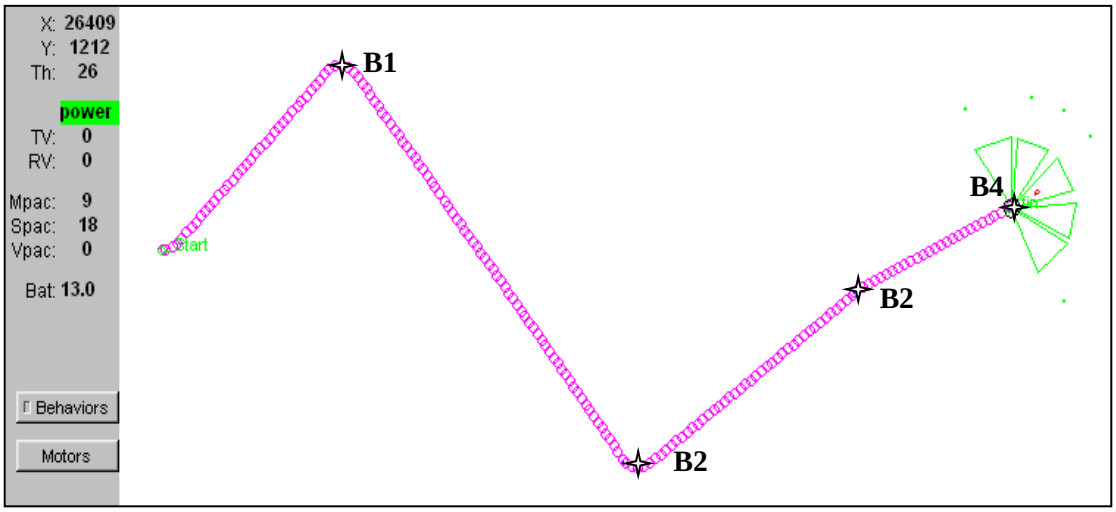


Figure III.6: Trajectoire du robot après apprentissage (validation).

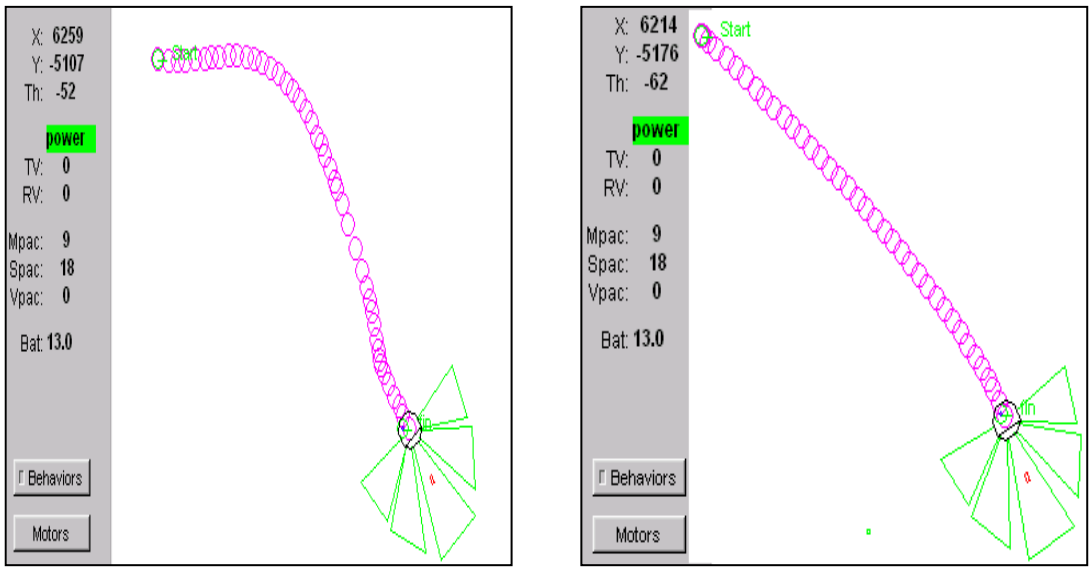
Dans l'étude précédente, nous avons appliqué l'algorithme d'apprentissage par renforcement FQL. Dans ce qui suit nous exploitons l'algorithme FACL pour réaliser le comportement « convergence vers le but ». L'apprenti et les conditions de l'expérimentation sont identiques à ceux des tests précédents.

Le tableau III.3 indique les valeurs des paramètres de l'algorithme FACL retenues dans cette simulation.

γ	λ	λ'	ψ	φ	θ	Sp
0.6	0.7	0.5	0.8	0.6	10	0.1

Table III.3 : Table des paramètres utilisés (algorithme FACL).

La figure III.7 montre les trajectoires du robot pendant et après la phase d'apprentissage, nous remarquons qu'après une phase d'exploration assez importante, le robot se dirige vers le point but.



Phase Apprentissage Phase Validation

Figure III.7 : Trajectoires de robot avant et après apprentissage par l'algorithme FACL « point but (6000,-5200) »

❖ Conclusion

Nous avons simulé le comportement « *convergence vers le but* » d'un robot mobile en utilisant les deux algorithmes d'apprentissage par renforcement FQL et FACL. Les expérimentations montrent qu'avec un bon choix des paramètres influençant l'apprentissage (taux d'apprentissage, facteur d'escompte, choix de la fonction de renforcement, politique d'exploration et d'exploitation) le contrôleur à six règles floues permet d'obtenir une convergence rapide vers le but.

Ces résultats montrent aussi la capacité de généralisation offerte par ces algorithmes d'apprentissage contrairement aux méthodes heuristiques d'élaboration des SIF.

III.2.2. Deuxième série d'expérimentation -évitement d'obstacles -

Le comportement développé dans la section précédente permet de réaliser la navigation d'un robot mobile autonome dans un environnement très peu encombré. Une telle condition est très restrictive vu la nature de l'environnement dont lequel le robot est amené à réaliser des tâches.

L'évitement d'obstacles est un comportement de base présent dans quasiment toutes les architectures de contrôle des robots mobiles. Il est indispensable pour permettre au robot de fonctionner dans un environnement contenant des obstacles pour gérer les écarts entre l'environnement modélisé et l'environnement réel qui peut présenter des obstacles imprévus ou non modélisés.

III.2.2.1 Principe du contrôle

Pour réaliser ce comportement, nous utilisons les informations de distances fournies par les capteurs à ultrasons dans les directions Frontale, Gauche et Droite. Les lectures des huit sonars du robot mobile sont donc regroupées en quatre groupes (D_0D_1, D_2D_3, D_4D_5, D_6D_7) comme le montre la figure III.8.

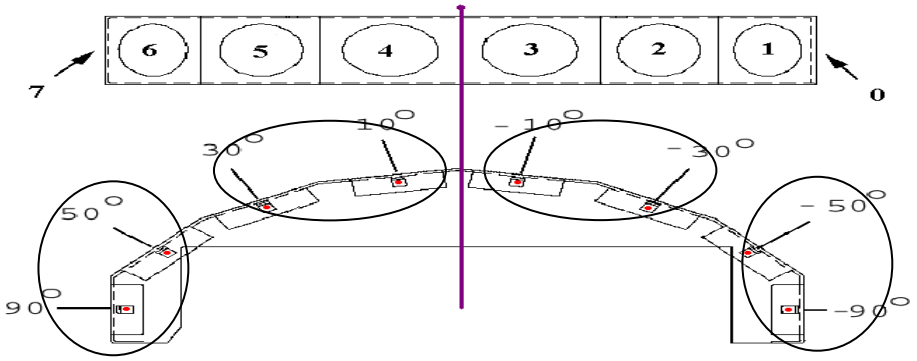


Figure III.8 : Regroupement des huit sonars de Pioneer II.

Les variables d'entrée de ce contrôleur sont respectivement les lectures de ces différents groupes, et les variables de sorties sont toujours la vitesse de rotation *Vrot*_**EB** et la vitesse de translation ***Vit*_EB (**figure III.9).

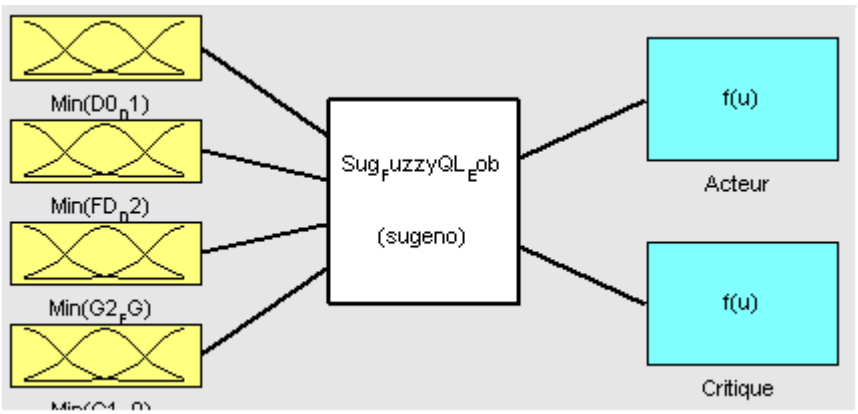


Figure III.9 : Contrôleur « d'évitement d'obstacles ».

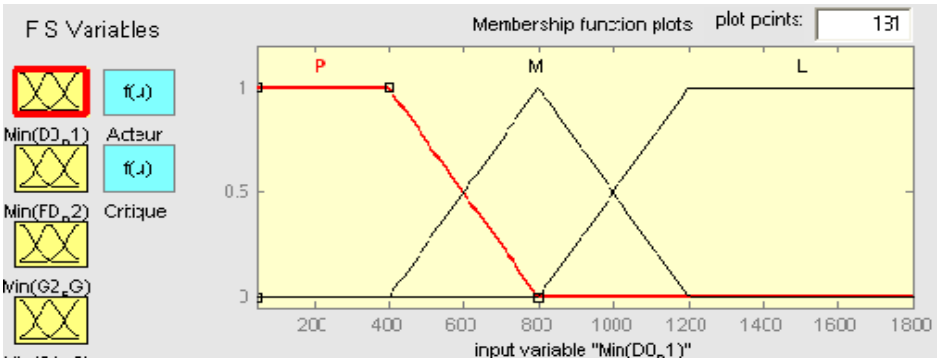


Figure III.10 : Fonctions d'appartenance des entrées.

III.2.2.2. L'apprenti

L'apprenti est un système d'inférence floue de type Sugeno d'ordre zéro. Les variables d'entrées sont les distances fournies par les trois sous-ensembles de trois groupes : des quatre groupes de sonars (S_0S_1, S_2S_3, S_4S_5 , S_6S_7) (figure III.8).

Les univers de discours sont décomposés en trois sous-ensembles flous (figure III.10).

La variable de sortie du contrôleur flou est la vitesse de rotation.

La vitesse de translation est déduite à partir de la distance minimale mesurée par les capteurs frontaux comme suit :

Les groupes de sonars sont déterminés sommes suit :

Sg1=min (S_0, S_1), Sg2=min (S_2, S_3), Sg3=min (S_4, S_5), Sg4=min (S_6, S_7)

$$\begin{cases} Vit\_EB = 0 \; if \; (\min(Sg2, Sg3) < 300mm) \\ Vit\_EB = 0.2 \times \min(Sg2, Sg3) \; if \; (\min(Sg2, Sg3) > 300mm \;\&\; \min(Sg2, Sg3) \geq 1800mm) \\ Vit\_EB = 250mm/s \; if \; (\min(Sg2, Sg3) \geq 1800mm) \end{cases} \quad \text{III.2}$$

On note que :

- La distance de sécurité robot obstacle est égale à 300mm
- Au-delà de la distance de 1800 mm l'obstacle est considéré très loin et ne présente aucun danger de collision

III.2.2.3 Les actions disponibles

L'ensemble des actions *U* commun à toutes les règles est constitué de cinq actions de vitesse de rotation {action a_1 = -20°/s, action a_2 = -10°/s, action a_3 = 0°/s, action a_4 = +10°/s et action a_5 = +20°/s}.

III.2.2.4 La fonction de renforcement

La fonction de renforcement retenue après plusieurs tests est définie comme suit :

$$\begin{cases} r = +1 \; if \; (\min(Sg1, Sg2) < \min(Sg3, Sg4) \;\&\; Vrot > 0 \\ r = +1 \; if \; (\min(Sg1, Sg2) \geq \min(Sg3, Sg4) \;\&\; Vrot \leq 0 \\ r = -1 \; ailleurs \end{cases} \quad \text{III.3}$$

III.2.2.5 Conditions d'expérimentation

Les valeurs des paramètres, obtenues après plusieurs essais, utilisés dans les algorithmes FQL et FACL sont représentées dans la table III.4 :

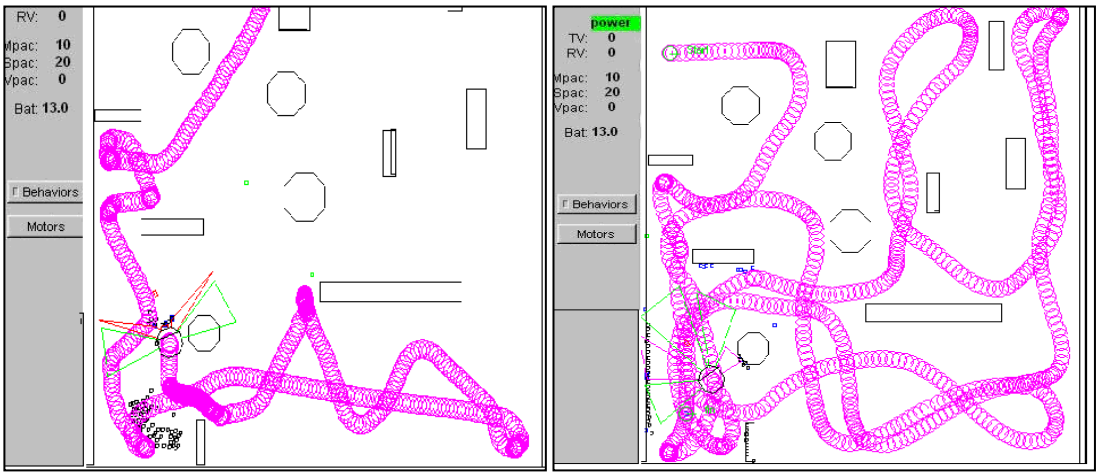
- Un taux d'apprentissage adaptatif β.
- Un taux d'escompte γ (pour évaluer l'influence du long terme dans la prédiction de la fonction d'évaluation)
- Une exploration et une exploitation S_P, θ
- Des traces d'éligibilités : facteur de proximité du critique λ, facteur de proximité de l'acteur λ'.

Paramètres	γ	λ	λ'	ψ	φ	θ	Sp
FQL	0.6	0.7	/	0.8	0.6	5	0.1
FACL	0.6	0.7	0.8	0.5	0.6	50	0.001

Table III.4 : Table des valeurs des paramètres utilisés.

III.2.2.6 Résultats de simulation

La figure III.11 montre les trajectoires du robot avant et après apprentissage pour l'algorithme FQL. Durant la phase d'apprentissage, le robot diminue sa vitesse de translation dans les coins serrés mais cette caractéristique disparaîtra dans la phase de validation.



Phase Apprentissage Phase Validation

Figure III.11 : Trajectoires du robot pendant et après l'apprentissage.

La figure III.12 représente d'autres essais, en utilisant l'algorithme d'apprentissage FQL, dans des environnements de type laboratoire (portes, tables, couloirs). Après la phase d'apprentissage le robot se déplace dans les différents endroits de l'environnement sans collision avec les obstacles.

Figure III.12 : Trajectoires du robot pendant et après apprentissage dans un environnement plus encombrés.

Dans le dernier essai, nous illustrons les trajectoires du robot obtenues en utilisant les deux algorithmes d'apprentissage FQL et FACL pour le même environnement.

Les trajectoires du robot après une phase d'apprentissage sont satisfaisantes pour les deux algorithmes d'apprentissage FQL et FACL, figure III.13.

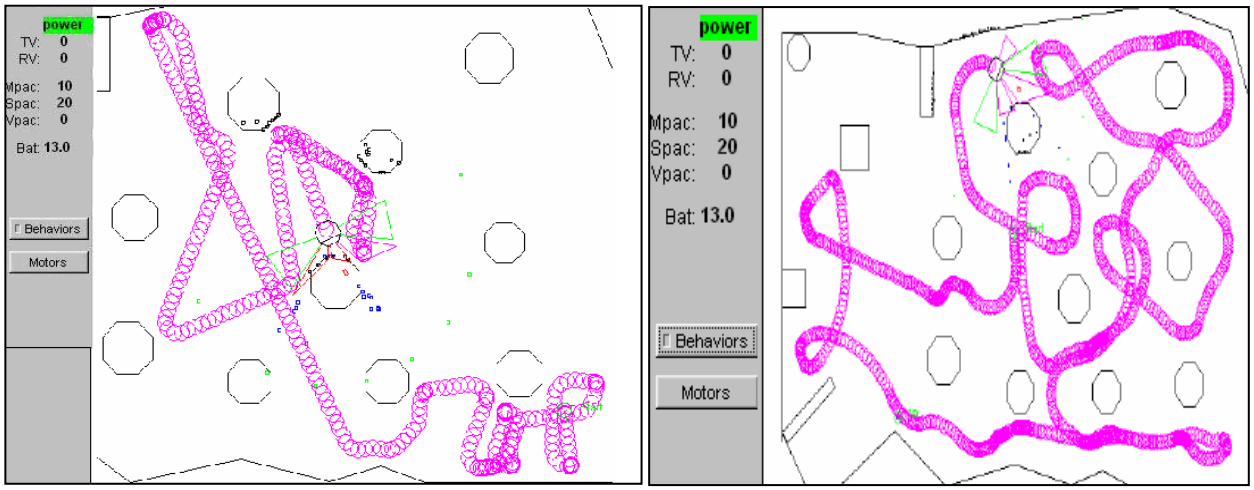


Figure III.13 : Trajectoires obtenues après apprentissage pour FACL et FQL respectivement.

❖ Conclusion

L'étude effectuée jusqu'ici, nous a permis d'élaborer des contrôleurs flous qui permettent d'éviter la collision avec les obstacles existant dans l'environnement. Les résultats obtenus par les deux algorithmes d'apprentissage par renforcement ont été satisfaisants.

La phase la plus délicate lors de l'implémentation de ces algorithmes d'apprentissage est la définition de la fonction de renforcement qui caractérise le comportement voulu. Les fonctions utilisées dans nos tests ont été obtenues après plusieurs essais.

III.3 Étude comparative entre les deux algorithmes FACL et FQL

Dans cette partie, nous allons effectuer une étude comparative entre les deux méthodes d'apprentissage par renforcement le fuzzy actor-critic learning FACL et le Fuzzy Q-learning FQL. Ces algorithmes reposent sur la méthode de prédiction les différences temporelles TD pour un apprenti de type SIF. L'objectif de ce travail est de rendre possible la mesure de l'influence des paramètres constituant ces méthodes et l'élaboration des règles de décision quant au choix de ces paramètres. Les simulations en ligne effectuées sur des comportements élémentaires de navigation réactive du robot mobile Pioneer II : « Convergence vers un but », ont mis en évidence l'influence de chaque paramètre constituant ces méthodes.

Les deux méthodes d'apprentissage par renforcement fuzzy actor critic (FACL), et fuzzy Q-Learning (FQL) s'appuient respectivement sur AHC [68] et Q-Learning [76]. Ces méthodes ont été conçues pour permettre un apprentissage efficace d'un système d'inférence floue lorsqu'il est impossible d'utiliser une source d'apprentissage plus informative qu'un critique. En effet, FACL et FQL permettent d'ajuster les conclusions du SIF de manière incrémentielle en se basant uniquement sur les signaux de renforcement (la partie prémisse, c'est-à-dire la perception de l'apprenant, est définie à l'aide de la connaissance à priori de l'expert).

III.3.1. Première série d'expérimentation

Pour étudier l'influence de divers paramètres des algorithmes FACL et FQL sur le comportement du robot, nous avons effectué une série d'expérimentation sur le comportement convergence vers un point but (c'est d'amener le robot de sa position actuelle à la position relative spécifiée par le contrôle) afin de mettre en évidence la contribution de chaque paramètre et son influence sur la rapidité et les performances de chaque algorithme.

L'objectif principale de cette première série de d'expérimentations est de mesurer les effets des valeurs du **taux d'apprentissage** et du **facteur d'escompte** et de leurs interactions sur les états.

III.3.1.1. Conditions d'application des deux algorithmes

- Un taux d'apprentissage fixe β.
- Avec et sans taux d'escompte γ (pour évaluer l'influence du court et du long terme dans la prédiction de la fonction d'évaluation).

III.3.1.2 L'apprenti

Nous utilisons un contrôleur flou de type Sugeno d'ordre zéro, où initialement les conclusions de la vitesse de rotation (*Vrot*) et celles du critique C_*Vrot* (v_i) sont toutes mises à zéro Voir figure III.2. Les entrées du contrôleur flou sont $\{\rho_b , \theta_{Rb}\}$ avec trois et deux fonctions d'appartenance, respectivement (figure III.3).

III.3.1.3 Les actions disponibles

L'ensembles des actions disponibles pour les deux variables de sorties vitesse de rotation et la vitesse de translation sont représentées par les tableaux suivants.

Ces actions sont communes à toutes les règles du contrôleur flou.

Vitesse de rotation *Vrot* (deg/s)						
-20	-10	-5	0	5	10	20

Table III.5 : Les actions de ***Vrot*.**

Vitesse de translation *Vit* (mm/s)		
00	150	350

Table III.6 : Les actions de ***Vit***.

III.3.1.4 La fonction de renforcement

La fonction de renforcement utilisée est identique à celle donné par la formule III.1.

III.3.1.5 Résultats et discussions

- **Cas où le facteur d'escompte** $\gamma = 0.9$

Les figures III.14 et III.15 représentent des captures d'écran des trajectoires effectuées par le robot avec les deux algorithmes.

Dans les simulations des figures III.14 et III.15 on a fixé le facteur d'escompte γ à une valeur inférieure à 1. Puis on fait varier les valeurs de taux d'apprentissage $\beta = 10^{-4}$ (valeur très faible) et β = 0.9 (valeur importante), ce choix de valeurs est effectué afin de montrer l'influence du taux d'apprentissage β sur le comportement du robot toute en considérant $\gamma \neq 1$ pour les deux algorithmes d'apprentissage FACL et FQL.

Pour les figures III.14 et III.15 (de gauche : $\beta = 10^{-4}$), le robot effectue une importante exploration de l'environnement et suit de longues trajectoires pour atteindre sa destination.

En augmentant le taux d'apprentissage (figures III.14 et III.15 de droite : β =0,9), l'algorithme FQL a fourni de meilleures actions que celles du FACL et le robot suit donc une trajectoire rectiligne avec un temps considérablement réduit pour atteindre le point objectif.

Les figures III.14 et III.15 montrent les trajectoires obtenues pour les mêmes valeurs du taux d'apprentissage des deux algorithmes, le robot parvient à sélectionner les meilleures actions à l'aide du FQL que le FACL. Ce choix s'interprète avec une trajectoire rectiligne du robot.

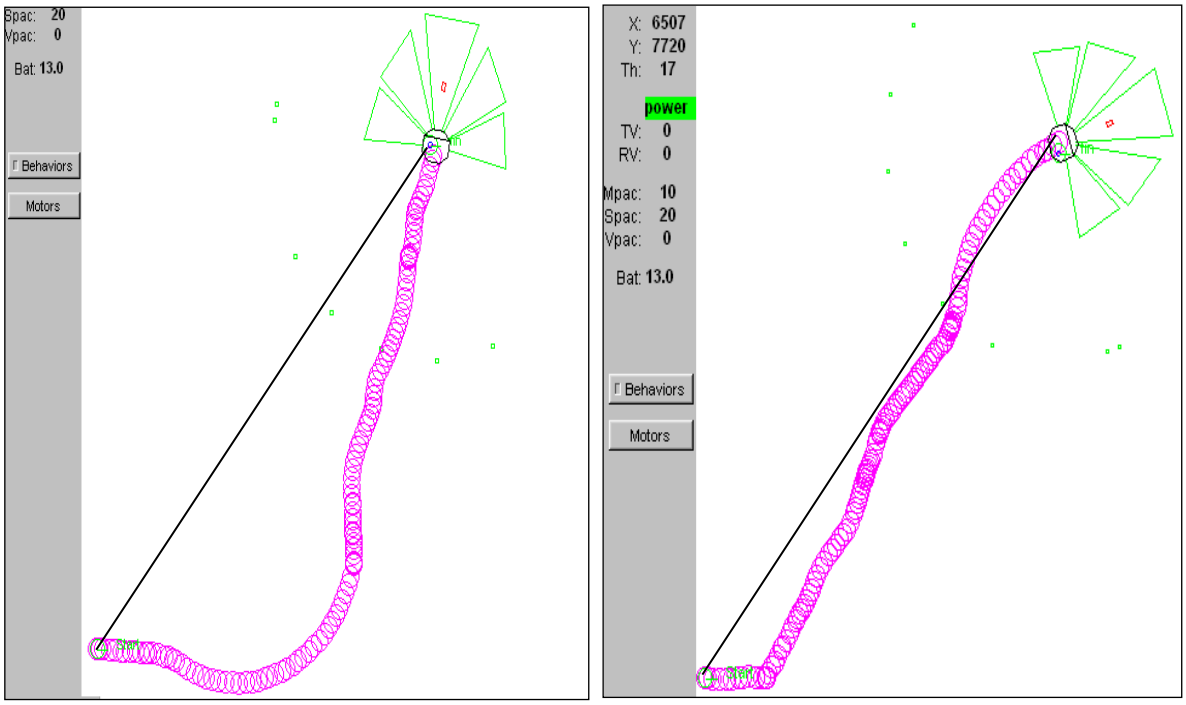


Figure III.14 : Trajectoires de robot pour $\beta = 10^{-4}$ (gauche) et β = 0.9 (droite) avec l'algorithme FACL et γ=0.9

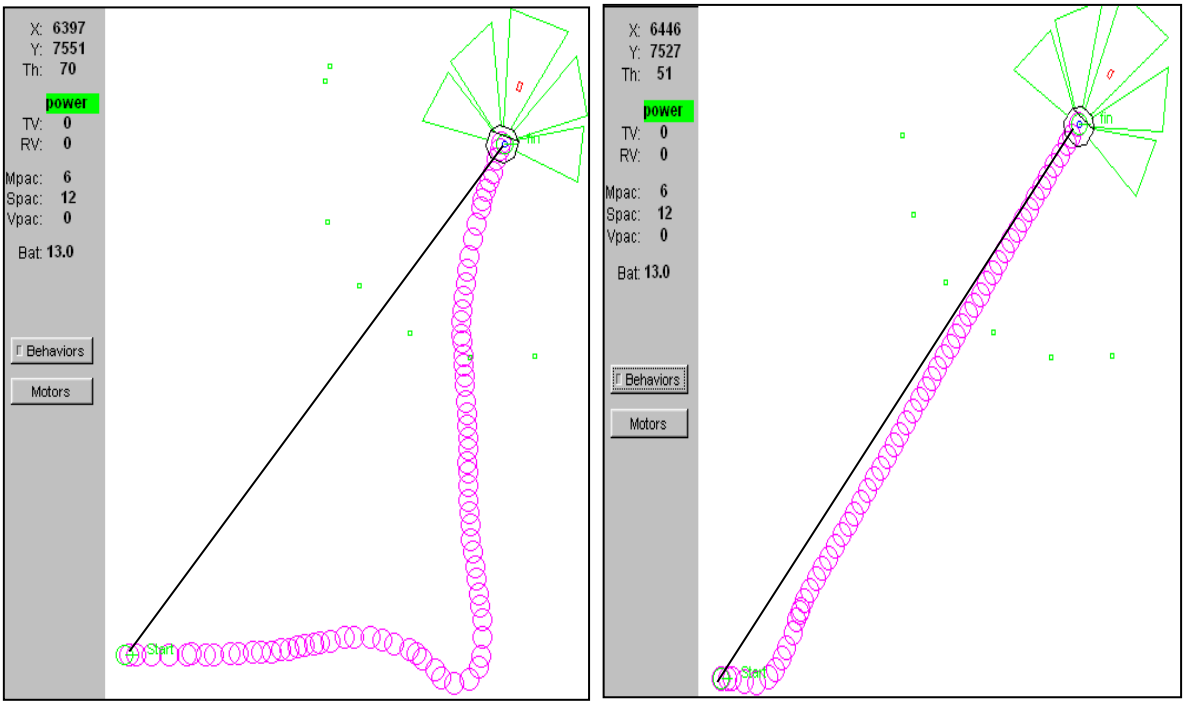


Figure III.15 : Trajectoires de robot pour $\beta = 10^{-4}$ (gauche) et β = 0.9 (droite) avec l'algorithme FQL et γ=0.9.

❖ **Cas où** $\gamma = 0.9$

- Si le taux d'apprentissage est élevé (β = 0.9), nous constatons que l'erreur TD effectue des variations brusques ainsi que le signal de renforcement (Figure III.16 a) pour FACL et (Figure III.17 a) pour FQL. Les valeurs de la fonction d'évaluation ainsi que les qualités des actions locales dans ce cas sont importantes.

- D'autre part, si le taux d'apprentissage est faible ($\beta = 10^{-4}$), l'erreur TD évolue de la même manière que le signal de renforcement principal (Figure III.16 b) et (Figure III.17 b). La fonction d'évaluation prend alors des valeurs plus élevées en utilisant le FQL que le FACL.

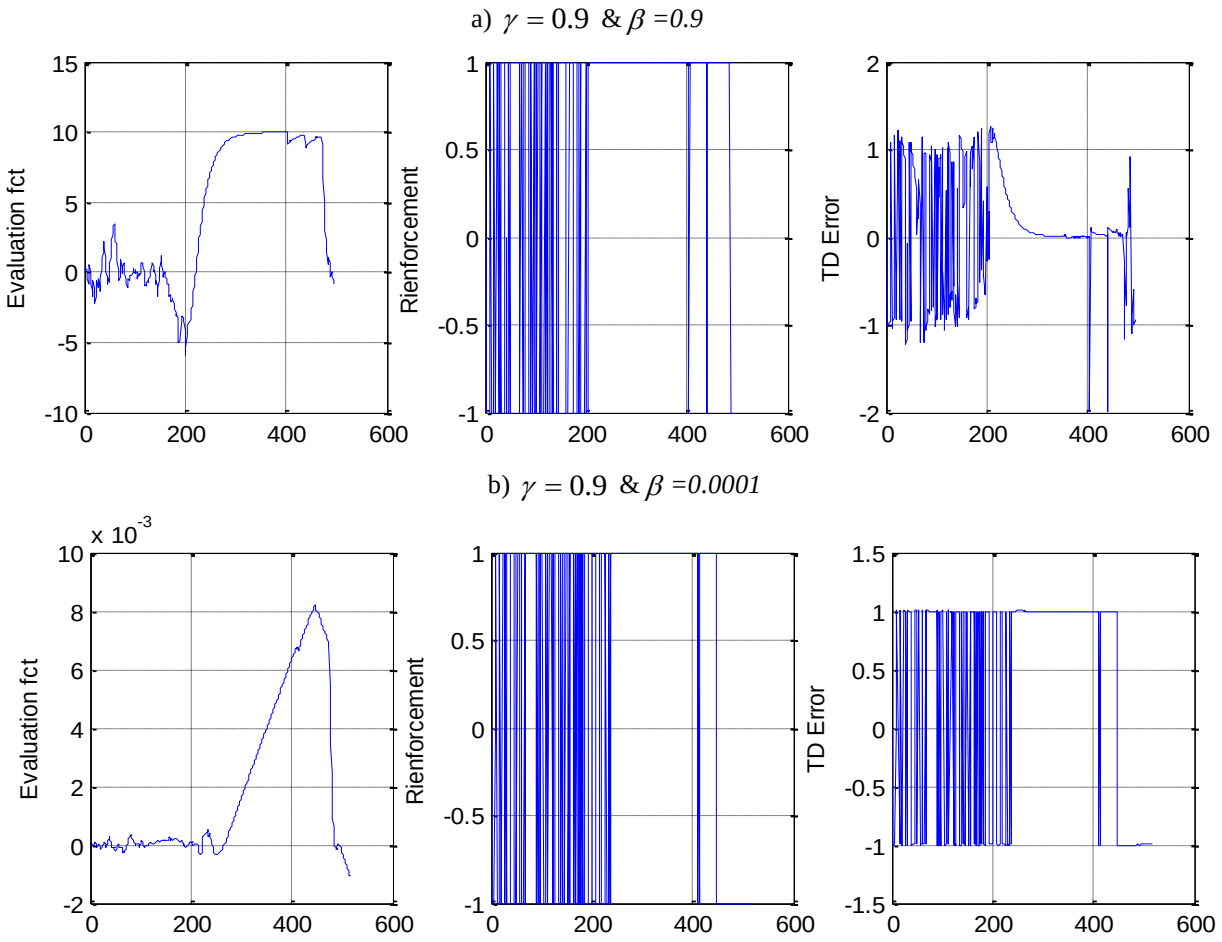


Figure III.16 : Évolution de la fonction d'évaluation, Renforcement et l'erreur TD en fonction du temps - Algorithme FACL -

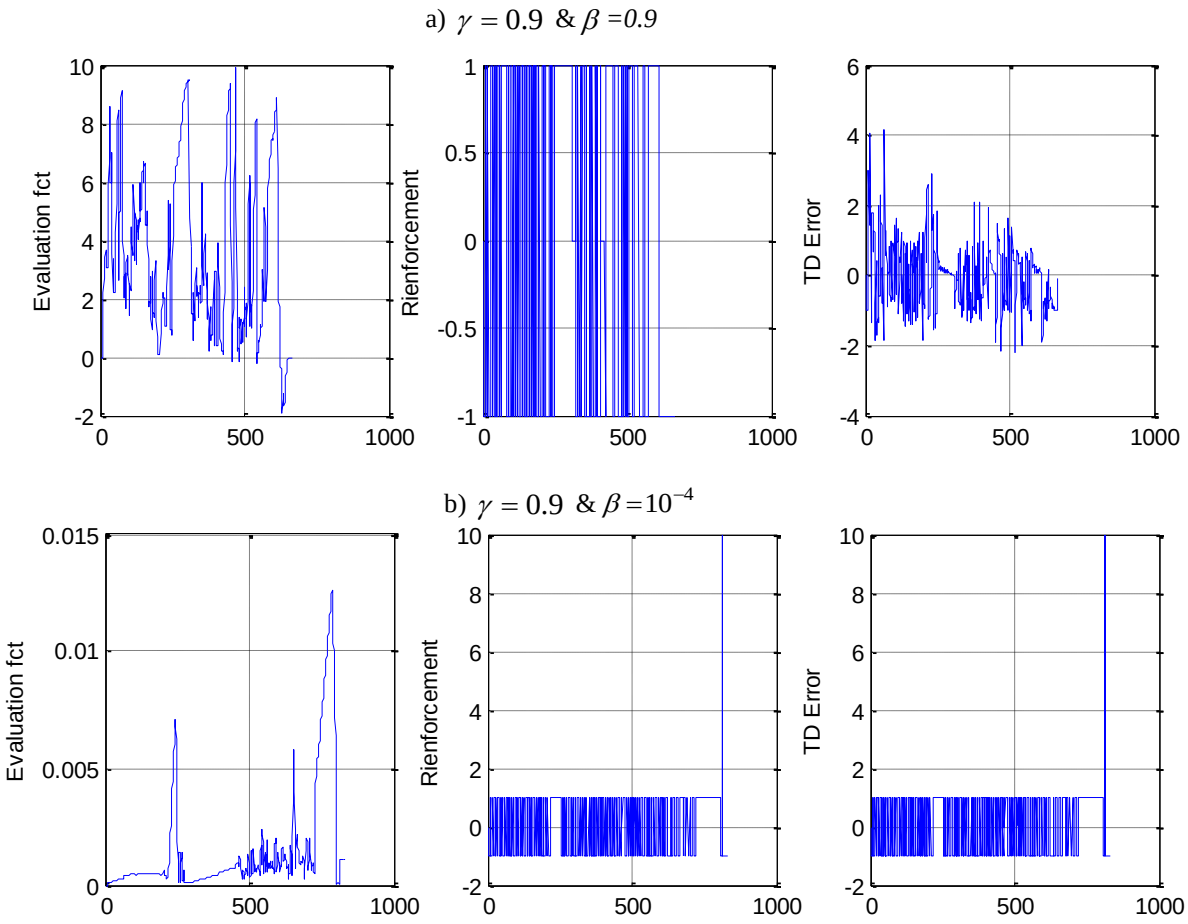


Figure III.17 : Évolution de la fonction d'évaluation, Renforcement et l'erreur TD en fonction du temps - Algorithme FQL –

❖ **Cas où le facteur d'escompte** $\gamma = 1$

Comme deuxième cas nous considérons $\gamma = 1$, les trajectoires obtenues sont données par les figures III.18 et III.19. En se basant sur le même principe du test précédant, on fixe γ à un puis on fait varier le taux d'apprentissage $\beta = 10^{-4}$ (valeur très faible) et β = 0.9 (valeur importante), ce choix de valeurs est effectué afin de montrer l'influence du taux d'apprentissage β sur le comportement du robot toute en considérant $\gamma = 1$ pour les deux algorithmes d'apprentissage FACL et FQL.

Nous constatons qu'avec un taux d'apprentissage faible ($\beta = 10^{-4}$), le robot effectue une exploration plus importante de l'environnement et suit de longues trajectoires pour atteindre le point objectif (figures III.18 et III.19 de gauche).

Par contre avec un taux d'apprentissage important (β =0,9), l'algorithme FQL permet de sélectionner les meilleures actions que celles du FACL et le robot suit donc une trajectoire rectiligne avec un temps considérablement réduit pour atteindre le point objectif (figures III.18 et III.19 de droite).

Ainsi pour les mêmes valeurs du taux d'apprentissage des deux algorithmes, le robot parvient à sélectionner les meilleures actions à l'aide du FQL que le FACL. Ce choix s'interprète avec une trajectoire rectiligne du robot.

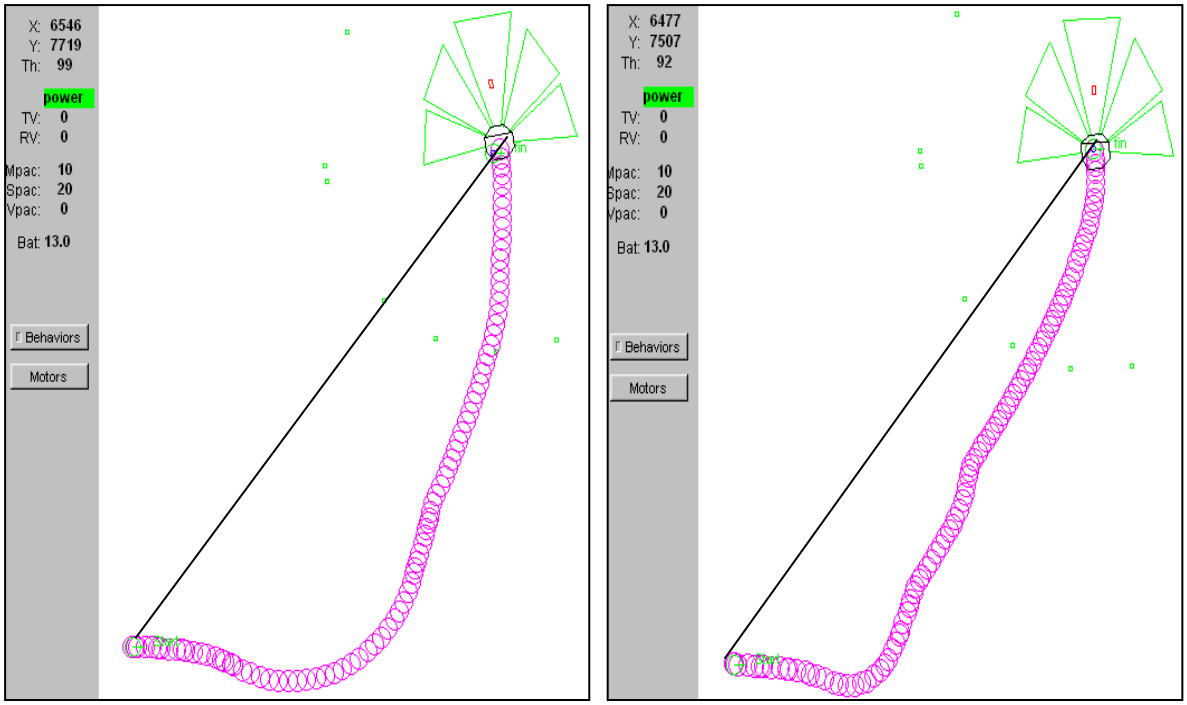


Figure III.18 : Trajectoires de robot pour $\beta = 10^{-4}$ (gauche) et β = 0.9 (droite) avec l'algorithme FACL.

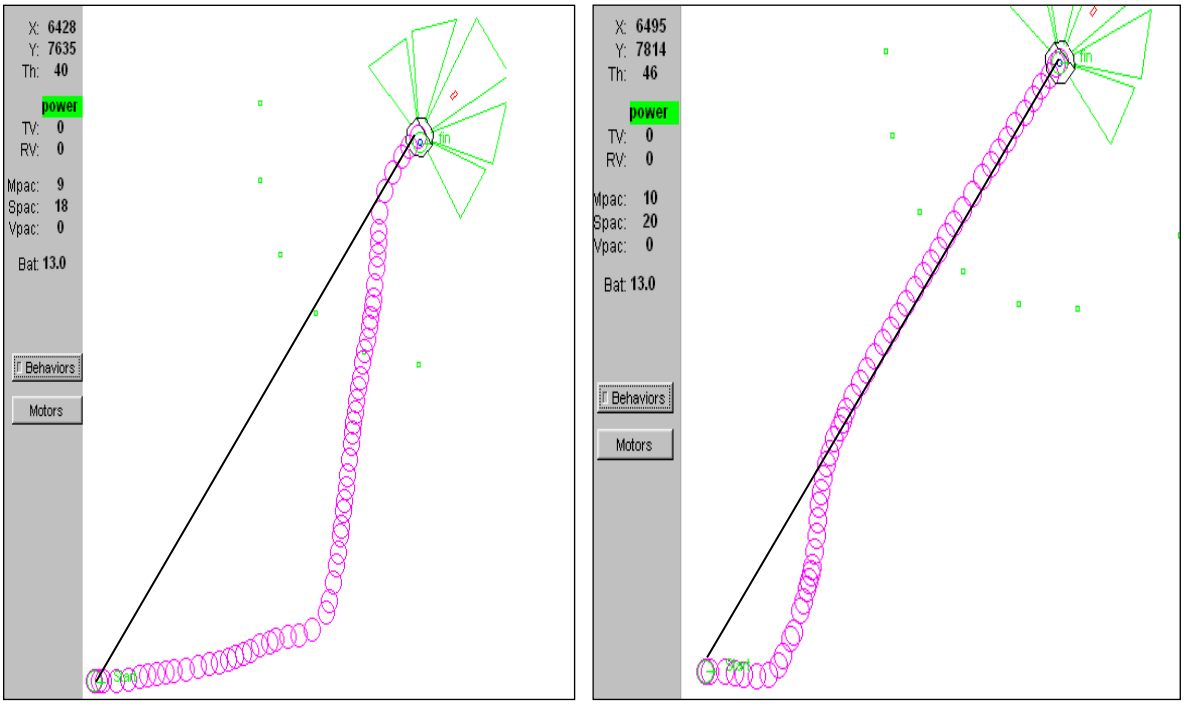


Figure III.19 : Trajectoires de robot pour $\beta = 10^{-4}$ (gauche) et β = 0.9 (droite) avec l'algorithme FQL.

❖ **Cas où** $\gamma = 1$

- Si le taux d'apprentissage est faible ($\beta = 10^{-4}$), l'erreur TD évolue de la même manière que le signal de renforcement principal (figure III.20 a) pour FACL et (et III.21 a) pour FQL. Le signe de cette erreur TD influe directement les qualités des actions locales élues. La fonction d'évaluation progresse de manière linéaire lorsque la même règle floue est activée et diminue fortement lorsque le système atteint un autre état non exploré : une autre règle est activée. Les valeurs de la fonction d'évaluation sont encore très basses. FACL réussit à sélectionner de meilleures actions que FQL. Le test est arrêté après environ 600 itérations lorsque le robot a atteint son objectif.

- D'autre part, si le taux d'apprentissage est élevé (β = 0,9), nous constatons que l'erreur TD n'a pas la même allure que le signal de renforcement (Figure III.20 b) (Figure III.21 b). Les valeurs de la fonction d'évaluation et les qualités sont importantes dans le cas de l'algorithme FQL. Les valeurs presque nulles de l'erreur TD ont provoqué une variation linéaire de la fonction d'évaluation jusqu'au changement de signe de cette erreur (les valeurs négatives impliquent une valeur décroissante de la fonction d'évaluation).

  Donc, dans cette expérience, nous obtenons pratiquement les mêmes résultats pour les algorithmes FACL et FQL. Cependant, pour l'algorithme FQL, les valeurs des qualités des actions locales sont proportionnelles à la valeur du taux d'apprentissage utilisé.

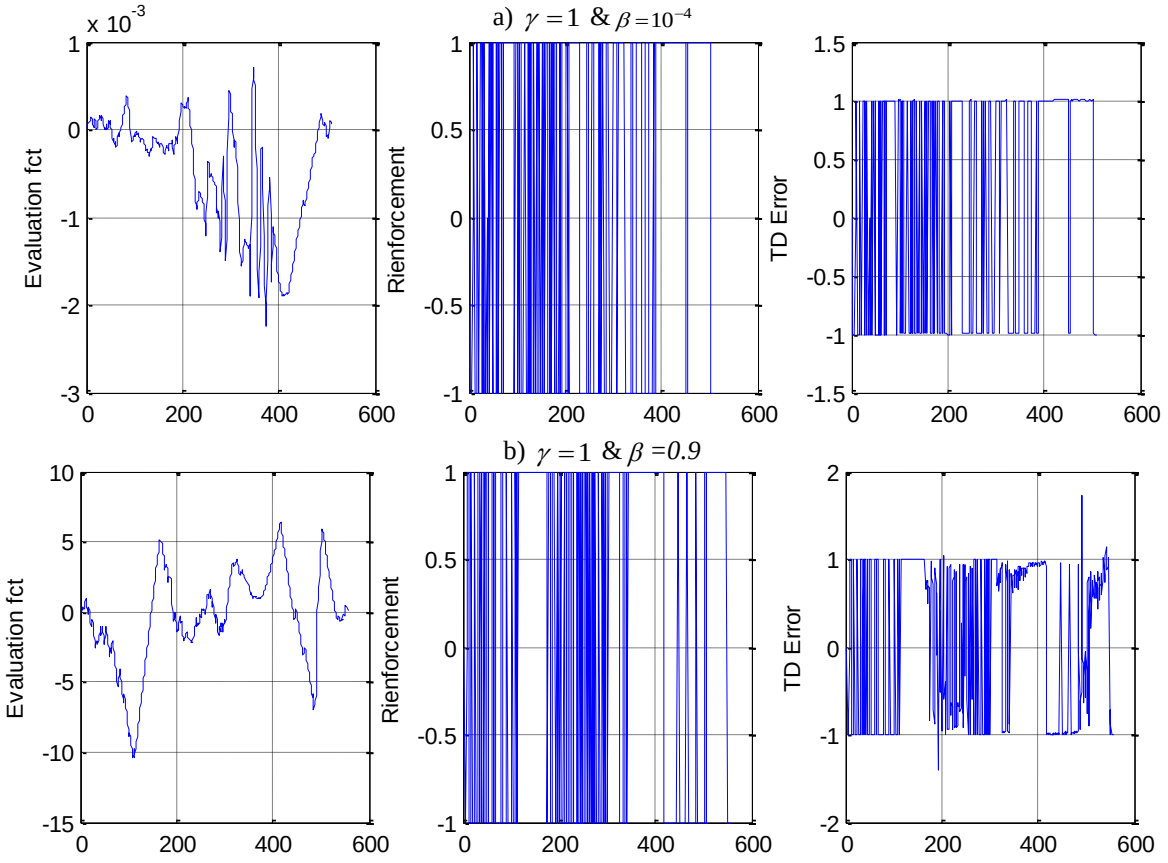


Figure III.20 : Évolution de la fonction d'évaluation, Renforcement et l'erreur TD - Algorithme FACL –

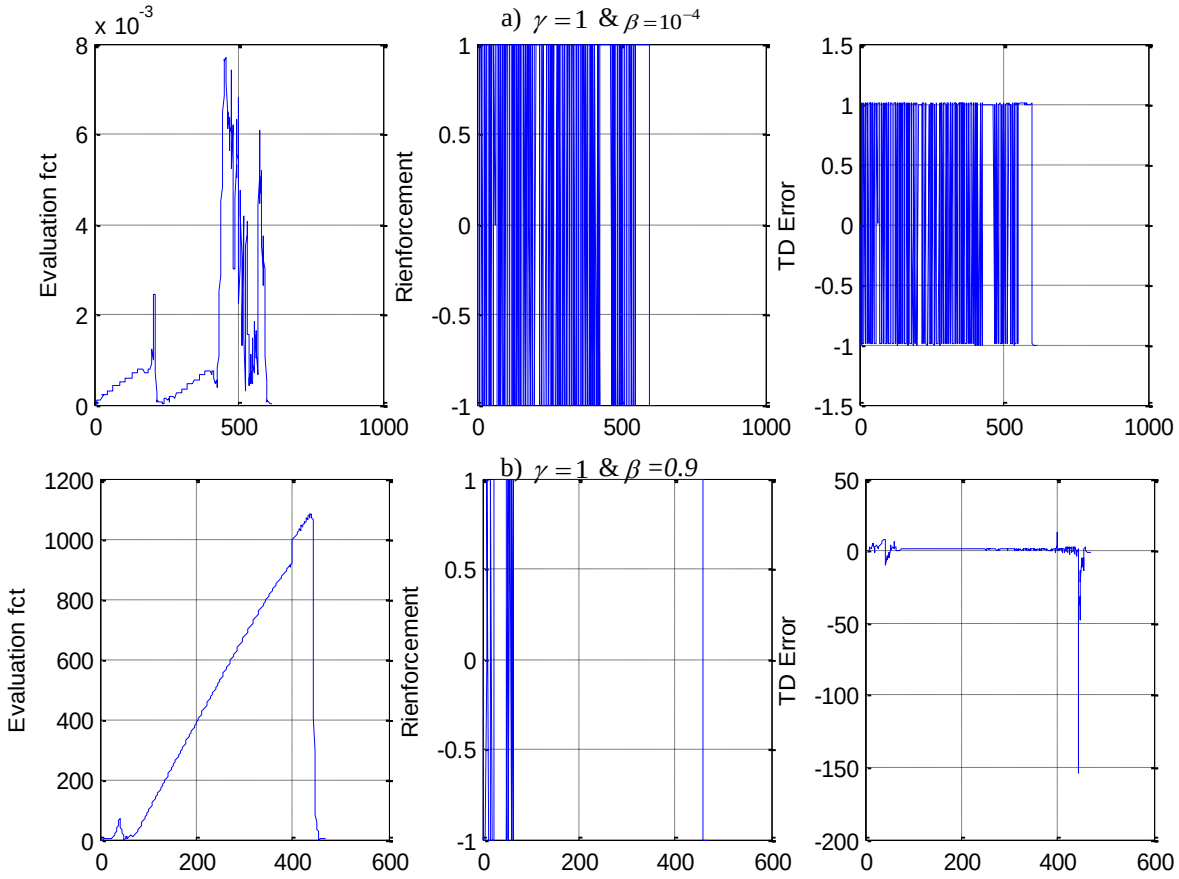


Figure III.21: Évolution de la fonction d'évaluation, Renforcement et l'erreur TD
- Algorithme FQL -

❖ **Cas où le facteur d'escompte $\gamma = 0.9$ et β adaptatif**

Les tests précédants montrent clairement l'influence du taux d'apprentissage β sur le comportement du robot, plus β est faible plus la trajectoire est longue et plus le robot explore des nouveaux états avant de sélectionner les actions qui ont les valeurs de qualité la plus élevées. Donc l'utilisation d'un taux d'apprentissage adaptatif devient une nécessité a fin de permettre au robot de tester que les actions qui rapportent des récompense (c'est-à-dire des renforcements positifs).

La figure III.22 (à gauche et à droite) illustre les trajectoires du robot mobile Pioneer II avec les algorithmes d'apprentissage FACL et FQL respectivement. Dans cette simulation nous avons considéré un taux d'apprentissage β adaptatif qui a été donné par la formule II.36 (Voir chapitre II § II.7.2.9), et un facteur d'escompte $\gamma = 0.9$. L'algorithme FQL preuve sa puissance dans le choix des meilleures actions par rapport à l'algorithme FACL. Ceci s'interprète par une trajectoire rectiligne et directe.

Les figures III.23 et III.24 montrent l'évolution de la fonction d'évaluation et le renforcement et l'erreur TD en utilisant l'algorithme FACL et FQL respectivement, ces courbes montrent clairement que le robot parcourt la distance point départ-point objectif en 400s en utilisant le FQL, tandis qu'avec l'algorithme FACL, il mit un temps plus important.

L'algorithme FQL trouve les meilleures actions tandis que le FACL continu à chercher les bonnes actions et ceci est montré par l'allure du signal de renforcement.

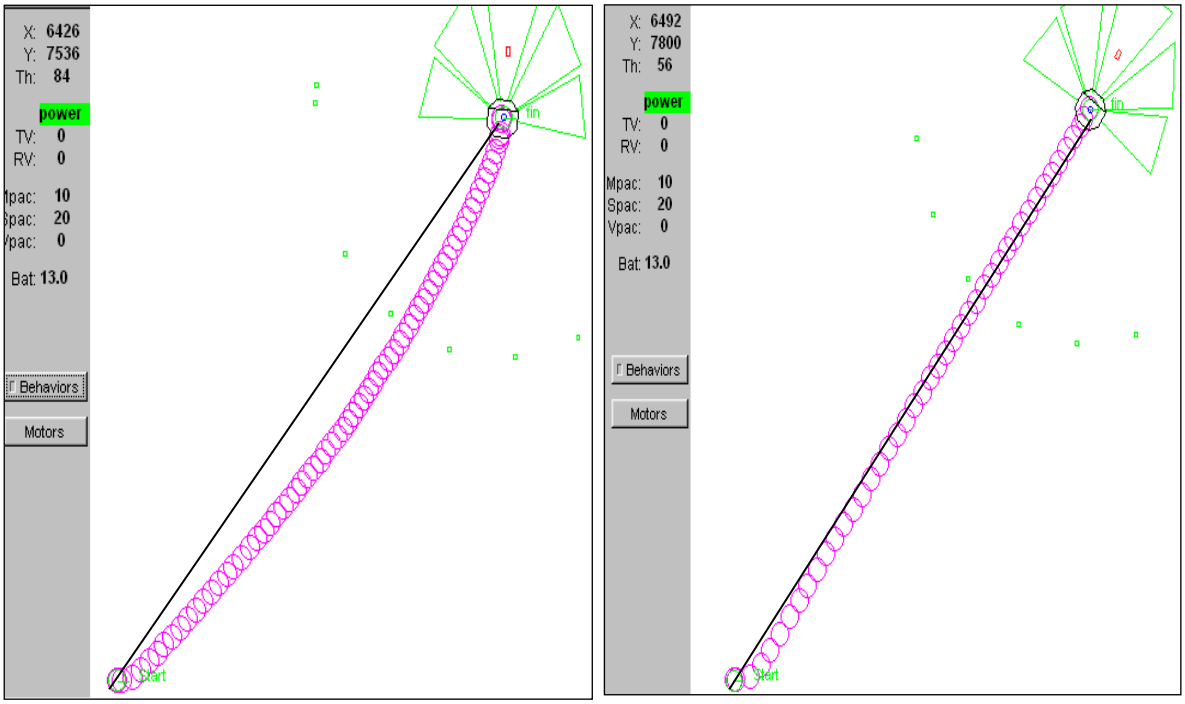


Figure III.22 : Trajectoires de robot FACL (gauche) et FQL (droite).

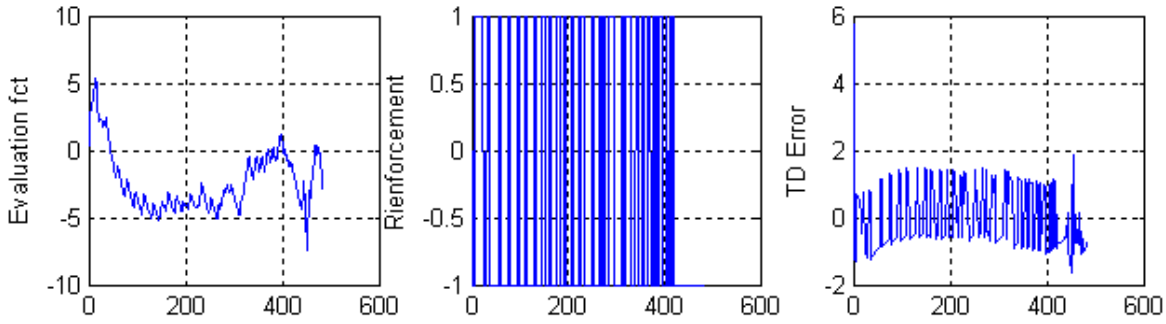


Figure III.23 : Évolution de la fonction d'évaluation, Renforcement et l'erreur TD
- Algorithme FACL –

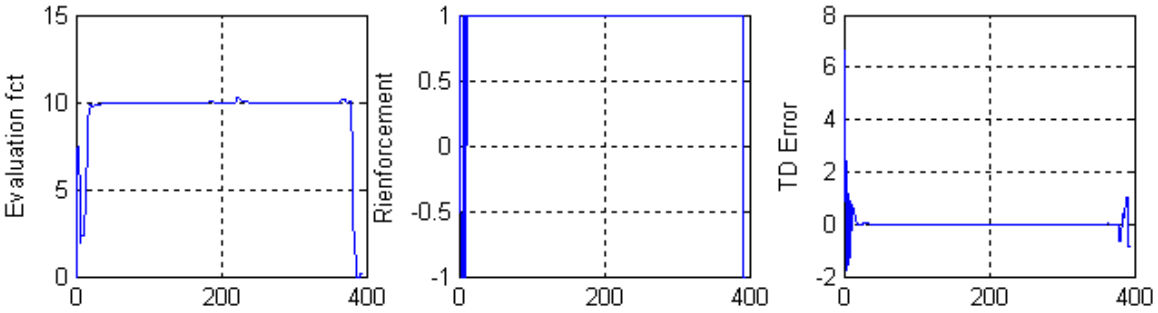


Figure III.24: Évolution de la fonction d'évaluation, Renforcement et l'erreur TD
- Algorithme FQL –

Les figures III.25 et III.26 illustrent la variation de la vitesse de rotation et la vitesse de translation du robot en utilisant FACL et FQL respectivement.

La figure III.26 montre que le robot effectue une rotation sur place dans la direction du point but puis avance avec une vitesse de translation égale à 200 mm/s. La figure III.25 montre les variations successives dans l'allure de la vitesse de rotation, ceci justifie les variations obtenues sur l'allure du renforcement (figure III.23), autrement dit le robot explore les actions parmi l'ensemble des actions définies et reçoit des renforcements qui peuvent être positifs ou négatifs selon la qualité de l'action testé tout au long de sa trajectoire.

Cette étude nous a permis d'explorer minutieusement l'influence des paramètres des algorithme FACL et FQL. L'utilisation d'un taux d'apprentissage adaptatif devient une nécessité. La considération des retours espérés (en termes de renforcements) dans la prédiction de la fonction d'évaluation influe sur la qualité des actions et le nombre des règles floues testées. Toutefois la détermination de la bonne combinaison des paramètres est alors assez délicate.

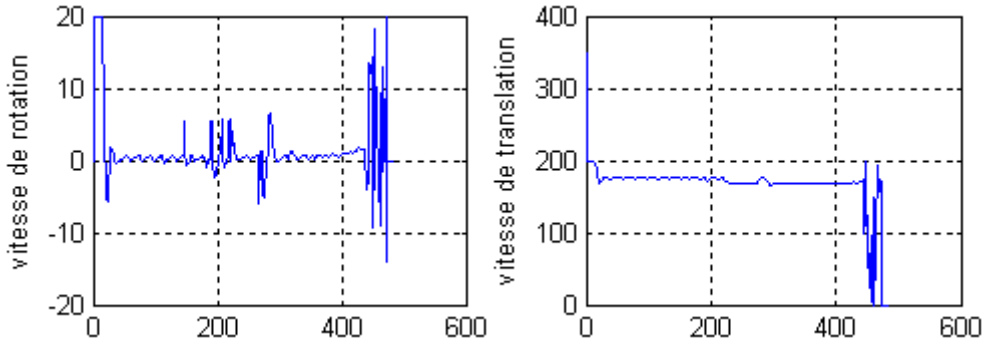


Figure III.25 : Évolution de la vitesse de rotation et la vitesse de translation
- Algorithme FACL –

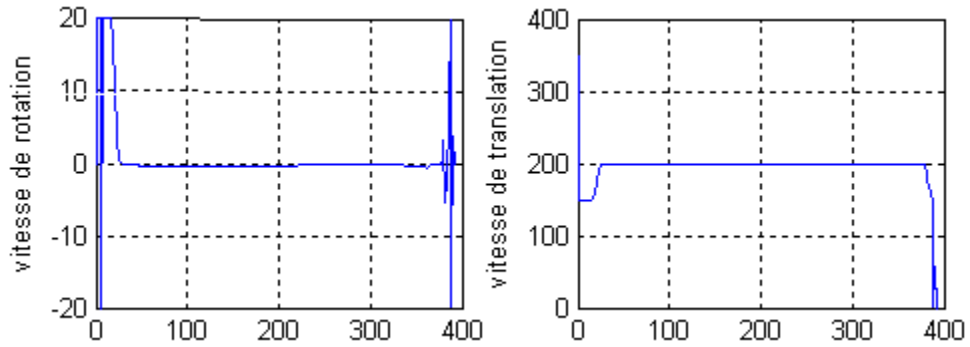


Figure III.26 : Évolution de la vitesse de rotation et la vitesse de translation
- Algorithme FQL –

III.3.2 Deuxième série d'expérimentation

L'étude réalisée dans cette seconde série de ce comportement permettra d'explorer l'influence de l'utilisation d'une politique gloutonne sur le comportement du robot en se basant l'algorithme FACL puis FQL.

III.3.2 .1 Conditions d'application des deux algorithmes

- Un taux d'apprentissage β adaptatif
- Un taux d'escompte $\gamma = 0.9$
- Une politique gloutonne (exploitation pure et pas d'exploration)

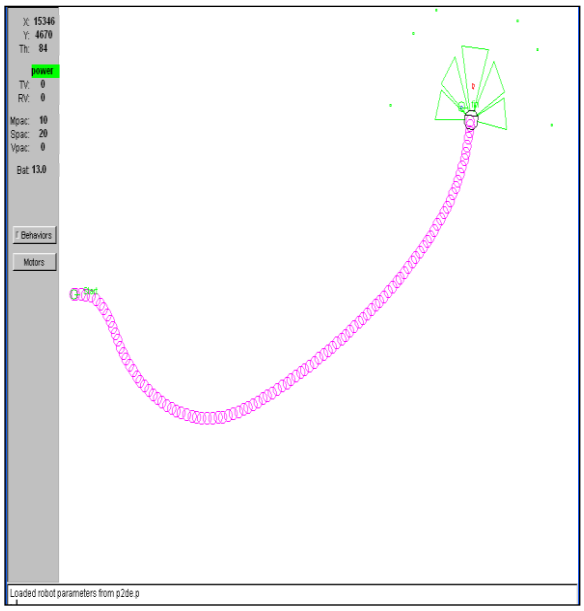
III.3.2 .2 La fonction de renforcement

La fonction de renforcement utilisée est identique à celle donné par la formule III.1.

III.3.2 .3 Résultats de la simulation

- **Phase d'apprentissage**

Les figures III.27 et III.28 sont des captures d'écran des trajectoires effectuées par le robot en phase d'apprentissage en utilisant respectivement les algorithmes FACL et FQL. Dans ce test on a considéré une politique gloutonne qui se base sur l'exploitation d'un ensemble des actions disponibles sans faire aucune exploration des actions qui rapportent des renforcements positives.



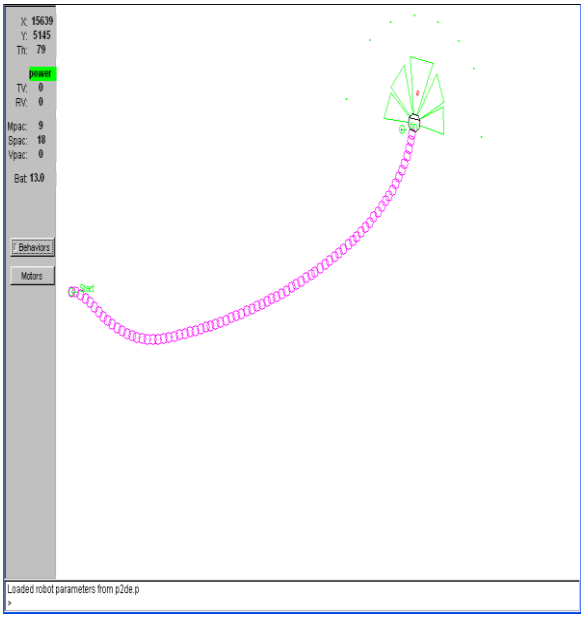


Figure III.27 : Trajectoire du robot avec FACL. **Figure III.28 :** Trajectoire du robot avec FQL.

Ces expériences montrent que dans le cas de l'algorithme FACL, le robot n'explore pas les actions à effectuer dans l'état t + 1, il exploite au contraire les actions disponibles, ces résultats donnent une trajectoire plus longue. D'autre part, avec l'algorithme FQL, la trajectoire est nettement améliorée.

- **Validation phase**

Les figures III.29 et III.30 sont des captures d'écran illustrant le type de trajectoires obtenues lors de la phase de validation des algorithmes appliqués. La figure III.30 illustre parfaitement l'insuffisance de la politique gloutonne qui a conduit dans ce cas à un échec : le robot ne converge pas vers la cible fixée en utilisant ces paramètres pour l'algorithme FQL. La figure III.29 montre que le robot a pu atteindre le point but tout en suivant une trajectoire non rectiligne en utilisant l'algorithme FACL.

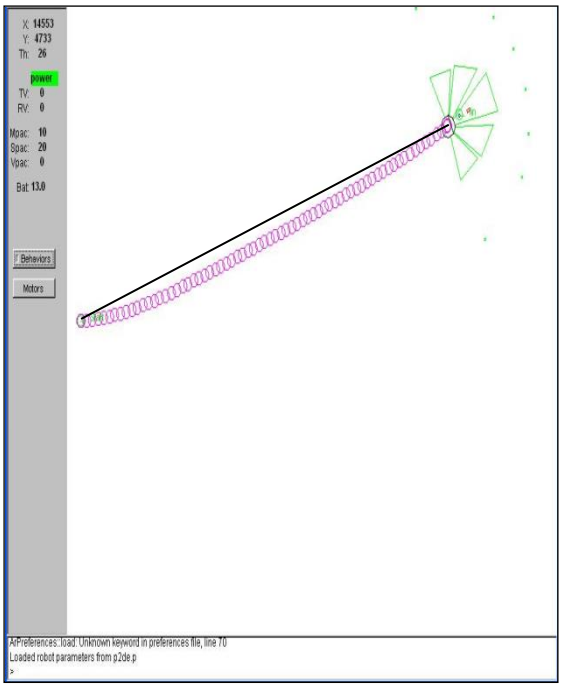


Figure III.29 : Trajectoire du robot avec FACL.

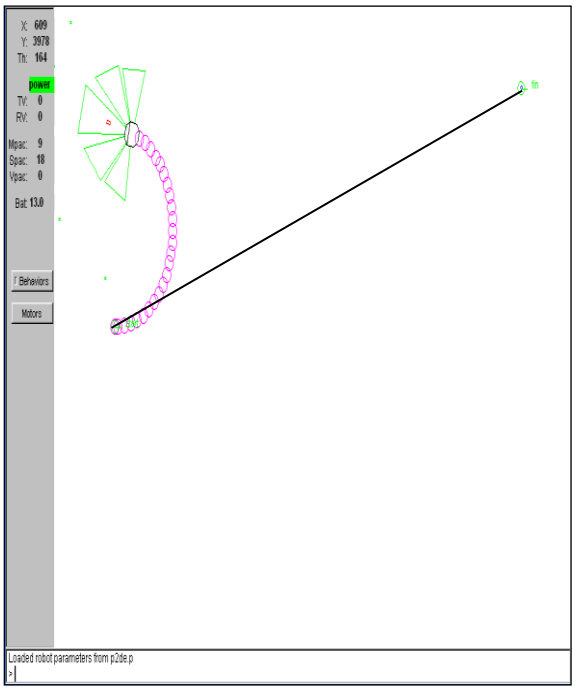


Figure III.30 : Trajectoire du robot avec FQL.

III.3.2.4 Discussion des résultats

Les simulations présentées ont montré l'influence du taux d'apprentissage et du facteur d'escompte. Le taux d'apprentissage affecte les valeurs des fonctions d'évaluation et les qualités des actions locales élues dans les règles floues tandis que le facteur d'escompte permet d'obtenir des valeurs finies lorsqu'il est inférieur à 1 et des valeurs non liées aux états précédents dans le cas $\gamma = 1$.

Il convient de noter que ces simulations montrent que la vitesse d'apprentissage dans le cas ou $\gamma < 1$ dépend directement de la valeur du taux d'apprentissageβ. Plus il est élevé (plus de 1) plus l'apprentissage est lent, en particulier avec l'algorithme FACL.

Grâce à l'utilisation de contrôleurs flou simple (convergence vers un objectif), nos expériences nous ont permis de mettre en évidence les principaux paramètres de ces méthodes (la vitesse d'apprentissage, facteur d'escompte, exploration / exploitation de la politique) sur la vitesse d'apprentissage et la performance finale de l'apprenti. Les traces d'éligibilité jouent un rôle important pour lequel elles ont été mises en place, à savoir la réduction du problème de la cession temporelle de crédits.

Un facteur de proximité élevé permet d'obtenir de meilleurs résultats d'apprentissage. Ces expériences ont également montré que l'utilisation de taux d'apprentissage adaptatifs dans FACL et FQL est nécessaire.

Enfin, ces résultats montrent que l'algorithme FACL donne de meilleurs résultats en termes de vitesse d'apprentissage par rapport à l'algorithme FQL alors que ce dernier représente une version compacte de FACL.

La supériorité de FACL provient probablement de la mise en œuvre du critique qui fournit à chaque pas de temps un critère plus riche, à travers la fonction d'évaluation, pour le choix des actions par rapport à celui représenté par les renforcements primaires.

Cependant, le choix de paramètres inappropriés de FACL entraîne également une dégradation des performances par rapport au FQL. Ce dernier n'est cependant pas dépourvu d'avantages : simplicité de mise en œuvre et choix restreint de ses paramètres.

III.4 Conclusion

Dans ce chapitre, nous avons utilisé deux méthodes d'apprentissage par renforcement, le Fuzzy Actor-Critic Learning (FACL) et le Fuzzy Q-Learning ((FQL). Ces dernières permettent l'adaptation d'apprentis de type SIF en réponse aux récompenses et punitions qu'ils reçoivent en cours d'interaction avec l'environnement. Les deux algorithmes FACL et FQL reposent sur la méthode de prédiction développée par Richard S. Sutton et sur les différences temporelles (TD). Elles constituent en fait des extensions de méthodes provenant de la communauté du Machine Learning, respectivement l'Adaptative Heuristic Critic (AHC) et le Q-Learning. En raison de la généralisation introduite par les sous-ensembles flous de l'apprenti, FACL et FQL rendent possible le traitement de systèmes ayant des états et des actions de type continu, et ce, avec une vitesse d'apprentissage accrue.

Dans le but de valider ces méthodes qui sont par nature heuristiques, nous avons effectué en première partie une phase d'expérimentations intensive sur des comportements élémentaires de navigation réactive du robot mobile Pioneer II : « Convergence vers un but », « Évitement d'obstacles ». Hormis l'aspect validation, ces expérimentations numériques nous ont permis de comparer les performances deux méthodes et de montrer que FACL et FQL sont à même de résoudre différentes classes de problèmes d'apprentissage par renforcement.

En outre, et afin de rendre possible la mesure de l'influence de chaque paramètre constituant ces méthodes nous avons effectué une étude comparative entre les deux algorithmes d'apprentissage par renforcement FACL et FQL. Nous avons simulé le comportement de "convergence vers l'objectif" d'un robot mobile à l'aide des deux algorithmes d'apprentissage par renforcement FQL et FACL. Les expériences effectuées ont montré qu'avec un bon choix de paramètres influençant l'apprentissage (vitesse d'apprentissage, facteur d'escompte, politique d'exploration et d'exploitation), le contrôleur -avec six règles floues- permet une convergence rapide vers l'objectif visé.

Ces résultats ont montré également la capacité de généralisation offerte par ces algorithmes d'apprentissage contrairement aux méthodes heuristiques de développement des systèmes d'inférences flous. Enfin, les derniers résultats montrent que l'algorithme FACL donne également des résultats satisfaisants, cependant un choix inapproprié des paramètres de cet algorithme entraîne une dégradation des performances par rapport à l'algorithme FQL.

Chapitre IV

Construction de carte 3D de l'environnement à l'aide de KINECT

La création commence par la vision.

Henri Matisse.

Chapitre IV

Construction de carte 3D de l'environnement à l'aide de KINECT

IV.1 Introduction

Pour jouir plus du confort et de sécurité, l'homme se fait appel aux robots pour intervenir dans différentes situations délicates : incendie, tremblement de terre, accident nucléaire, déminage, intervention policière, renseignements …etc. Dans toutes ces situations le robot vient en aide à l'homme en explorant des zones inaccessibles ou trop dangereuses.

Les robots doivent donc, d'une part, être suffisamment autonomes pour remplir leur mission sans la supervision de l'homme, et d'autre part être capables de fournir à l'utilisateur des informations sur la topographie du lieu, le plus souvent, sous la forme d'une carte.

Le problème de l'autonomie d'un robot dans la tâche d'exploration d'un environnement inconnu est appelé SLAM (Simultaneous Localization And Mapping). C'est un domaine de recherche très actif depuis plusieurs décennies et beaucoup de solutions existent pour la cartographie 2D. Le défi ici est de réaliser la modélisation en 3D de l'environnement en se basant uniquement sur les informations visuelles [78].

IV.2 Mise en œuvre de la plateforme expérimentale

L'objectif visé dans cette partie est de développer un module permettant de construire une carte de l'environnement d'exploration du robot. Autrement dit, le module doit permettre au robot de dresser la carte d'un environnement inconnu au fur et à mesure qu'il l'explore. Cette carte permettra à l'utilisateur humain de récupérer en temps réel des informations sur les lieux explorés par le robot.

À travers ce projet, nous voulons développer un programme qui permet de collecter les données RGB-D de la Kinect pour en faire une reconstruction 3D automatique et précise de l'environnement du robot en temps réel, c'est-à-dire pendant sa mobilité.

IV.2.1 Matériels et logiciels utilisés

Le matériel utilisé dans nos tests est constitué principalement d'une caméra RGB-D (la Kinect) installée sur le robot mBot (Annexe B) commandé à distance par notre programme par Bluetooth.

Nous avons aussi utilisé un PC avec un processeur Intel (R) CoreTM i3, une RAM de 6Go et un système d'exploitation 64 bits (Windows 10 professionnel). La figure IV.1 illustre la plateforme expérimentale utilisée pour valider notre algorithme.

Figure IV.1 : Constitution de la plateforme.

Le langage de programmation que nous avons utilisé est le Visual Studio C Sharp qui est un langage de programmation sous objets orientés.

IV.2.2 L'environnement d'évolution (Indoor)

Nous avons choisi comme environnement interne une salle de cour (R18C) dans le rez-de-chaussée du Bloc A au niveau de la faculté des hydrocarbures et de la chimie. La salle s'étale sur une superficie de 70 m² (12.5m de longueur et 5.5 m de largeur). La figure IV.2 présente quelques aperçus de la salle :

Figure IV.2 : L'environnement d'évolution du robot.

IV.3 Filtres linéaires

Un filtre linéaire transforme un ensemble de données d'entrée en un ensemble de données de sortie selon une opération mathématique appelée convolution. Lorsqu'il s'agit de données numérisées comme dans le cas du traitement d'image, la relation entre les valeurs des pixels de sortie et celle des pixels d'entrée est décrite par un tableau de nombres, généralement carré, appelé matrice de convolution. Le temps de calcul est souvent réduit lorsqu'on veut séparer un filtre en deux filtres dont la convolution mutuelle permet de le reconstituer. Cette remarque est utilisée en particulier pour créer un filtre à deux dimensions à partir de deux filtres à une seule dimension (vecteurs) dans le sens horizontal et le sens vertical.

IV.4 Extraction et association des points d'intérêt

Un point d'intérêt est un point qui doit avoir des caractéristiques plus significatives que les autres. Ces points correspondent à des changements bidimensionnels du signal. Nous pouvons en citer, les coins en 'L' et les points de jonction en 'T'. Beaucoup de travaux ont été menés pour l'extraction de ces points dans l'image. Les détecteurs de points d'intérêts ont plusieurs champs d'application. En effet, nous les retrouvons dans plusieurs domaines de la vision par ordinateur, que ce soit en reconnaissance des formes, ou encore dans la reconstruction 3D d'une scène.

Le détecteur d'Harris est un détecteur de coins (voir annexe). Dans l'image de profondeur acquise par la Kinect, la majorité des coins détectés par Harris disposent une profondeur nulle, donc ne sont pas fiables pour nos calculs, en plus il n'assure pas une bonne répétabilité des points d'intérêts entre les images successives. Tous ces inconvénients nous ont amené à choisir un autre détecteur de point d'intérêts robuste et efficace l'algorithme "SURF" car il est plus précis, il assure une bonne répétabilité des points.

IV.5 Algorithme adopté

Contrairement aux méthodes et aux techniques de reconstruction des cartes 3D de l'environnement d'un robot mobile développées jusqu'à présent, l'algorithme, que nous proposons et que nous avons développé sous le langage C Sharp, se base dans cet ordre sur les filtres SURF, RANSAC et le classifieur KNN et l'algorithme ICP (Figure IV.3). Ce choix va servir à minimiser une erreur globale qui est l'erreur moyenne de fusion des différents nuages de points captés pour chaque vue, et ainsi apporter plus de précision à notre reconstruction quelles que soient les transformations entre les vues.

Il existe plusieurs méthodes de détection/description de points d'intérêt (SIFT « Scale Invariant Feature Transform », SURF « Speeded Up Robust Feature », Harris…etc.) Nous avons choisi la méthode SURF pour sa robustesse, son efficacité, et son invariance face au changement d'échelle, rotation d'image, illumination et transformation affine (voir annexe). La détection par la méthode SURF se fait en trois étapes :

- Détection des extremas de la matrice hessienne.
- Localisation des points d'intérêts.
- Calcul du descripteur SURF

La détermination des points d'intérêt par l'algorithme SURF consiste à déterminer une relation entre les points détectés dans l'image à l'instant k et leurs correspondants de l'image à l'instant k+1, cette relation doit être robuste face au changement d'échelle, condition d'acquisition d'image, rotation, translation…etc.

Pour cela nous avons fait l'association par le descripteur SURF. Elle est basée sur la comparaison entre leurs vecteurs descripteurs. Si cette distance est inférieure à un certain seuil, les deux points d'intérêt sont associés. Plusieurs distances peuvent être utilisées, dans notre travail nous utilisons la distance euclidienne pour des raisons de simplicité.

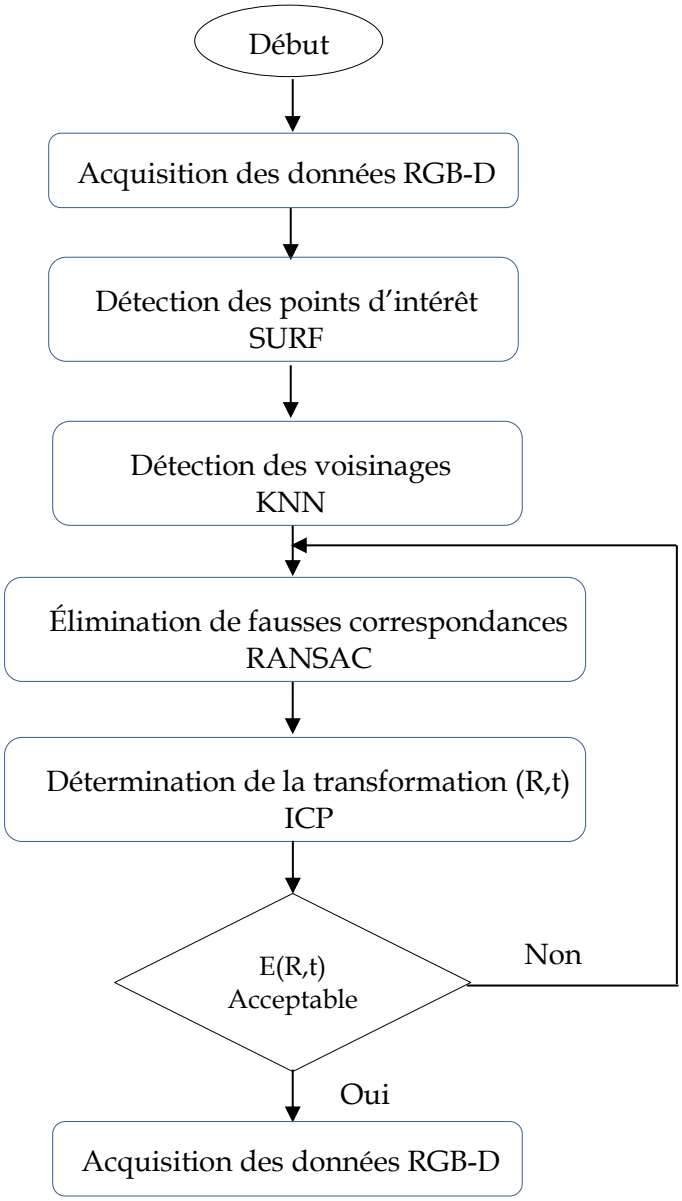


Figure IV.3 : Organigramme de l'algorithme de la reconstruction 3D appliqué.

IV.5.1 SURF

Nous avons appliqué l'algorithme SURF pour détecter les caractéristiques d'un descripteur. Il est partiellement inspiré du descripteur SIFT mais, il est plus rapide et plus robuste [79].

Dans le présent travail, nous utilisons SURF uniquement pour la détection des points d'intérêt.

La figure IV.4 présente un exemple d'application du SURF sur 2 images :

Figure IV.4 : Exemple d'application du SURF.

Les cercles représentent les points d'intérêt détectés et dont les cercles rouges indiquent les points clairs dans l'espace foncé, et inversement pour cercles bleus.

La taille du cercle présente l'échelle du point tandis que le trais vert présente son orientation.

IV.5.2 KNN

K points voisins les plus proches (k-nearest neighbor) est l'un de ces algorithmes qui sont très simples à comprendre, mais fonctionne incroyablement bien dans la pratique. En outre, il est étonnamment polyvalent et ses applications vont de la vision à des protéines par exemple, à la géométrie algorithmique et à des graphiques etc…

Dans notre cas, en appliquant KNN sur les points d'intérêt détectés par SURF avec K = 5, nous découvrons les cinq points d'intérêt les plus proches de chaque point d'intérêt de la première image dans la deuxième image et les relier, et vice-versa (voir annexe A).

La figure IV.5 représente un exemple d'application du KNN sur les points d'intérêt des deux images de la figure IV.4.

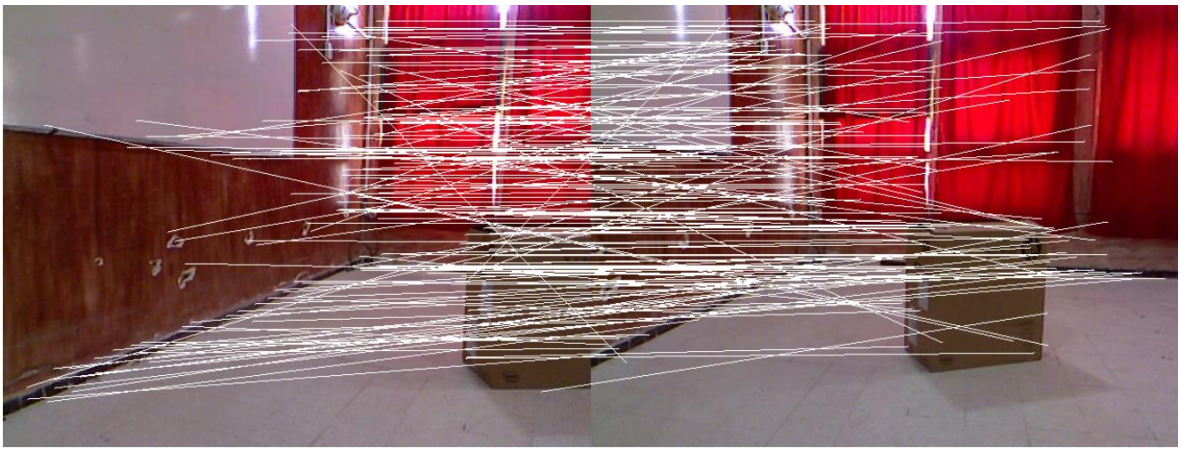


Figure IV.5 : Détermination des points d'intérêt avec KNN.

IV.5.3 RANSAC

RANSAC est une abréviation qui signifie « RANdom SAmple Consensus », c'est une méthode pour estimer les paramètres de certains modèles mathématiques. C'est une méthode itérative utilisée, généralement, lorsque l'ensemble de données observées peut contenir des valeurs aberrantes (outliers) (voir annexe). Il s'agit d'un algorithme non-déterministe dans le sens où il produit un résultat correct avec une certaine probabilité, et donne des résultats très approchés lorsque le nombre d'itérations est élevé [80].

Alors, après KNN, nous utilisons RANSAC pour éliminer les fausses correspondances entre les points d'intérêt des deux images et pour estimer la transformation entre les deux images [79].

La figure IV.6 représente un exemple d'application du RANSAC sur les points d'intérêt des deux images de la figure IV.4.

Figure IV.6 : Élimination des fausses correspondances avec le RANSAC.

IV.5.4 Algorithme ICP

L'algorithme ICP (Iterative Closest Point) est une méthode dite "itérative" de recalage 3D. Cet algorithme consiste à calculer, de façon itérative, la matrice de transformation recalant le mieux deux (ou plusieurs) ensembles de données 3D (points 3D, courbes ou surfaces).

Une connaissance approximative de cette matrice est, par ailleurs, nécessaire, pour son initialisation.

Le principe, simple, de cet algorithme est d'itérer les deux étapes du recalage : la mise en correspondance des données et l'estimation de la transformation de repères entre les nuages 3D à recaler. Au bout de chaque itération, l'algorithme fournit une liste de points appariés et une estimation de la transformation de repère entre les données à recaler. Cette transformation est utilisée dans l'itération suivante pour la mise à jour de la liste des points appariés. Ces derniers serviront, à leur tour, pour calculer une nouvelle estimation de la transformation. Ces étapes sont répétées jusqu'à la convergence de l'algorithme. Celui-ci converge lorsque l'erreur résiduelle de distance entre les points appariés est inférieure à un certain seuil.

Il existe plusieurs variantes de cet algorithme. En effet, de nombreuses améliorations ont été apportées pour chaque étape de celui-ci [80].

Dans notre travail, nous avons les deux nuages de points à fusionner et la correspondance entre eux (le résultat de RANSAC). Donc, on peut utiliser une variante simple de l'ICP pour estimer la transformation entre les deux nuages.

Les principales étapes de l'algorithme ICP sont : [78] [79]

1. À partir de deux ensembles de points correspondants

$$X = \{x_1, \dots \dots, x_n\}$$
$$P = \{p_1, \dots \dots, p_n\}$$

2. Calculer le centre de masse de chaque nuage

$$\mu_x = \frac{1}{N_x}\sum_{i=1}^{N_x} x_i \qquad \mu_p = \frac{1}{N_p}\sum_{i=1}^{N_p} p_i \qquad \text{IV.1}$$

3. Calcul de la soustraction entre les cordonnées des centres de masse de chaque nuage de points avec tous les points dans les deux ensembles de points avant de calculer la transformation.
4. Les ensembles de points qui en résultent sont :

$$X' = \{x_i - \mu_x\} = \{x_i'\}$$
$$P' = \{p_i - \mu_x\} = \{p_i'\} \qquad \text{IV.2}$$

5. On calcule la matrice :

$$W = \sum_{i=1}^{N_p} x'_i {p'_i}^T \qquad \text{IV.3}$$

6. La décomposition en valeurs singulières (SVD) de W est effectuée comme suit:

$$W = U\begin{bmatrix}\sigma_1 & 0 & 0\\ 0 & \sigma_2 & 0\\ 0 & 0 & \sigma_3\end{bmatrix}V^T \qquad \text{IV.4}$$

où $U, V \in R^{3*3}$ sont unitaires, et $\sigma_1 \geq \sigma_2 \geq \sigma_3$ sont les valeurs singulières de W.

7. La matrice de rotation R et le vecteur de translation *t* sont :

$$R = UV^T$$
$$t = \mu_x - R\mu_p \qquad \text{IV.5}$$

8. Calcul du nouveau nuage P'' :

$$P'' = Rp_i + t \qquad \text{IV.6}$$

9. Calcul de la valeur minimale de la fonction d'erreur à (R, t) comme suit :

$$E(R,t) = \sum_{i=1}^{N_p}(\|x'_i\|^2 + \|y'_i\|^2 - 2(\sigma_1 + \sigma_2 + \sigma_3) \qquad \text{IV.7}$$

10. Si E n'est pas acceptable il faut reprendre à partir de l'étape 1 avec les nuages X et P''

Pour les étapes 8 et 9, nous avons implémenté une erreur globale à minimiser :

$$E(R,t) = \frac{1}{N_p}\sum_{i=1}^{N_p}\left\|x_i - Rp_i - t\right\| \qquad \text{IV.8}$$

Si E n'est pas acceptable on refait RANSAC et ICP pour les deux nuages X et P, parce que RANSAC choisit les points aléatoirement pour l'estimation de la transformation entre eux ce qui donne des résultats différents à chaque fois qu'on le ré exécute.

Donc l'idée principale est d'augmenter le nombre d'itérations de RANSAC au lieu d'augmenter celle d'ICP pour augmenter la probabilité d'avoir une correspondance plus juste des points d'intérêt.

IV.6 Description de l'interface du programme développé

Le programme développé dans ce mémoire et que nous appelons « Kinect 3D Application », est un programme écrit en langage C Sharp sous l'environnement Windows, intégrant tous les algorithmes et procédures développés (connexion avec la Kinect, acquisition des données RGB-D, application des filtres etc...). Son interface conviviale permet l'interaction avec l'utilisateur à travers un ensemble d'onglets, de boutons et de boîtes de dialogue qui accordent le choix et la variation des paramètres techniques liés à l'application.

IV.6.1 L'onglet Welcome

Figure IV.7: L'onglet Welcome de l'interface développé.

- Le bouton Start permet de se connecter et se déconnecter à la Kinect, comme il permet de choisir une seule caméra s'il y en a plusieurs. Le label au-dessus affiche le Statut de la Kinect.
- Le bouton Exit déconnecte la Kinect et ferme le programme.

IV.6.2 L'onglet RGB View

L'onglet RGB View (Figure IV.8) sert à afficher les données RGB de la Kinect en temps réel.

Figure IV.8: L'onglet RGB View.

- Le bouton Snap Shot permet de capter une image RGB et l'enregistrer dans un fichier JPEG ou Bitmap.

IV.6.3 L'onglet Depth View

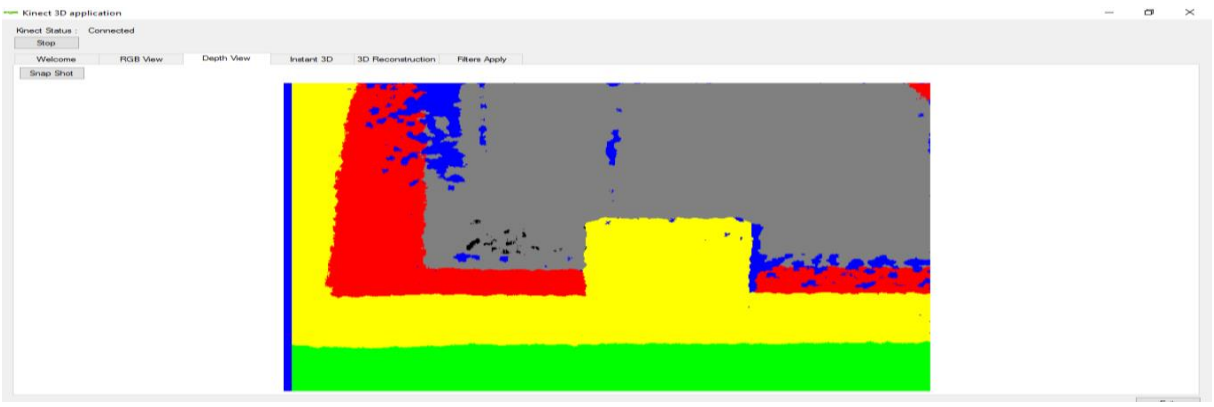


Figure IV.9: L'onglet Depth View.

- Le bouton Snap Shot permet la capture d'une image colorée du Depth et l'enregistrer dans un fichier JPEG ou Bitmap.

L'onglet DepthView (Figure IV.9) affiche les données de profondeur de la Kinect en temps réel sous forme de couleurs indiquant la distance des points par rapport à la Kinect, par le biais d'un code que nous avons inséré dans notre programme, comme cité dans l'algorithme de la Figure IV.10.

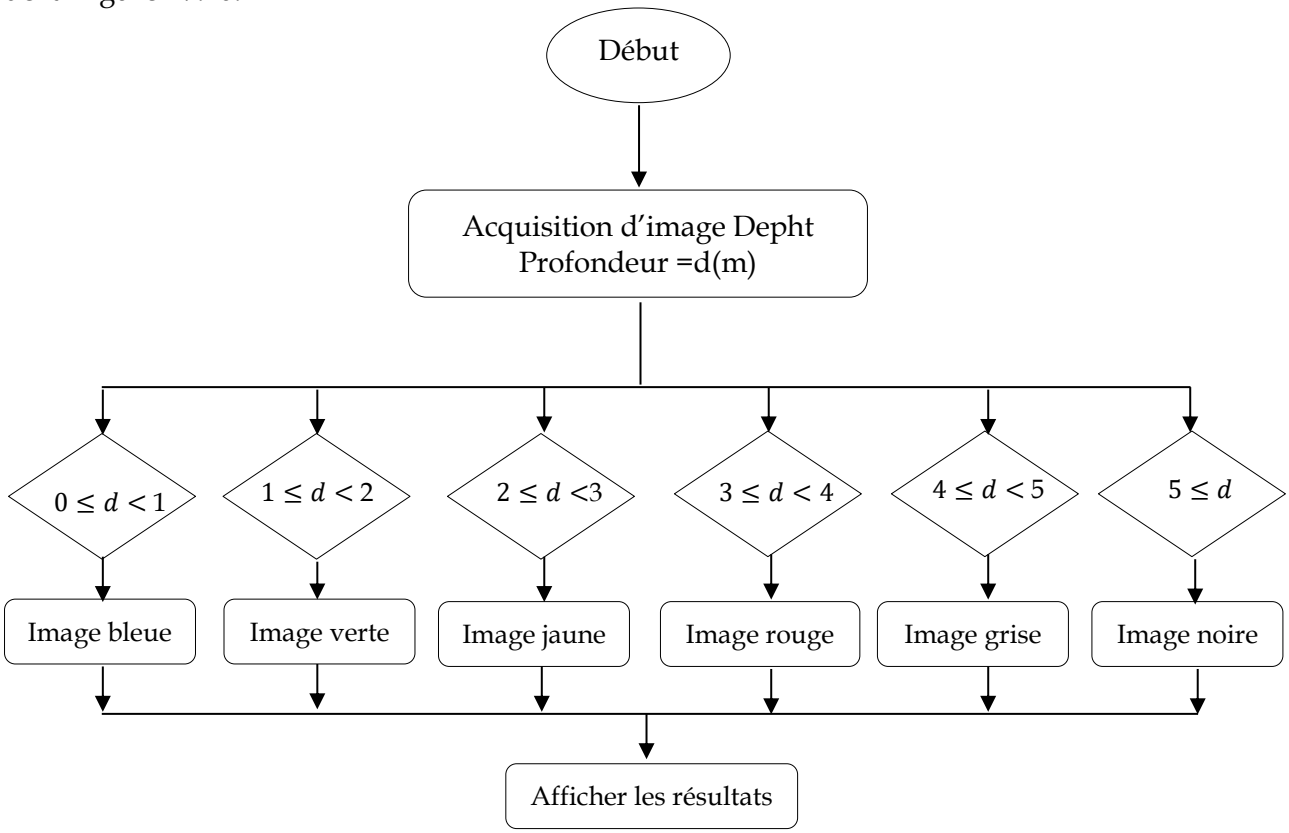


Figure IV.10 : Organigramme de la vision du Depth par la Kinect.

Le tableau IV.1 indique le code des couleurs utilisées.

La distance *d (m)*	$0 \leq d < 1$	$1 \leq d < 2$	$2 \leq d < 3$	$3 \leq d < 4$	$4 \leq d < 5$	$5 \leq d$
La couleur	Bleue	Verte	Jaune	Rouge	Grise	Noire

Tableau IV.1 : Code couleur du DepthView.

IV.6.4 L'onglet Instant 3D

Cet onglet permet de réaliser une reconstruction 3D instantanée au moment même de l'affichage de la vue (Figure IV.12).

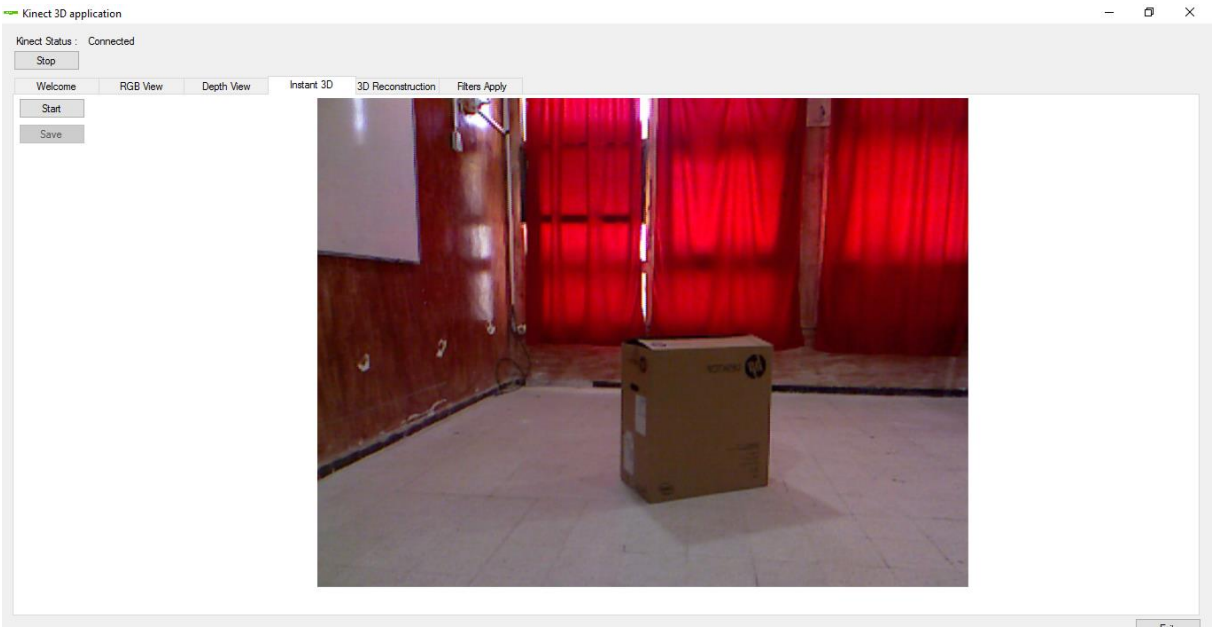


Figure IV.12 L'onglet Instant 3D.

- Le bouton Start permet de démarrer cette reconstruction, l'afficher et l'arrêter.
- Le bouton Save sauvegarde les données RGB et XYZ (les coordonnées cartésiennes) de chaque point de la vue dans un fichier texte.

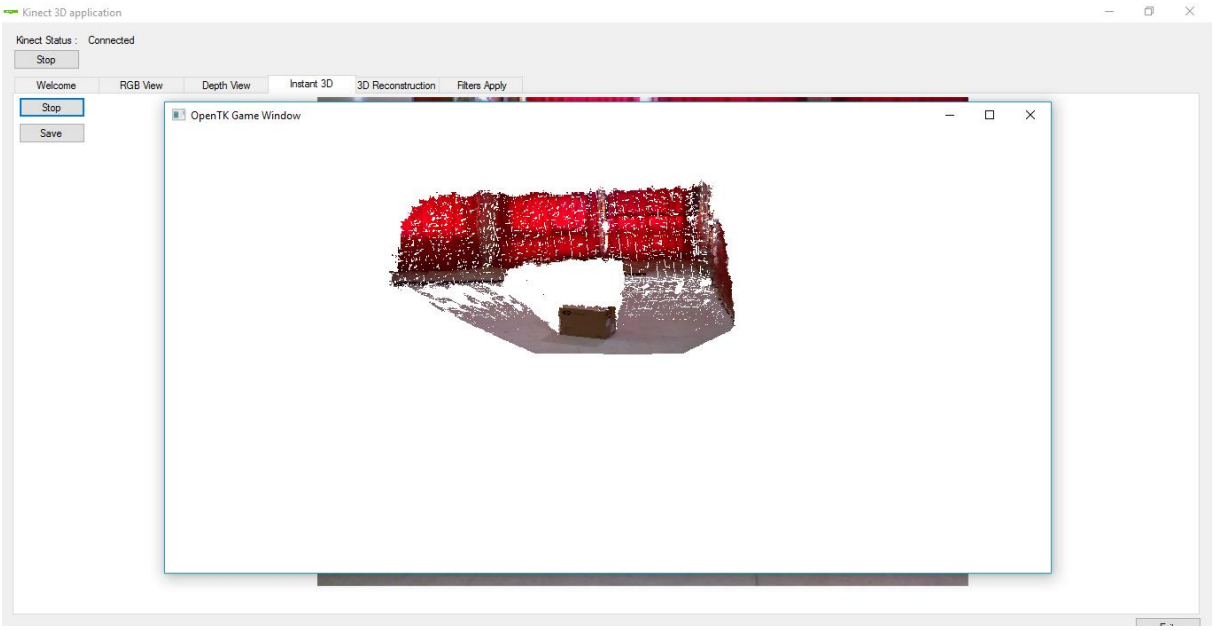


Figure IV.13 : Résultat d'une reconstruction 3D instantanée.

IV.6.5 L'onglet 3D Reconstruction

C'est l'onglet fondamental dans notre travail puisqu'il nous permet de réaliser notre objectif : faire le recalage de plusieurs vues successives (Figure IV.14).

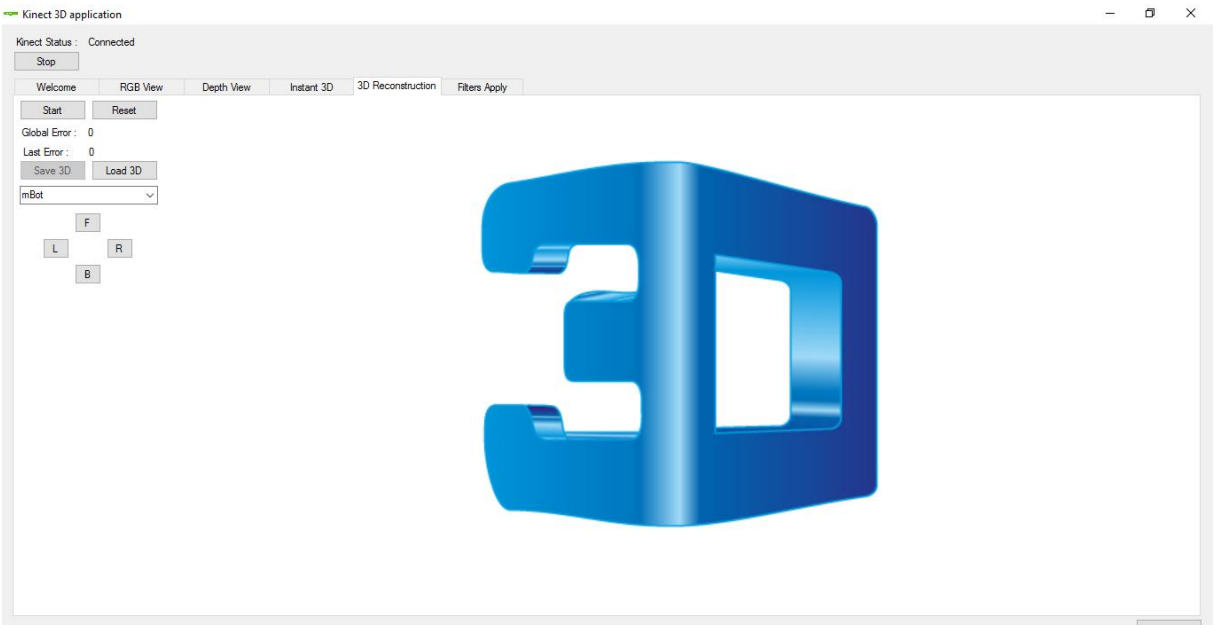


Figure IV.14 : L'onglet 3D Reconstruction.

- Le bouton Start permet de démarrer et d'arrêter l'algorithme (présenté dans IV.3), ainsi que d'afficher les résultats.
- Le bouton Reset est utilisé pour recommencer une nouvelle reconstruction 3D.
- Le label au-dessous affiche l'erreur moyenne de la fusion de toutes les vues et l'erreur moyenne de la fusion de la dernière vue.
- Le bouton Save 3D sauvegarde les données RGB et XYZ de chaque point de la reconstruction dans un fichier texte comme le bouton Load 3D recharge ces données depuis le fichier texte et les affiche.
- Le comboBox au-dessous affiche les périphériques Bluetooth et permet de se connecter avec mBot.
- Les boutons F et B envoient des commandes à mBot pour avancer ou reculer tandis que les boutons R et L envoient des commandes pour une rotation à droite ou à gauche du robot.

IV.6.6 L'onglet Filters Apply

Cet onglet permet de faire l'acquisition des données RGB-D de deux vues, leur appliquer les filtres SURF, KNN, RANSAC et afficher les résultats obtenus. Comme il applique la fusion 2D (2D Blend) ou 3D à travers l'algorithme ICP.

On observe les résultats successivement comme suit :

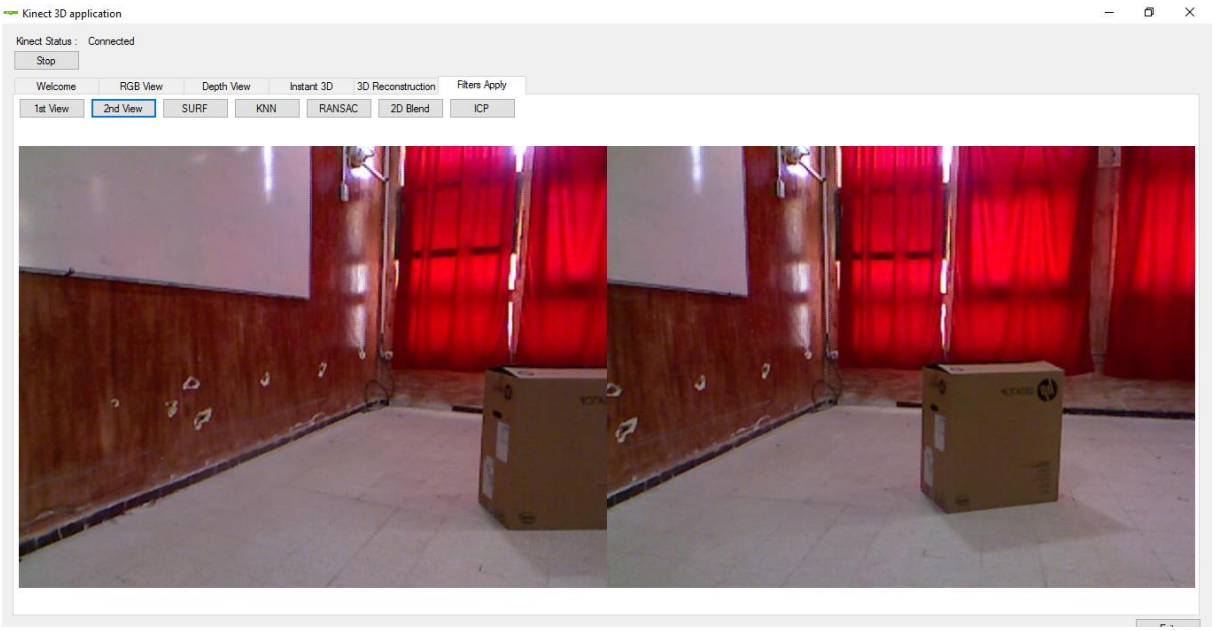


Figure IV.15 Acquisition des données de deux vues.

- Le bouton 1st View capte la première vue.
- Le bouton 2nd View capte la deuxième vue.

Figure IV.16 : Application du SURF.

- Le bouton SURF applique et affiche les résultats du filtre SURF : extraction des points d'intérêt et calcule leurs descripteurs.

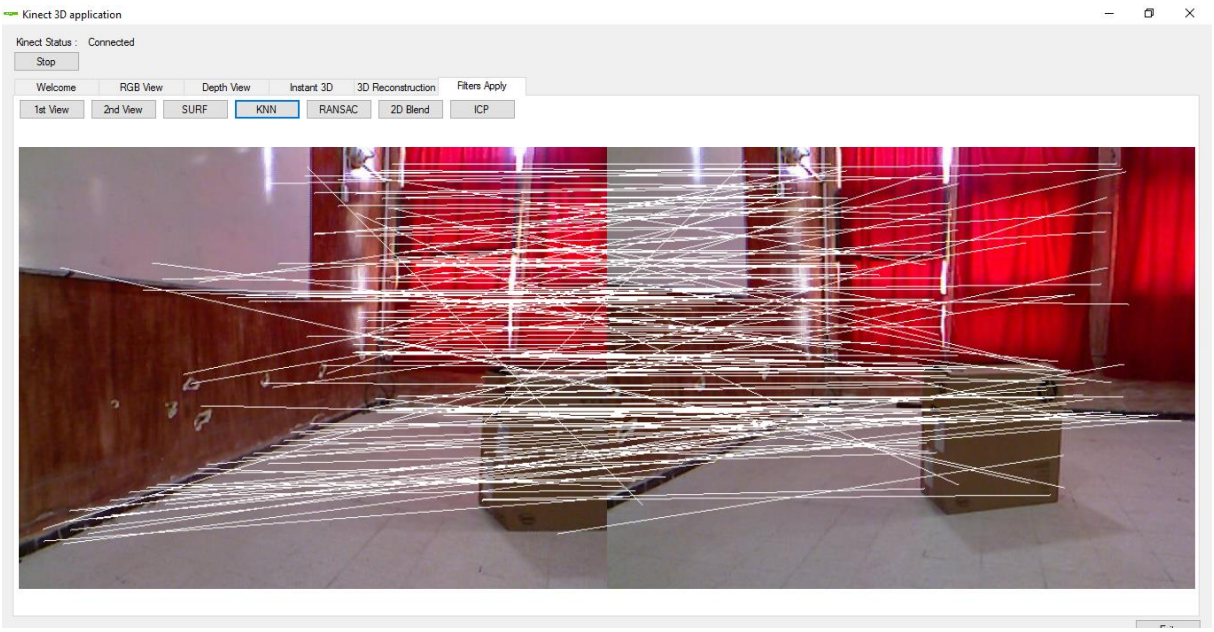


Figure IV.17 : Application du KNN.

- Le bouton KNN applique et affiche les résultats du filtre KNN : permet de relier chaque point d'intérêt de la première vue avec les 5 points d'intérêt voisins les plus proches de la deuxième. Et vice-versa.

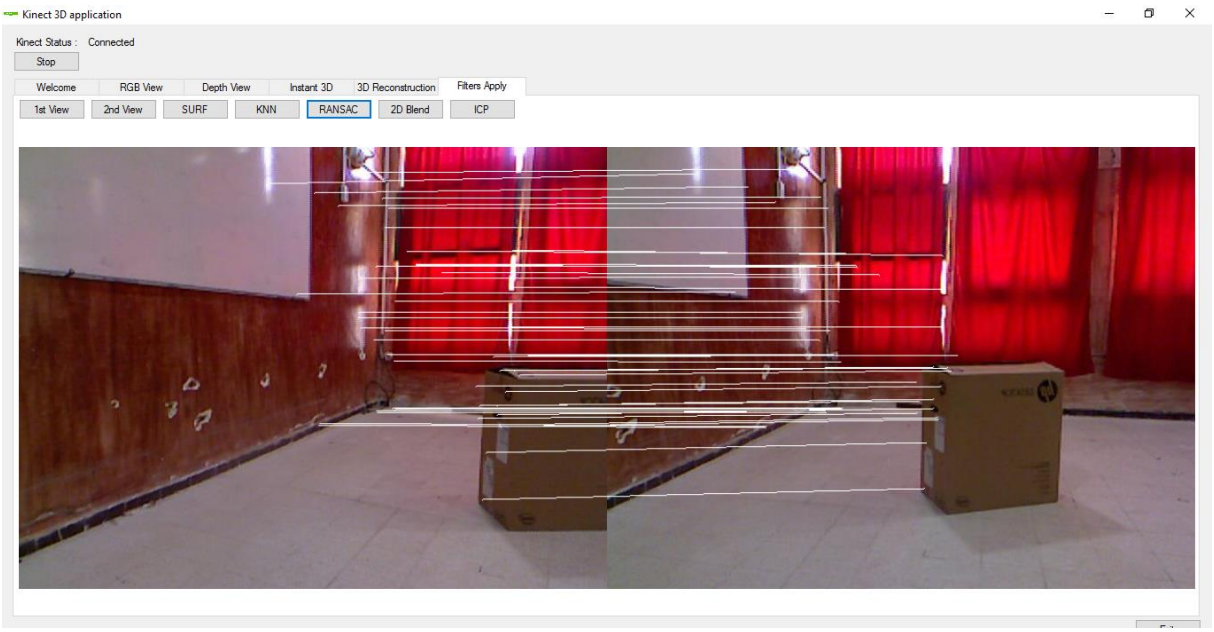


Figure IV.18 Application du RANSAC.

- Le bouton RANSAC applique et affiche les résultats du filtre RANSAC : permet d'éliminer les fausses liaisons et maintenir les liaisons les plus justes.

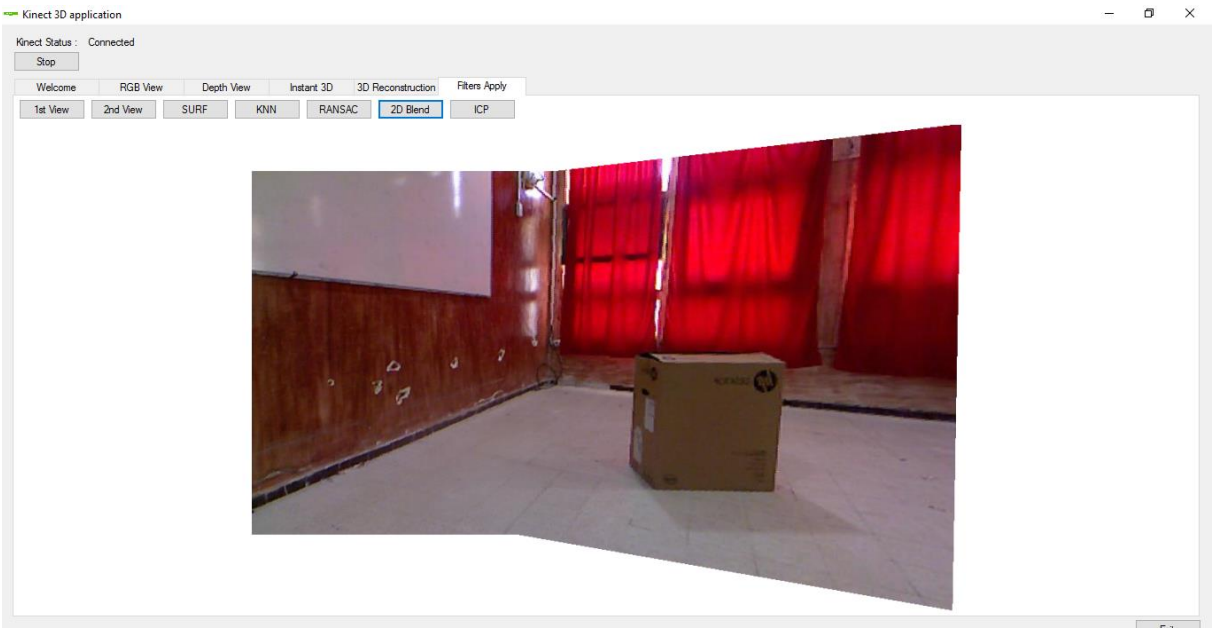


Figure IV.19 : La fusion 2D des deux vues.

- Le bouton 2D Blend fusionne les 2 vues en 2D selon les résultats du RANSAC.

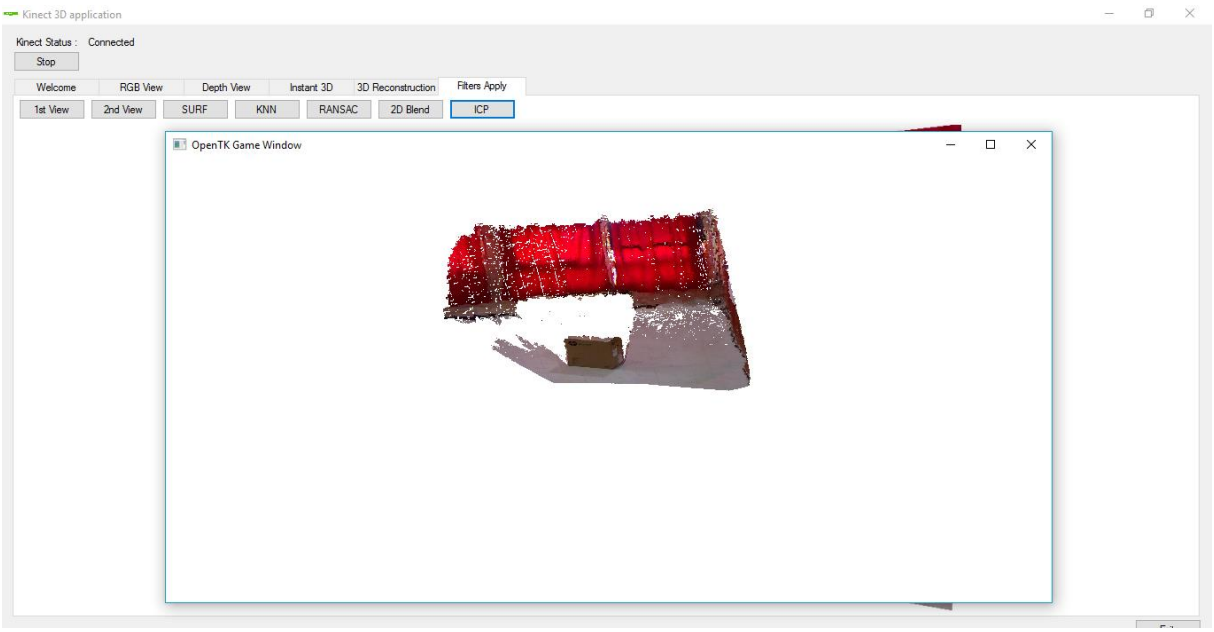


Figure IV.20 : La fusion 3D des deux vues.

- Le bouton ICP fusionne les 2 vues en 3D en appliquant l'algorithme ICP et affiche les résultats.

IV.7 Séries de reconstruction 3D

Dans ce qui suit, nous allons présenter les résultats de 2 exemples d'application de notre programme.

IV.7.1 Carte 3D d'une salle sans obstacles

Figure IV.21 (a) : Les différentes phases de la reconstruction 3D de la salle sans obstacles.

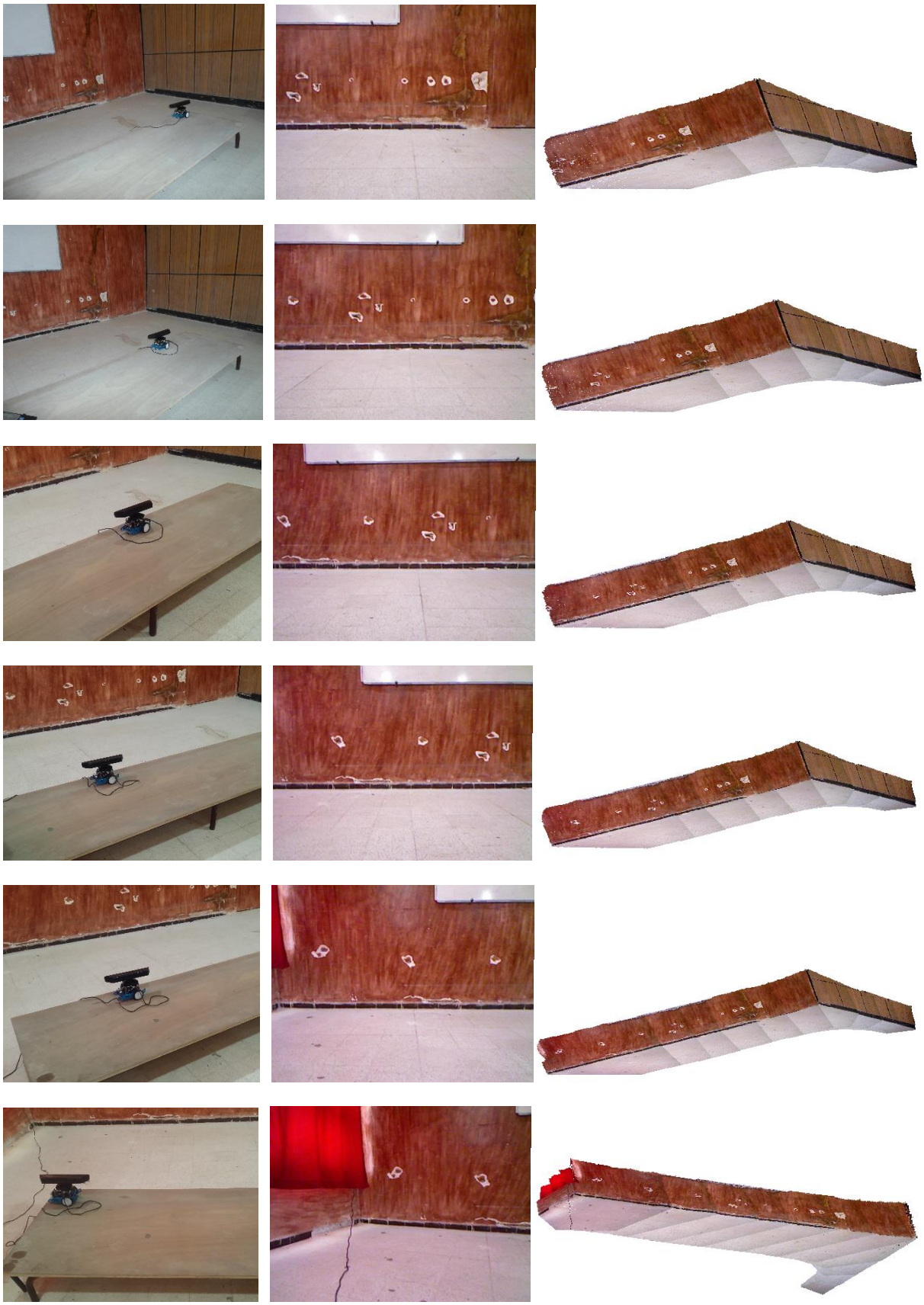


Figure IV.21 (b) : Les différentes phases de la reconstruction 3D de la salle sans obstacles.

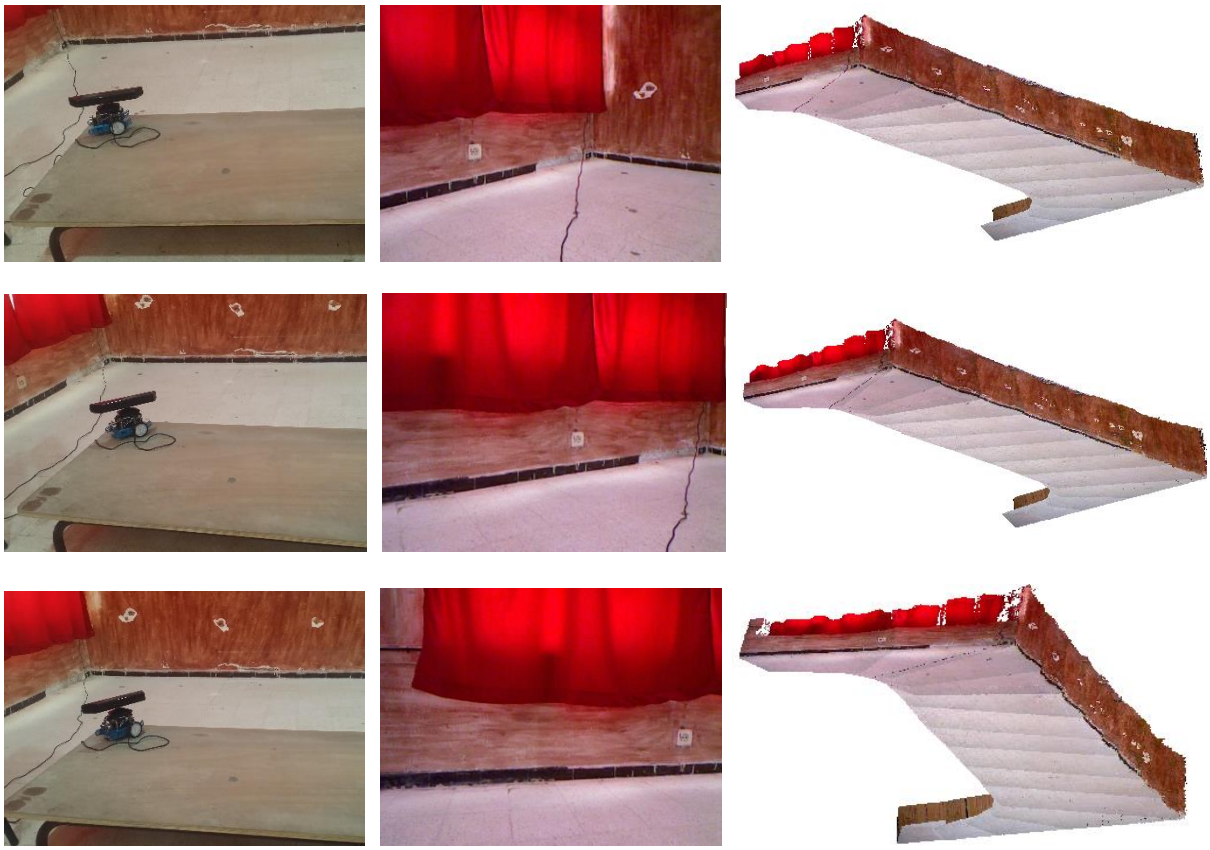


Figure IV.21 (c) Les différentes phases de la reconstruction 3D de la salle sans obstacles.

- **La vue globale de la reconstruction**

La figure IV.22 représente la vue globale obtenue à la fin de processus de reconstruction de la carte de l'environnement d'évolution du robot mobile.

On remarque l'algorithme appliqué fournit une carte fidèle de l'environnent du test.

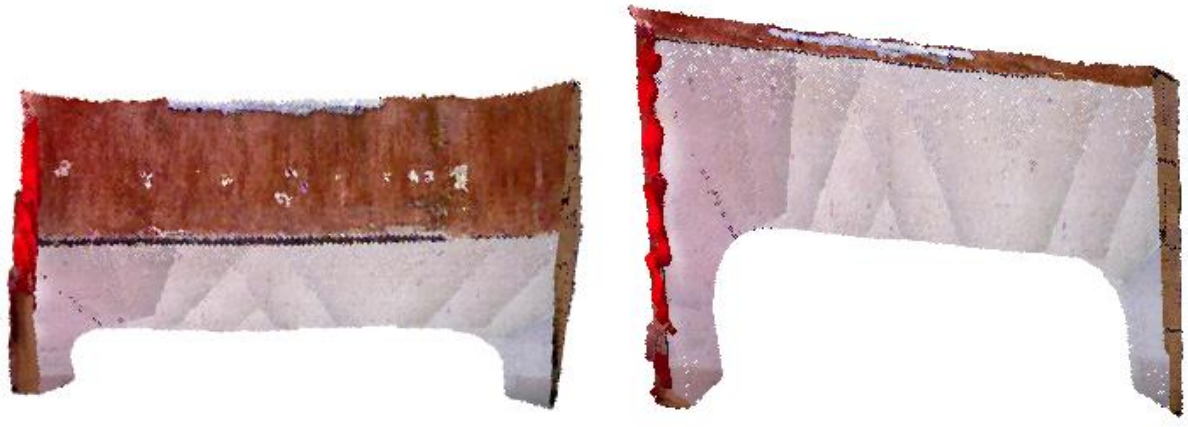


Figure IV.22 : La vue globale de la reconstruction.

L'erreur moyenne globale de fusion des vues obtenue est de : 13.021 mm.

Le tableau suivant (Tableau IV.2) montre l'erreur moyenne de fusion de chaque vue et l'évolution de l'erreur moyenne globale :

Vue N°	L'erreur moyenne de fusion avec la vue précédente (mm)	L'erreur moyenne globale de fusion de toutes les vue
1	0	0
2	6.087	6.087
3	10.945	8.516
4	18.011	11.681
5	8.053	10.774
6	11.419	10.903
7	15.847	11.727
8	15.353	12.245
9	16.749	12.808
10	15.724	13.132
11	12.842	13.103
12	19.824	13.714
13	10.486	13.445
14	7.933	13.021

Tableau IV.2 : Évaluation de l'erreur moyenne de la fusion des vues et l'erreur moyenne globale.

On remarque que l'erreur moyenne globale augmente, ceci étant dû au nombre de vues. Cette erreur ne dépasse pas la valeur de 30 mm fixée par programmation comme un seuil (Figure IV.3).

IV.7.2 Carte 3D d'une salle avec obstacle

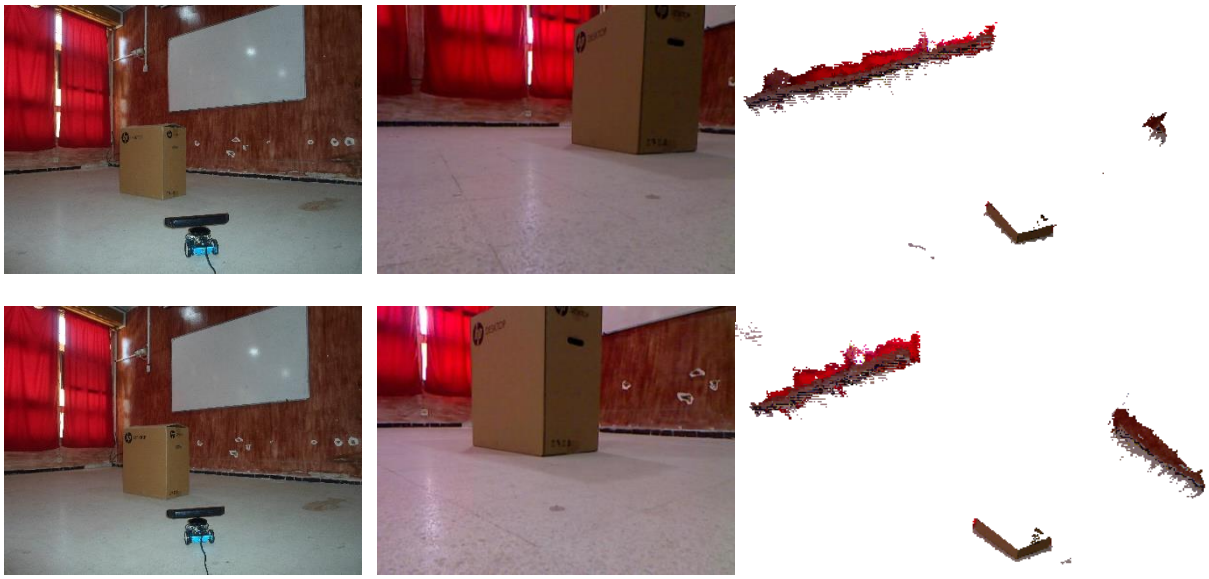


Figure IV.23 (a) : Les différentes phases de la reconstruction 3D de la salle avec obstacle.

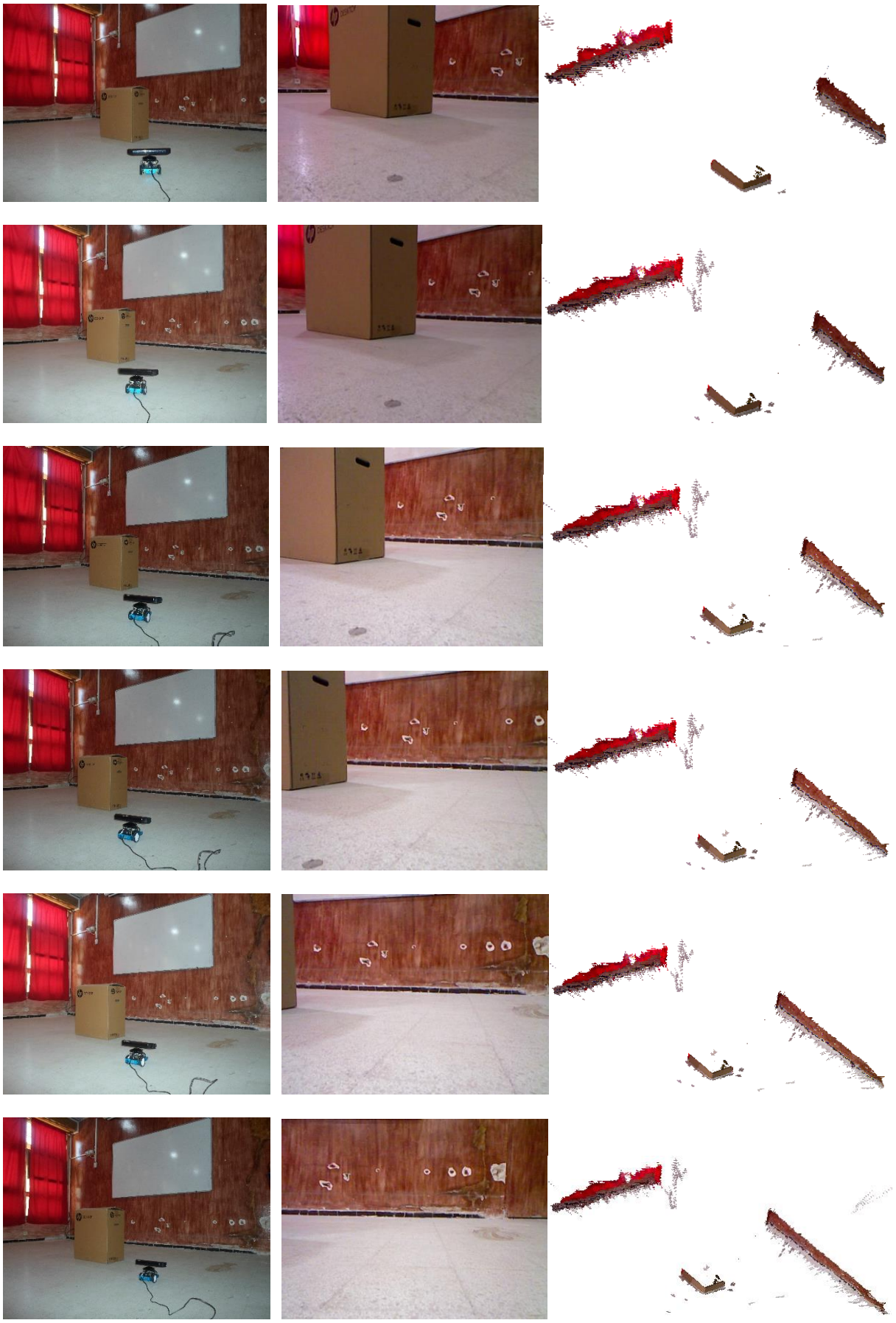


Figure IV.23 (b) Les différentes phases de la reconstruction 3D de la salle avec obstacle.

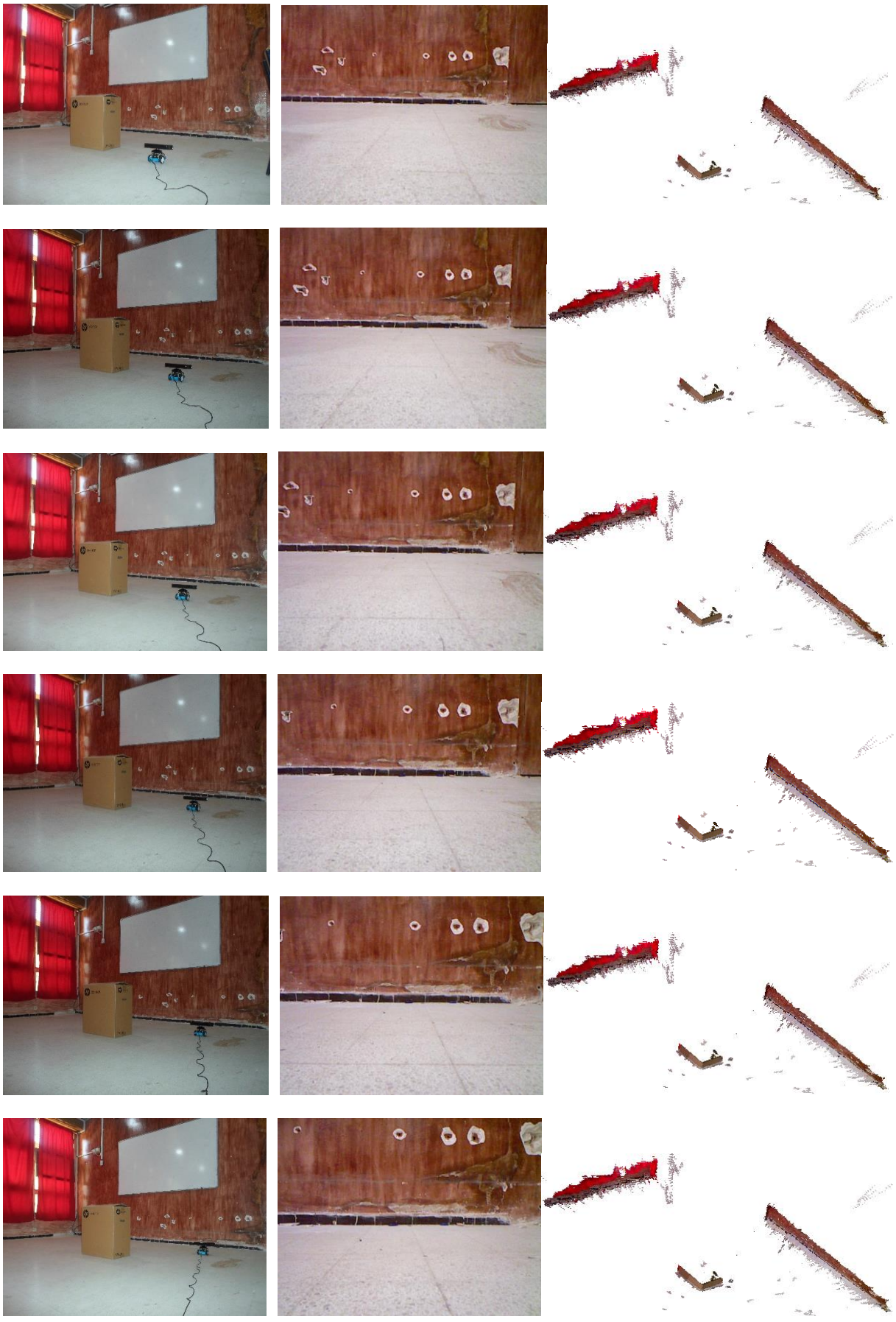


Figure IV.23 (c) : Les différentes phases de la reconstruction 3D de la salle avec obstacle.

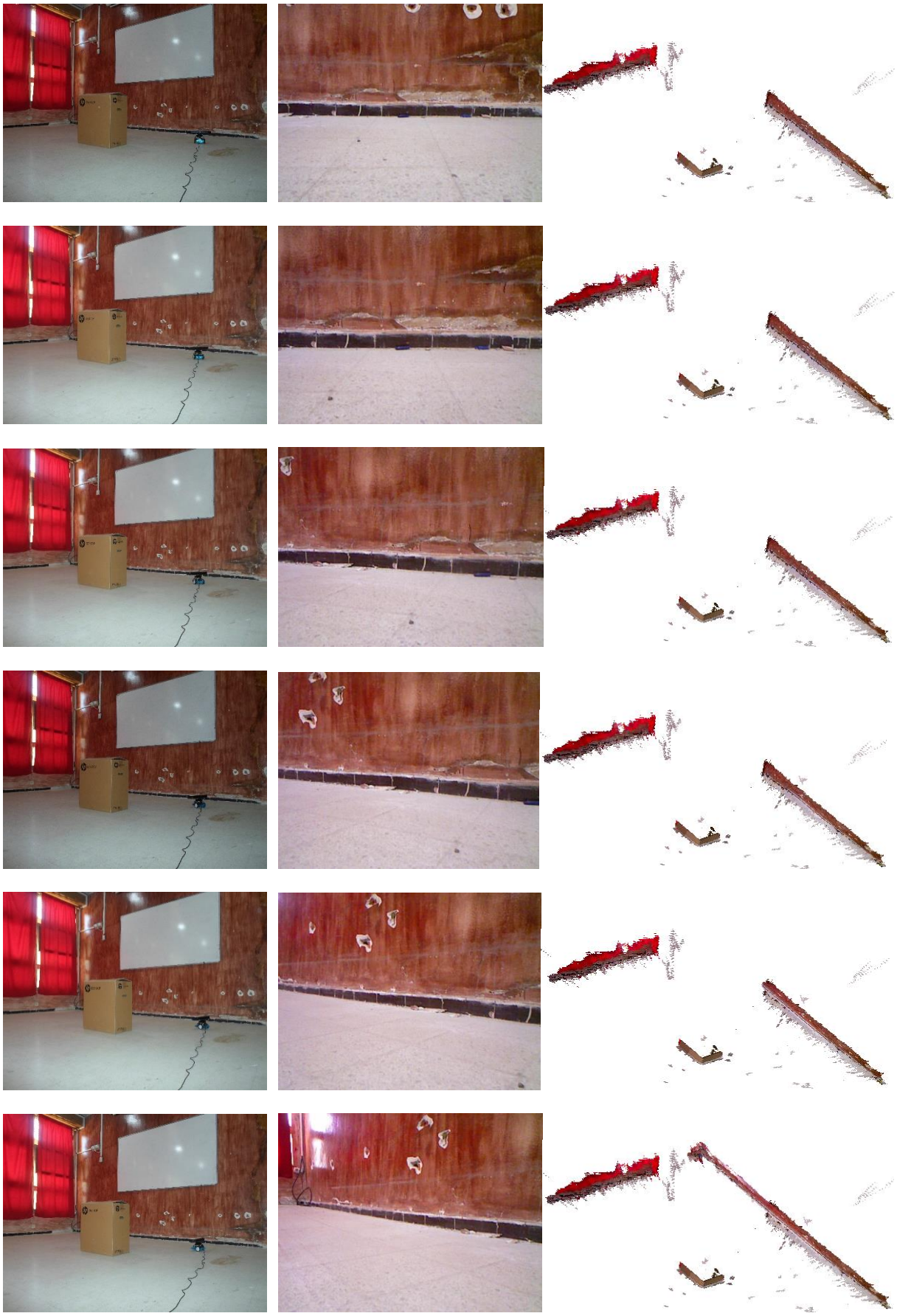


Figure IV.23 (d) : Les différentes phases de la reconstruction 3D de la salle avec obstacle.

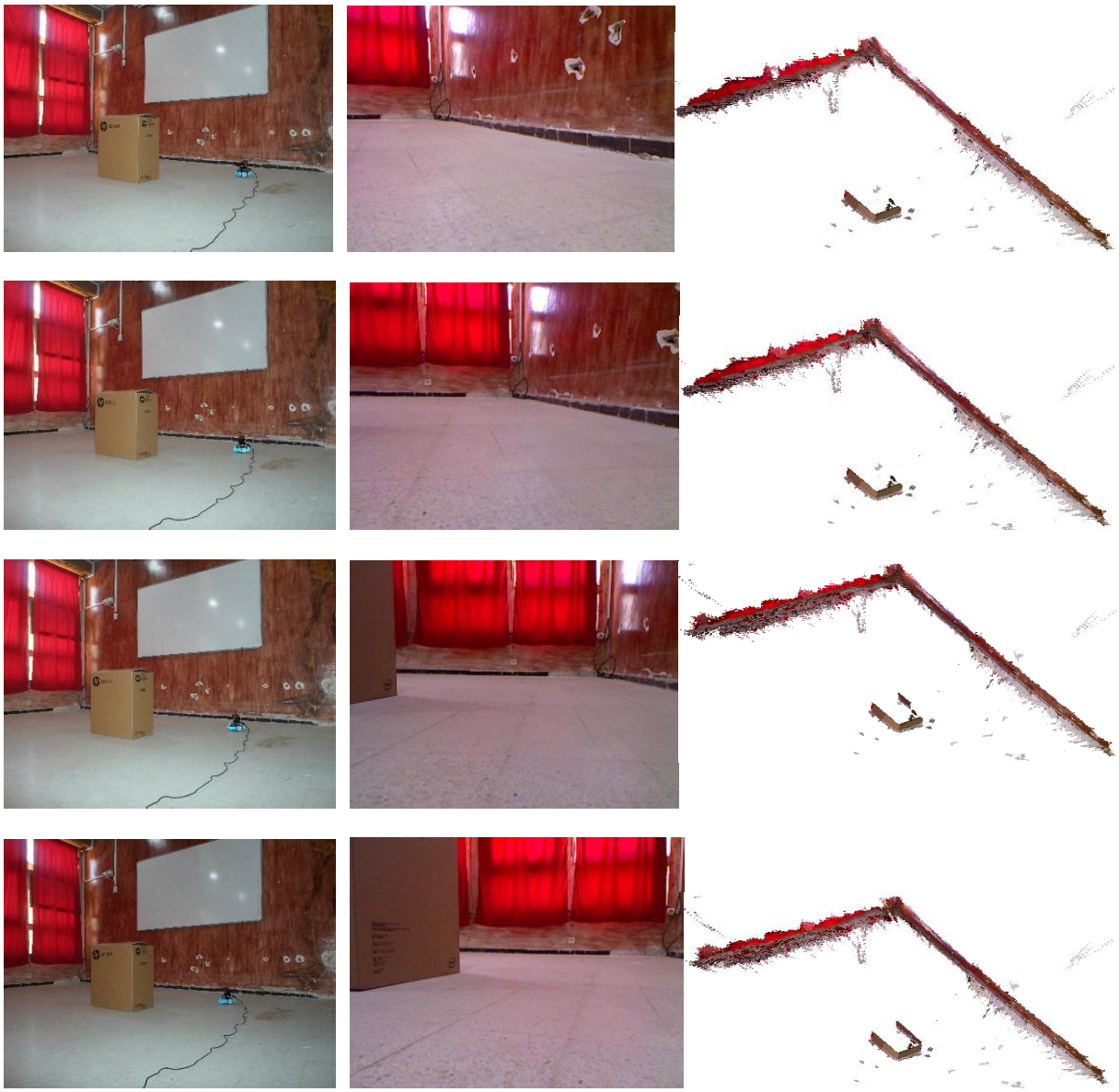


Figure IV.23 (e) : Les différentes phases de la reconstruction 3D de la salle avec obstacle.

- **La vue globale de la reconstruction**

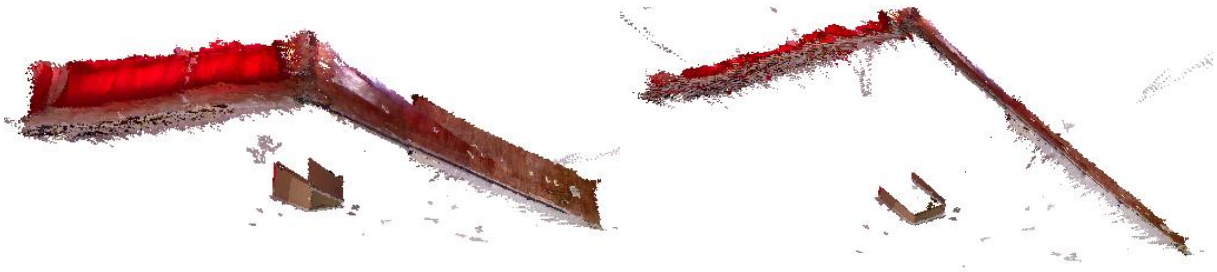


Figure IV.24 La vue globale de la reconstruction.

L'erreur moyenne globale de fusion des vues obtenue est de : 14.701 mm.

Le tableau suivant (Tableau IV.3) montre l'erreur moyenne de fusion de chaque vue et l'évolution de l'erreur moyenne globale :

Vue N°	L'erreur moyenne de fusion avec la vue précédente (mm)	L'erreur moyenne globale de fusion de toutes les vue
1	0	0
2	29.963	29.963
3	28.319	29.141
4	23.775	29.095
5	21.401	27.172
6	18.791	25.496
7	17.546	24.171
8	18.172	23.314
9	8.290	21.436
10	10.108	20.177
11	8.894	19.049
12	8.043	18.048
13	6.621	17.096
14	5.335	16.191
15	4.192	15.334
16	2.830	14.501
17	2.968	13.780
18	3.908	13.199
19	4.201	12.699
20	6.705	12.384
21	16.178	12.573
22	27.385	13.279
23	29.860	14.032
24	29.437	14.701

Tableau IV.3 : Évaluation de l'erreur moyenne de la fusion des vues et l'erreur moyenne globale.

IV.8 Discussion des résultats

Les résultats présentés précédemment montrent que l'algorithme utilisé pour fournir la carte d'environnement en 3D donne des résultats acceptables.

L'erreur moyenne de fusion qui a un minimum de 2.830 mm et ne dépasse pas 29.963 mm. Cette erreur est très acceptable en robotique mobile et surtout dans la navigation autonome.

Les traits qui apparaissent dans la reconstruction de la figure IV.25 indiquent les champs de vision de la Kinect. Ceci apparait dans la fusion des vues uniquement mais affectent pas sur les vecteurs X, Y et Z de la reconstruction.

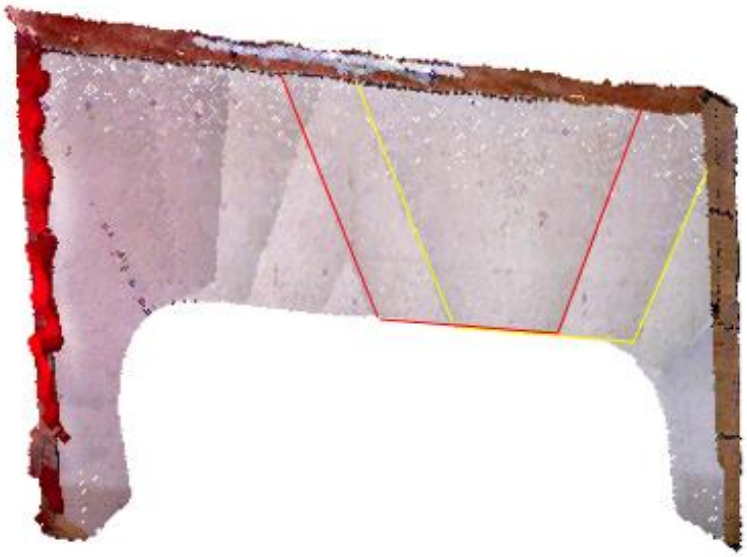


Figure IV.25 : Indication de 2 champs de vision de la Kinect dans la reconstruction.

Dans le début de la reconstruction de la salle avec obstacle, l'obstacle dans le milieu de la salle coupe le champ de vision de la Kinect, ce qui explique l'absence de la reconstruction derrière le carton comme le montre la figure IV.26.

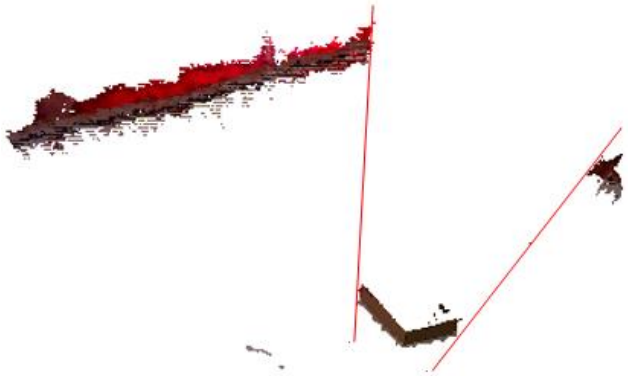


Figure IV.26 : Le champ de vision de la Kinect coupé par un obstacle.

Après la navigation du robot, l'algorithmes appliqué complète la reconstruction de la partie située derrière l'obstacle (figure IV.27).

Figure IV.27 : Reconstruction de la partie manquante après la navigation.

IV.9 Conclusion

Dans ce dernier chapitre, nous avons évalué et validé notre algorithme de cartographie 3D proposé expérimentalement en utilisant un capteur Kinect embarqué sur le robot mBot. L'algorithme présenté est implémenté en utilisant le Visuel C#. Les tests ont été effectués au niveau de la salle R18C dans notre faculté.

Les résultats expérimentaux montrent qu'on obtient des très bons résultats de l'algorithme de traitement, et cela même si on augmente la complexité de la scène observée (avec obstacles).

Toutes les solutions intelligentes ont des avantages et des inconvénients, la combinaison de plusieurs méthodes nous a permis non seulement de surpasser les inconvénients d'une méthode ou d'une autre, mais aussi d'implémenter un savoir-faire de haut niveau pour un robot (dans le cas de la robotique). Dans le cas générale les solutions basées sur cette approche permettent la résolution des problèmes plus complexes, il suffit de conduire une analyse exacte du problématique.

De nombreuses améliorations restent à faire dans le domaine de la cartographie 3D. En premier lieu, concernant l'algorithme décrit, des efforts doivent être fait pour simplifier les modèles et décroître les coûts de calcul. En ce qui concerne le filtrage, les approximations faites sur les modèles et les perturbations suggèrent l'emploi de techniques plus robustes.

De plus, on souhaite que le programme élaboré soit validé sur des environnements dynamiques c'est-à-dire des scènes où il existe des obstacles mobiles et/ou la nature du sol n'est pas fixe.

Conclusion générale

Dans ce travail, le problème d'apprentissage par renforcement (AR) et la reconstruction de carte 3D de l'environnement d'un robot mobile a été abordé.

Tous d'abords, nous avons présenté un aperçu sur la robotique mobile et les concepts de base des approches utilisées. Au début, des définitions et des concepts généraux sur l'autonomie du robot mobile ont été présentés. Les types des robots mobiles avec les domaines d'utilisation, les capteurs utilisés et les techniques récente utilisées dans la vision ont été exposés. L'architecture de la plateforme expérimentale utilisée dans nos premiers travaux est également présentée.

Les premiers travaux présentés dans cette thèse portent sur l'apprentissage des comportements réactifs, qui permettent de générer des trajectoires au robot en se basant sur une réactivité entre la perception et l'action sans aucune modélisation de l'environnement. Le but général de cette étude est de faire « *apprendre* » à un robot mobile deux comportements simples : « convergence vers un but » et « évitement d'obstacles ».

Les séries d'expérimentation réalisées dans cette première partie nous ont permis de constater l'influence des constantes et des paramètres caractérisant ces méthodes d'apprentissage FACL er FQL. Pour cela, nous avons effectué une étude comparative entre ces deux algorithmes FACL et FQL. Nous avons simulé le comportement de "convergence vers le but" d'un robot mobile à l'aide des deux algorithmes. Les expériences ont montré qu'avec un bon choix de paramètres influençant l'apprentissage (vitesse d'apprentissage, facteur d'escompte, choix de la fonction de renforcement, politique d'exploration et d'exploitation), le contrôleur flou avec six règles permet une convergence rapide vers l'objectif visé.

Ces résultats montrent également la capacité de généralisation offerte par ces algorithmes d'apprentissage contrairement aux méthodes heuristiques de développement des SIF. Enfin, les derniers résultats montrent que l'algorithme FACL donne également des résultats satisfaisants, cependant un choix inapproprié des paramètres de cet algorithme entraîne une dégradation des performances par rapport à l'algorithme FQL.

En seconde partie, nous avons proposé et développé un algorithme de cartographie 3D en utilisant une Kinect embarquée sur le robot mobile mBot. Afin de réaliser cette reconstruction 3D il est nécessaire d'exécuter plusieurs traitements :

– L'acquisition des images,

– L'extraction des points d'intérêt de ces images,

– Trouver les correspondances entre ces points d'intérêt,

– Le recalage et la fusion de ces images, dans un nuage de points 3D,

– La construction d'un maillage à partir de ce nuage de point.

Notre contribution dans ce domaine consiste à réaliser une reconstruction 3D en temps réel d'un environnement d'intérieur en se basant sur un ensemble de filtres linéaires. Après avoir décrit le matériel utilisé dans cette étude à savoir la caméra Kinect et le robot mobil mBot, nous avons expliqué en détail notre expérience : l'acquisition des données de la Kinect. Cette étape permet de capter l'image vue par la Kinect embraquée sur le robot mobile dans une position donnée. Par la suite la fusion de deux images : pour cela nous avons choisi le filtre SURF qui permet d'extraire des points d'intérêt de chaque image de façon plus robuste et plus rapide. Puis, nous nous sommes servis du KNN pour joindre les points d'intérêt des deux images entre elles et du RANSAC pour éliminer les fausses liaisons. Enfin, pour réaliser la reconstruction 3D, nous avons utilisé l'algorithme ICP qui est indispensable pour la fusion de nuages de points 3D et pour déterminer la transformation entre les deux vues.

L'inconvénient majeur du RANSAC est de choisir des points d'intérêt de façon aléatoire, ce qui est -parfois- source de fausses reconstructions. Pour remédier à ce problème, nous avons intégré, à notre algorithme de reconstruction 3D, une fonction qui permet le calcul d'une erreur moyenne qui ne doit pas dépasser 3cm, au cas où cette erreur est supérieure à 3cm une réexécution de RANSAC et du ICP se fait automatiquement et se répète à plusieurs reprises, le processus s'arrête quand l'erreur moyenne est inférieure à 3cm. Nous avons procédé de la sorte, pour augmenter les itérations de RANSAC et ainsi augmenter la probabilité d'avoir une correspondance plus exacte des points d'intérêt.

Nous avons réalisé plusieurs tests dans différents environnements afin de vérifier la robustesse de l'algorithme développé, les tests ont été effectués dans un environnement d'intérieur.

Les résultats expérimentaux obtenus en se basant sur l'algorithme développé sont très satisfaisant , et cela même si on augmente la complexité de la scène observée (présence d'obstacle).

Toutes les solutions intelligentes ont des avantages et des inconvénients, la combinaison de plusieurs méthodes nous a permis non seulement de surpasser les inconvénients d'une méthode ou d'une autre, mais aussi d'implémenter un savoir-faire pour réaliser une cartographie 3D. Dans le cas général, les solutions basées sur cette approche permettent la résolution des problèmes plus complexes, il suffit de conduire une analyse exacte de la problématique.

Enfin, nous pensons que le travail présenté dans cette thèse ouvre des nouvelles perspectives selon les principales directions suivantes :

Pour l'apprentissage par renforcement nous proposons :

- Incorporation de la connaissance a priori nécessaire à la convergence de l'apprentissage,
- Utilisation d'autre type de fonction de renforcement (continue, floue),
- L'optimisation des contrôleurs flous par les algorithmes de colonies de fourmis,
- Adaptation automatique de la partie prémisse des contrôleurs flous,
- La comparaison des méthodes développées avec d'autres algorithmes stochastiques tels que les algorithmes génétiques.

Pour la reconstruction 3D nous proposons :

- L'utilisation d'un filtre pour diminuer l'erreur de fusion des vues lors de la reconstruction 3D.
- Réaliser la reconstruction 3D en démultipliant le nombre de Kinect.
- Utilisation d'un autre type d'erreur à minimiser.

Références bibliographiques

[1] A.Habeeb : « *Introduction to Artificial Intelligence* ».Thesis Mansoura University September 2017.

[2] P.H.Winston : « *Artificial Intelligence* ».book, third edition, Massachusetts institute of technology, ISBN 0-201-53377-4, 1993.

[3] M. GHAOUI. : « *Planification d'un mouvement pour un robot mobile* ». Thèse de magister Université de Batna année 1997.

[4] F. Boufera : « *Contribution Des Outils de l'Intelligence Artificielle dans la Robotique Mobile* ». Thèse, Université d'Oran1 Ahmed ben bella,2014

[5] O. Djekoune : « *Localisation Et Guidage Du Robot Mobile Atrv2 Dans Un Environnement Naturel* ». Thèse, Université des Sciences et de technologie Houari Boumediene, 2010.

[6] Z.Laouici: « *la modélisation d'un système de coopération et communication dans la navigation des robots mobiles* » thèse, Université d'Oran1 Ahmed ben bella,2015

[7] F. Morbidi : « *Localisation et navigation de robots* », Chapitre université de Picardie jules verie,2015.

[8] K. Niechwiadowicz, Z. Khan : « *Robot Based Logistics System for Hospitals – Survey*»,[digital communication and networks, vol 1, issue 2, pp 102-111, 2015.

[9] J.C. Anderson : « *Mobile robot navigation* », thèse Université Technique de Danemark Septembre 2007.

[10] M.D. Rossetti, R.A. Felder, A.Kumar : « *Simulation of robotic courier deliveries in hospital distribution services* »,Health Care Management science 3,pp 201-213,2000.

[11] D.Lhomme-Desages «*Commande d'un robot mobile rapide à roues non directionnelles sur sol naturel*», thèse Université Paris 6 Pierre et Marie Curie. 2008

[12] P.Kucsera : « *Industrial Component-based Sample Mobile Robot System*», Acta Polytechnica Hungarica, Vol. 4, No. 4,pp 69-81,2007

[13] H. Choset, K. M. Lynch, S. Hutchingson, G. Kantor, W. Burgand, L. E.Kavraki, and S. Thrun : « *Principles of Robot Motion: Theory, Algorithms and Implementations*», 1st ed. MIT Press, 2005.

[14] J.Schiff, A,Kulkarni, D.Bazo, V.Duindam : «*Actuator networks for navigating an unmonitored mobile robot*», Conference International on Automation Science and Engineering, 2008

[15] V.Ganapathy, S.C. Yun, J. Ng: «*Fuzzy and neural controllers for acute obstacle avoidance in mobile robot navigation* ». IEEE/ASME International Conference on Advanced Intelligent Mechatronics Suntec Convention and Exhibition Center, pp1236-1241, 2009.

[16] P.K. Mohanty, D. R. Parhi: « *A New Intelligent Motion Planning for Mobile Robot Navigation using Multiple Adaptive Neuro-Fuzzy Inference System* ». Appl. Math. Inf. Sci.8, No. 5, pp 2527-2535, 2014

[17] M. Elhoseny, A. Tharwat, A. E. Hassanien : « *Bezier Curve Based Path Planning in a Dynamic Field using Modified Genetic Algorithm*». Journal of Computational Science, Volume 25, pp. 339-350,2018.

[18] W. Zhangqi, Z. Xiaoguang, H. Qingyao : « *Mobile Robot Path Planning based on Parameter Optimization Ant Colony Algorithm*». Procedia Engineering, Volume 15, pp. 2738-2741,2011.

[19] P.K. Das, H.S. Behera, B.K. Panigrahi : « *A hybridization of an improved particle swarm optimization and gravitational search algorithm for multi-robot path planning* ». Swarm and Evolutionary Computation, Volume 28, pp. 14-28, 2016.

[20] W. H. I.Sabban , L. F Gonzalez, R. N, Smith, G. F. Wyeth: « *Wind-energy based path planning for electric unmanned aerial vehicles using Markov decision processes* ». In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems. IEEE. 2012

[21] A. Konar, G.I.Chakraborty, , S. J. Singh, L. C.Jain , A. K.Nagar: « *A Deterministic Improved Q-Learning for Path Planning of a Mobile Robot* », IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS: SYSTEMS, VOL. 43, NO. 5, 2013.

[22] M.Zhu, M. Otte, P. Chaudhari, E .Frazzoli, : « *Game theoretic controller synthesis for multi-robot motion planning Part I: Trajectory based algorithms*. IEEE International Conference on Robotics & Automation (ICRA)Hong Kong Convention and Exhibition Center May 31 - June 7, 2014. Hong Kon. 2014

[23] H.Zhang : « *Mobile Robotics Mobile robot classification continued*», TAMS, Department of Informatics, University of Hamburg, Germany.2012

[24] B.Bayle : « *Robotique Mobile*», chapitre , Télécom Physique Strasbourg Université de Strasbourg,2007.

[25] R. Siegwart, I. R. Nourbakhsh: « *Introduction to autonomous mobile robots*», MIT press, 2004.

[26] R.Carona, A. P. Aguiar, J.Gaspar : « *Control of unicycle type robots, Tracking, Path Following and Point Stabilization*», in Proc. of IV Jornadas de Engenharia Electrónica e Telecomunicações e de Computadores, pp180-185, Lisbon, Portugal November 2008.

[27] L.Jaïem: « *Contribution à l'autonomie des robots : vers la garantie de performance en robotique mobile autonome par la gestion des ressources matérielles et logicielles*», Thèse en Robotique. Université Montpellier, 2016.

[28] J.Angeles, F.C.Park : « *Performance evaluation and design criteria* », Springer Handbook of Robotics. Springer,pp. 229–244. 2008.

[29] A. Jacoff, E. Messina, J.Evans: « *Experiences in Deploying Test Arenas for Autonomous Mobile Robots*», Proceeding of the Performance Metrics for Intelligent System Workshop.2001.

[30] R. Siegwart, M. Chli, M. Rufl : « *Autonomous Mobile Robots* »,chapitre : Autonomous Mobile Robots, , ETH,Zurich, Spring 2017

[31] A. Affoyon, L. Stephanes lima : « *Robot Mobile pour Arduino »,Rapport Projet 12 IMA4* polytechnique de Lille. France.2016.

[32] P. Gierlak, K. Krzysztof, K, D. Szybicki: « *Mobile crawler robot vibration analysis in the contexts of motion speed selection*», International LTD. Journal of vibr-engineering, Vol. 19, Issue 4. ISSN 1392-8716.pp 2403-2412, Juin 2017.

[33] R.Airton. J. da Silva, S.Frank: « *Design and Control of a Two-Wheeled Robotic Walker for Balance Enhancement* »,International Conference on Rehabilitation Robotics June 24-26, 2013 Seattle, Washington USA.

[34] J.Shin, A.Rusakov, B. Meyer : « *Smart Walker: an intelligent robotic walker*», Journal of Ambient Intelligence and Smart Environments 1, pp 1-16, 2016.

[35] C. D.Remy, O. Baur, M. Latta, A. Lauber, M. Hutter , M. A. Hoepflinger, C. Pradalier, R. Siegwart : « *Walking crawling with ALoF - A robot for autonomous locomotion on four legs»,* 13th International Conference on Climbing and Walking Robots and the Support Technologies for Mobile Machines, Nagoya, Japan, 31 Aout - 03 Septembre 2010.

[36] T.Yanagida, E.L,Rajesh, T. Pathmakumar, K. Elangovan, M. Iwase: « *Design and Implementation of A Shape ShiftingRolling–Crawling–Wall-Climbing Robot »,* Appl. Sci.2017,7, 342; doi:10.3390/app7040342,pp 1-21,2017.

[37] E. Colle, Y.Rybarczyk, P. Hoppenot : « *ARPH: An assistant robot for disabled people», International* Conference on Systems, Man and Cybernetics, 2002 , Volume: 1, Hammanet, Tunisia, 6-9 October 2002.

[38] H. Seok Hong, Sung.Y. Jung, J.H. Jung, B.G. Lee, J. W.Kang,D. J. Park,M. J. Chung : « *Development of Work Assistant Mobile Robot System for the Handicapped in a Real Manufacturing Environment»,* 9th International Conference on Rehabilitation Robotics, June 28 - July 1, 2005, Chicago, IL, USA.

[39] G. Capi, H. Toda: « *A New Robotic System to Assist Visually Impaired People»,* international *Symposium* on Robot and Human Interaction Communication, Atlanta, Georgia. 31 July - 3 August 2011.

[40] J.Na : « *The blind interactive guide system using RFID-based indoor positioning system»,* 10th Int. Conf. on Computers Helping People with Special Needs, Linz, Austria, July 11-13, Lecture Notes in Computer Science, Vol.4061, Springer Berlin, pp.1298-1305. 2006.

[41] S. Vatsal: « *Floor Cleaning Robot with Mobile-App or Autonomous*», Mini Project, Department of Electronics & Communication Engineering Indus Institute of Technology and Engineering, India, May 2015.

[42] L. Kuotsan, W. Chulun : « *A Technical Analysis of Autonomous Floor Cleaning Robots Based on US Granted Patents*», European International Journal of Science and Technology Vol. 2 No. 7 September 2013.

[43] S.Muruganandhan, G.Jayabaskaran, P.Bharathi: « *LabVIEW-NI ELVIS II based Speed Control of DC Motor*», International Journal of Engineering Trends and Technology (IJETT) Volume 4 Issue 4, April 2013.

[44] P.Szegedi, P. Koronvary B. Békési: « *The use of robots in military operations* », Scientific research and education in the air force in the air force. AFASES 2017.

[45] G.Mie : « *Military robots of the present and the future* », AARMS Technology Vol. 9, No. 1 125–137. 2010.

[46] E. Samuelsen, K. Glette, J.Torresen : « *A hox gene inspired generative approach to evolving robot morphology* », the 15th Annual Conference on Genetic and Evolutionary Computation, pp. 751–758. 2013

[47] E. Samuelsen, K. Glette : « *Some distance measures for morphological diversification in generative evolutionary robotics* » The 16th Annual Conference on Genetic and Evolutionary Computation, pp. 721–728. 2014.

[48] A. Oualid Djekoune : « *Localisation Et Guidage Du Robot Mobile Atrv2 Dans Un Environnement Naturel* », Robotique. Université des Sciences et de la Technologie Houari Boumediene, 2010.

[49] P. Bonnifait : « *Localisation précise en position et attitude des robots mobiles d'extérieur à évolutions* », Robotique. Ecole Centrale de Nantes. France. 1997.

[50] T. Krajník, V. Vonasek, D. Fišer, Jan Faigl: « *AR-Drone as a Platform for Robotic Research and Education*», International Conference on Research and Education in Robotics, EUROBOT 2011, Prague, Czech Republic, June 2011.

[51] Y. Onmek, J Triboulet, S Druon, A Meline, S Louis, B Jouvencel : « *3D underwater reconstruction of archeology object with a mono camera camcorder: The case study at Gourneyras lake*», OCEANS 2017-Aberdeen, pp 1-6.2017.

[52] J.F. Brethe : « *Modélisation de la précision micrométrique des robots manipulateurs pilotés à* l'aide *d'informations extéroceptives*», 20ème Congrès Français de Mécanique Besançon,pp1-6, 29 août au 2 septembre 2011.

[53] C.F. Olson , L.H. Matthies : « *Maximum likelihood rover localization by matching range maps*», Proc. of the IEEE Int. Conf. On Robotics and Automation ICRA'98, p. 272-277, May 1998.

[54] A. R. SAINT-VINCENT : « *Perception et Modélisation de l'Environnement d'un Robot Mobile: une Approche par Stéréovision*», Thèse de l'Université P. Sabatier, Toulouse, France, Octobre 1986.

[55] J. Takeno, U. Rembold : « *Stereovision systems for autonomous mobile robots*», Proc. Int. Conf. *Intelligent* Autonomous Systems, pp. 26-41, 1995.

[56] E. Krotov, M. Hebert : « *Mapping and positioning for a prototype lunar rover*», Proc. IEEE – "Int.Conf. on Robotics and Automation, Nagoya, Aichi, Japan, p. 2913-2919, May 1995.

[57] J.Gluckman , S. K. Nayar : « *Catadioptric Stereo Using Planar Mirrors*», International Journal of Computer Vision 44(1), pp 65–79, 2001.

[58] D. Catuhe: «*Programming with the Kinect for Windows Software Development Kit*. », Microsoft Press,USA. ISBN: 978-0-7356-6681-8 ,2012.

[59] A.Ufkes, M.Fiala: «*Visual Odomtry Using 3-Dimensional Video Input* », the Canadian Conference on Computer and Robot Vision, 2011.

[60] J. Ashley, J.Webb: «*Beginning Kinect Programming with the Microsoft Kinect SDK*», North Carolina State University, 2012. DOI: 10.1007/978-1-4302-4105-8.

[61] R. S. Sutton and A. G. Barto: «*Reinforcement Learning: An Introduction*», MIT Press, Cambridge, MA, 1998.

[62] P. Y. Glorennec: «*Reinforcement Learning: an Overview*», ESIT 2000, Aachen, Germany, September 2000.

[63] J. Ferber: «*Les Systèmes Multi Agents: Vers une Intelligence Collective*», Inter Edition, Université Pierre et Marie Curie Paris 6, 1995.

[64] L. Cherroun : « *Navigation Autonome d'un Robot Mobile par des Techniques Neuro-Floues*», Thèse, université Biskra. 2014.

[65] J. A. Anderson: « *An Introduction to Neural Networks*», Bradford - MIT Press, 1995.

[66] E. Gauthier: « *Utilisation des Réseaux de Neurones Artificiel pour la Commande d'un Véhicule Autonome*», Thèse de Doctorat, Institut National Polytechnique de Grenoble, 1999.

[67] J. Günther, P.M. Pilarski , G,Helfrich, H. Shen, K. Diepold : « *Intelligent laser welding through representation, prediction, and control learning: An architecture with deep neural networks and reinforcement learning*», Mechatronics vol 34, pp 1–11, 2016.

[68] F. Fathinezhad, V. Derhami, M. Rezaeian: « *Supervised Fuzzy Reinforcement Learning for Robot Navigation*», Applied Soft Computing Journal , pp1-17,2015.

[69] F. Lachekhab: « *Planification et navigation à base de la logique floue d'un robot mobile dans un environnement partiellement connu* », thèse de Magister, École Militaire Polytechnique, Alger, 2005.

[70] M. L. Puterman: « *Markov Decision Processes-Discrete Stochastic Dynamic Programming*», *John* Wiley and Sons, New York, NY, 1994.

[71] R. E. Bellman: « *Dynamic Programming*», Princeton University Press, Princeton, NJ, 1957.

[72] L. Jouffe : « *Apprentissage de Système D'inférence Floue par des Méthodes de Renforcement* », *Thèse* de Doctorat, IRISA, Université de Rennes I, Juin 1997.

[73] L. Jouffe: « *Actor-Critic Learning Based on Fuzzy Inference System*», Proc of the IEEE International Conference on Systems, Man and Cybernetics, pp.339-344, Beijing, China, 1996.

[74] Q. Wei, Frank L. Lewis, Q.Sun: « *Discrete-Time Deterministic Q -Learning: A Novel Convergence Analysis*»,IEEE Transactions on Cybernetics; Volume: 47, Issue: 5,pp 1224 - 1237; May 2017.

[75] H. Kesten: « *Accelerated Stochastic Approximation* », Annals of Mathematical Statistics, vol *29*, 1958, pp. 41-59.

[76] C. Watkins, P. Dayan: « Q-*Learning Machine Learning »,* Springer pp.279-292, 1992.

[77] A. Prakash Moon, K. K. Jajulwar: « *Design of adaptive fuzzy tracking controller for Autonomous navigation system»,* International Journal of Recent Trend in Engineering and Research. 2016.

[78] H. Ghazouani : «*Navigation visuelle de robots mobiles dans un environnement d'intérieur»,* thèse de Université de Montpellier ,France, 2012.

[79] C.Zhengrong , H. Zhang: «*Local-SURF based 3D scene reconstruction for mobile robot»,*2nd International Conference on Information Technology and Electronic Commerce. 2014.

[80] K. Qian , X. Ma ; F. Fang ; H. Yang: *« 3D environmental mapping of mobile robot using a low-cost depth camera»,*IEEE International Conference on Mechatronics and Automation, Takamatsu, Japan. 2013.

Annexe A

1. Le détecteur de Harris

Les points d'intérêt sont des points dans l'image qui se distinguent par leurs saillances et qui peuvent être localisés facilement dans des images successives et ne se perdent pas facilement, ce qui les rend facile à détecter et à suivre dans le temps. Les coins sont souvent utilisés pour identifier des objets dans une scène ou utilisés dans la mesure de déplacement d'objets ou aussi dans la stéréoscopie. Un certain nombre d'algorithmes ont été élaborés on peut citer principalement Le détecteur de Harris connu sous le nom de Harris est une version modifiée du détecteur Plessey. C'est un algorithme moins sensible aux bruits et n'a besoin de calculer que la première dérivée de l'image. Harris a défini une mesure du coin par l'opérateur suivant :

$$H(x,y) = \det(C) - \alpha Trace^2(C)$$

$$C = w_G(r,\sigma) \times \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \quad et \quad w_G(r,\sigma) un \ \ filtre \ gaussien$$

2. Détermination des points d'intérêt par l'algorithme SURF

Cette étape consiste à déterminer une relation entre les points détectés dans l'image à l'instant k et leurs correspondants de l'image à l'instant k+1, cette relation doit être robuste face au changement d'échelle, condition d'acquisition d'image, rotation, translation…etc.

Pour cela nous avons fait l'association par le descripteur SURF. Elle est basée sur la comparaison entre leurs vecteurs descripteurs. Si cette distance est inférieure à un certain seuil, les deux points d'intérêt sont associés. Plusieurs distances peuvent être utilisées, dans notre travail nous utilisons la distance euclidienne pour des raisons de simplicité.

P^K : Nombre de points détectés dans l'image à l'instant k.

P^{K+1}: Nombre de points détectés dans l'image à l'instant k+1.

$d(i,j)$: Distance euclidienne entre le descripteur du $i^{éme}$ point d'intérêt de l'image à l'instant k et le $j^{éme}$ point d'intérêt de l'image à l'instant k+1.

S : seuil pour associer les points d'intérêt (deux points i et j sont associés si et seulement si $d(i,j) < \mathrm{S}$.

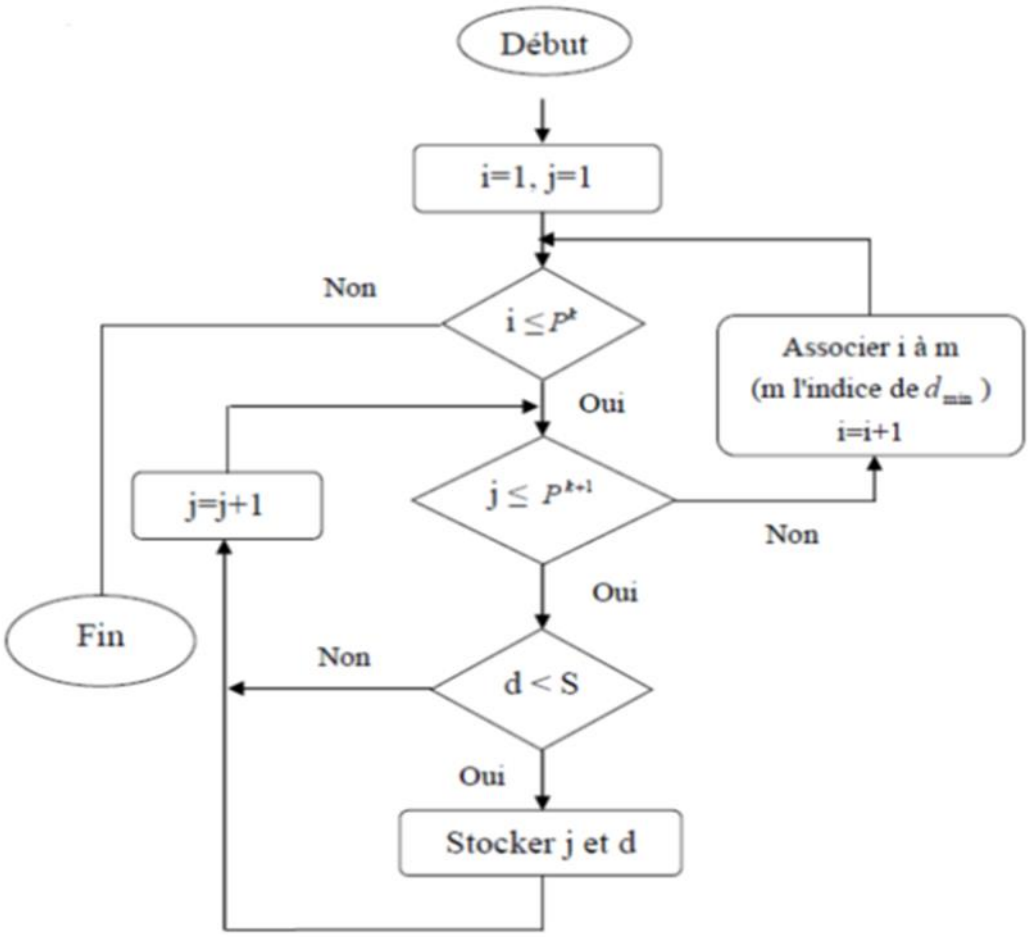


Figure 1 : Organigramme d'association des points d'intérêt.

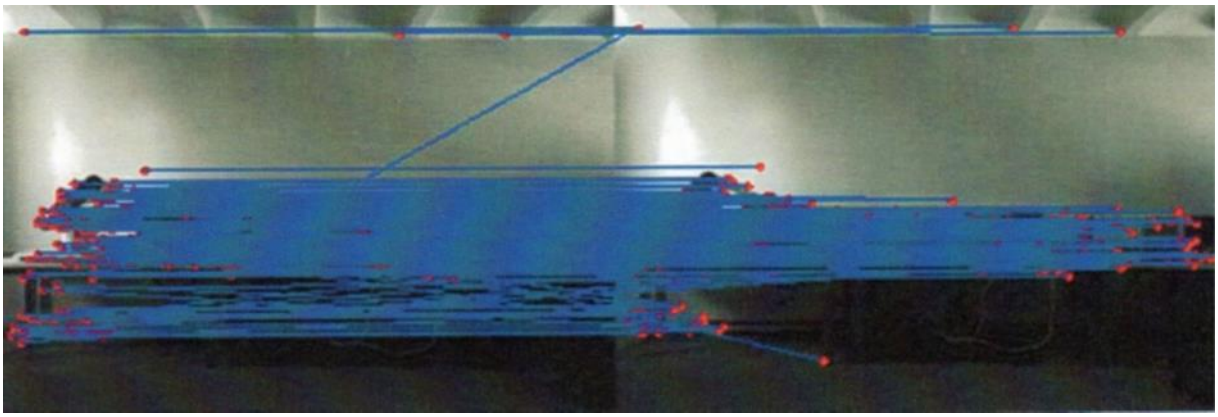


Figure 2 : Détermination des points d'intérêt par SURF avec un seuil S = 1.

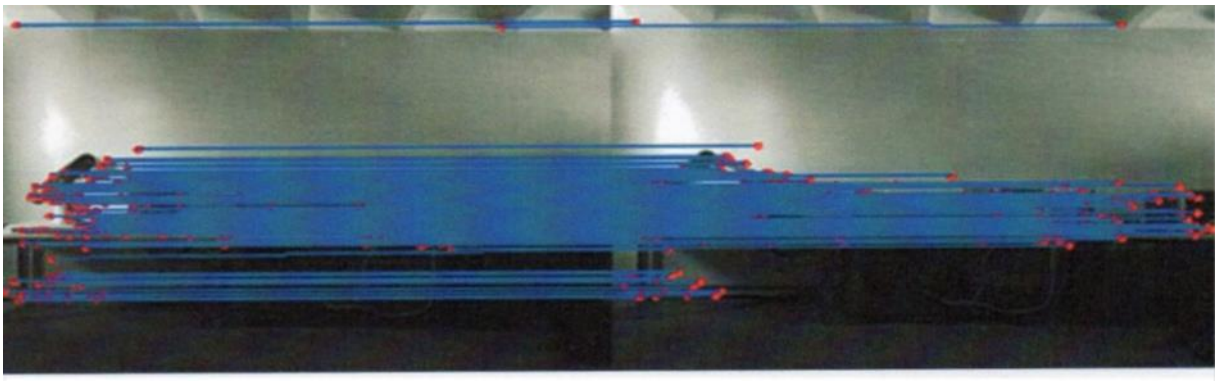


Figure 3 : Détermination des points d'intérêt par SURF avec un seuil S = 0.5.

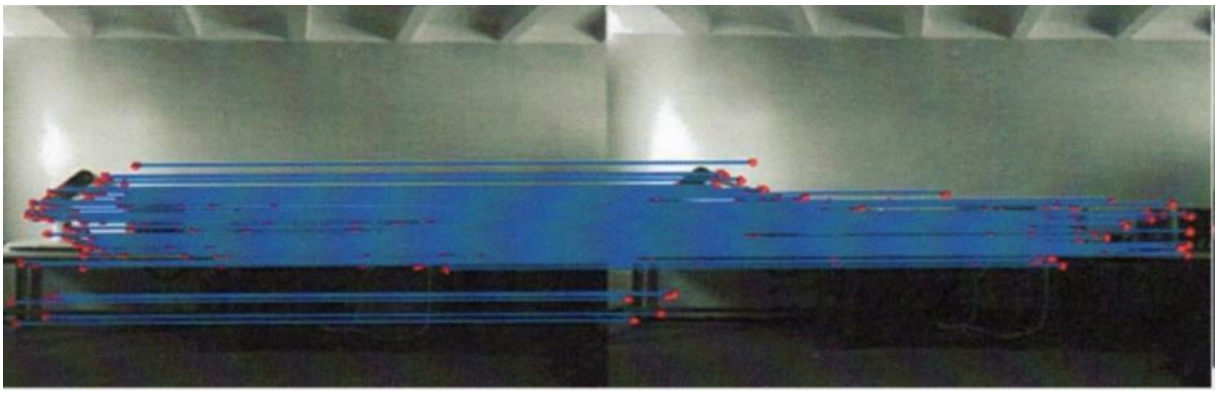


Figure 4 : Détermination des points d'intérêt par SURF avec un seuil S = 0.2.

Le Tableau suivant montre le nombre de points associés avec différents seuils. Nous remarquons que le nombre de points diminue avec la diminution du seuil sur la distance.

Seuil	1	0.8	0.6	0.5	0.4	0.2	0.1
Nombre de points matchés	291	258	236	225	210	192	144

Tableau 1 : Résultats de la détermination des points d'intérêt par SURF.

3. RANSAC

Cette méthode est classiquement utilisée pour extraire un modelé d'un ensemble de données bruitées. Dans le cas présent, RANSAC est utilisé pour déterminer quels sont les points qui permettent d'estimer la transformation entre les deux nuages le plus correctement possible. Autrement dit, RANSAC permet d'éliminer les fausses associations.

RANSAC est basé sur le principe suivant : un ensemble de points dans chaque nuage sont tirés au hasard pour déterminer une base. La transformation liant les deux bases est alors calculée par une des méthodes d'estimation. Les nuages sont recollés suivant cette transformation et la distance entre chaque couple de points est estimée. Un score est alors attribué à la transformation : si un nombre important de points se superposent, le score est

élevé, sinon il est faible. Finalement, la transformation qui possède le plus haut score est conservée, car elle permet de recoller plus de points de l'ensemble, donc elle a plus de chance d'être correcte. Cette méthode permet en plus d'évaluer la transformation liant les deux nuages de point et de filtrer les points qui n'entrent pas dans le modèle principal.

3.1 Exemple

Un exemple simple est l'ajustement d'une ligne située dans un plan (2D) à une série d'observations. On suppose que cet ensemble contient à la fois des points pertinents (inliers), c'est-à-dire, les points qui peuvent être approximativement ajustés à une ligne, et des points aberrants (outliers), les points qui sont éloignés de ce modèle de ligne. Un simple traitement par une méthode des moindres carrés donnera une ligne qui est mal ajustée aux points pertinents. En effet, la droite s'ajustera de manière optimale à tous les points, y compris les points aberrants. La méthode RANSAC, par contre, peut générer un modèle qui ne tiendra compte que des points pertinents, à condition que la probabilité, lorsqu'on tire des points au hasard, de ne sélectionner que des points pertinents, soit suffisamment élevée. Cependant, il n'y a aucune garantie d'obtenir cette situation, et il existe un certain nombre de paramètres de l'algorithme qui doivent être soigneusement choisis pour maintenir ce niveau de probabilité suffisamment élevé.

Figure 5 : Un jeu de données avec de nombreuses valeurs aberrantes pour lequel une ligne doit être ajustée.

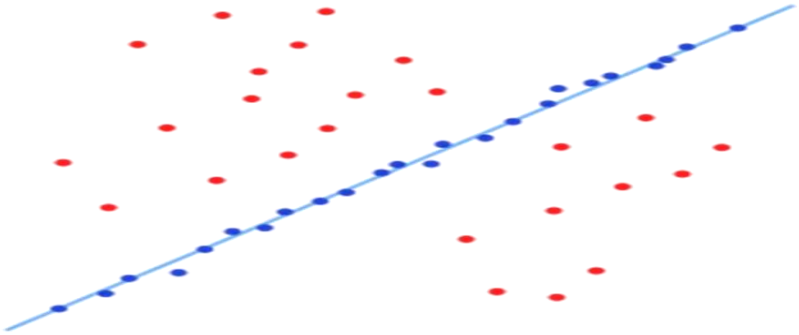


Figure 6 : Ligne ajustée avec la méthode RANSAC, les valeurs aberrantes n'ont aucune influence sur le résultat.

Les données d'entrée de l'algorithme RANSAC sont un ensemble de valeurs des données observées, un modèle paramétré qui peut expliquer ou être ajusté aux observations, et des paramètres d'intervalle de confiance.

RANSAC atteint son objectif en sélectionnant itérativement un sous-ensemble aléatoire des données d'origine. Ces données sont d'hypothétiques données pertinentes et cette hypothèse est ensuite testée comme suit :

Un modèle est ajusté aux données pertinentes hypothétiques, c'est-à-dire que tous les paramètres libres du modèle sont estimés à partir de ce sous-ensemble de données.

- Toutes les autres données sont ensuite testées sur le modèle précédemment estimé. Si un point correspond bien au modèle estimé, alors il est considéré comme une donnée pertinente candidate.
- Le modèle estimé est considéré comme correct si suffisamment de points ont été classés comme données pertinentes candidates.
- Le modèle est ré-estimé à partir de ce sous-ensemble des données pertinentes candidates.
- Finalement, le modèle est évalué par une estimation de l'erreur des données pertinentes par rapport au modèle.
- Cette procédure est répétée un nombre fixe de fois, chaque fois produisant soit un modèle qui est rejeté parce que trop peu de points sont classés comme données pertinentes, soit un modèle réajusté et une mesure d'erreur correspondante. Dans ce dernier cas, on conserve le modèle réévalué si son erreur est plus faible que le modèle précédent.

3.2 Notations et Algorithme

L'algorithme RANSAC générique, en pseudo code, fonctionne comme suit :

Entrées :

Data - un ensemble d'observations

modele - un modèle qui peut être ajusté à des données

n - le nombre minimum de données nécessaires pour ajuster le modèle

k - le nombre maximal d'itérations de l'algorithme

t - une valeur seuil pour déterminer si une donnée correspond à un modèle

d - le nombre de données proches des valeurs nécessaires pour faire valoir que le modèle correspond bien aux données

Sorties :

meilleur_modèle - les paramètres du modèle qui correspondent le mieux aux données (ou zéro si aucun bon modèle a été trouvé)

meilleur_ensemble_points - données à partir desquelles ce modèle a été estimé

```
    meilleure_erreur - l'erreur de ce modèle par rapport aux données
  itérateur := 0
  meilleur_modèle := aucun
  meilleur_ensemble_points := aucun
  meilleure_erreur := infini
   Tant que itérateur < k
    points_aléatoires := n valeurs choisies au hasard à partir des données
    modèle_possible := paramètres du modèle correspondant aux points_aléatoires
    ensemble_points := points_aléatoires
  Pour chaque point des données pas dans points_aléatoires
      Si le point s'ajuste au modèle_possible avec une erreur inférieure à t
        Ajouter un point à ensemble_points
      Si le nombre d'éléments dans ensemble_points est > d
      (Ce qui implique que nous avons peut-être trouvé un bon modèle, on teste
maintenant          dans quelle mesure il est correct)
      modèle_possible := paramètres du modèle réajusté à tous les points de
ensemble_points
      erreur := une mesure de la manière dont ces points correspondent au
modèle_possible
      Si erreur < meilleure_erreur
        (Nous avons trouvé un modèle qui est mieux que tous les précédents,
        le garder jusqu'à ce qu'un meilleur soit trouvé)
        meilleur_modèle := modèle_possible
        meilleur_ensemble_points := ensemble_points
        meilleure_erreur := erreur
   incrémention de l'itérateur
  Retourne meilleur_modèle,
  meilleur_ensemble_points,
  meilleure_erreur
```

3.3 Variantes possibles de l'algorithme RANSAC

- Arrêter la boucle principale si un modèle assez bon a été trouvé, c'est-à-dire avec une erreur suffisamment petite ; cette solution peut épargner du temps de calcul, au détriment d'un paramètre supplémentaire ;

- Calculer erreur directement à partir de modèle_possible, sans ré-estimation du modèle à partir d'ensemble_points ; cette solution peut faire gagner du temps au détriment de la comparaison des erreurs liées à des modèles qui sont estimés à partir d'un petit nombre de points et donc plus sensibles au bruit.

3.4 Paramètres

Les valeurs des paramètres t et d doivent être fixées conformément aux exigences spécifiques liées à l'application et à l'ensemble de données qui peuvent être éventuellement fondées sur l'évaluation expérimentale. Le paramètre k (le nombre d'itérations), cependant, peut être déterminé à partir d'un résultat théorique. Soit p la probabilité que l'algorithme RANSAC pendant une itération sélectionne uniquement des données pertinentes dans l'ensemble des données d'entrée, lorsqu'il choisit les n points à partir desquels les paramètres du modèle seront estimés. Lorsque cela se produit, le modèle qui en résulte est susceptible d'être pertinent, donc p donne la probabilité que l'algorithme produise un résultat correct. Soit w, la probabilité de choisir un point pertinent à chaque fois qu'un seul point est sélectionné, c'est-à-dire w = nombre de points pertinents dans les données / nombre de points dans les données. Il est fréquent que w ne soit pas connu à l'avance, mais que l'on puisse en estimer une valeur approximative. En supposant que les n points nécessaires pour l'estimation d'un modèle sont sélectionnées de manière indépendante, w^n est la probabilité que l'ensemble des n points correspond à des points pertinents et $1 - w^n$ est la probabilité qu'au moins un des n points soit un cas aberrant, un cas qui implique qu'un mauvais modèle sera estimé à partir de cet ensemble de points. Cette probabilité à la puissance de k est la probabilité que l'algorithme ne choisisse jamais un ensemble de n points qui seraient tous pertinents et cela doit être égal à 1 - p.

Par conséquent, $1 - p = (1 - w^n)^k$ qui, en prenant le logarithme des deux côtés, conduit à :

$$k = \frac{\log(1-p)}{\log(1-w^n)} \qquad (1)$$

Il convient de noter que ce résultat suppose que les n points de données sont sélectionnés de façon indépendante, c'est-à-dire, qu'un point qui a été sélectionné une fois est remis et peut être sélectionné à nouveau dans la même itération. Cela n'est pas souvent une approche pertinente et la valeur calculée pour k devrait être prise comme une limite supérieure dans le cas où les points sont choisis sans remise. Par exemple, dans le cas de la recherche d'une ligne qui s'ajuste à la série de données illustrée sur la figure ci-dessus, l'algorithme RANSAC choisit généralement deux points à chaque itération et calcule le modèle_possible comme la ligne qui relie ces deux points et il est alors important que les deux points soient distincts.

Pour gagner en qualité, l'écart type ou des multiples de celui-ci peuvent être ajouté à k.

L'écart type de k est défini comme

$$SD(k) = \frac{\sqrt{1-w^n}}{w^n} \qquad (2)$$

3.5 Avantages et inconvénients

Un avantage de RANSAC est sa capacité à calculer de manière robuste les paramètres du modèle, c'est-à-dire qu'il peut estimer les paramètres avec un degré élevé de précision, même si une quantité importante de valeurs aberrantes (outliers) est présente dans les données. Un inconvénient de RANSAC est qu'il n'y a pas de limite supérieure sur le temps de calcul de ces paramètres. Lorsqu'une limite est utilisée (un nombre maximal d'itérations), la solution obtenue peut ne pas être la solution optimale. Un autre inconvénient de RANSAC est qu'elle suppose de fixer des seuils spécifiques au problème traité.

RANSAC ne peut estimer qu'un seul modèle à un ensemble de données particulier. Comme pour toute approche à modèle unique, lorsque deux (ou plusieurs) modèles coexistent, RANSAC peut ne parvenir à trouver ni l'un ni l'autre.

3.6 Applications

L'algorithme RANSAC est souvent utilisé dans le domaine de la vision par ordinateur, par exemple, pour résoudre simultanément les problèmes de mise en correspondance et d'estimer la matrice fondamentale liée à une paire stéréo de caméras.

4. KNN

L'algorithme KNN est utilisé dans de nombreux domaines :

- La reconnaissance de formes.
- La recherche de nouveaux biomarqueurs pour le diagnostic.
- Algorithmes de compression.
- Analyse d'image satellite.
- Marketing ciblé.

4.1 Principe de fonctionnement

Le principe de cet algorithme de classification est très simple. On lui fournit un ensemble de données d'apprentissage D, une fonction de distance d et un entier k. Pour tout nouveau point de test x, pour lequel il doit prendre une décision, l'algorithme recherche dans D les k points les plus proches de x au sens de la distance d, et attribue x à la classe qui est la plus fréquente parmi ces k voisins.

4.2 Exemple

Dans l'exemple suivant on a 3 classes, le but est de trouver la valeur de la classe de l'exemple inconnu x. On prend la distance Euclidienne et $k = 5$ voisins. Des 5 plus proches voisins, 4 appartiennent à ω1 et 1appartient à ω3, donc x est affecté à ω1, la classe majoritaire.

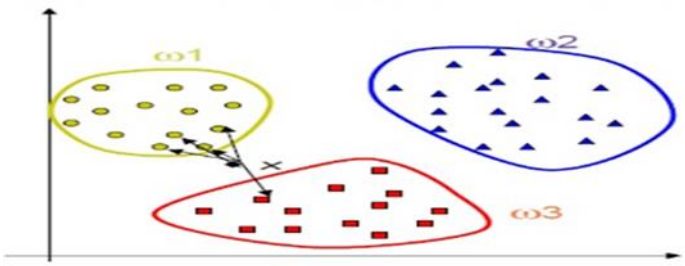


Figure 7 : Exemple montrant le principe du fonctionnement de KNN.

4.3 Comment choisir la valeur de K ?

K=1 : frontières des classes très complexes

- Très sensible aux fluctuations des données (variance élevée).
- Risque de sur-ajustement.
- Résiste mal aux données bruitées. K=n : frontière rigide.
- Moins sensible au bruit.
- Plus la valeur de k est grande plus le résultat d'affectation est bien réalisé.

4.4 Mesures de distance

Mesures souvent utilisées pour la distance dist(xi, xj), la distance Euclidienne: qui calcule la racine carrée de la somme des différences carrées entre les coordonnées de deux points :

$$d(x,y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \quad (3)$$

La distance de Manhattan : qui calcule la somme des valeurs absolues des différences entre les coordonnées de deux points :

$$d(x,y) = \sum_{i=1}^{n}|x_i - y_i| \quad (4)$$

La distance de Minkowski qui est une métrique de distance générale.

$$d(x,y) = \sqrt[q]{\sum_{i=1}^{n}|x_i - y_i|^q} \quad (5)$$

4.5 Notations et Algorithme

Soit D = {(x', c), c ∈ C} l'ensemble d'apprentissage. Soit x l'exemple dont on souhaite déterminer la classe.

Algorithme

Début

Pour chaque ((x', c) ∈ D) faire

Calculer la distance dist(x, x')

Fin

Pour chaque {x' ∈ kppv(x)} faire

Compter le nombre d'occurrence de chaque classe

Fin

Attribuer à x la classe la plus fréquente ;

Fin

4.6 Avantages

- Apprentissage rapide.
- Méthode facile à comprendre.
- Adapté aux domaines où chaque classe est représentée par plusieurs prototypes et où les frontières sont irrégulières (ex. Reconnaissance de chiffre manuscrits ou d'images satellites).

4.7 Inconvénients

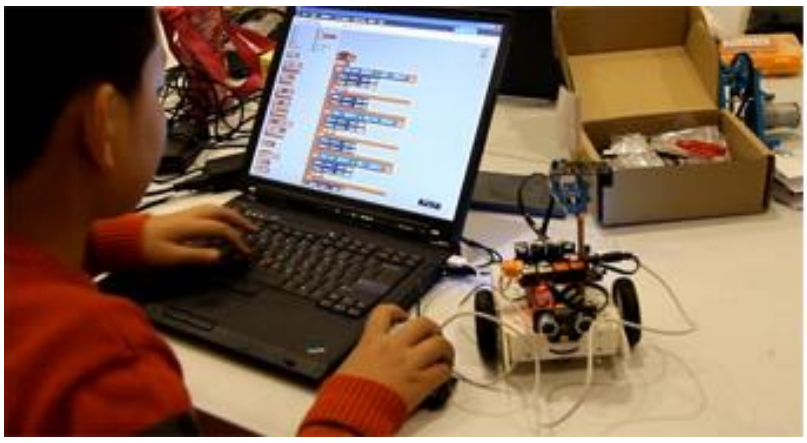
- Prédiction lente car il faut revoir tous les exemples à chaque fois.
- Méthode gourmande en place mémoire.
- Sensible aux attributs non pertinents et corrélés.
- Particulièrement vulnérable au fléau de la dimensionnalité.

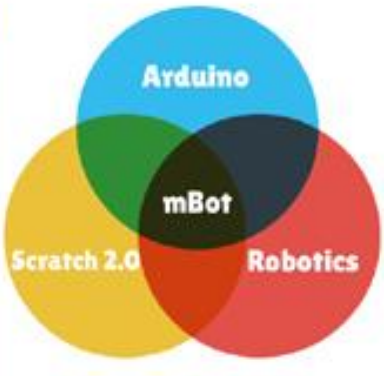
Annexe B
Spécifications techniques du mBot

Robot "mBot-Blue (Bluetooth Version)"

Cette base robotique compatible arduino est probalement la base présentant le meilleur rapport qualité / prix / performance / possibilité évolutive / simplicité de montage et d'utilisation du marché ! Livrée sous la forme d'un kit à assembler, la base "mBot" est idéalement conçue pour les collèges, les lycées, les IUT, etc... afin de permettre aux étudiants d'apprendre les bases techniques de la robotique.

Cette dernière est composée de diverses structures mécaniques en aluminium anodisé de couleur "bleue" associées à 2 moteurs à courant continu, à des roues à pneus gomme, à une roue folle, à une platine programmable compatible arduino et à différents modules périphériques.

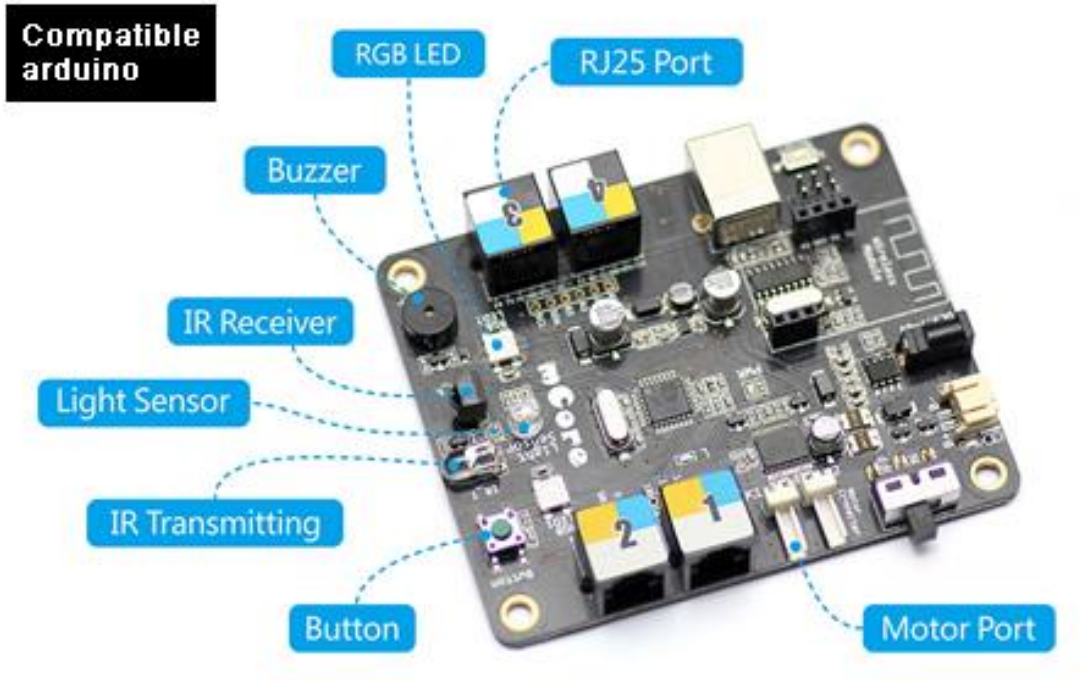




La base "mBot" met à votre disposition un outil pédagogique permettant de mettre en œuvre des applications robotiques avec une programmation sur environnement Arduino ou sur environnement graphique de type Scratch™.

Base programmable compatible Arduino

La platine principale du robot "mBot" est architecturée sur un cœur compatible Arduino UNO-328 (base ATmega328) associée à un contrôleur de moteur spécialisé lequel sera directement capable de piloter 2 moteurs "cc".



La platine dispose d'une prise USB afin que vous puissiez la programmer via l'environnement de l'Arduino. Elle dispose également :

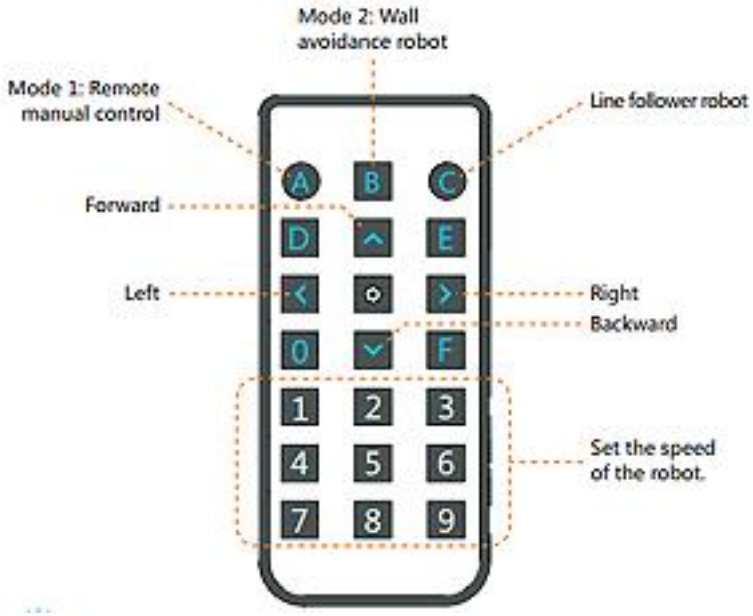
- ✓ D'un Buzzer
- ✓ De 2 Leds RGB
- ✓ D'une Led de transmission IR
- ✓ D'une Led de réception IR
- ✓ D'un capteur de lumière
- ✓ D'un bouton poussoir
- ✓ De 2 connecteurs permettant de la relier aux moteurs du robot
- ✓ D'un module Bluetooth™ embrochable (livré)
- ✓ D'un interrupteur "M/A"
- ✓ D'un connecteur permettant de la relier à un support de pile (le support est livré)
- ✓ D'un connecteur pour la relier à une batterie LiPo (batterie non livrée)
- ✓ De 4 connecteurs RJ45 permettant de lui adjoindre des modules d'extension.

Sont également livré avec le robot "mBot" :

- Un module de détection ultrason

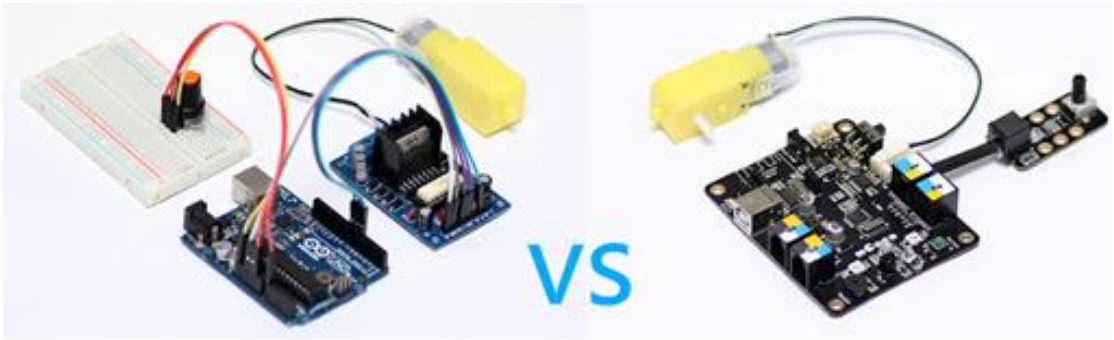
- Un module de suivi de ligne au sol

- Une télécommande IR (pile à ajouter)



Pourquoi choisir la solution mBot ?

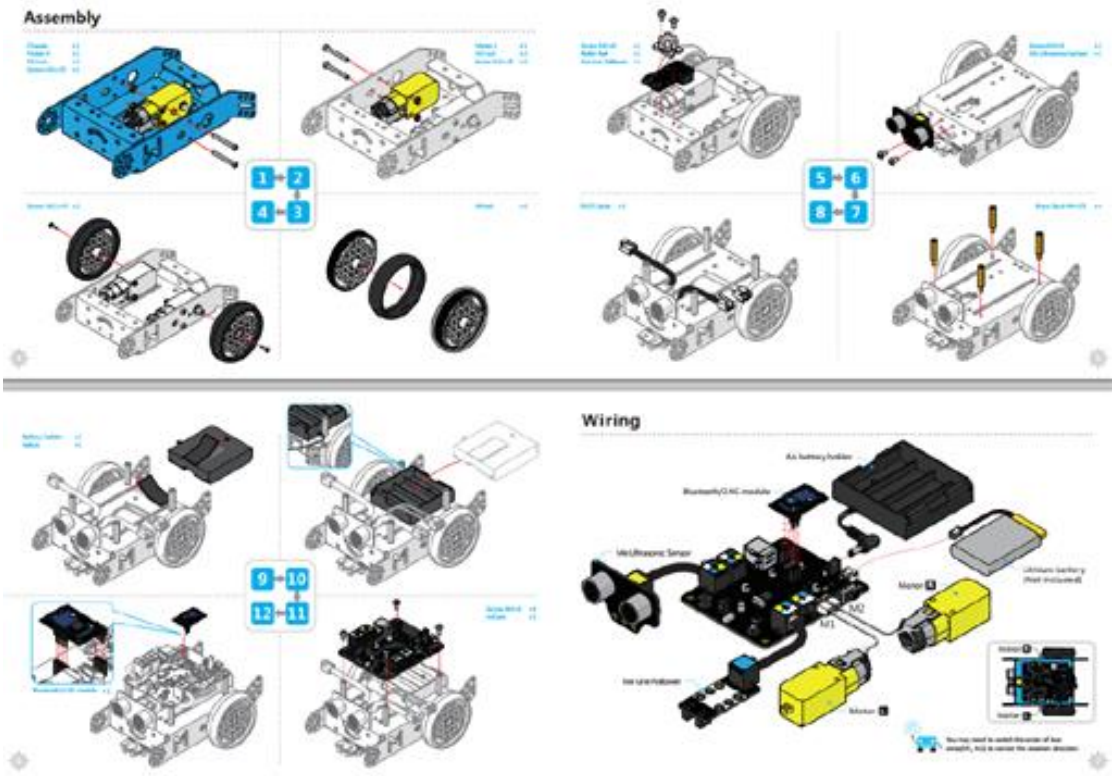
Choisir une base "mBot" vis à vis d'une solution "standard" de type Arduino, c'est choisir un système plus simple, plus rapide et moins cher à mettre en œuvre.



L'exemple ci-dessus montre la configuration nécessaire (à gauche) pour piloter un moteur à courant continu à partir d'une platine Arduino standard. Il vous sera ainsi nécessaire de disposer d'une platine Arduino, d'une interface de puissance, d'un moteur, d'un système de connexion et d'un potentiomètre pour modifier la vitesse de rotation du moteur. Sur la photo de droite vous avez l'équivalent avec une solution mBot !

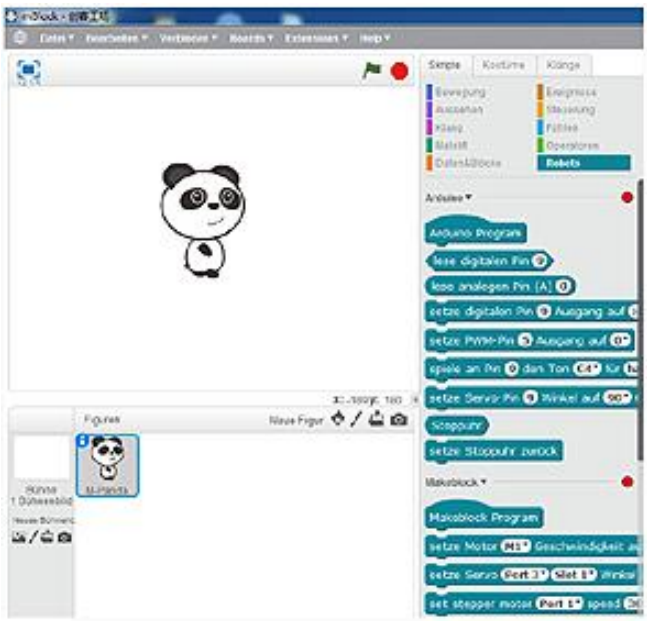
Assemblage simple et rapide

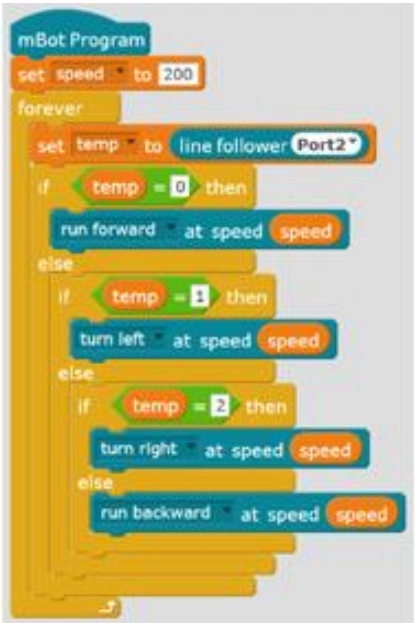
Choisir une base "mBot" c'est choisir un dispositif simple à assembler (sans opération de soudure) et simple à utiliser. L'ensemble des outils étant livrés avec le robot. Le kit comprend même des pignons de rechange pour les moteurs (pour les moins soigneux d'entre vous !)



Programmation en langage graphique Scratch™

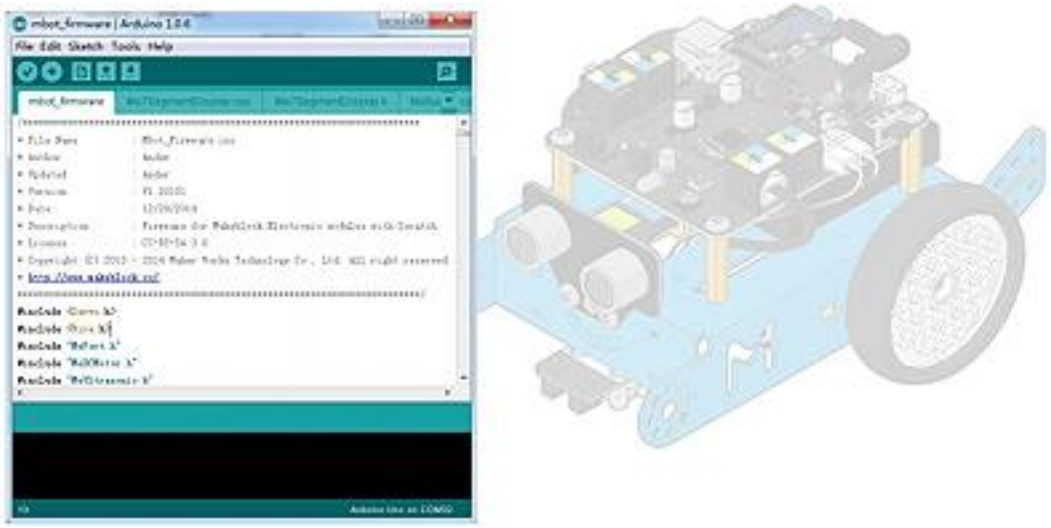
Votre robot "mBot" est facilement programmable en langage graphique Scractch™





Programmation en langage "C" sous environnement Arduino

Votre robot "mBot" est facilement programmable en langage "C" sous l'environnement des Arduino.



Exemples d'utilisations

Votre robot "mBot" dispose de nombreux exemples d'applications lui permettant :

- ✓ De suivre un tracé au sol (une feuille avec un tracé en "8" est à ce titre livré avec le robot)
- ✓ De se déplacer en évitant les obstacles grâce à son capteur ultrason
- ✓ De jouer des notes de musiques
- ✓ De piloter ses leds RGB
- ✓ De piloter votre base à l'aide de sa télécommande IR