Coordination de plates-formes robotiques autonomes, en environnement inconnu pour la recherche et le sauvetage

Nicolas Gauville

UNIVERSITÉ DE LORRAINE

Département de formation doctorale en informatique — École doctorale IAEM Lorraine

Coordination de plates-formes robotiques autonomes, en environnement inconnu pour la recherche et le sauvetage

THÈSE

présentée et soutenue publiquement le 18 Novembre 2022

pour l'obtention du

Doctorat de l'université de Lorraine
(spécialité informatique)

par

Nicolas Gauville

Composition du jury

Président	Julien Bourgeois, Professeur, Université de Bourgogne Franche Comté
Rapporteurs	Philippe Mathieu, Professeur des Universités, Lille 1
	Julien Bourgeois, Professeur, Université de Bourgogne Franche Comté
Examinateurs	Caroline Chanel, Enseignant-Chercheur ISAE Toulouse
	Véronique Serfaty, Responsable Innovation du pole numérique de l'Agence de l'innovation de défense (PhD)
Directeur	François Charpillet, *Directeur de recherche, Inria, Loria*
Encadrants	Christophe Guettier, Safran Electronics & Defense
Encadrants invités	Dominique Maltese, Safran Electronics & Defense

SAFRAN Inria Loria Laboratoire lorrain de recherche en informatique et ses applications

Sommaire

Table des figures

Remerciements

Je tiens tout d'abord à remercier les membres du jury, dont mes rapporteurs Julien Bourgeois et Philippe Mathieu, et mes examinatrices Caroline Chanel et Véronique Serfaty d'avoir accepté de siéger dans mon jury de thèse, d'avoir pris le temps de lire mon travail ainsi que pour tous les retours sur ce dernier. Vos commentaires bienveillants et l'intérêt que vous avez manifesté pour mon travail m'ont permis de conclure cette aventure sur une note très positive.

Je remercie également François Charpillet, Directeur de recherche au centre de l'Inria Nancy, qui m'a permis d'avoir l'opportunité de réaliser ce travail, et qui a encadré et suivis mes recherches. Je remercie Safran Electronics & Defense, qui a financé cette thèse, et notamment mes deux encadrants Safran Christophe Guettier et Dominique Maltèse. Ces trois encadrements ont été très complémentaires, souvent dans des directions différentes, et m'ont toujours une grande liberté dans les pistes explorées.

J'ai eu la chance de pouvoir effectuer ce long travail dans un laboratoire où l'ambiance est amicale et bienveillante. J'aimerais remercier les nombreux autres doctorants, notamment dans les équipes Larsen et MFX, qui m'ont accompagné au cours de ces années de thèse, et notamment Jimmy Etienne, pour son soutien ainsi que les échanges d'idées. Je remercie également chaleureusement Yassine El Khadiri, Clélie Amiot, Jacques Zhong et Lucien Renaud, ainsi que les autres doctorants de l'équipe pour les nombreux moments de convivialité passés tout au long de la thèse.

Je remercie enfin mes parents, qui m'ont donné le goût d'apprendre, et qui m'ont soutenu tout au long de mes études. J'ai été très heureux, après tant d'années, de pouvoir les inviter à ma soutenance de thèse.

Première partie

Contexte et état de l’art

Chapitre 1

Introduction

Sommaire

1 Introduction

Le travail présenté ici s'est déroulé dans le cadre d'une thèse *CIFRE* financée par *Safran Electronics & Defense* en soutien du projet FURIOUS (FUturs systèmes Robotiques Innovants en tant qu'OUtilS au profit du combattant embarqué et débarqué) de la Direction Générale de l'Armement (*DGA*). Ce projet vise à produire trois démonstrateurs de robots (eRider, Vicking et Jaguar) ainsi qu'un drone, pouvant être intégrés à une section d'infanterie. Ces trois robots sont de taille très différentes, de la taille d'un véhicule à celle d'un petit robot pouvant passer des portes. L'objectif principal est de tester l'intégration de robots autonomes en appui d'une section d'infanterie.

L'intégration de robots autonomes dans un cadre militaire, dont les stratégies et la gestion ont été développées et éprouvées pendant des siècles, pose de nouvelles problématiques stratégiques et éthiques. Il n'est pas question de reproduire le comportement humain, ni de concevoir un système dont on contrôlerait chaque action de manière précise à la manière d'un véhicule télécommandé, mais plutôt d'utiliser l'intelligence artificielle pour prédire et répondre aux besoins de la situation, tout en profitant des capacités et atouts différents d'un système robotique. Dans le cadre d'une mission de recherche et de sauvetage par exemple, un robot peut potentiellement prendre des risques que l'on ne pourrait pas faire prendre à un être humain, ou résister à des conditions difficiles et dangereuses. Les drones utilisés par les pompiers lors de l'incendie de Notre-Dame en sont un bon exemple. L'enjeu est avant tout de développer des robots capables de

s'intégrer à une stratégie globale pour compléter l'action humaine. L'utilisation de robots hétérogènes suppose aussi de profiter des différents atouts de chacun pour développer une stratégie collective efficace, basée sur la collaboration des robots entre eux, et avec les êtres humains travaillant avec eux.

FIGURE 1.1 – Robot eRider développé par Safran Electronics & Défense dans le cadre du projet FURIOUS.

D'un point de vue éthique, la France a fait le choix de ne pas envisager de robot autonome armé jusqu'ici, et il n'est donc pas question d'action offensive décidée par une intelligence artificielle, comme d'autres pays ont pu le faire avec des drones ayant la capacité de tirer sans intervention humaine. Une partie significative des recherches liées à ces problématiques pourrait ainsi s'avérer utile dans d'autres cadres, telles que vérifier l'absence d'être vivant dans un bâtiment en feu pour éviter de mettre en danger des pompiers, cartographier un grand bâtiment, explorer d'autres planètes, etc.

Ce travail fait suite au projet *Cart-O-Matic* [93]. Cart-O-Matic était l'un des cinq projets fondés par l'Agence Nationale de la Recherche (ANR) pour sa participation au concours de robotique organisé par la *Délégation générale pour l'armement* (DGA). Ce concours intitulé « Défi *CAROTTE* » avait pour objectif de définir un système robotique capable d'explorer un environnement intérieur inconnu et identifier des objets localisés dans cet environnement. Le projet Cart-O-Matic a choisi de déployer un système multirobot. Cart-O-Matic a remporté le concours final en 2012.

Le cadre du projet *FURIOUS* vise à répondre à de nombreux problèmes à l'aide d'un groupe de robots autonomes, dont celui de l'exploration, qui consiste à visiter un environnement de manière exhaustive.

Cette thèse s'intéresse plus particulièrement à l'exploration de milieux intérieurs par une flotte de plusieurs robots autonomes, avec comme objectifs de limiter les commu-

nications entre les robots et fonctionner avec une capacité de calcul restreinte tout en exploitant efficacement les données perçues par les robots. Les approches développées dans ce domaine trouvent des applications militaires et civiles, qu'il s'agisse de contrôler un robot de nettoyage, des robots de recherche et de sauvetage dans des situations d'incendie ou de catastrophe environnementale, ou encore de soutien militaire.

La première partie de cette thèse dresse un panorama des différentes méthodes et outils employés dans le cadre de l'exploration autonome, et présente différentes approches développées dans l'état de l'art. La seconde partie présente les différentes approches proposées au cours de cette thèse.

2 Systèmes embarqués

Les systèmes embarqués sont généralement constitués d'un ensemble matériel et logiciel destiné à effectuer une tâche précise. Les robots utilisés dans l'exploration spatiale sont des exemples marquants de ce type de système. En effet, ils doivent répondre à de nombreuses contraintes typiques des systèmes embarqués telles que :

- l'autonomie : la téléopération de tels systèmes peut être compliquée du fait du délai de réponse potentiellement long, ceux-ci se trouvant parfois très loin de leurs opérateurs induisant des délais incompressibles.
- la fiabilité : les réparations sur place sont souvent exclues, les systèmes embarqués doivent donc généralement fonctionner jusqu'à la fin de leur mission sans intervention humaine. Dans de nombreux cas, la défaillance d'un système embarqué peut provoquer des fortes pertes économiques ou humaines (dans le cas d'un stimulateur cardiaque par exemple, ou d'un robot *Rover* explorant la planète Mars)
- une faible consommation d'énergie : de façon à fonctionner longtemps et en autonomie, les ressources énergétiques (souvent une batterie) sont limitées. Cette contrainte affecte alors souvent les capacités de calculs
- la limitation des communications : dans le cadre militaire par exemple, toute communication peut permettre à une entité ennemie de repérer le dispositif et ainsi compromettre sa mission. De plus, de nombreux éléments matériels peuvent altérer ou bloquer les communications entre les robots (longues distances, blocs de pierre ou murs épais, etc.), nécessitant alors des approches robustes aux pertes de communications.

Aujourd'hui, les systèmes embarqués sont présents dans de nombreux domaines, incluant l'électroménager, les télécommunications, les équipements militaires ou médicaux.

3 Systèmes distribués

Utiliser une flotte de robots pour effectuer une mission d'exploration apporte de nombreux avantages. En termes de robustesse, utiliser plusieurs robots pourra permettre de continuer la mission et la mener à bien même en cas de défaillance d'une partie du groupe, par exemple si un robot est détruit, tombe dans un trou ou se bloque dans un obstacle. Lorsque l'espace à explorer est grand, utiliser plusieurs robots permet également un gain d'efficacité notable, en permettant de répartir l'espace à explorer entre les différents robots.

Une flotte de robots explorant un bâtiment est un système distribué si chaque robot possède son propre contrôleur, et prend alors ses décisions en fonction de ses perceptions et des communications qu'il effectue avec les autres robots. Éviter une approche centralisée, où tous les robots seraient contrôlés à distance par une seule machine, apporte de nombreux avantages. Les robots peuvent ainsi continuer l'exploration même en cas de perte de communication, les délais entre la prise de décision et l'action sont réduits (ne nécessitant pas de contrôle à distance), etc.

Les systèmes distribués représentent aujourd'hui une part importante des systèmes informatiques dans de nombreux domaines. En effet, ces derniers ne font pas intervenir de nœud particulier dont la défaillance entraînerait un dysfonctionnement de l'ensemble du système. Cette caractéristique leur permet une grande extensibilité tout en garantissant une robustesse importante. Ces qualités justifient de l'intérêt croissant qui leur est porté et leur présence notable dans les architectures logicielles et matérielles à travers le monde. Internet, les cartes graphiques et les systèmes multirobots en sont trois exemples.

Les systèmes distribués sont également à l'origine de nouvelles difficultés : un grand nombre d'algorithmes devient plus complexe lorsque l'on souhaite les exécuter de manière distribuée, y compris pour des problèmes simples tels que le tri d'une liste d'éléments, et paralléliser un algorithme est alors une tâche souvent difficile. En effet, concevoir un algorithme parallèle est souvent moins intuitif et plus difficile. Expliquer un algorithme parallèle simple comme l'algorithme de tri bitonique [6], est souvent moins aisé qu'un algorithme non parallèle. La parallélisation permet cependant un gain de performance notable, d'autant plus grand que la part de l'algorithme exécutée parallèlement est grande (loi d'Amdahl [5]).

Un algorithme parallèle est un algorithme utilisant plusieurs processeurs (matériels ou logiciels) simultanément sur une même machine. Une approche distribuée utilise plusieurs machines communiquant ensemble via un réseau. Un système décentralisé est une approche distribuée dans laquelle il n'y a pas de nœud central dont la défaillance entraînerait la défaillance de l'ensemble du système.

Ces caractéristiques s'appliquent également dans le cadre de l'exploration autonome. Un système distribué composé de différents robots bénéficiera d'une robustesse accrue, pouvant supporter la défaillance d'un robot au cours de l'exploration ou la perte de communication avec l'un d'eux. La possibilité d'utiliser un certain nombre de robots pourra permettre un meilleur passage à l'échelle, par exemple en augmentant le nombre de robots si la surface à explorer est grande.

L'utilisation d'un groupe de robots apporte également de nombreuses difficultés. Les données récoltées par les robots doivent alors être fusionnées. Dans le cas de l'exploration, il s'agit d'assembler les différentes cartes de l'environnement produites par les différents robots, ce qui suppose une localisation des robots dans un repère commun. De manière a rendre l'exploration efficace, les robots doivent également se répartir correctement l'espace à explorer, et ainsi éviter que plusieurs robots explorent le même lieu. Les robots sont également susceptibles de perdre leur communication au fil de l'exploration, notamment si l'espace à explorer est grand ou s'ils sont séparés par des obstacles épais. Ce type de problème peut également être abordé en utilisant des communications de proche en proche, un robot pouvant faire office de relais de communications entre deux robots situés de part et d'autre, permettant à la fois d'améliorer la robustesse des communications, et d'utiliser des moyens de transmission à plus courte portée moins énergivore et moins détectable. Enfin, pouvoir utiliser un grand nombre de robots implique d'avoir recours à une méthode dont la complexité des calculs ne dépen-

dra pas, ou peu, du nombre de robots.

4 Problématiques de l'exploration

Explorer un environnement inconnu consiste généralement à découvrir et cartographier un lieu donné ; c'est un problème abordé par l'humanité depuis des millénaires, nécessitant de surmonter de nombreuses difficultés. En effet, la cartographie peut être réalisée de multiples manières selon le choix de la représentation envisagée, et nécessite de mettre en oeuvre des mécaniques de localisation dans cette représentation.

Ces difficultés ont pu être appréhendées de différentes manières selon l'objet cartographié ; des outils tels que les boussoles, le système GPS ont par exemple permis une meilleure localisation dans l'espace.

La localisation et la représentation de l'environnement ne sont cependant pas les seuls critères à prendre en compte pour aborder le problème de l'exploration. En effet, c'est un problème répondant à des problématiques variées, et les données récupérées pour construire la carte seront ainsi de nature très différentes [21]. La construction d'une carte du ciel regroupera généralement des données sur des étoiles en coordonnées astronomiques, les cartes du monde contiendront quant à elles des informations à la fois géographiques mais aussi d'éventuelles frontières définies par l'homme, etc.

De plus, les enjeux de l'exploration et de la cartographie sont également fortement disparates. Il s'agit souvent de produire une carte permettant de se déplacer et localiser efficacement dans un environnement [27, 105, 122], mais il peut également être question de géopolitique, où encore de situation d'urgence, comme lorsqu'il s'agit de plan d'évacuation d'un bâtiment.

Le système de localisation et de représentation des données, le type d'informations récoltées et les objectifs qui ont mené à l'exploration et la cartographie sont ainsi fortement variés, produisant un champ de recherche large s'étendant à de nombreux domaines.

Les problématiques abordées dans cette thèse sont ainsi nombreuses, et correspondent à plusieurs sous-domaines de la robotique autonome. Nous pouvons notamment distinguer :

- l'exploration, visant à découvrir un environnement, souvent de manière exhaustive,
- la cartographie, dont l'objectif est de produire une carte,
- la recherche, visant à trouver un objet où lieu donné dans un environnement,
- la patrouille, où l'objectif est de minimiser le temps entre deux passages à un même endroit, tout en parcourant régulièrement l'ensemble de l'environnement.

Chacune de ces problématiques apporte également de nombreux sous problèmes, tels que le problème de la galerie d'art (où du musée) [13], le problème de l'itinéraire du gardien [17], la recherche de chemin optimal où la planification de trajectoire, etc.

5 Exploration en robotique et intelligence artificielle

L'exploration d'un environnement inconnu est également un problème majeur pour les systèmes multi-agents et la robotique mobile autonome, abordé sur le plan théorique et pratique. De même que lorsqu'il est abordé par l'homme, répondre à ce problème nécessite un moyen de représenter l'espace tout en étant capable de se localiser dans

cette représentation, et les différents moyens d'y parvenir dépendent grandement de l'objet à cartographier. Les objectifs suivis sont également de nature variés et détermineront la stratégie choisie pour explorer l'environnement.

De manière générale, les objectifs de la cartographie en robotique diffèrent de l'exploration humaine ; autrement dit, l'objectif pour les robots n'est pas de reproduire les différents travaux de cartographie menés par l'homme, mais d'acquérir la capacité de cartographier différents environnements avec ou sans représentation explicite.

L'intérêt de robots capables d'explorer et cartographier un environnement est double : d'abord, cela leur permet d'avoir une présentation interne de l'espace et ainsi de pouvoir s'y localiser et se déplacer de leur position à un objectif donné. Ce premier point permet à un robot d'être autonome face à un environnement initialement inconnu, sans nécessairement chercher à communiquer la carte obtenue, et trouve de nombreuses applications, telles que les robots aspirateurs, capables de déterminer un chemin pour nettoyer la totalité d'une surface, retourner à son chargeur de façon autonome, etc.

Dans d'autres cas, l'exploration vise, en plus de se localiser et déplacer de façon autonome, à produire et restituer une carte. La cartographie autonome à l'aide de robots est particulièrement intéressante dans les situations où le terrain à cartographier est difficilement accessible où potentiellement dangereux pour l'homme, et/ou qu'un contrôle des robots à distance est difficile. De nombreuses applications concrètes existent, tels que l'exploration de la planète Mars [34, 104] (où le contrôle à distance implique un délais de transmission trop important), la cartographie sous-marine, la recherche d'une source olfactive [109, 115], l'exploration de zones ennemies dans un cadre militaire (de façon à éviter un risque humain, par exemple dans un terrain miné), où encore la recherche et le sauvetage après une catastrophe naturelle [76] (par exemple, la recherche d'une victime dans un bâtiment en feu).

Ainsi, de la même façon que pour l'homme, l'exploration et la cartographie à l'aide de robots utilisera différents capteurs et représentations de l'espace selon la nature de l'environnement à explorer, et différentes stratégies selon l'objectif suivis.

6 Exploration multirobots

En robotique ou en intelligence artificielle, l'exploration peut également être menée à l'aide d'un groupe de robots. Les avantages peuvent être multiples : répartition de l'espace entre les différents robots (permettant ainsi une exploration plus rapide), où encore l'utilisation d'une flotte de robots non homogène, permettant par exemple à certains robots d'aller plus vite, d'avoir des capteurs plus ou moins précis, ou d'être capable de franchir différents obstacles. Par exemple, un drone aura l'avantage d'avoir plus facilement accès à différents emplacements d'un bâtiment, même avec un sol fortement dégradé, mais aura généralement une autonomie énergétique moindre.

Le cas de l'exploration multirobots ouvre un champ de problématiques plus vaste encore. De manière générale, les communications entre les différents robots devront être prises en compte, ainsi que la fusion des différentes cartes générées par les robots. Les algorithmes d'exploration multirobots considèrent ainsi d'autres aspects que ceux mono-robot. De plus, en cas de flotte non homogène, la répartition des robots en fonction de leurs différentes caractéristiques constituera un facteur de plus à prendre en compte.

7 Problèmes liés

En robotique, le problème de l'exploration vise à explorer un environnement inconnu le plus rapidement possible à l'aide d'un ou plusieurs robots mobiles. C'est une question proche du problème de couverture [94], où l'on souhaite qu'un ou plusieurs robots visitent de manière exhaustive un environnement, sans laisser de zone accessible non-visitée. Les algorithmes de couverture sont généralement regroupés en deux catégories : « *en ligne* », où les robots n'ont pas de connaissance de l'environnement *a-priori*, et « *hors-ligne* » où les robots connaissent déjà la carte [108].

Un autre problème proche est celui de la patrouille, où les robots cherchent à minimiser l'intervalle entre deux passages dans chacun des endroits d'un lieu donné [88]. Il s'agit donc de produire un chemin permettant de passer par un ensemble de points permettant l'observation régulière de l'ensemble de l'environnement.

Ces différents problèmes reposent sur une problématique commune qu'est le *SLAM* [75, 96, 120], qui consiste à se localiser dans une représentation de l'environnement construite au fil de l'exploration.

De très nombreux algorithmes ont été développés pour répondre à ces différents problèmes. Il est difficile de dresser un portrait global de toutes les approches de l'état de l'art tant ces problèmes peuvent être abordés sous différents angles et avec différentes contraintes. En effet, les objectifs principaux (vitesse de l'exploration, exhaustivité, robustesse aux erreurs de localisation, ...), les contraintes considérées (communication restreinte, asynchronisme, capteurs disponibles) où encore la façon de représenter le monde (grille d'occupation, cartes topologiques, ...) diffèrent fortement selon les approches.

8 Contributions

Les objectifs principaux de cette thèse sont de développer de nouvelles approches d'exploration autonome multirobots à la fois efficaces et économes en calculs et en communications.

Dans cette thèse, nous nous sommes concentrés sur les problématiques de contrôle et d'assignation d'objectifs pour les robots. De nombreuses approches de l'état de l'art fournissent des solutions avec différents niveaux d'efficacité et différents coûts. Une partie des contributions de cette thèse vise à s'approcher de l'efficacité des approches globales (considérant l'ensemble des informations perçues par les robots depuis le début de l'exploration) tout en minimisant les calculs et données échangées, en s'inspirant d'approches locales (ne considérant que les informations immédiatement perceptibles par les robots, où d'éventuelles traces déposées au sol par ces derniers).

Dans de nombreux scénarios, il est également souhaitable de pouvoir configurer les algorithmes de façon à en adapter le résultat souhaité. Par exemple, dans le cadre d'une opération de recherche et de sauvetage, certains éléments peuvent être connus à l'avance, tels qu'une position approximative d'éventuelles victimes, ou des zones trop dangereuses à éviter, même pour les robots.

Nous avons également développé une approche visant à prendre en compte des informations de plus haut niveau d'abstraction, telles que les salles et couloirs d'un bâtiment. De cette façon, nous pouvons tirer parti d'informations supplémentaires en permettant aux robots de mieux comprendre l'environnement dans lequel ils se trouvent.

Ce type de représentation des données permet également de partager moins d'informations que les représentations métriques, tout en minimisant les erreurs de planification.

Enfin, nous avons proposé une adaptation de notre approche par « frontières locales » de façon à intégrer les informations sémantiques à l'aide d'une approche par contraintes. Cette façon de procéder permet de paramétrer l'algorithme selon les besoins de la situation, permettant par exemple de privilégier les couloirs aux salles, de définir des zones plus où moins prioritaires dans l'exploration, etc.

8.1 Nouvelles approches

Deux nouvelles approches ont été proposées au cours de cette thèse : l'approche par « *frontières locales* » visant à explorer un environnement de manière exhaustive à l'aide d'une flotte de robots autonomes en minimisant les calculs et communications, et une approche construisant une *carte sémantique* de l'environnement, de façon à mieux utiliser les informations perçues, en inférant des informations de plus haut niveau, telles que la segmentation des pièces et couloirs d'un bâtiment.

Frontières locales

L'approche par *frontières locales* développée au cours de cette thèse a été inspirée des approches par frontières [26, 61] et d'algorithmes fourmis et Brick & Mortar tels que *MDFS* [41], *BoB* [108] et *BMILRV* [102]. L'objectif de cette approche est de permettre une exploration efficace, tout en réduisant significativement les calculs et les données échangées entre les robots, et en maintenant une robustesse importante aux pertes de communications.

Cette approche a été publiée et présentée dans un article aux 27èmes Journées Francophones sur les Systèmes Multi-Agents (*JFSMA*) en 2019 [134] et a reçu le prix du meilleur article. Cette approche est présentée dans le chapitre 7 Exploration par stigmergie.

Cartes sémantiques

La seconde contribution principale de cette thèse est l'approche par « carte sémantique » visant à produire une carte des pièces et couloirs d'un bâtiment, permettant ainsi de minimiser les communications (en ne partageant qu'un graphe beaucoup plus léger qu'une représentation métrique de l'environnement), et de réduire les erreurs de planification de trajectoires (une commande du type « aller au fond du couloir » étant moins soumise à d'éventuelles dérives de capteurs qu'une commande métrique telle que « avancer de 10 mètres et tourner de 90 degrés dans le sens horaire). Nous avons également proposé une adaptation de l'approche par frontières locales utilisant cette carte sémantique.

L'adaptation de l'approche par « frontières locales » repose sur un système de propagation de flots [12] permettant de rendre l'algorithme aisément paramétrable pour différentes situations. Le chapitre 8 Exploitation de données sémantiques présente la méthode de segmentation sémantique de l'environnement et l'adaptation de l'approche par frontières locales.

8.2 Simulateurs

Trois simulateurs ont été développées au cours de cette thèse pour répondre à différents besoins selon les algorithmes : un simulateur simple sur grille développé en Python au début de la thèse, un second simulateur utilisant *ROS* (Robot Operating System) visant à permettre de tester les algorithmes sur des robots réels, et un simulateur 3D plus proche des conditions réélles, développé pendant le confinement pour permettre d'avoir des résultats plus fidèles à la réalité malgré les restrictions d'accès au laboratoire. Ces trois simulateurs et leurs caractéristiques sont présentés dans le chapitre 6 Méthodologie.

9 Plan de la thèse

La première partie de cette thèse présente un état de l'art général de l'exploration multirobots autonome.

- Le chapitre 2 Représentation du monde présente différentes représentations couramment utilisées en robotique pour définir l'environnement des robots.
- Le chapitre 3 Capteurs et extraction de données aborde les principaux capteurs utilisés en robotique autonome dans le cadre de l'exploration, et l'extraction des données utiles à partir de ces derniers.
- Le chapitre 4 présente différentes approches de l'exploration autonome de l'état de l'art, tandis que le chapitre 5 Intégration de données sémantiques se concentre sur les approches basées sur l'inférence de données sémantiques.

La seconde partie aborde les principales contributions de cette thèse :

- Le chapitre 6 Méthodologie présente les différents simulateurs développés au cours de cette thèse et utilisés pour tester les approches proposées, ainsi que les hypothèses générales considérées dans cette thèse.
- Le chapitre 7 Exploration par stigmergie présente l'approche par « frontières locales » et différentes versions visant à optimiser les données échangées et améliorer l'efficacité de l'exploration.
- Le chapitre 8 Exploitation de données sémantiques aborde l'approche par segmentation sémantique de l'environnement, et l'adaptation de l'approche par frontières locales aux cartes sémantiques.

Enfin, la partie 3 présente une synthèse du travail réalisé et des perspectives à explorer.

Chapitre 2

Représentation du monde

Sommaire

1 Introduction

Il existe de nombreuses façons de représenter le monde pour un robot. En 1992, ANGELOPOULOU et al. proposent une liste d'une quinzaine de façons de représenter l'environnement pour un robot. Malgré le développement de nombreuses nouvelles méthodes de représentation de l'environnement [21], cette liste donne encore aujourd'hui un panorama approprié des principales approches couramment utilisées. De manière générale, les représentations utilisées sont choisies selon l'approche utilisée, les données utiles dans le contexte de la mission et les capteurs disponibles sur le robot. Ce chapitre présente les principales représentations du monde couramment utilisées en robotique mobile.

2 Dimension du problème

De nombreux critères sont à prendre en compte pour choisir la façon dont on représentera les perceptions des robots, et différentes manières de les représenter sont possibles selon le contexte.

2.1 Données métriques ou symboliques

Les données métriques sont généralement utilisées pour définir l'environnement du robot. En effet, de nombreux capteurs produisent des données métriques (LiDARs, caméras RGB-D, etc.), et ces dernières peuvent ainsi aisément être enregistrées pour produire une représentation métrique de l'environnement [18, 19].

D'autres approches visent à produire des données symboliques, souvent avec un plus haut niveau d'abstraction, représentant par exemple les passages reliant différents lieux, le type d'obstacle rencontré, etc. L'utilisation de données symboliques nécessite généralement un traitement supplémentaire des données des capteurs de façon à les interpréter, mais produit plusieurs avantages, notamment un poids souvent plus léger, et une utilisation plus efficace par les algorithmes (ayant accès à des données souvent plus concises, et fournissant des informations de plus haut niveau d'abstraction) [15].

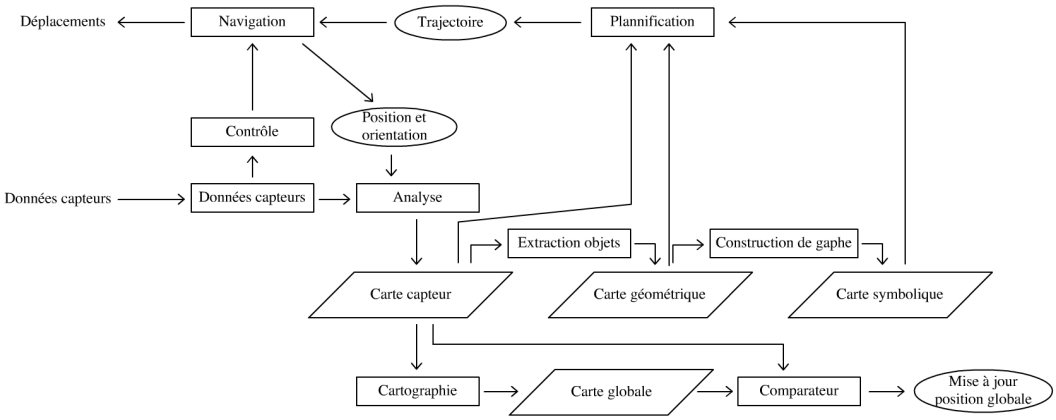


FIGURE 2.1 – Représentation simplifiée de l'architecture utilisée dans [15].

La figure 2.1 présente l'architecture utilisée par [15] pour construire et utiliser une carte symbolique à partir des données des capteurs (sonars).

2.2 Incertitude

Les données représentées peuvent également être probabilistes, dans la mesure où les données perçues sont soumises à de nombreuses erreurs et donc incertaines. Dans la plupart des cas, les données seront corrigées au fil des déplacements des robots. Certaines approches associent un niveau d'incertitude aux données représentées [19].

2.3 Deux et trois dimensions

Les données peuvent être représentées en deux ou trois dimensions selon le contexte. Dans les premières représentations utilisées, les robots ont souvent été restreints à

un environnement plat, et une représentation en deux dimensions de l'environnement s'avère alors suffisante pour explorer efficacement un environnement.

D'autres approches ne peuvent pas se contenter d'une représentation bidimensionnelle de l'environnement, dans le cas d'un terrain non plat, si le robot se déplace en trois dimensions (drones [149], exploration sous-marine [144]), où encore si les données à enregistrer où prendre en compte vont au-delà des objets au sol (utilisation de caméras RGB-D ou de LiDARs multicouches [142]).

3 Données représentées

Avant de nous intéresser à la façon de représenter la carte et les données associées à celle-ci, commençons par voir différentes informations utiles pouvant être perçues ou mémorisées par les robots.

3.1 Occupation de l'espace

Une approche simple de la représentation du monde est d'établir une carte permettant de savoir, pour un point donné, s'il est occupé (par un obstacle) ou non. L'objectif est ici d'obtenir une carte de navigation pour permettre au robot de se déplacer dans l'espace et de reconnaître son environnement. Pour un point donné, différentes informations pourront être considérées : d'abord la probabilité que ce point soit occupé par un obstacle, mais aussi éventuellement des informations sur son accessibilité (capacité du robot à atteindre le point considéré). Dans de nombreux cas, il est difficile de savoir avec certitude pour chaque point donné s'il est occupé ou non à cause de l'incertitude des capteurs utilisés (souvent LiDAR ou caméra RGB-D), on préfère ainsi utiliser une probabilité d'occupation, plutôt qu'une valeur booléenne indiquant si une zone est occupée ou non.

3.2 Visibilité et accessibilité

Un point donné d'un lieu peut être visible (par les capteurs du robot) depuis différents endroits de ce lieu. Cette information est utile dans certains contextes, par exemple dans une problématique de patrouille ou de surveillance d'une carte, et diffère de l'accessibilité ou de l'occupation de la carte. Il peut donc être utile de mémoriser, sur une carte spécifique, la visibilité d'un robot pour un point donné. Cette information supplémentaire peut également permettre d'améliorer la précision de la carte construite au fil de l'exploration.

De manière générale, l'accessibilité dépend essentiellement des capacités du robot (sa taille, s'il roule, saute, vole, glisse, etc). De nombreux travaux dans ce domaine visent à déterminer la capacité du robot à de déplacer à un endroit donné selon le type de robots utilisés, ainsi que le problème considéré. Les problématiques d'accessibilité sont en effet nombreuses, qu'il s'agisse de la détection d'obstacles éloignés [46], de l'exploration d'autres planètes en faible gravité [124], où encore la navigabilité avec des obstacles mobiles [141].

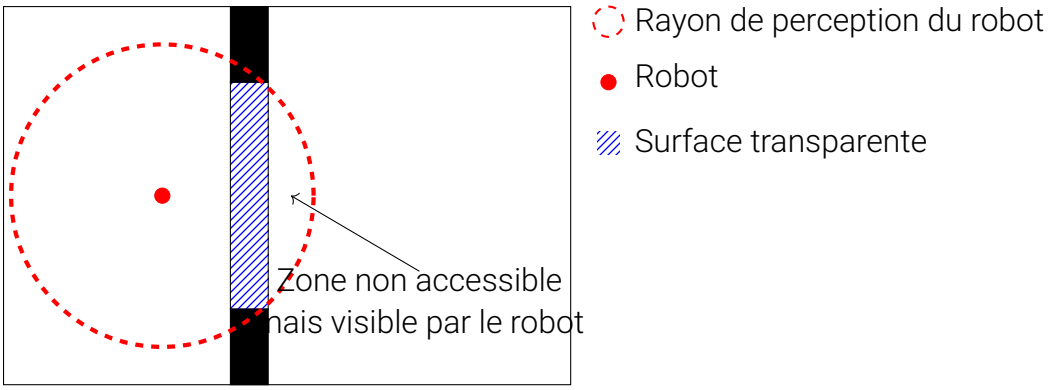


FIGURE 2.2 – Exemple de situation ou une partie de l'espace est visible, mais non accessible par un robot.

3.3 Informations sémantiques

Lorsque l'objectif de l'exploration s'étend au-delà de la navigation (cartographie, recherche d'objets, etc.), la carte produite inclura généralement d'autres informations que l'occupation ou l'accessibilité de l'environnement du robot, telle que la nature des obstacles qui ont été perçus par un robot, des informations sur les différentes salles d'un bâtiment, des zones considérées dangereuses pour un robot (par exemple, un sol boueux risquant d'enliser un robot à roues), etc.

Les informations sémantiques peuvent ainsi permettre d'éviter certains problèmes spécifiques aux cartes métriques. Par exemple en donnant des informations de navigation du type « aller au fond du couloir puis tourner à droite »au lieu de « avancer de 12,5m puis tourner de 90 degrés dans le sens horaire », la trajectoire effectuée ne souffrira pas d'une éventuelle erreur de localisation du robot de quelques mètres ou degrés.

De plus, les informations sémantiques identifiées peuvent permettre de guider le robot. Par exemple, si le robot cherche un objet se trouvant sur un four, l'identification d'une pièce avec un lit et une commode pourra lui indiquer que l'objet cherché a peu de chances de se trouver ici, et que passer à nouveau par une porte augmentera sa probabilité de se rapprocher de l'objet.

L'utilisation de données symboliques avec un plus haut niveau d'abstraction de l'environnement peuvent ainsi apporter de nombreux avantages :

- meilleure compréhension de l'environnement, en ayant des informations sur les objets rencontrés, la structure d'un bâtiment (salles, couloirs), etc,
- réduction des informations mémorisées (les informations symboliques pouvant être plus simples que les données métriques) dans certains cas,
- simplification des algorithmes de navigation, en utilisant moins de données (par exemple, un simple graphe au lieu d'une grille d'occupation détaillée).

Ce type de représentation rapproche également le comportement et les capacités des robots des comportements humains, où les choix sont effectués à partir de représentations symboliques de l'environnement et d'une compréhension globale de celui-ci. Cela nous rapproche ainsi du domaine de la robotique cognitive, c'est-à-dire visant à faire réaliser à des robots des tâches qui, pour l'homme, nécessitent la mise en œuvre de fonctions cognitives telles que la perception, la motricité, l'apprentissage, le raisonnement où encore la cognition spatiale [113].

4 Structures de données

La représentation du monde générée par les robots au fil de l'exploration pourra ainsi être représentée de nombreuses manières. Dans certains cas, plusieurs couches, éventuellement représentées de manières différentes, pourront être utilisées de manière concurrente pour enregistrer différents types de données. Commençons par voir les différentes structures couramment utilisées pour représenter l'environnement des robots.

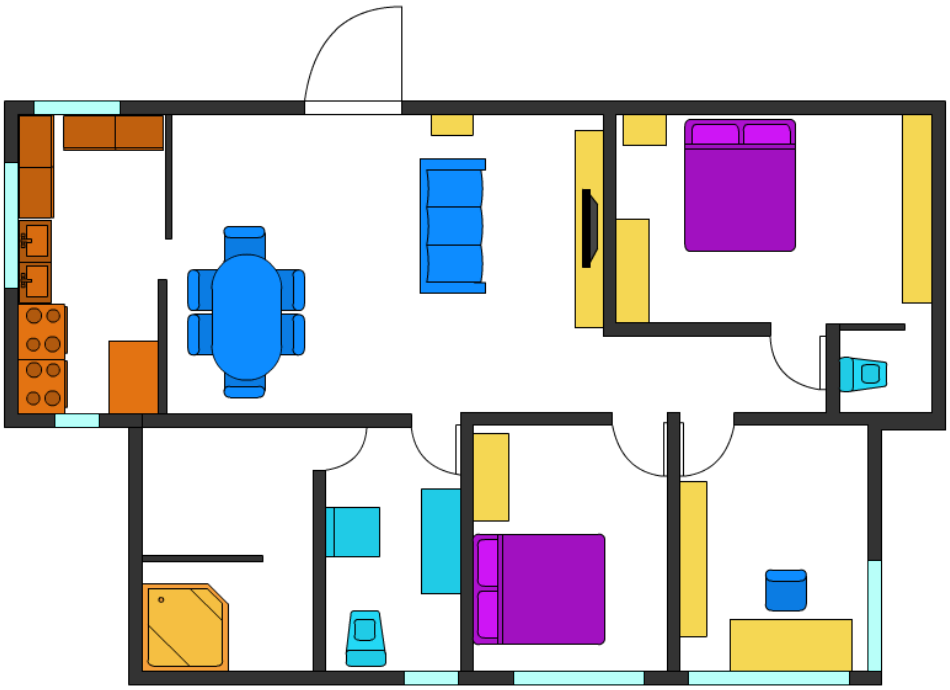


FIGURE 2.3 – Exemple de carte d'un appartement à explorer

Supposons que l'espace à explorer est un appartement, représenté figure 2.3. Nous représenterons cet appartement avec différentes structures de données dans cette partie.

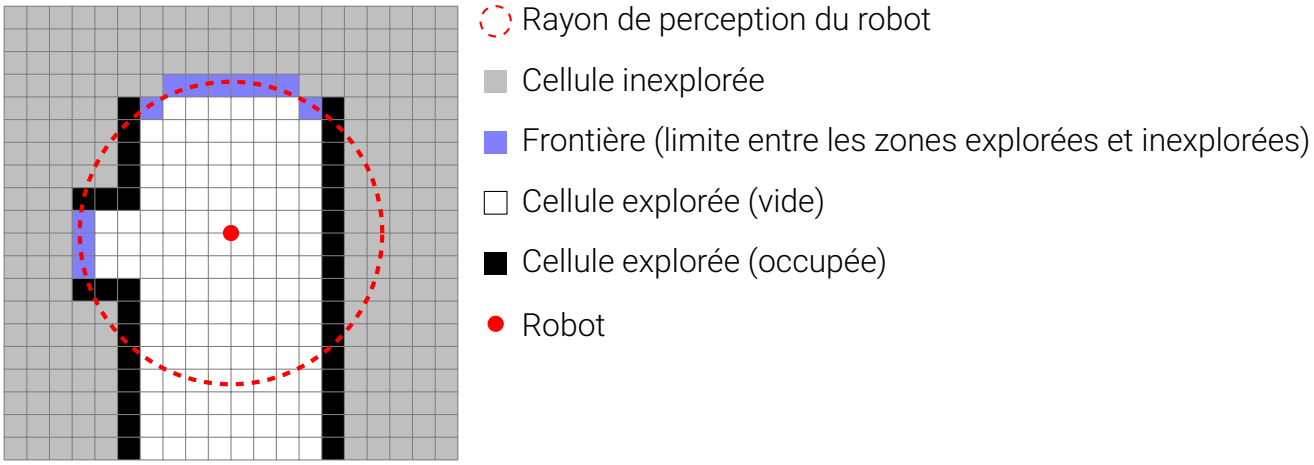


FIGURE 2.4 – Représentation d'une grille d'occupation d'un robot

4.1 Grilles et voxels

L'une des représentations les plus courantes est la grille d'occupation, qui consiste à diviser le monde en cases et noter, pour chacune d'elle, la probabilité qu'elle soit occupée ou non par un obstacle. Cette représentation convient particulièrement lorsque les robots sont équipés de capteurs précis et à longue portée (par exemple des LiDARs), et peut fonctionner en deux ou trois dimensions [95].

Il est également possible d'adapter la taille des cases aux différents besoins où à la géométrie de l'environnement [77]. Les grilles d'occupation ont l'avantage d'être à la fois un modèle adapté aux capteurs (il est facile de remplir une grille d'occupation à l'aide des observations envoyées par un capteur laser), et à de nombreux algorithmes couramment utilisés, une grille pouvant de plus être vue comme un graphe si nécessaire. La figure 2.4 représente une grille d'occupation pour un robot (dans un cas non probabiliste, où une cellule vue est occupée ou inoccupée de façon certaine).

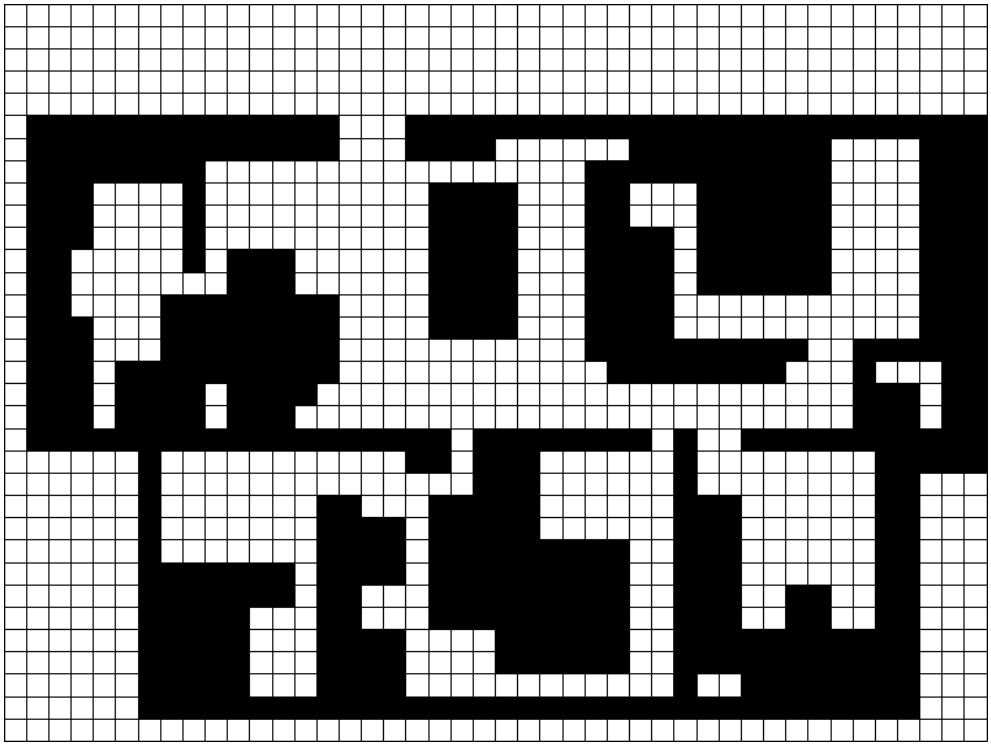


FIGURE 2.5 – Grille d'occupation de l'appartement

En revanche, les grilles d'occupation ne donnent qu'une information restreinte de l'espace : les zones occupées (par un objet, un mur, etc.) ou non, et n'apportent aucune information sémantique de ce qui a été observé par les robots.

La représentation des données sous forme de grille ne se limite pas à l'occupation ou non de l'espace. Cette représentation est également appropriée pour le stockage d'autres types d'informations, comme les cartes d'accessibilité et visibilité, ou des informations sémantiques décrites dans la partie 3.

La figure 2.5 représente une grille d'occupation correspondant à l'appartement (figure 2.3). Ici, de nombreux objets et murs sont entre deux cases, et forcent donc deux cases à être vues comme "obstacles". Dans certains cas, cet échantillonnage bloque alors des zones en réalité potentiellement accessibles par le robot.

Voxel

Dans le cas d'un espace à trois dimensions, un voxel se substituera à la grille. Un voxel est une grille tridimensionnelle [138, 140]. Cette représentation peut être utilisée de la même façon qu'une grille, mais permet le stockage des informations en trois dimensions, ce qui peut être particulièrement utile dans le cas où les robots perçoivent et/ou peuvent se déplacer en hauteur.

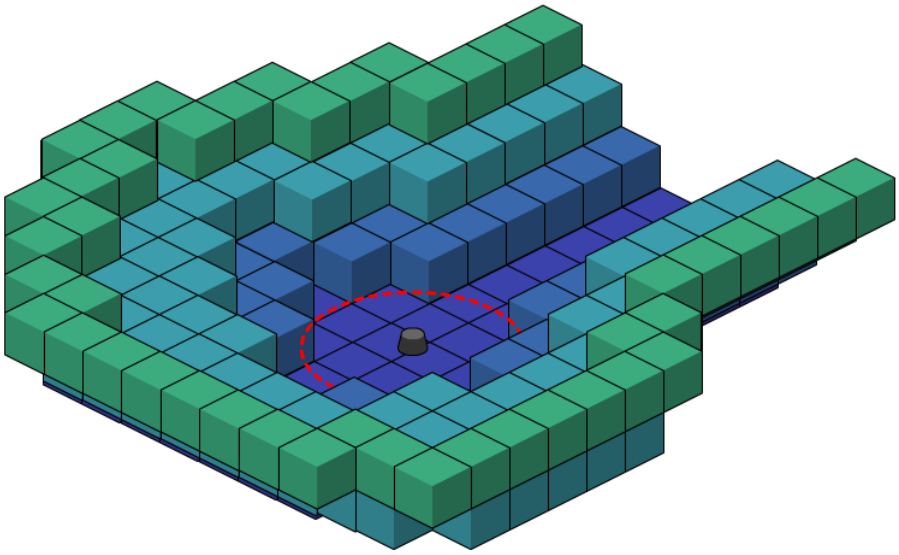


FIGURE 2.6 – Carte d'occupation représentée par un voxel

Cette représentation est souvent utilisée pour les drones. De nombreux algorithmes d'exploration plane fonctionnant sur des grilles peuvent être adaptés au cas de drones se déplaçant dans l'espace avec les voxels.

Limitations

L'un des principaux défauts des grilles et voxels réside dans la discrétisation de l'espace. Une grille avec des cellules trop étroites pourra être trop lourde (en mémoire, calcul ou communication), et une grille avec des cellules trop larges posera de nombreux problèmes. Le choix de la taille des cellules dépendra du robot (notamment sa taille et ses capacités de mouvement) et de la nature de l'espace à explorer.

4.2 Quadtree et octree

Quadtree

Un Quadtree [11] est une structure de donnée sous forme d'arbre (avec relations de parenté), qui permet de palier aisément aux problèmes de granularité décrits précédemment avec les grilles d'occupation et voxels.

Le principe général est de partir d'une grille large, et de subdiviser les cases lorsque celles-ci contiennent des informations de natures différentes. Si l'on utilise un quadtree pour représenter l'occupation de l'espace, les zones totalement libres ou totalement occupées pourront ainsi être de grandes cellules, tandis que les zones frontières autour

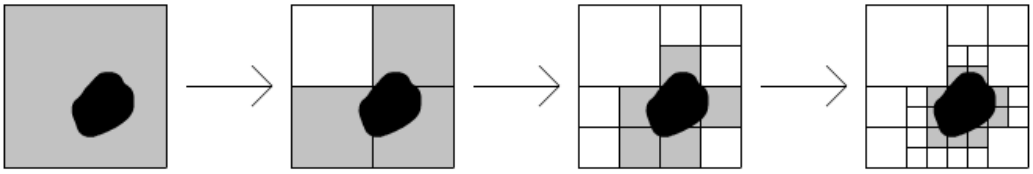


FIGURE 2.7 – Construction itérative d'un octree représentant l'occupation de l'espace

des obstacles seront subdivisées jusqu'à obtenir une discrétisation de l'espace suffisamment précise. La figure 2.7 représente la construction itérative d'un quadtree délimitant un obstacle (représenté en noir). Les cases grisées contiennent l'obstacle, et sont ainsi subdivisées au fil des itérations.

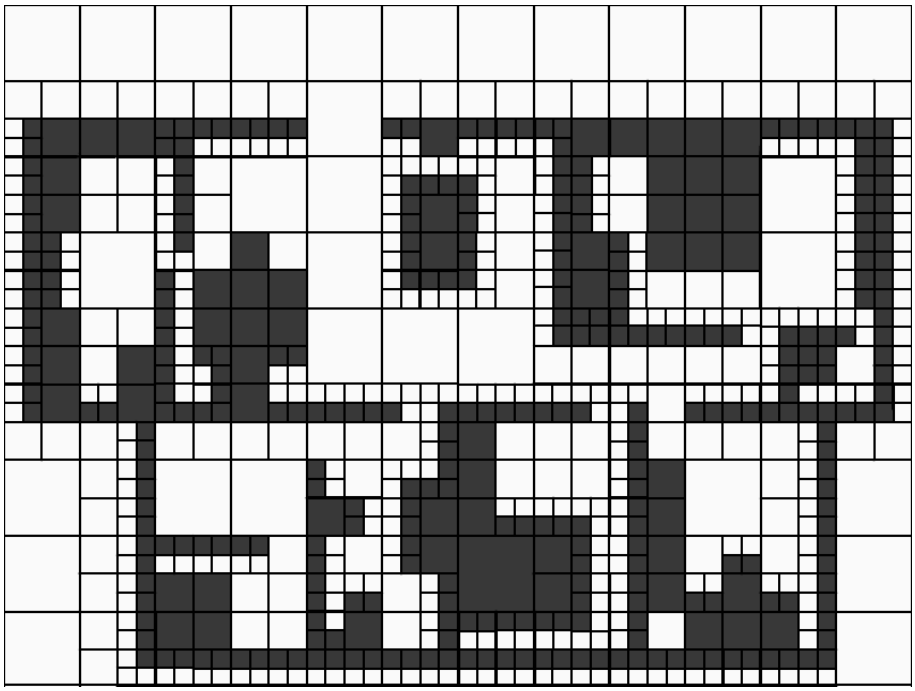


FIGURE 2.8 – Carte d'occupation de l'appartement avec un quadtree

Cette méthode est donc un compromis, permettant une représentation de l'espace simple et légère tout en conservant une précision importante lorsque c'est nécessaire [44].

Octree

De la même façon que pour la grille d'occupation, cette méthode fonctionnera en 2D ou en 3D. Dans le cas de la 3D, une cellule de l'octree pourra se subdiviser en 8 cubes (là où les cases d'un quadtree se subdivisent en 4 cellules dans le cas 2D) [9].

Les octrees sont utilisés dans de nombreux domaines, tels que la génération de modèles, l'addition-soustraction de géométries ou encore la simulation de phénomènes physiques [10, 45].

L'utilisation d'un octree peut poser des difficultés, telles que trouver efficacement le voisinage d'une cellule, mais de nombreuses solutions ont été proposées pour faire face à ces difficultés [74, 38].

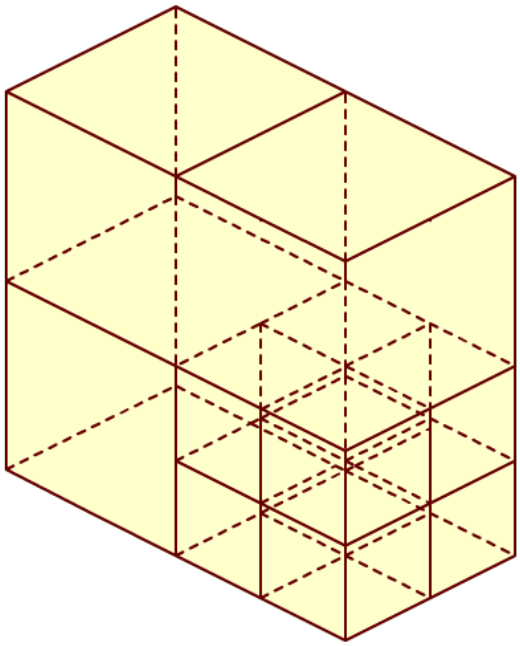


FIGURE 2.9 – Représentation d'un Octree, avec l'une des cellules subdivisées en 4 sous-cellules.

En exploration autonome sur un environnement 3D (par exemple d'un milieu sous-marin ou avec des drones), les octrees permettent de réduire la quantité de données nécessaires pour maintenir une carte, tout en évitant les problèmes de granularité posés par des cellules trop grandes [57, 99, 100, 95].

4.3 Addition-soustraction de chemins ou géométries

De nombreux algorithmes et bibliothèques logicielles permettent de travailler avec des chemins ou géométries en réalisant différentes opérations binaires (addition, soustraction, union, différence). En deux dimensions, des bibliothèques logicielles telles que Clipper [1] réalisent aisément ce type d'opération. En trois dimensions, de nombreuses bibliothèques permettent le même type d'opération, habituellement nommées CSG (Constructive Solid Geometry).

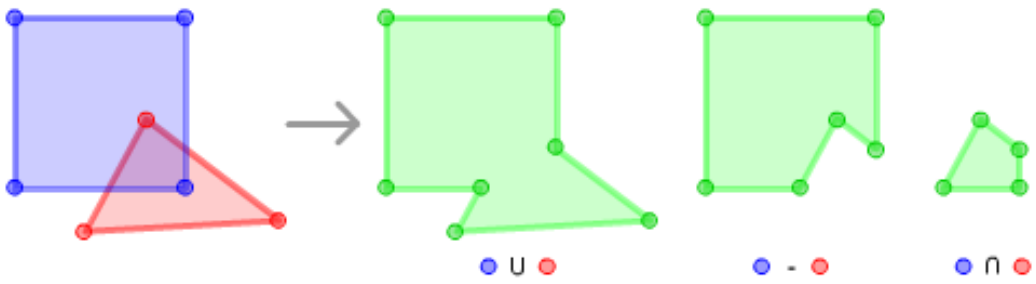


FIGURE 2.10 – Exemples d'opérations sur des chemins (union, différence et intersection).

En deux dimensions, les tracés sont représentés par un ensemble de coordonnées de points. On peut alors obtenir l'union, l'intersection ou la différence de deux tracés, ou

1. Page de la bibliothèque logicielle Clipper http ://www.angusj.com/delphi/clipper.php

réaliser différentes opérations comme appliquer une marge pour grossir ou réduire la surface d'une forme. La figure 2.10 montre un exemple de deux formes (en rouge et en bleu) et du résultat obtenu pour leur union, différence et intersection.

Grâce à ce type d'opérations, il est possible de construire une carte à l'aide d'un ensemble de géométries ou tracés, englobant les espaces accessibles et inaccessibles par exemple [150, 125].

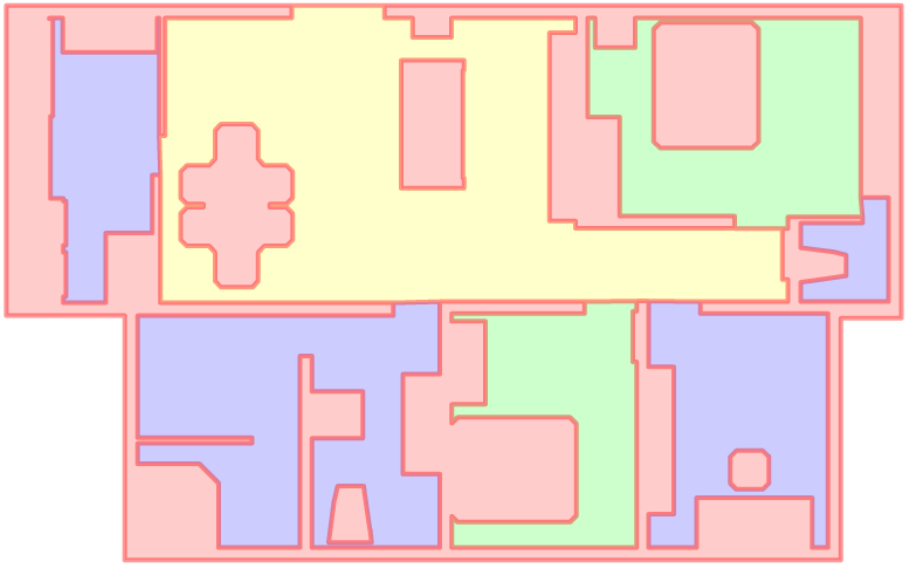


FIGURE 2.11 – Représentation de l'appartement sous forme d'ensemble de tracés

La figure 2.11 présente l'appartement représenté par un ensemble de chemins clos, correspondants aux obstacles et différentes pièces. Ce type de représentation permet d'avoir une carte avec un faible volume de données (uniquement les points représentant les contours des zones libres/occupées), et les tracés peuvent être simplifiés en supprimant les points inutiles ou redondants, souvent avec un simple appel de fonction des bibliothèques d'addition-soustraction de géométries.

Pour la planification sur ce type de représentation, différentes méthodes sont possibles, comme la triangulation de la zone non occupée [20] puis l'utilisation d'un algorithme de pathfinding sur la zone triangulée [101].

4.4 Graphes

Représenter la carte sous forme d'un graphe permettant de connaître différentes positions entre lesquelles les robots peuvent naviguer peut simplifier significativement les algorithmes utilisés pour l'exploration ou la couverture d'une carte [85, 121].

Le chapitre 8 propose un algorithme pour construire un graphe composé des différentes pièces et portes qui les relient au fil de l'exploration.

4.5 Arbres couvrants

À partir d'un graphe, il est également possible de déterminer un arbre couvrant, c'est-à-dire un sous-graphe sans cycles passant par l'ensemble des nœuds de l'arbre.

La figure 2.13 représente un graphe (à gauche) et un arbre couvrant de ce graphe (à droite). Le *théorème de Kirchhoff* permet de calculer de nombre d'arbres couvrants pour un graphe quelconque [66].

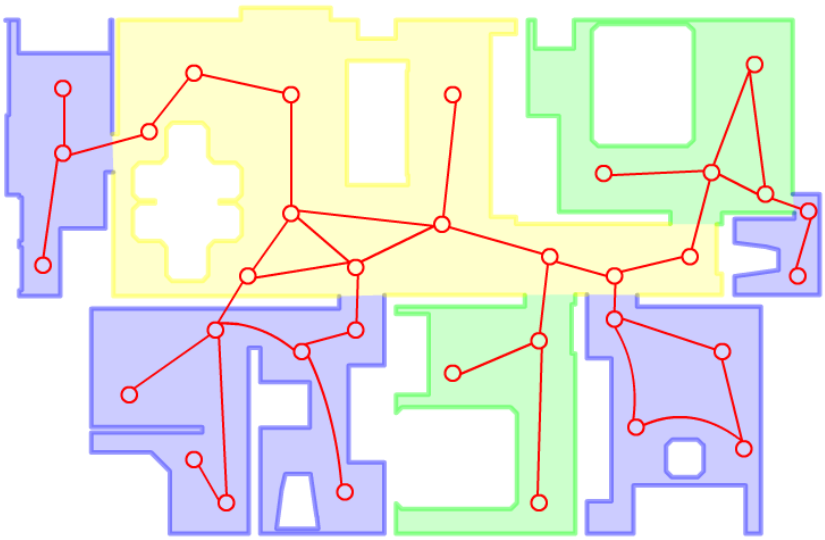


FIGURE 2.12 – Représentation de l'appartement sous forme d'un graphe de navigation

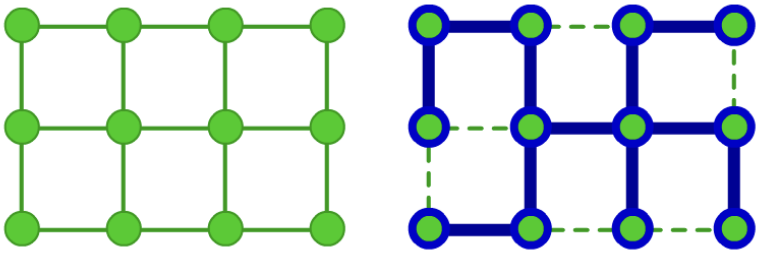


FIGURE 2.13 – Représentation d'une grille sous forme de graphe non orienté (à gauche), et d'un arbre couvrant de ce graphe (à droite, les arrêtes plus épaisses correspondent aux arrêtes de l'arbre couvrant)

Si l'on pondère un graphe, de nombreux algorithmes permettent de calculer un arbre couvrant de poids minimal de ce graphe, c'est à dire où la somme du poids des arrêtes est minimal [148]. Obtenir un graphe couvrant de poids minimal peut-être utile en vue l'exploration ou la couverture, et plusieurs approches utilisent ainsi ce type de représentation (voir chapitre 4. Approches de l'exploration, partie 6. Couverture de graphes).

4.6 Informations sémantiques

Les cartes sémantiques, plus proches d'une représentation humaine, consistent en un ensemble de lieux distincts reconnaissables par les robots, reliés entre eux par des arcs indiquant les déplacements possibles d'un lieu à un autre [33, 76, 127].

Ce type de représentation est généralement utilisé dans l'objectif de simplifier l'environnement, qu'il s'agisse de contraintes de mémoire ou de calculs. Dès 1996, THRUN et al. proposent une approche hybride utilisant à la fois une carte topologique et une grille d'occupation [25, 126]. L'approche proposée dans le chapitre 8 Exploitation de données sémantiques utilise une carte sémantique de l'environnement.

5 Contraintes et objectifs

La plupart des approches et des représentations utilisées visent à répondre à différentes contraintes inhérentes à la robotique. De manière générale, pour des raisons de coût et de batterie, la consommation de mémoire, de calculs et d'énergie est limitée. L'environnement à explorer peut être de différentes natures, nécessitant alors des approches prenant en compte ces particularités : certains matériaux peuvent fausser significativement les capteurs (miroir ou vitre pour un capteur laser par exemple), le sol peut poser des problèmes de franchissabilité, de longs couloirs peuvent rendre difficile une localisation précise, etc. Dans le cas d'un groupe de robots, d'autres difficultés s'ajouteront, telles que la fusion des données des différents robots, le maintien de la communication malgré d'éventuels obstacles les séparant ainsi que la répartition du terrain à explorer.

6 Conclusion

Le choix de la représentation du monde ainsi que des données représentées dépendra donc de la nature de la mission à effectuer, mais aussi des capteurs disponibles sur les robots.

Dans le cadre de cette thèse, nous utiliserons des Turtlebots équipés de LiDARs mono couches. L'utilisation de grilles d'occupation est particulièrement adaptée pour construire facilement une carte représentant l'occupation de l'environnement, et est généralement fournie par la plupart des modules de *SLAM*.

En vue de limiter les données échangées, l'utilisation d'un graphe réduit la quantité d'informations nécessaires. Nous verrons dans le chapitre 7 Exploration par stigmergie comment adapter notre algorithme d'exploration multirobots de façon à ne plus partager de grille d'occupation, mais une simple liste de positions successives.

Dans le chapitre 8 Exploitation de données sémantiques nous construirons une carte sémantique de l'environnement en identifiant les différentes pièces et salles d'un bâtiment au fil de l'exploration, de façon à tirer parti de plus d'informations symboliques et réduire encore la quantité d'informations partagées.

Chapitre 3

Capteurs et extraction de données

Sommaire

1 Introduction

Pour pouvoir percevoir leur environnement, les robots doivent être munis de capteurs et exploiter efficacement les signaux de ces derniers. Utiliser différents types de

capteurs permet d'avoir plus d'informations à exploiter, mais aussi de palier aux défauts de chacun d'eux. Par exemple, un miroir dans une pièce pourra facilement tromper un capteur optique, mais sera moins propice à fausser un capteur à ultrasons ; un capteur laser aura généralement une portée bien plus élevée qu'un capteur à ultrasons, etc. Utiliser un ensemble cohérent de capteurs et les exploiter conjointement est ainsi essentiel pour limiter les erreurs et percevoir correctement l'environnement, tout en ayant un maximum d'informations à exploiter.

1.1 Proprioception et exteroception

En robotique mobile, deux grandes familles de capteurs peuvent être considérées. Les capteurs proprioceptifs effectuent des mesures relatives aux déplacements du robot. Les accéléromètres, les gyroscopes, les inclinomètres, ou les capteurs mesurant l'angle des roues d'un robot sont de bons exemples de capteurs proprioceptifs. Les capteurs spécifiques à l'état du robot, par exemple pour connaître son niveau de batterie, font également partie des capteurs proprioceptifs.

À l'inverse, les capteurs extéroceptifs effectuent des mesures de l'environnement. Les LiDARs et caméras RGB-D en sont de bons exemples. Ces capteurs permettent de connaître l'environnement, mais seront également utiles à la localisation relative du robot. Par exemple, la variation des données d'un LiDAR peut permettre de reconstituer les différents obstacles entourant le robot au fil du temps, et ainsi d'inférer le déplacement du robot entre de mesures à partir des variations de position de ces obstacles.

De manière à construire une carte de l'environnement tout en nous repérant dans cette même carte, nous utiliserons habituellement à la fois les capteurs proprioceptifs et extéroceptifs, ainsi que les commandes envoyées aux actionneurs. En associant ces trois types d'informations, il est possible de réduire significativement les erreurs de localisation et de perception [48].

1.2 Cartographie et localisation simultanées (SLAM)

L'un des problèmes fondamentaux de la robotique mobile est la cartographie et la localisation simultanées [51, 118]. Même lorsqu'il ne s'agit pas de cartographie, il est essentiel pour un robot mobile de pouvoir se repérer dans le monde qui l'entoure. Concrètement, il n'est pas possible pour un robot mobile de cartographier un environnement sans pouvoir connaître sa position dans la carte qu'il est entrain de créer.

C'est une problématique plus vaste que l'odométrie, qui consiste à estimer le déplacement réel du robot, en tenant compte des commandes envoyées aux actionneurs, et éventuellement des capteurs sur ces derniers (estimant par exemple la rotation réelle des roues). L'odométrie est également propice aux erreurs de dérive (décalage progressif entre la position réelle et la position estimée), et une bonne localisation nécessite donc une approche plus globale, incluant différentes sources d'informations. Par exemple, repasser par un endroit déjà connu peut permettre de corriger une éventuelle dérive de capteurs (fermeture de boucle).

En anglais, la cartographie et la localisation simultanées sont généralement abrégées *SLAM* (Simultaneous Localization and Mapping). De nombreuses bibliothèques logicielles fournissent un module de *SLAM* intégré, fournissant directement une carte et la position du robot relative à cette carte à partir des données des capteurs dispo-

nibles, telles que *gmapping* [1] sur ROS [2] qui fonctionne avec les données d'un LiDAR. Ces dernières années, de nombreuses approches utilisant des caméras plutôt que des capteurs laser ont été développées [107].

Dans une problématique multirobots, il faut alors localiser les différents robots dans un repère commun, et fusionner les données acquises au fil des déplacements. Différentes approches ont été étudiées pour les *SLAM* multirobots [49, 96, 55, 120].

2 Erreurs de mesure

De manière générale, différents types d'erreurs surviennent lors de l'utilisation d'un capteur quelconque, chacun pouvant être traité d'une manière différente. L'utilisation de différents types de capteurs permettra souvent de mieux tolérer les erreurs de mesure, chacun d'entre eux ayant tendance à obtenir des erreurs accidentelles dans des circonstances différentes.

2.1 Erreurs systémiques

Les erreurs systémiques correspondent à des erreurs qui affectent la totalité des mesures effectuées par un capteur. Elles peuvent être constantes, renvoyant alors une mesure biaisée d'une valeur fixe, comme une balance mal tarée qui ajoutera (ou soustraira) toujours la valeur non nulle qu'elle indique lorsqu'elle est vide. De manière générale, la calibration des capteurs est le meilleur moyen de pallier à ce problème. Notons qu'il est possible de corriger les erreurs de calibration au cours d'une exploration [23, 123].

Une erreur systémique n'est pas toujours constante, et peut ajouter un biais variable selon la valeur mesurée, rendant plus difficile la calibration et sa détection. Il est alors nécessaire de faire une calibration sur différentes mesures pour identifier et corriger les données du capteur.

2.2 Propagation des erreurs

Au-delà des erreurs dues à de mauvaises calibrations, il est également possible d'avoir une dérive des capteurs dans certains cas spécifiques. Un cas typique est le parcours d'un long couloir par un robot. Dans un couloir long et monotone, les observations se ressembleront, et identifier correctement la position du robot à l'aide d'un LiDAR ou d'une caméra s'avérera difficile, n'ayant pas de repère se déplaçant par rapport à la position du robot. Dans ce type de cas, s'il n'y a pas d'autres sources d'informations pour mieux localiser le robot (notamment d'odométrie), ou que ces derniers sont faussés (par exemple par un sol glissant), la position estimée du robot pourra se retrouver de plus en plus éloignée de la réalité au fil du temps.

Différentes approches sont possibles pour contrer ce type de problème. De manière générale, revenir dans un environnement déjà visité permet de corriger une éventuelle dérive de capteur, et se réaligner au repère initial. Pour ce faire, il est nécessaire de correctement reconnaître un lieu déjà visité, qu'on y revienne en revenant sur nos pas ou depuis un autre endroit (problème de la fermeture de boucle dans les algorithmes de SLAM) [65].

1. Gmapping http ://wiki.ros.org/gmapping
2. Robot Operating System https ://www.ros.org

L'utilisation de cartes topologiques (définissant différents lieux et les chemins d'accès entre eux) permet également de limiter ce type de problème, en évitant alors les données métriques. Différentes approches ont été développées visant à privilégier les cartes métriques locales, et une carte topologique reliant les différentes cartes métriques, pour obtenir des données locales précises tout en évitant les problèmes de dérive [143]. La figure 3.1 montre un exemple des conséquences de ce type d'erreur sur une grille d'occupation. Le chapitre 8 propose différentes approches de représentation sémantique de l'environnement de façon à pallier à ce type d'erreur.

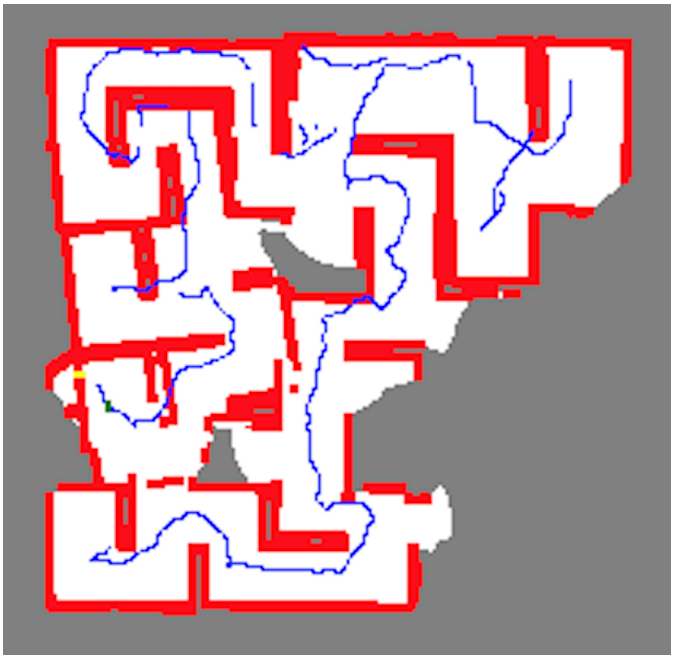


FIGURE 3.1 – Grille d'occupation d'un robot lors d'une simulation sur ROS. Les obstacles sont en rouge et les trajectoires en bleu. On peu constater que certains murs sont perçus non alignés, du fait d'erreur de localisation angulaire au fil de l'exploration.

2.3 Erreurs accidentelles

Les erreurs accidentelles peuvent également être divisées en plusieurs catégories. Quel que soit le type de capteur utilisé, des valeurs aberrantes peuvent être renvoyées de temps à autre, ne correspondant pas à la réalité, et étant potentiellement très éloignées des valeurs correctes. Ce type d'erreur est facile à éliminer : les valeurs aberrantes étant souvent très éloignées des valeurs correctes, la simple comparaison aux mesures précédentes permet de les repérer et de les éliminer [64, 43]. Pour ce type d'erreur, la multiplication des mesures au fil du temps et un filtrage des valeurs reçues suffit pour reconnaître les fausses mesures. Lors de la construction d'une grille d'occupation, les approches probabilistes permettent également de prendre en compte les erreurs, assignant aux cellules une probabilité d'occupation plutôt qu'une valeur fixe, de façon à prendre en compte l'incertitude potentielle de certaines mesures.

Des erreurs peuvent également se produire dans certaines circonstances selon le type de capteur utilisé. Par exemple, un capteur laser pourra être affecté par un miroir, une vitre, ou différents matériaux renvoyant la lumière ; un capteur à ultrasons par des tissus et matériaux absorbants efficacement les sons, etc. Pour ce second type d'erreur accidentelle, une des meilleures approches est d'utiliser conjointement différents types

de capteurs. Ainsi, une erreur accidentelle de l'un des capteurs peut être détectée et corrigée par un autre capteur, insensible au problème ayant provoqué l'erreur. Le couplage d'un capteur visuel (LiDAR ou caméra) à un capteur d'un autre type (ultrasons, capteur de contact) est souvent une solution efficace.

2.4 Erreurs liées aux hypothèses

Des erreurs peuvent également survenir dans le cas où les conditions réelles ne vérifient pas les hypothèses considérées lors de la définition de l'approche.

Par exemple, de nombreuses approches utilisent des grilles d'occupation en deux dimensions, en supposant que le sol est plat et que l'environnement du robot peut ainsi être restreint à un plan. Si le sol est penché, où que le robot se déplace sur un obstacle qui n'est pas plat, les perceptions de ce dernier seront alors différentes du fait de son inclinaison, et produiront des données erronées dans le repère à deux dimensions.

3 Extraction de données

Les capteurs qui équipent les robots dans une mission d'exploration ont différents objectifs : d'une part, permettre à ces robots d'effectuer correctement leur mission, en étant aptes à percevoir et se localiser dans l'environnement, et d'autre part à récolter les informations souhaitées au fil de l'exploration. Dans les deux cas, il sera nécessaire d'extraire les informations utiles à partir des données brutes récoltées par les capteurs au fil du temps (obstacles présents, zones accessibles, informations sémantiques, etc.).

3.1 Odométrie et estimation de la position

De très nombreuses sources d'informations permettent d'obtenir une estimation de la position d'un robot au fil du temps, telles que les capteurs GPS, les retours du système de déplacement du robot (roues codeuses, consommation de courant des moteurs ...), et différents capteurs extéroceptifs et proprioceptifs (boussole, accéléromètre, LIDAR et caméras).

L'approche la plus courante est le recours à un module *SLAM* (*Simultaneous Localization and Mapping*) qui donne une estimation de la position d'un robot au cours du temps en fusionnant différentes sources de données et de manière parallèle à la construction de la carte du robot (la carte ne pouvant être construite sans estimation de la position du robot et réciproquement). Ainsi, l'odométrie est issue de capteurs proprioceptifs, dans un repère propre au robot, tandis que le *SLAM* utilise des capteurs extéroceptifs, et fournis une localisation dans la carte construite par le robot, et donc relative aux obstacles et objets identifiés par le ou les robots.

En pratique, l'estimation de la position par un module *SLAM* est assez précise, mais peut être soumise à une dérive parfois importante pour différentes raisons ; par exemple de longs couloirs uniformes peuvent rendre difficile l'estimation de la distance parcourue, et les rotations successives d'un robot produisent régulièrement des dérives d'angles potentiellement significatives [75]. Pour qu'une mission d'exploration soit menée à bien, la justesse du SLAM sera alors déterminante.

Un problème classique des algorithmes de SLAM est la fermeture de boucle. Supposons qu'un robot suive différents couloirs jusqu'à revenir à sa position de départ, comme illustré sur le schéma de gauche de la figure 3.2, mais que ce dernier subisse une dérive

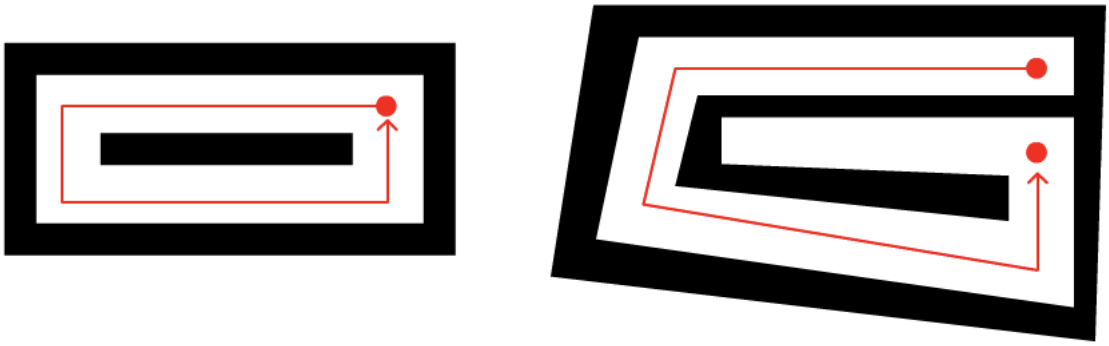


FIGURE 3.2 – Illustration du problème de fermeture de boucle suite à la dérive de capteurs.

de capteurs et accumule ainsi une erreur de localisation au fil de son exploration. La partie droite de la figure 3.2 montre ce à quoi pourrait ressembler la carte perçue par le robot ainsi que sa trajectoire dans cette dernière. Dans cet exemple, la carte produite est incohérente, et le robot ignore qu'il est revenu à son point de départ. Si d'autres données lui permettent alors de savoir qu'il est à nouveau à sa position de départ, il est alors possible de corriger la carte produite et ainsi régler l'incohérence qui s'en est suivie [78].

3.2 Obstacles

Quels que soient les capteurs utilisés, la détection des obstacles est l'une des composantes essentielles de la robotique autonome. Dans la majorité des cas, les obstacles fixes détectés seront mémorisés, souvent à l'aide d'une grille d'occupation (voir chapitre 2 sur la représentation du monde). Les obstacles seront généralement détectés à l'aide d'un LiDAR, d'une caméra (classique ou RGB-D) et/ou de capteurs ultra-sons.

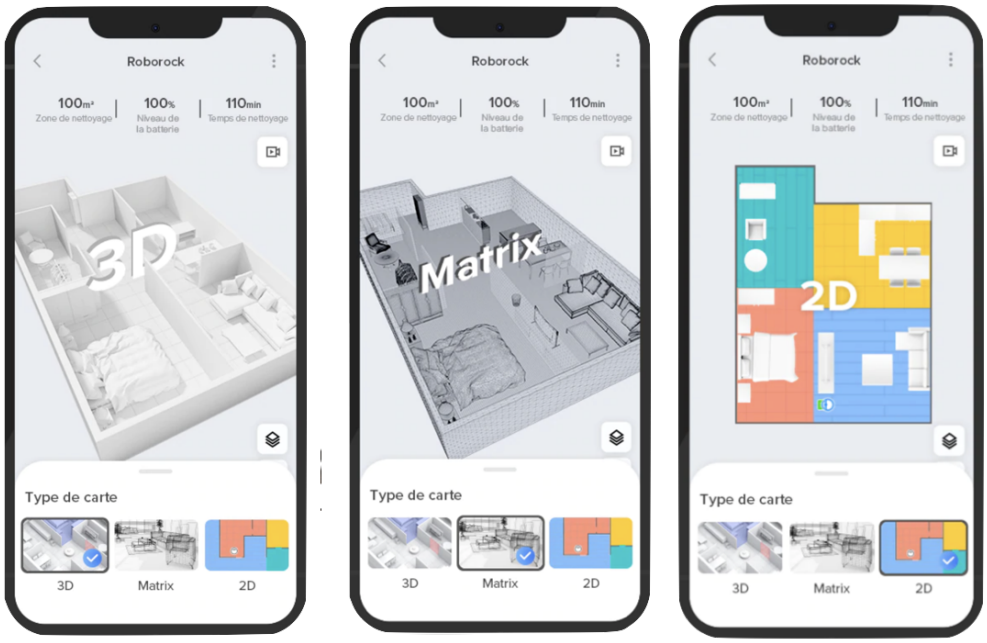


FIGURE 3.3 – Interface de l'application mobile de gestion du robot aspirateur Roborock Série S7 MaxV tirée du site https://fr.roborock.com/pages/roborock-s7-maxv

De nombreux robots prendront également en compte des capteurs de contact et les données issues du SLAM pour détecter des obstacles bloquants (empêchant le passage du robot) sur des zones ayant à tort été détectées comme accessibles (par exemple un petit obstacle bloquant le passage du robot, mais trop bas pour être vu par le faisceau d'un LiDAR, induisant ainsi le robot en erreur). La grille d'occupation ou la carte topologique permettront ensuite au robot de planifier ses déplacements en tenant compte des obstacles connus de son environnement.

Ce type de technologie est aujourd'hui largement utilisé dans des produits commerciaux disponibles au grand public, notamment les robots aspirateurs. Par exemple, le robot *Roborock Série S7 MaxV*, dont l'interface mobile est présentée figure 3.3 utilise un LiDAR, un capteur de contact et une caméra RGB. À l'aide de ces capteurs, il est capable de cartographier une maison ou un appartement sur plusieurs étages. Il peut reconnaître les différentes pièces et meubles, permettant ainsi à l'utilisateur de lui indiquer des zones à prioriser, ou d'en interdire d'autres. Sa caméra et son LiDAR lui permettent également de reconnaître différents objets, lui évitant ainsi de déplacer des objets, se coincer avec un câble, ou encore de rétracter sa serpillière lorsqu'il passe sur un tapis.

3.3 Données topologiques

Les données topologiques représentent une couche d'abstraction supplémentaire des données perçues par les robots. Il s'agit d'identifier des voies d'accès, des chemins, permettant le passage d'une position à une autre, permettant ainsi au robot de construire un graphe représentant les différents trajets qu'il peut effectuer dans son environnement. Une carte topologique présente différents avantages : de manière générale, ces données seront plus légères en mémoire qu'une grille d'occupation. D'un point de vue algorithmique, l'exploration d'un graphe peut être effectuée par de nombreux algorithmes très simples et potentiellement plus légers que ceux s'appliquant à une grille. Enfin, une carte topologique pourra être moins soumise aux données métriques, pouvant ainsi corriger certaines erreurs de mesure décrites au début de ce chapitre.

3.4 Reconnaissance d'objets

Au-delà de la détection d'obstacles et des données sémantiques, il est également possible d'essayer de reconnaître les objets de l'environnement d'un robot, notamment à l'aide de caméras ou caméras RGB-D.

Reconnaître les objets environnants peut présenter divers avantages. L'un des objets peut être un objectif de la mission (trouver un objet ou une personne dans un lieu), identifier certains objets peut permettre au robot d'éviter des risques, ou répondre à des besoins spécifiques (comme les robots aspirateurs qui vont éviter de s'approcher des cordes et fils qui risqueraient de les bloquer). La reconnaissance d'un objet peut également guider le robot dans son exploration, par exemple, si un robot doit trouver une chambre, l'identification d'un réfrigérateur peut lui indiquer qu'il est peu probable qu'il soit dans la bonne pièce [139].

De nombreuses méthodes existent pour reconnaître des objets [67, 68, 119] allant des algorithmes de classification aux réseaux de neurones. Des algorithmes tels que *YOLO* (*You Only Look Once*) [117], permettent de reconnaître et étiqueter en temps réel des objets présents sur une image.

4 Capteurs

Comme nous l'avons vu au début de ce chapitre, il est possible de choisir parmi de très nombreux capteurs pour équiper un robot mobile en vue d'une mission d'exploration. Ces capteurs peuvent être de différentes qualités et précisions, et intégrer différents fonctionnements. Cette partie dresse un portrait rapide des principaux capteurs utilisés dans ce type d'utilisation.

4.1 Capteur laser

Les *LiDARs* (*LIght Detection And Ranging*) sont couramment utilisés en robotique mobile. Ces derniers sont aujourd'hui abordables et disponibles sur une large gamme de prix et de principes de fonctionnement. De nombreux appareils grand public en sont équipés, tels que les smartphones haut de gamme ou les robots aspirateurs.

Dans le cas des LiDARs utilisés en robotique mobile, il s'agit généralement d'un laser permettant d'estimer la distance des objets environnants, faisant des tours réguliers à 360 degrés (avec un éventuel angle mort du fait des spécifications du LIDAR utilisé et/ou de la structure du robot). Ce type de capteur renvoie généralement un ensemble de nombres, correspondant aux distances mesurées sur chacun des angles parcourus. Ces LiDARs rotatifs peuvent être monocouches ou multicouches (mesurant la distance sur différentes hauteurs). D'autres types de LiDARs existent, comme ceux équipant les *iPhones* utilisés notamment pour la reconnaissance faciale (FaceID) et le suivi de visage (pour les effets d'appareil photo, le découpage de la scène avec la simulation d'un effet Bokeh, etc.).

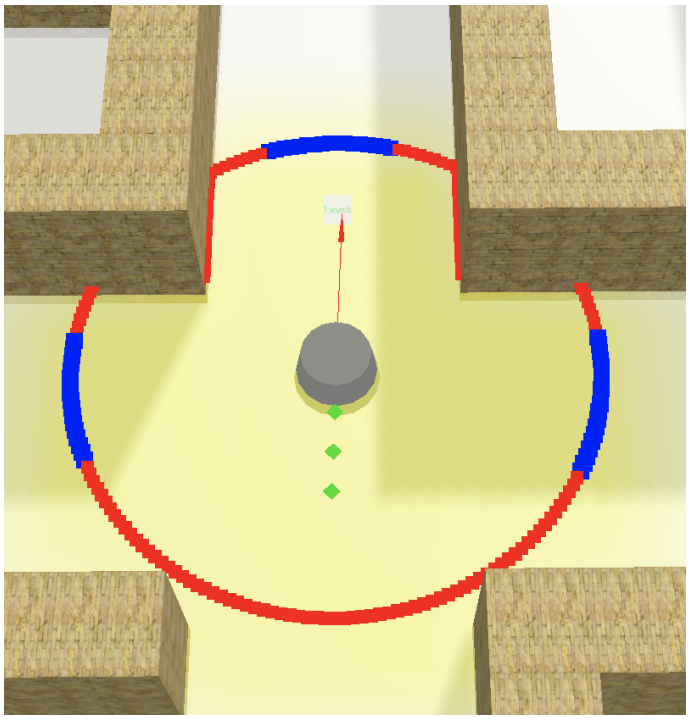


FIGURE 3.4 – LIDAR d'un robot sur le simulateur.

Dans les simulateurs développés au cours de cette thèse et présentés chapitre 6, les robots simulent un LiDAR en effectuant des lancés de rayon sur un nombre fixé d'angles à chaque itération. L'intervalle d'angle entre deux points et le nombre de points mis à jour

à chaque instant sont deux paramètres des simulateurs. De façon à se rapprocher d'un LiDAR réel, il est possible de bruiter les données reçues, ou de considérer des surfaces/-matériaux spécifiques qui affectent les données (par exemple un matériau réfléchissant renvoyant une valeur faussée). Enfin, la distance maximale d'un point projeté (rayon de perception du robot) est également choisie.

La figure 3.4 montre un exemple d'un LIDAR sur un robot du simulateur 3D. Les points projetés sont représentés en rouge ou bleu selon s'ils sont considérés comme destinations éligibles par l'algorithme fonctionnant sur le robot. La figure 3.5 montre un LiDAR 3D compatible *ROS* disponible sur le marché.

FIGURE 3.5 – Lidar 3D 360° Outdoor de la société Robosense, 16 nappes proposant une portée de 150m, compatible ROS.

4.2 Caméra RGB-D

Les caméras et caméras RGB-D font partie des capteurs usuels en robotique autonome. Ces dernières peuvent permettre d'identifier des objets, mais aussi se substituer à l'utilisation de LiDARs dans certains robots.

Les caméras RGB-D correspondent à des caméras qui perçoivent également la distance de chaque pixel, correspondant donc à un LiDAR multicouche avec perception des couleurs. Dans le grand public mais aussi en robotique, les capteurs de type *Kinekt* sont de bons exemples de caméras RGB-D. La Kinekt, initialement développée pour les jeux vidéos puis abandonnée, est encoure aujourd'hui utilisée pour la capture d'animations et de mouvements ainsi que comme capteur pour la robotique autonome ; un paquet *ROS* (Robot Operating System) prenant en charge la *Kinekt* existe également.

La figure 3.7 montre le rendu des deux caméras d'un robot lors d'une simulation (les différents niveaux de gris de la caméra distance correspondent à une échelle de distance par rapport au robot pour chaque pixel).

Ce type de capteur intervient généralement en complément d'un LiDAR, et peut aider le robot à avoir une meilleure appréciation de l'environnement dans lequel il se situe.

FIGURE 3.6 – Caméra RGB-D Kinect produite par Microsoft de 2010 à 2017, encore utilisée aujourd'hui dans le domaine de la robotique

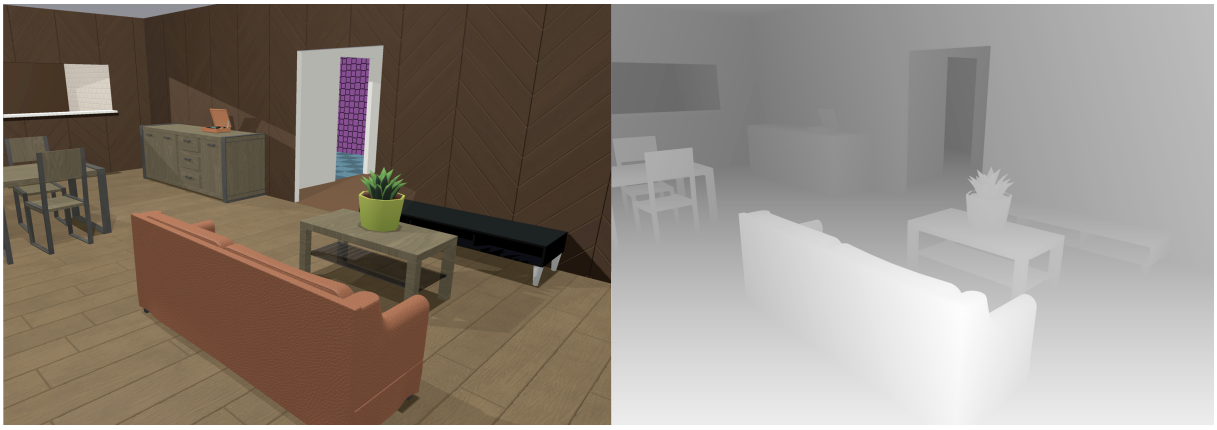


FIGURE 3.7 – Caméras d'un robot du simulateur

4.3 Capteur à ultrasons

Les capteurs à ultrasons, tout comme les LiDARs, font rebondir une onde sur les objets environnants pour mesurer la distance aux obstacles. À la différence des LiDARs, ces derniers ne sont généralement pas rotatifs, et font office de capteurs de proximité [130], de façon à percevoir les objets proches, et non scanner l'ensemble de l'environnement sur une distance importante.

Comme nous l'avons vu, l'utilisation de capteurs à ultrasons en plus d'autres types de capteurs permet de réduire les erreurs de détection, notamment pour les objets ayant des propriétés particulières vis-à-vis de la lumière [147, 71, 83].

La figure 3.8 montre un exemple de capteur de distance à ultrasons. Ce type d'appareil fonctionne sur le même principe que l'écholocalisation chez les dauphins, en émettant un son rebondissant sur les obstacles avoisinants.

4.4 Boussole

Pour un ou plusieurs robots autonomes, l'utilisation d'une boussole numérique permet d'avoir un repère angulaire fixe. En intérieur, une boussole numérique peut cepen-

FIGURE 3.8 – Capteur de distance à ultrasons 40kHz par Adafruit Industries LLC

dant être facilement perturbée par certains équipements électromagnétiques, mais cette particularité peut également être utilisée pour mieux localiser un robot [32].

4.5 Accéléromètre

Les accéléromètres font également partie des capteurs couramment utilisés, pour mesurer l'accélération d'un objet. Comme les gyroscopes, ils équipent aujourd'hui de nombreux smartphones et robots, où encore dans différents contrôleurs de jeux vidéos comme les manettes de *Wii*, les montres connectées pour la mesure d'activités sportives, les appareils photo pour la stabilisation de l'image, etc.

Les accéléromètres sont constitués d'un système masse-ressort, permettant de mesurer l'influence des accélérations sur les mouvements de la masse, sur un à trois axes.

En robotique, ils peuvent constituer un capteur supplémentaire aidant à mieux localiser le robot, notamment dans une problématique d'odométrie [54].

4.6 Gyroscope

Les gyroscopes sont des capteurs d'inclinaison, permettant de mesurer l'inclinaison d'un objet sur trois axes, à l'aide d'un disque dont l'axe de rotation est libre.

Comme les accéléromètres, les gyroscopes sont largement utilisés dans les appareils grand public, par exemple pour faire tourner l'interface d'un téléphone (en portrait ou paysage) selon l'inclinaison du téléphone.

Pour un robot, la connaissance de l'inclinaison de ce dernier peut s'avérer très utile pour adapter les données des autres capteurs. Par exemple, un robot équipé d'un LiDAR percevra différemment les obstacles si le sol est en pente, et la connaissance de cette inclinaison permettra de corriger les données perçues.

4.7 Microphone

Les microphones de qualité moyenne sont des capteurs très bon marché pouvant s'avérer très utiles en complément d'autres capteurs, notamment dans une problématique multirobots.

Un ou plusieurs microphones peuvent également être utilisés pour différents objectifs, tels que localiser et trouver une source sonore, continue ou intermittente [116].

4.8 Centrale inertielle

Les centrales inertielles sont des instruments de mesure de position et mouvement (vitesse et accélération linéaire et angulaire) utilisés dans les systèmes de navigation, par exemple dans les avions. Les centrales inertielles viennent généralement en complément d'un *GPS* pour améliorer la précision de la localisation. Ils équipent également souvent les véhicules militaires, de façon à pallier à un éventuel brouillage *GPS* en temps de guerre. Dans le civil, ces derniers équipent également souvent les avions pour éviter les problèmes de perte de GPS durant les phases d'approche (fusion de données visuelles et de données provenant de la centrale inertielle).

FIGURE 3.9 – Centrale inertielle *GI550* du fabriquant *ASC* décrite sur le site *PM Instrumentation* https://www.pm-instrumentation.co.

Une centrale inertielle est composée de trois accéléromètres ainsi que trois gyromètres, ainsi qu'un module de calcul utilisant ces six capteurs.

5 Fusion de données

La fusion et l'interprétation des données des différents capteurs sont des aspects essentiels de la perception en robotique. De nombreuses approches existent pour traiter les données perçues, en extraire les données utiles, et fusionner les informations perçues par les différents capteurs. Les principales méthodes utilisées seront présentées dans cette section.

5.1 Filtrage Bayésien

Le filtrage Bayésien consiste à discriminer des données en groupes en utilisant le théorème de Bayes [1], définissant la probabilité qu'un évènement survienne à partir d'un autre évènement indépendant déjà produit. Il est défini comme suit :

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Avec :

- $P(B) \neq 0$
- A et B deux évènements de probabilité $P(A)$ et $P(B)$
- $P(A|B)$ et la probabilité conditionnelle que l'évènement A se produise étant donné que l'évènement B s'est produit

À partir de ce théorème, le filtrage Bayésien consiste à filtrer des données en utilisant la probabilité qu'une donnée appartienne à un groupe et des probabilités conditionnelles selon différentes caractéristiques de la donnée considérée. Le filtrage Bayésien est notamment utilisé dans les filtres antispam, de façon à discriminer les messages indésirables des autres, en tenant compte des mots contenus dans les messages [28].

La probabilité qu'un message soit indésirable sachant qu'il contient un mot M est alors déterminée de la manière suivante :

$$P(S|M) = \frac{P(M|S)P(S)}{P(M|S)P(S) + P(M|H)P(H)}$$

Avec :

- $P(M|S)$ la probabilité que le message soit indésirable, sachant qu'il contient le mot M
- $P(S)$ la probabilité qu'un message soit indésirable (ratio du nombre de messages indésirables sur le nombre de messages total)
- $P(M|S)$ la probabilité que le mot M apparaisse dans un message indésirable (nombre de messages indésirables contenant le mot M divisé par le nombre total de messages indésirables)
- $P(H)$ la probabilité qu'un message ne soit pas indésirable (et donc égal à $1-P(S)$)

Une application plus concrète du filtre Bayésien aux données de capteurs (observations bruitées) pour l'estimation de l'état d'un système est le filtrage de Kalman, décrit dans la section suivante.

5.2 Filtres de Kalman

Les filtres de Kalman [4] permettent d'avoir une estimation de l'état d'un système (par exemple, la position d'un robot) à partir de données perçues (par exemple, nuage de points issu d'un LiDAR) au fil du temps, par nature bruitées et/ou incomplètes. Les filtres de Kalman sont ainsi couramment utilisés dans les modules de *SLAM*, mais également dans d'autres domaines tels que la météorologie ou la finance.

Pour établir une prédiction, le filtre de Kalman utilise les données actuelles des capteurs ainsi que la prédiction de l'état précédent, et n'utilise donc pas l'historique des données des capteurs (mais dispose de la mémoire du filtre qui les a intégrés). La prédiction est produite en deux étapes : estimation de l'état courant à partir de l'état précédent (prédiction précédente), puis correction de cette prédiction en utilisant les données perçues à l'état actuel.

De façon formelle, le calcul d'une prédiction $\dot{x}_{t,t}$ et de l'estimation de la précision de la prédiction $p_{t,t}$ (matrice de covariance de l'erreur) au temps t est effectué ainsi :

$$\begin{cases} \dot{x}_{t,t} = \dot{x}_{t,t-1} + K_t(z_t - \dot{x}_(t, t-1)) \\ p_{t,t} = (1 - K_t)p_{t,t-1} \end{cases}$$

Avec :

- $\dot{x}_{t,t}$ la prédiction de l'état du système au temps t
- $p_{t,t}$ la mesure de l'incertitude de l'état au temps t
- K_t le gain de Kalman, correspondant au ratio de l'importance des mesures face aux prédictions

En robotique, les filtres de Kalman répondent ainsi à de nombreux problèmes, allant du filtrage des données des capteurs à l'extraction des données utiles.

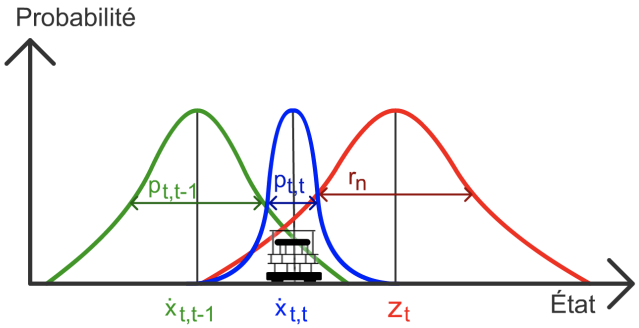


FIGURE 3.10 – Représentation des mesures et prédictions pour la position d'un robot.

5.3 Filtrage particulaire

Les filtres particulaires constituent une alternative aux filtres de Kalman, et sont parfois utilisés conjointement pour filtrer les données perçues. Ces derniers s'adressent au problème modélisé sous forme de modèle de Markov caché [14], et effectuent un échantillonnage aléatoire (méthode de Monte-Carlo [2]) de façon séquentielle.

6 Conclusion

De très nombreux capteurs peuvent être utilisés pour équiper les robots en vue d'une exploration autonome. Ces derniers seront choisis selon la taille des robots, les capacités de calculs, la batterie disponible et la nature de la mission. Les LiDARs et caméras RGB-D figurent parmi les plus couramment utilisés, y compris dans les exploitations grand public comme les robots aspirateurs. L'utilisation de différents capteurs nécessite également d'interpréter et de fusionner les données de ces derniers, pour établir une représentation unifiée et cohérente du monde entourant le robot, évitant ainsi les différentes erreurs et imprécisions des différents capteurs. Acquérir une perception correcte de l'environnement nécessite ainsi des capteurs adaptés, mais aussi un bon traitement des signaux perçus par ces derniers.

Chapitre 4

Approches de l'exploration

Sommaire

1 Introduction

De très nombreuses approches ont été proposées pour répondre au problème de l'exploration multirobots. C'est un domaine vaste du fait de la multitude de contraintes, hypothèses et objectifs possibles. Ces approches peuvent être classées sur de nombreux critères. Certaines considèrent un partage complet d'informations, incluant la totalité de la carte explorée, tandis que d'autres minimisent la quantité et la régularité des échanges. Le point de vue choisi diffère également selon les approches, abordant l'exploration comme un problème de planification, d'apprentissage ou utilisant un algorithme réactif. Ce chapitre présente quelques exemples représentatifs des différents types d'approches classiques utilisées pour l'exploration multirobots.

2 Approches fourmis

Dès le début des années 90, Dorigo propose de s'inspirer des colonies de fourmis dans le cadre des systèmes multi-agents et multirobots [80]. En effet, les comportements complexes qu'une colonie peut adopter, comme par exemple trouver une source de nourriture, puis le chemin le plus court pour y accéder, fascine tant les fourmis sont, elles, individuellement simples et ne disposent que de très peu d'informations (ces dernières étant presque aveugles, et ayant des perceptions très localisées). La surprise face aux comportements de stigmergie des insectes sociaux comme les fourmis ou les termites persiste encore aujourd'hui, malgré la compréhension des comportements collectifs. Les algorithmes fourmis permettent la collaboration de nombreux agents simples pour la réalisation d'une tâche complexe. Chaque agent adopte un comportement réactif face à des perceptions locales de l'environnement et le dépôt et la perception de phénomènes. L'environnement participe alors au phénomène, permettant le dépôt des phénomènes ainsi que leur évaporation progressive. Cette classe d'algorithme constitue aujourd'hui un domaine de recherche important ayant pu fournir des solutions à des problèmes industriels concrets [133].

En robotique autonome, un comportement réactif simple et une absence de communication directe entre les robots sont deux propriétés souhaitables. En effet, un comportement réactif simple permet une économie de ressources, de calculs et de batterie, et ainsi un coût moindre et une meilleure autonomie, et restreindre les communications permet un meilleur passage à l'échelle lorsque l'on augmente le nombre de robots. En revanche, le dépôt et la perception de phénomènes s'évaporant au fil du temps s'avèrent plus difficiles à mettre en œuvre. En 2007, Garnier et al. proposent une implémentation d'un algorithme de foragement à l'aide de phéromones simulées par des points lumineux projetés par un vidéoprojecteur [62], tandis qu'en 2011 Beaufort et al. proposent une utilisation plus précise de la lumière à l'aide d'une table lumineuse [84]. Des algorithmes comme *EVAP* [56] ont été proposés pour répondre au problème de patrouille à l'aide de phéromones. Nous pouvons également citer les travaux de Fatès, qui proposent des algorithmes permettant de réunir des agents à un point de rendez-vous en utilisant des mécanismes de réaction-diffusion simples, sans avoir recours à une autorité globale [58, 81].

2.1 Algorithme EVAP

L'algorithme *EVAP* [56] vise à répondre au problème de la patrouille de robots en environnement inconnu. Les robots peuvent percevoir les phéromones qui les entourent et en déposer, et l'environnement joue un rôle actif, permettant l'évaporation de ces phéromones au cours du temps. Une quantité importante de phéromones à un endroit signifie un passage récent d'un robot, ces dernières jouent alors un rôle répulsif, permettant aux robots de s'orienter vers les zones non visitées, ou visitées il y a le plus longtemps. Un autre algorithme proche, VAW (*Vertex Ant Walk*) [37] existe également, où les agents inscrivent une date sur les cellules visitées au lieu d'y déposer une quantité de phéromones s'évaporant, mais le principe de diriger les robots vers les zones n'ayant pas reçu de visite depuis le plus de temps reste similaire. Le comportement est alors semblable, mais les propriétés des deux algorithmes diffèrent dans le sens où l'algorithme VAW ne suppose pas de rôle actif de l'environnement, en faisant ainsi un algorithme *Brick & Mortar* et non un algorithme fourmi. Cette approche a l'avantage d'être locale en termes de décision. Une version *Brick & Mortar* de cet algorithme a également été proposée [72, 30].

2.2 Algorithme CLiNG

Une approche similaire à l'algorithme *EVAP*, *CLInG* [40], considère quant à elle l'inactivité des cellules (intervalles de temps sans passage d'aucun robot) se propageant dans l'environnement au fil du temps. Le modèle *CLInG* est alors plus complexe, nécessitant que l'environnement propage l'information, là ou *EVAP* suppose simplement une évaporation, soit une diminution progressive des quantités de phéromones. L'algorithme *CIInG* se montre cependant plus efficace dans certaines situations, notamment au début de l'exploration et lorsque le nombre de robots est réduit.

3 Approches Brick & Mortar

Les approches fourmis sont simples, élégantes et efficaces, mais supposent un rôle actif de l'environnement, permettant aux phénomènes de s'évaporer au fil du temps, qui s'avèrent souvent difficiles à réaliser concrètement. En effet, l'utilisation de « phéromones réelles »(produits chimiques s'évaporant naturellement, poussières de capteurs larguées dans l'environnement, phéromones digitales, etc.) est difficile, et l'utilisation de phéromones virtuelles nécessite des calculs constants sur l'ensemble de l'espace parcouru, ce qui est coûteux en mémoire et en calculs. Certains travaux s'appuient sur des puces déposées au sol, mais ce type d'approche n'est pas toujours idéal ou envisageable. Un ensemble d'approches étroitement liées aux algorithmes fourmis, *Brick & Mortar*, permettent d'utiliser un marquage au sol sans que ce dernier évolue au fil du temps, évitant ainsi la nécessité d'une mise à jour régulière de données.

En 2007, Ferranti et al. proposent un algorithme fonctionnant sur ce principe [59], où le robot évolue dans un monde à cases sur lesquelles il note des informations lors de son passage. C'est un algorithme réactif qui ne prend en compte que les données locales (capteurs et marquage), mais ne s'adapte pas à la distance du champ de perception du robot. En 2015, Andries et al. proposent une nouvelle version de cet algorithme tenant compte de l'ensemble du champ de perception du robot, mais ce nouvel

algorithme est alors nettement plus complexe, sensible aux asynchronismes, et alors difficilement applicable dans la réalité.

3.1 Algorithme MDFS (Multiple Depth First Search)

L'algorithme *MDFS* [60] proposé par FERRANTI, TRIGONI et LEVENE est un algorithme simple d'exploration exhaustive reposant uniquement sur un marquage au sol et la lecture, au fil de l'exploration, d'éventuelles informations laissées sur la case occupée par les robots.

Les robots partagent une grille permettant d'annoter l'environnement, et peuvent noter sur chacune des cases trois informations :

1. Son état, parmi un exemple d'état { *inexploré, exploré, visité, obstacle* }
2. L'identifiant d'un robot
3. La direction de la cellule précédemment occupée par le robot qui l'a marquée avec son identifiant

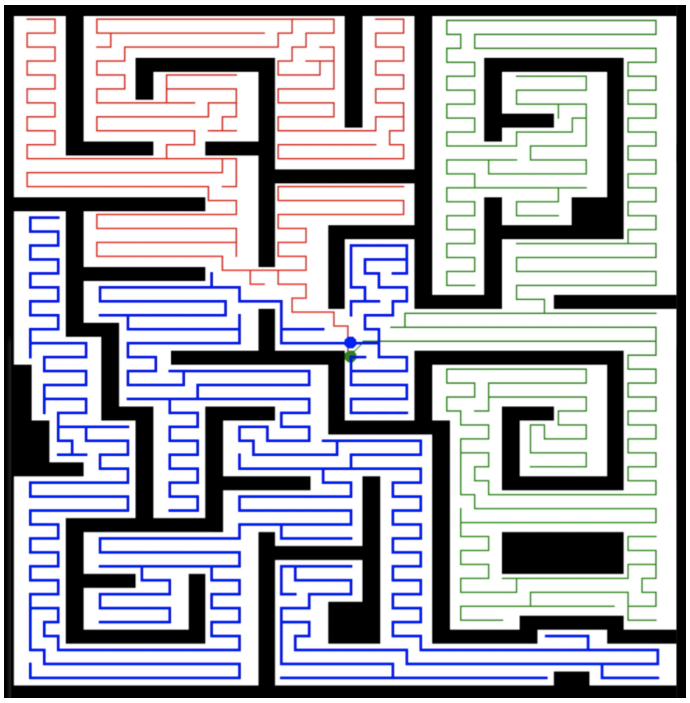


FIGURE 4.1 – Trajectoires obtenues lors d'une simulation avec 4 robots utilisant l'algorithme MDFS.

L'algorithme fonctionne de la manière suivante :

1. si la cellule actuelle est inexplorée
   - Marquer la cellule actuelle comme *explorée*
   - Annoter la cellule actuelle avec l'identifiant du robot et la direction de la cellule précédente
2. Si il y a des cellules voisines inexplorées, se déplacer vers une des cellules du voisinage inexplorées de façon aléatoire
3. Sinon
   (a) Si la cellule actuelle est marquée avec l'identifiant du robot

— Marquer la cellule actuelle comme visitée

— Revenir à la cellule précédente (grâce au marquage de direction)

(b) Sinon

— Aller sur une des cellules voisines inexplorées de façon aléatoire, en évitant la cellule précédente (sauf si c'est la seule disponible)

La version simple décrite ici fonctionne à condition que les robots ne se bloquent pas mutuellement à cause d'une boucle dans la carte explorée. Les auteurs proposent également un système de contrôle de boucle, complexifiant l'algorithme, mais permettant d'éviter ce problème. Le système de contrôle de boucle sera explicité dans la partie 3.4 sur l'algorithme *BMILRV*, ce dernier étant basé sur l'algorithme *MDFS*.

3.2 Algorithme Bob

L'algorithme Bob [108] vise à répondre au problème de couverture, en utilisant des trajectoires en boustrophédon (une trajectoire effectuant des sillons dans un sens et dans l'autre, à la manière de la plupart des aspirateurs automatiques). L'approche propose un algorithme de *backtracking* permettant aux robots de rejoindre efficacement le prochain point « utile »lorsque le chemin suivi ne permet plus d'accéder à une zone non explorée.

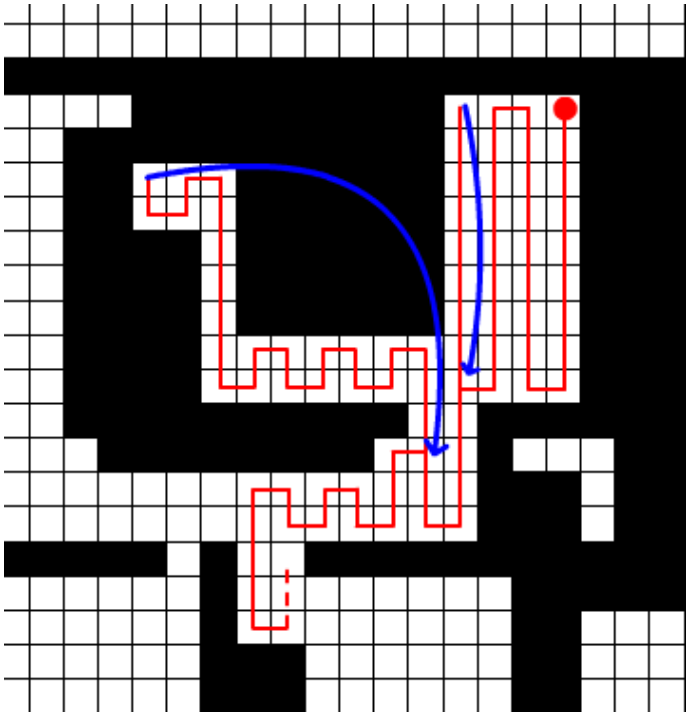


FIGURE 4.2 – Trajectoire obtenue avec l'algorithme BoB. Les traits rouges correspondent à la trajectoire du robot, et les flèches épaisses bleues aux retours arrière (backtracking).

La figure 4.2 représente une trajectoire typique de cet algorithme. Le point rouge en haut à droite représente le point de départ du robot, qui effectue alors une trajectoire en boustrophédon jusqu'à se retrouver bloqué en haut à gauche de la salle dans laquelle il se trouve. Il effectue alors un *backtracking* (flèche bleue arrondie) pour revenir sur ses pas jusqu'au dernier point ou une autre sortie était possible.

Les trajectoires obtenues par l'algorithme BoB ressemblent à celles obtenues avec *MDFS* (boustrophédon), mais avec un backtracking plus avisé, réduisant la distance parcourue par les robots.

3.3 Algorithme LRTA* (Learning Real Time A*)

L'algorithme *Learning Real Time A** (*LRTA**) [16], proposé par KORF en 1987, et repris et adapté de nombreuses fois [22, 47], est un algorithme de *path-finding* (recherche de chemin). Le principe est de mettre à jour l'heuristique pour la position actuelle de l'agent, puis se déplacer vers le noeud voisin ayant le coût (déplacement et heuristique sur le point d'arrivée) minimal, de façon itérative jusqu'à trouver le noeud de destination. Dans cette approche, lorsqu'un agent se retrouve dans un voisinage trop éloigné de son objectif (d'après son heuristique), le marquage le fera alors revenir en arrière (sa position précédente étant alors le meilleur choix local). Le marquage déposé (correspondant aux estimations locales des noeud), permet ainsi implicitement aux robots de revenir sur leurs pas lorsque c'est le choix le plus intéressant à effectuer, ce qui peut être un comportement souhaitable, notamment dans le cadre de l'exploration multi-robots.

3.4 Algorithme BMILRV (Brick and Mortar Improved)

En 2015, ANDRIES et CHARPILLET proposent l'algorithme *BMILRV* (Brick & Motar Improved, Long Range Vision) [102] avec comme objectif de permettre à un algorithme type *Brick & Mortar* de fonctionner en tenant compte de l'ensemble du champ de perception des robots, et non plus une seule case de leur grille d'occupation.

De façon à ne traiter que les informations locales (à l'intérieur du champ de perception des robots) marquées sur la grille tout en garantissant une exploration exhaustive, le regroupement des robots à la fin de l'exploration et l'évitement des blocages, de nombreux états et conditions sont ajoutés à l'algorithme. La figure 4.3 montre les trajectoires obtenues sur un labyrinthe avec 4 robots.

Fonctionnement de l'algorithme

L'algorithme *BMILRV*[102] utilise de nombreux marquages sur la grille d'occupation : un état parmi un ensemble d'états { *inexploré, exploré, fermé, obstacle, rendez-vous* }, une valeur indiquant si un robot contrôle une cellule donnée, la direction prise lors du dernier passage sur une cellule, et une variable booléenne indiquant si un robot donné est déjà passé sur une cellule. Les robots peuvent également être dans différents modes parmi { *détection de boucle, contrôle de boucle, fermeture de boucle, nettoyage, pause, arrêt* }.

De manière informelle, l'algorithme *BMILRV* consiste à maintenir une liste de cellules ouvertes permettant d'accéder à toutes les zones inexplorées de la carte, tout en fermant progressivement toutes les cellules qui peuvent l'être (qui ne bloqueront pas le passage vers une zone inexplorée), évitant ainsi de repasser dans les zones déjà visitées.

La figure 4.4 montre l'état d'une grille au cours d'une simulation de l'algorithme *BMILRV*. Les cellules blanches correspondent aux cellules inexplorées, les grises aux cellules fermées.

Lorsque des boucles sont présentes dans le chemin constitué par les cellules explorées, les robots peuvent les fermer en plusieurs étapes permettant d'éviter de bloquer

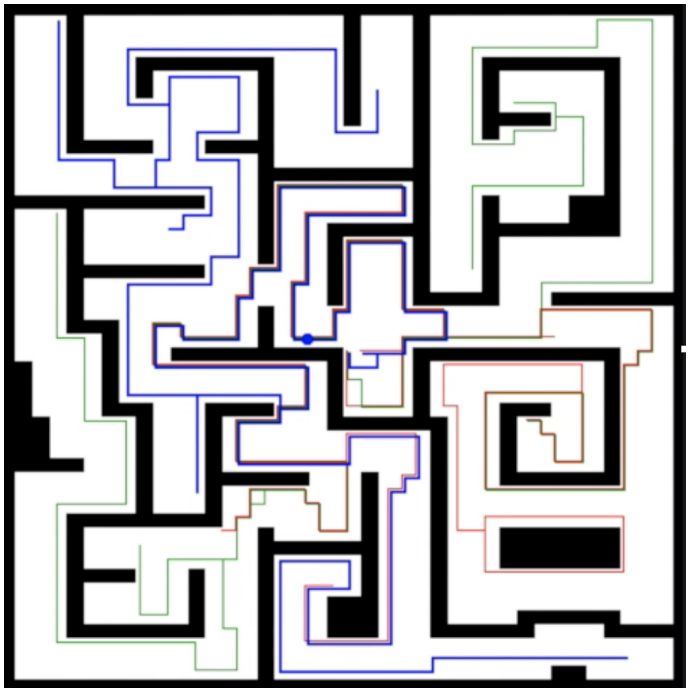


FIGURE 4.3 – Trajectoires obtenues lors d'une simulation avec 4 robots utilisant l'algorithme BMILRV.

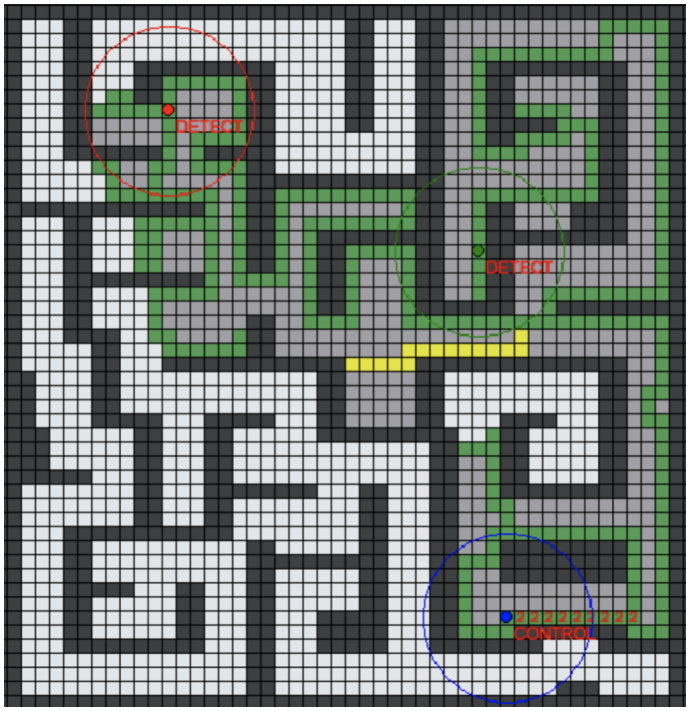


FIGURE 4.4 – État de la grille au cours de l'exploration avec l'algorithme BMILRV.

un robot. Le robot détectant la boucle (entrant dans une cellule explorée sur laquelle il est déjà passé en venant de la même direction), va alors passer en mode « *contrôle de boucle* » et parcourir cette boucle pour la marquer comme contrôlée avec son identifiant, vérifiant ainsi qu'il est le seul à chercher à la fermer à ce moment. S'il parvient à marquer la totalité de la boucle, il effectue alors un second passage en mode « *fermeture* » ou il va fermer les cellules de la boucle jusqu'à la première intersection, et enfin parcourir la boucle une troisième fois en sens inverse en mode « *nettoyage* » pour enlever les traces

indiquant qu'il contrôle la boucle. S'il ne parvient pas à contrôler la boucle, alors il passe directement en mode « *nettoyage* » si le robot qui la contrôle a une priorité supérieure, ou en mode « *pause* » si il est prioritaire, en attendant que l'autre robot ait nettoyé ses traces sur la boucle.

Inconvénients de la méthode

- **Fermeture de boucles** : au fil de l'exploration, les robots maintiennent une liste de cellules ouvertes (en vert sur la figure 4.4) sur lesquelles ils s'autorisent à repasser, qui permettent d'accéder à toutes les zones inexplorées. Lorsqu'un robot souhaite fermer un morceau du chemin composé par ces cellules ouvertes, il doit, en présence de boucles, s'assurer qu'aucun autre robot ne se trouve sur ladite boucle pour éviter de le bloquer sur un morceau de chemin qui deviendrait alors isolé. Pour se faire, un robot ayant détecté une boucle va la parcourir une première fois pour la marquer avec son identifiant (s'il tombe sur un autre identifiant en parcourant la boucle, alors le robot avec l'identifiant le plus faible reculera en effaçant son identifiant tandis que l'autre attendra), une seconde fois pour fermer les cellules et une troisième pour effacer son identifiant. Chaque boucle dans la carte nécessite alors au moins trois parcours d'un robot, et peut en provoquer beaucoup plus si plusieurs robots tentent de fermer une même boucle, ou partie de boucle, simultanément. Ainsi, plus il y a de boucles (généralement dues à des obstacles isolés), plus la durée d'exploration sera longue.
- **Asynchronisme et erreurs de localisation** : Bien que les robots marquent des cellules dans l'ensemble de leur champ de vision (de façon à fermer les cellules qui n'ont pas besoin d'être visitées sans avoir à se déplacer sur chacune d'elles), leur déplacement se font d'une case à la fois. Si un robot se trouve isolé sur une cellule ouverte entourée de cellules fermées, par exemple suite à une erreur de localisation ou une différence de vitesse entre deux robots, alors il sera bloqué et l'exploration exhaustive et le regroupement des robots ne fonctionneront plus. Cet algorithme est donc difficilement applicable dans la réalité.

4 Approches par frontières

Le problème de l'exploration peut également être envisagé comme un problème de planification. Il s'agit alors d'assigner un objectif à chacun des robots, en tenant généralement compte de la totalité des informations obtenues par tous les robots.

En 1997, YAMAUCHI propose l'exploration par frontières, où l'on détermine les limites entre les zones explorées et inexplorées de la carte, et dirige chaque robot vers l'une d'elles. L'algorithme est adapté à l'utilisation de plusieurs robots l'année qui suit [29], avant d'être repris par de nombreux autres auteurs.
Le principe de l'algorithme est le suivant :

1. Les robots partagent une carte globale ou les cellules sont dans un état parmi { *exploré, inexploré, obstacle* }, initialisées à l'état *inexploré*.
2. À chaque itération, les robots marquent l'ensemble des cellules perceptibles depuis leur position comme *explorée* ou *obstacle* selon leur perception.
3. Les cellules qui séparent les zones explorées et inexplorées sont alors identifiées comme frontières.

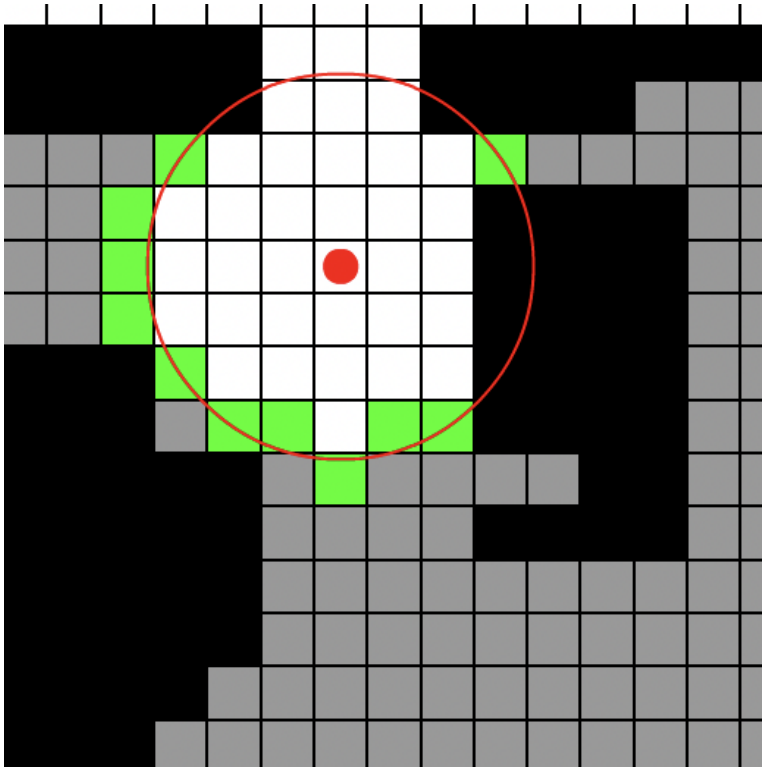


FIGURE 4.5 – Exécution de l'algorithme d'exploration par frontières avec une grille d'occupation sur un robot. Le point rouge représente le robot, et le cercle rouge qui l'entoure son champ de perception. Les cellules noires correspondent aux obstacles, les blanches aux cellules explorées, les grises sont inexplorées, et les frontières sont représentées en vert.

4. Chaque robot est alors assigné à une frontière, vers laquelle il va se déplacer.

Cette approche permet de choisir plusieurs paramètres qui auront un effet significatif sur le coût et les performances de l'exploration, dont notamment :

- **l'algorithme d'assignation** qui détermine à quelle frontière chaque robot est assigné. Le choix devant généralement aboutir à un consensus, de nombreux algorithmes d'assignation sont coûteux en calculs et en communication.
- **la position choisie comme objectif pour une frontière donnée**, une frontière pouvant être un ensemble de cellules connexes. Il est par exemple possible de choisir la cellule « centrale »de la frontière, celle la plus proche du robot assigné, où encore une cellule à une distance d'un obstacle légèrement inférieure au champ de perception du robot.
- **la condition de réassignation**, qui détermine quand on doit réexécuter l'algorithme d'assignation. Ce paramètre peut être défini par intervalle de distance, de temps, ou selon une condition précise comme le fait d'atteindre la position donnée comme objectif. Une réassignation trop fréquente peut parfois provoquer des oscillations, un robot pouvant choisir successivement deux frontières dans des directions opposées.

Cette approche permet donc d'utiliser différents algorithmes pour assigner une frontière à un robot donné, ainsi que différentes conditions de réassignation des frontières. Des algorithmes d'assignation très différents ont alors pu être proposés, des plus simples comme l'assignation de chaque frontière au robot le plus proche, aux plus complexes comme le système d'enchères de BERHAULT et al. dans [39].

4.1 Assignation de frontières

De nombreux algorithmes d'assignation ont été proposés au fil des années. En 2012, BAUTIN et al. proposent l'algorithme *MinPos* [89], consistant à compter pour chaque couple robot-frontière le nombre d'autres robots plus proches de cette frontière, tandis que FAIGL et al. proposent une solution basée sur le problème du voyageur de commerce. Plusieurs études comparent alors les différentes stratégies possibles, comme FAIGL et al. dans [105] où 5 stratégies d'assignation sont comparées sur la vitesse de l'exploration, ou JAIN et al. dans [122] qui comparent différentes stratégies sur différents critères. Différentes stratégies sont alors développées en vue de répondre à des contraintes et objectifs spécifiques, comme la répartition de l'espace entre les robots dans [70, 126] où la consommation d'énergie [129, 136].
Les approches proposées incluent :

- **Greedy** : Algorithme d'assignation utilisé dans l'article original de YAMAUCHI [31]. Chaque robot est assigné à la frontière la plus proche de sa position. Certains articles utilisent une version légèrement modifiée de cet algorithme, ou chaque objectif non assigné est assigné au robot le plus proche, mais avec un ordre aléatoire pour les robots.
- **Assignation itérative** : on considère chaque tuple $<$ *robot*, *frontière* $>$ et on calcule pour chacun la distance entre le robot et la frontière du tuple. La liste des tuples est alors triée par distance croissante, et chaque frontière non assignée est alors attribuée au robot le plus proche n'ayant pas encore de frontière assignée.
- **Assignation hongroise** (*MUNKRES*) : on considère une matrice de coûts de taille $n * m$ pour m robots et n objectifs. Si $m > n$, l'algorithme d'assignation itérative est utilisé, tandis que si $m < n$ la matrice de coûts est agrandie avec des robots virtuels ajoutés, avec un coût de distance très élevé pour les objectifs.
- **Voyageur de commerce (MTSP)** [90] (*multiple travelling salesman problem*, problème du voyageur de commerce) [87, 98] ou l'algorithme *MTSP* est adapté à l'assignation du robot. Les frontières sont d'abord regroupées en clusters à l'aide de l'algorithme *K-Means*, et l'objectif assigné à chaque robot est ensuite défini en fonction du coût de distance de l'algorithme TSP.
- **MinPos** [89] : pour chaque tuple $<$ *robot*, *frontière* $>$, on compte le nombre d'autres robots plus proches de la frontière donnée que le robot du tuple, et on trie ensuite les tuples par valeur croissante.

4.2 Condition de réassignation

Différentes conditions peuvent être choisies pour réassigner les frontières au fil de l'exploration. Parmi elles, les conditions les plus couramment utilisées incluent :

- *réassignation immédiate* : les frontières sont réassignées à chaque changement de cellule des robots. Cette condition est coûteuse en calculs, provoquant des exécutions de l'algorithme d'affectation très fréquentes.
- *Intervalle de distance* : l'algorithme d'assignation est exécuté à chaque fois que les robots ont parcouru un intervalle de distance donné, par exemple leur rayon de perception.
- *Intervalle de temps* : l'algorithme d'assignation est exécuté à intervalles réguliers.
- *Par objectif* : l'algorithme est exécuté lorsqu'un robot atteint l'objectif (la position) qui lui avait été donné.

5 POMDP

Pour répartir les différents robots et définir le prochain objectif, des POMDP [3] décentralisés (DEC-POMDP) peuvent également être utilisés. MATIGNON et al., qui ont également participé au défi CAROTTE, ont proposé une approche locale minimisant les communications entre robots dans [91].

Dans leur approche, une fonction de valeur distribuées (*DVF*) est utilisée par chaque agent, en utilisant les données perçues localement et une communication limitée entre agents. Chaque agent calcule localement une stratégie visant à minimiser les interactions entre robots et maximiser la couverture spatiale de l'ensemble des robots.

Ce type d'approche n'utilise pas réellement de *Dec-POMDP*, mais une heuristique locale à chaque robot et plus simple à calculer.

5.1 Dec-POMDP

Un Dec-POMDP est un POMPD décentralisé (processus décisionnel de Markov partiellement observable décentralisé). Il est composé d'un tuple $(S, \{A_i\}, T, R, \{\Omega_i\}, O, \gamma)$ correspondant à :

- S un ensemble d'états du système
- $\{A_i\}$ l'ensemble des actions réalisables par l'agent i. L'ensemble A correspond à l'ensemble des assignations d'actions possibles pour tous les agents i
- $T(s, a, s') = P(s'|s, a)$ la matrice des des probabilités de transition conditionnelles entre les états.
- $R : S \times A \to \mathbb{R}$ la fonction de récompense
- Ω_i l'ensemble des observations pour l'agent i
- $O(s', a, o) = P(o|s', a)$ l'ensemble des probabilités d'observation conditionnelles
- $\gamma \in [0, 1]$ le facteur d'actualisation.

À chaque étape :

1. Chaque agent i choisit une action $a_i \in A_i$
2. L'état du système est mis à jour suivant les probabilités de la matrice de transition T, selon l'état actuel et l'action a (ensemble des actions a_i de tous les agents i)
3. L'observation est mise à jour selon l'état actuel s et l'action commune a suivant les probabilités conditionnelles O.

6 Couverture de graphes

Les algorithmes visant à répondre au problème de couverture hors ligne (sans connaissance de l'environnement à priori) permettent également de répondre au problème de l'exploration, la différence entre les deux n'étant qu'une question d'intention (explorer dans un cas, produire un chemin couvrant l'environnement dans l'autre). Dans la littérature, ces deux problèmes seront souvent abordés de manière différente par des communautés distinctes : l'exploration comme un problème de robotique autonome, et la couverture hors-ligne comme un problème plus théorique.

De nombreux algorithmes à la fois simples et locaux ont été proposés pour répondre au problème de la couverture [52, 97, 108]. VIET et al. proposent *BA** puis *BoB* [97, 108],

deux algorithmes de couverture hors ligne qui répondent au problème de l'exploration, utilisant une représentation du monde sous forme de graphe ou de grille.

JENSEN et al. ont proposé une comparaison de différents algorithmes locaux avec des groupes de robots de différentes tailles dans des cas où les communications sont restreintes dans [135]. Notons que ces approches sont comparables, en terme de fonctionnement, aux approches dites « Brick & Mortar ».

6.1 Arbres couvrants

Les arbres couvrants (présentés chapitre 2. Représentation du monde, partie 4.5. Arbres couvrants) peuvent être utilisés de façon à explorer ou couvrir une carte en robotique autonome.

En 2001, GABRIELY et RIMON proposent l'approche *STC* (Spanning Tree Coverage) pour explorer un environnement avec un robot mobile. Ils proposent trois versions de l'algorithme, une hors-ligne où le robot connaît son environnement à-priori, une en-ligne, et une version fourmi où le robot utilise un marquage au sol.

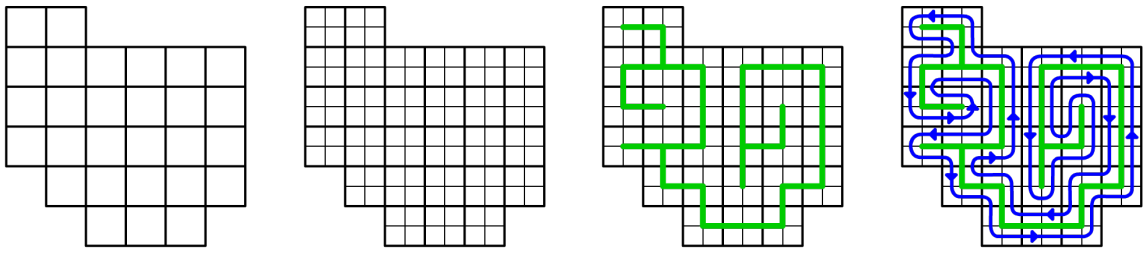


FIGURE 4.6 – Étapes de l'algorithme *STC*.

La figure 4.6 représente les différentes étapes de l'algorithme *STC*. À partir d'une grille de cellules non-occupées (a), on construit un graphe où chaque cellule est un noeud, reliée à ses quatre voisines (si elles existent). On construit alors un arbre couvrant de ce graphe (en vert dans la figure 4.6. c). Enfin, chaque cellule est subdivisée en quatre sous-cellules (4.6. b), et on réalise alors un parcours de dans le sens anti-horaire de l'arbre (en bleu figure 4.6 d).

Depuis, de nombreuses approches ont été proposés en utilisant les arbres couvrants pour l'exploration multirobots [92, 69, 50].

7 Apprentissage et DeepLearning

De nombreux travaux plus récents utilisent des algorithmes d'apprentissage et de DeepLearning pour répondre à des problèmes d'exploration ou de réalisation de tâches par un groupe de robots dans un environnement inconnu.

De nombreux sous-problèmes se posent pour adapter efficacement les algorithmes d'apprentissage par renforcement aux problèmes concrets, notamment en termes de ressources et capacité de calculs nécessaires. Différentes approches visent ainsi à accélérer l'apprentissage, tout en essayant d'apprendre différentes capacités telles que l'exploration autonome.

Les environnements 3D des jeux vidéos fournissent des conditions de simulation et d'apprentissage idéales, et de nombreuses approches les utilisent pour développer des

agents capables de réaliser différentes tâches apprises avec des algorithmes d'apprentissage par renforcement [114, 112].

En 2019, BEECHING et al. proposent une méthode pour tester des algorithmes d'apprentissages sur un environnement simplifié (cartes de jeu vidéo), permettant une exécution, un apprentissage et des mesures de performances rapides avant des adaptations sur des cas plus concrets (photoréalistes) [131].

8 Topologie de l'environnement

Que l'objectif soit d'explorer exhaustivement un environnement ou de se déplacer dans un environnement partiellement connu, le type de lieu à explorer peut donner des informations supplémentaires sur la meilleure façon de s'y prendre. Par exemple, dans un grand bâtiment, il est habituel de trouver de grands couloirs d'accès permettant de parcourir les grands axes. La structure du bâtiment est également souvent composée d'éléments similaires répétés à différents endroits, comme les toilettes, des salles de classe dans une école, etc. En tenant compte de ces informations, il est alors possible de faciliter l'exploration et la navigation dans un lieu dont les caractéristiques probables peuvent alors être prédites.

9 Problèmes spécifiques

De nombreuses travaux cherchent à apporter des solutions répondant à des contraintes concrètes spécifiques, et utilisent des approches différentes selon les particularités à considérer. Par exemple, dans le cas d'applications militaires des problèmes d'exploration et de calcul de trajectoire, la programmation par contrainte à souvent été utilisée [33, 73, 127]. D'autres sous-domaines, comme l'exploration sous-marine, sont également confrontés à des contraintes spéficiques (autres types de capteurs et actionneurs, système de localisation, etc) et sont également souvant abordés sous des angles différents [136].

Le partage d'informations entre les robots figure également parmi les problèmes spécifiques de l'exploration, et une grande partie de l'état de l'art suppose ces dernières efficaces et illimitées. BENAVIDES OLIVERA propose une stratégie d'exploration multiobjectif autoadaptative (AMMO) [132], visant à tenir compte de la performance de l'exploration et du niveau de connectivité du réseau, en intégrant l'avis d'un opérateur humain qui pondère les critères au cours de la mission de façon à en optimiser l'efficacité. Grâce à cette approche, il est possible de permettre à des robots d'explorer des objectifs proches de façon à maintenir leurs communication et ainsi réduire les déconnexions sans diminuer significativement les performances des l'exploration.

10 Bilan des différentes approches

Un nombre important d'approches ont ainsi été proposées au fil des années, visant à aborder l'exploration sur différents aspects. Chaque approche apporte son lot de forces et faiblesses, et nombre d'entre elles peuvent être associées pour s'adapter au mieux aux conditions du problème. Nous allons ici résumer les différentes approches et moyens de les associer et différencier.

10.1 Approches locales et globales

L'utilisation d'informations uniquement locales ou non est une façon de discriminer les approches de l'exploration. Les algorithmes de type *fourmis* et *Brick & Mortar* regroupent la majorité des approches locales de l'exploration. Ces approches sont souvent moins performantes que les approches globales, mais nécessitent généralement beaucoup moins de calculs, et permettent parfois de partager beaucoup moins d'informations entre les robots (comme dans le cas de l'approche « frontières locales »où seul l'historique des positions est partagé).

Les approches locales sont ainsi particulièrement adaptées lorsque le nombre de robots est grand ; permettant ainsi de limiter les communications et calculs (des robots moins puissants pouvant être moins coûteux, et ainsi permettre un nombre plus élevé de robots). Certaines approches, comme les frontières locales, ont également l'avantage d'avoir une complexité indépendante du nombre de robot et de la taille de l'espace à explorer, permettant ainsi un très grand nombre de robots.

Approches locales	**Approches globales**	**Hybrides**
Ant Colony Optimisation [80]	Frontières [27]	Roy et Dudek [36]
Evap [56]	MSP [82]	
MDFS [60]	Bidder agent [53]	
BoB [108]	Rooker et Birk [63]	

Table 4.1 – Classification des approches locales et globales

10.2 Fourmis et Brick & Mortar

Les approches par stigmergie peuvent être séparées en deux groupes : fourmis et Brick & Mortar. Les approches fourmis ont l'inconvénient de supposer une évolution des traces déposées au fil du temps, et donc un rôle actif de l'environnement. Les traces déposées étant généralement simulées, cette évolution nécessite alors des calculs supplémentaires.

Notons cependant que l'évolution des traces étant généralement uniforme, cette opération peut souvent être réalisée de façon peu coûteuse en parallélisant les calculs (avec une carte graphique par exemple).

Fourmis	**Brick & Mortar**
Evap [56]	BMILRV [60]
ACO [80]	Frontières locales [134]

Table 4.2 – Classification des approches fourmis et Brick & Mortar

10.3 Partage d'informations

La façon de partager les informations entre les robots regroupe différents critères de discrimination des différentes approches. Les algorithmes de l'état de l'art son plus ou moins sensibles aux asynchronismes et pertes de communication, nécessitent de partager différents types d'informations (et quantités de données), avec plus ou moins de régularité.

La quantité et la régularité du partage d'informations différent grandement selon les approches. Par exemple, les approches par frontières nécessitent de partager la totalité de la carte et les positions des différents robots pour calculer une assignation pour chaque robot, tandis que l'approche par frontières locales ne nécessite que le partage de l'historique des positions, et ne nécessite pas de connaissance sur la carte explorée.

Certaines approches sont également peu robustes aux problèmes de communication et d'asynchronismes, comme l'algorithme *BMILRV*, où des robots peuvent se retrouver bloquer en cas de perte temporaire de communications.

10.4 Représentation du monde

Les approches de l'état de l'art n'utilisent pas tous les mêmes représentations du monde. Selon les approches, la représentation utilisée est également plus ou moins contrainte. Par exemple, l'algorithme *BMILRV* nécessite de représenter la carte sous forme d'une grille, avec différentes informations marquées sur chaque case de la grille. À l'inverse, les approches par frontières ou frontières locales peuvent utiliser différentes représentation du monde, à partir du moment où il est possible de calculer les frontières entre les zones explorées et inexplorées.

Approche	Informations partagées	Régularité
Frontières [27]	Carte + positions	À chaque prise de décision
BMILRV [60]	Cartes (3 données/case)	Déplacement de une case
Frontières locales [134]	Historique des positions	Prise de décision ou moins souvent

TABLE 4.3 – Informations partagées par différentes approches

10.5 Objectifs

Les objectifs visés par les différentes approches de l'état de l'art ne sont pas toujours les mêmes. Si une performance optimale est parfois l'objectif, permettre à un groupe de robots d'explorer de façon autonome avec un ensemble de contraintes plus stricte est également courant. De plus, différents critères de mesures permettent de discriminer les algorithmes, et certaines approches visent particulièrement à minimiser les communications, les calculs ou la mémoire utilisée. Le tableau 4.4 résume les objectifs principaux de plusieurs algorithmes de l'état de l'art.

Approche	Objectif(s) visé(s)
Frontières [27]	Rapidité et exhaustivité de l'exploration
BMILRV [60]	Preuve de concept d'exploration par stigmergie
Frontières locales [134]	Communication et calculs réduits, exploration exhaustive
Evap [56]	Patrouille par stigmérgie

TABLE 4.4 – Objectifs visé dans les différentes approches

10.6 Tableau récapitulatif

Le tableau ci-dessous regroupe différentes approches développées dans ce chapitre, avec pour chacune d'elles le type d'approche, les informations partagées et la représen-

tation du monde utilisée.

Algorithme	Type	Communications	Représentation
ACO [80]	Fourmis	Stigmergie	Graphe
Evap [56]	Fourmis	Stigmergie	Grille
MDFS [60]	Brick&Mortar	Stigmergie	Grille
BoB [108]	Brick&Mortar	Stigmergie	Grille
Frontières [27]	Planification	Carte + positions	Grille
Frontières locales [134]	Brick&Mortar	Historique positions	Grille, continu
MSP [82]	Brick&Mortar	Graphe	Graphe
Bidder agent [53]			
ROOKER et BIRK [63]		Centralisée ou agent-agent	
ROY et DUDEK [36]			

TABLE 4.5 – Tableau récapitulatif de différentes approches de l'état de l'art

11 Conclusion

De très nombreuses approches ont été proposées pour répondre au problème de l'exploration autonome. Ces dernières peuvent être discriminées selon de très nombreux critères, notamment en fonction des objectifs, de la représentation du monde adoptée, ou encore des informations partagées entre les robots. De nombreux algorithmes ont également été adaptés plusieurs fois pour modifier leurs contraintes ou leurs performances. Chaque approche possède ses propres avantages et inconvénients, permettant différents niveaux de performances selon les contraintes associées. Lorsqu'un nouvel algorithme est proposé, il convient généralement de le mesurer à une sélection d'autres algorithmes qui répondent aux même contraintes, et/ou qui utilisent un fonctionnement similaire.

Chapitre 5

Intégration de données sémantiques

Sommaire

1 Introduction

L'utilisation de données sémantiques permet une meilleure interprétation de l'environnement, en considérant des données de plus haut niveau que les données métriques, et en apportant une meilleure compréhension de l'environnement du robot. L'identification des salles, portes et couloirs permet ainsi de représenter le monde de façon plus simple et efficace, en évitant d'enregistrer ou partager des informations qui n'aideront pas les robots à se repérer et se déplacer jusqu'à leurs objectifs. Comme nous l'avons vu, se repérer dans une carte non métrique peut également palier à différents problèmes de dérive des capteurs. Ces informations peuvent également permettre de mieux guider et répartir les robots, en tenant compte des règles usuelles de l'architecture humaine pour mieux prédire la structure de l'environnement.

D'autres informations peuvent également être considérées, comme le type d'objets présents dans une pièce, pouvant ainsi indiquer au robot ce qu'il est susceptible de trouver dans les environs.

Ce chapitre s'intéresse à différentes approches sémantiques de l'état de l'art. Le chapitre 8 décrit quant à lui les propositions de cette thèse pour l'exploration multirobots intégrant des données sémantiques.

2 Exploration ciblée

2.1 Exploration sur différentes échelles

De nombreux planificateurs intègrent différentes cartes, dites locales et globales, de façon à représenter les données avec différents niveaux de détail, permettant ainsi de

s'adapter au mieux aux contraintes locales précises et aux aléas (objets en mouvement, obstacle inattendu), tout en planifiant à long terme les trajectoires. Dans la même idée, KORPAN et EPSTEIN proposent un modèle à deux niveau : une carte privilégiant les grands axes et permettant de prévoir les trajectoires optimales dans un grand bâtiment, et des cartes locales plus précises [143].

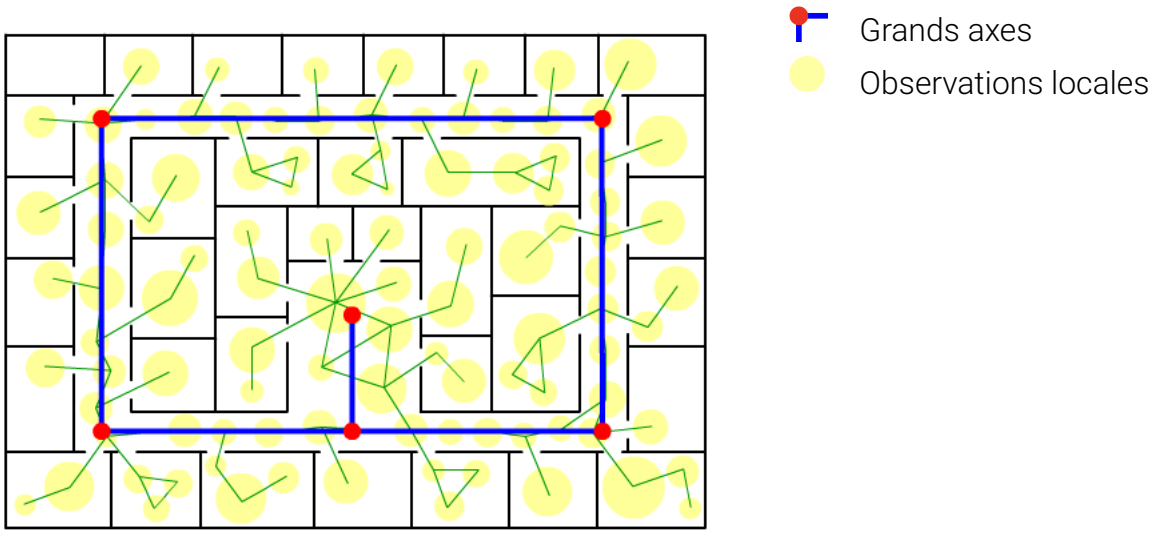


FIGURE 5.1 – Carte des grands axes (en bleu) et des observations locales (en jaune)

Les grands axes (ou couloirs) sont identifiés par rapport à leur ratio longueur/largeur. L'efficacité des trajectoires trouvées grâce à cette méthode est d'autant plus grande que les bâtiments de taille importante sont généralement structurés de façon à ce que de grand couloirs permettent de naviguer efficacement dans le bâtiment.

Les deux cartes sont ainsi utilisées par un planificateur global et un planificateur local de la façon suivante :

- Le planificateur global utilise une représentation des grands axes du bâtiment. Lorsqu'un objectif est à une distance importante du robot, il est probable que le chemin d'accès à l'objectif passe majoritairement par de grands couloirs, et non un enchaînement de petites pièces. Ces grands axes sont alors identifiés (comme couloirs) avec des informations de plus haut niveau (longueur, topologie entre les couloirs), et permettent de guider le robot vers une zone à proximité de son objectif.
- Le planificateur local utilise lui des cartes métriques, représentant les obstacles identifiés à proximité du robot, et permettant ainsi l'évitement d'obstacles et le guidage du robot sur une petite zone. Les cartes métriques locales sont localisées dans un repère global utilisé par le planificateur global.

Cette approche utilise ainsi des données sémantiques et métriques de façon à mieux guider les robots dans des espaces grands et complexes, en utilisant des données de plus haut niveau pour la planification à long terme, et des données métriques pour la navigation à court terme. Cette exploitation des données sémantiques, en plus de réduire la quantité de données à considérer pour la planification à long terme, tiens compte de l'architecture usuelle des bâtiments pour mieux prédire les chemins pouvant mener un robot à un objectif donné.

2.2 Habitat

Habitat Sim [1] est un simulateur 3D dédié à la recherche en intelligence artificielle embarquée développé par Facebook [146, 137]. Une compétition est organisée chaque année, *Habitat Challenge* [128] constitué de deux tâches : *PointNav* dans lequel un robot est placé aléatoirement dans un environnement inconnu, et doit naviguer vers des coordonnées spécifiques relatives à son point de départ, en utilisant uniquement une caméra RGB-D. Pour la seconde tâche, *ObjectNav*, le robot doit naviguer jusqu'à un type d'objet demandé (par exemple une chaise), avec en plus de la caméra RGB-D une boussole et un GPS indiquant la position et la rotation relatives au point de départ du robot.

FIGURE 5.2 – Simulateur 3D développé pendant la thèse sur une carte utilisée avec Habitat Sim, réalisée à partir d'un lieu réel scanné.

Le simulateur 3D développé pendant cette thèse (décrit dans le chapitre 6 Méthodologies) a été conçu de manière a être compatible avec les jeux de données couramment utilisés avec Habitat Sim (Habitat-Matterport 3D Research Dataset, Gibson, 3DSceneGraph [2]). La figure 5.2 montre un exemple d'exécution sur une de ces cartes. Des environnements modélisés à la main (plus légers que les scans réalisés à partir de lieux réels) ont cependant été préférés lors de l'utilisation seule du LiDAR pour des raisons de performances et de poids de fichier.

2.3 Recherche d'objectif spécifique

CHAPLOT et al. ont proposé une méthode pour chercher, dans un environnement inconnu, un objet d'une catégorie donnée [139]. Dans cette approche, le robot identifie les différents objets présents dans son environnement à l'aide d'une caméra RGB-D. Il associe alors chacun d'eux à différentes catégories selon les salles dans lesquels ils

1. https ://github.com/facebookresearch/habitat-sim
2. Datasets utilisés avec Habitat Sim
https ://github.com/facebookresearch/habitat-sim/blob/main/DATASETS.md

sont susceptibles de se trouver, de façon à estimer la probabilité qu'un objet à trouver (objectif) se situe aux alentours des objets identifiés.

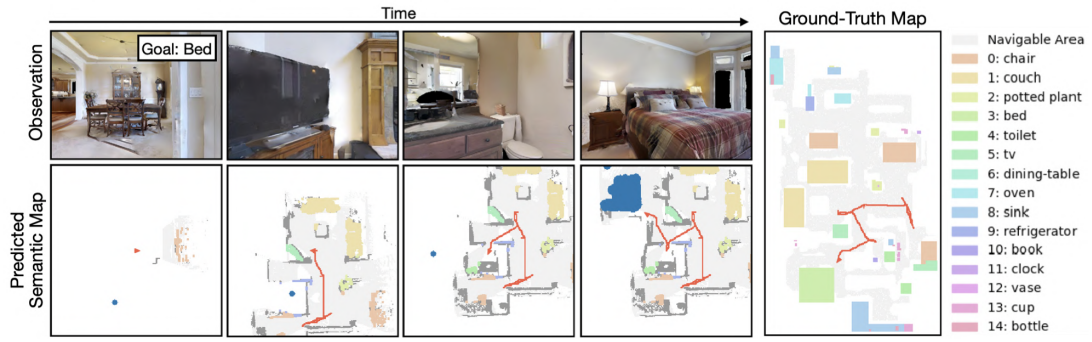


FIGURE 5.3 – Figure reprise de l'article « *Object Goal Navigation using Goal-Oriented Semantic Exploration* »[139]
représentant un exemple de trajectoire de leur model SemExp dans une scène du DataSet Gibson.

L'identification du type d'objets présents dans une pièce permet ainsi au robot de mieux comprendre son environnement, et ainsi prédire plus efficacement la probabilité de trouver son objectif dans un lieu donné, permettant alors de trouver un objet plus rapidement.

3 Conclusion

L'exploitation de données sémantiques peut ainsi apporter de nombreux avantages en robotique autonome. Les cartes sémantiques permettent un plus haut niveau d'abstraction de l'environnement, et donc une économie de poids tout en facilitant leur exploitation. La prise en compte des données ajoutées et inférées permet également de mieux exploiter les informations perçues par les robots, en attribuant un sens aux données exploitées. Les possibilités d'interprétation et d'exploitation des données sémantiques sont nombreuses selon le contexte et les objectifs visés, mais une bonne utilisation de ces dernières donne un avantage certain.

Deuxième partie

Propositions

Chapitre 6

Méthodologie

Sommaire

1 Introduction

De façon à pouvoir tester différents algorithmes d'exploration, comparer les algorithmes proposés à ceux de l'état de l'art ou encore mieux comprendre le comportement de différents modèles, il est nécessaire de simuler et/ou tester différentes approches sur robots réels.

Au cours de ma thèse, trois systèmes différents ont été utilisés. Dès les premiers jours de ma thèse, j'ai développé un simulateur très simple en Python pour pouvoir tester les algorithmes des articles que je lisais, de façon à mieux comprendre le fonctionnement de ces derniers. Ce simulateur a également permis de tester les premiers algorithmes d'exploration que j'ai proposés. Ce premier simulateur fonctionne sur de simples grilles constituant les cartes à explorer, et permet de tester rapidement différents algorithmes d'exploration. Il est disponible sur le Gitlab de l'Inria.

Après avoir proposé une nouvelle approche pour l'exploration (frontières locales), nous avons souhaité implémenter cette dernière sur des robots réels, et l'approche proposée a été implémentée sur ROS dans cet objectif. L'implémentation sur ROS permet une exécution sur robot réél ou en simulation avec Rviz et Stage. Olivier Rochel, ingénieur à l'Inria, m'a aidé à configurer et utiliser les Turtlebots pour ces expérimentations.

Avec les confinements successifs et les restrictions d'accès au laboratoire qui ont fait suite à l'épidémie de *COVID-19*, j'ai dû revenir à l'utilisation de simulations pour permettre de travailler à distance. De façon à se rapprocher des conditions réelles, j'ai alors développé un nouveau simulateur, cette fois en 3 dimensions dans un monde continu. Ce dernier simulateur n'utilise plus de grilles, mais de véritables cartes 3D, réalisées manuellement avec un logiciel de modélisation 3D, où enregistrées à l'aide de LIDARs et

caméras RGB-D sur des robots réels. De façon à rester compatible avec les deux précédents simulateurs développés, il est également compatible avec les grilles utilisées par ces derniers, en les convertissant à la volée en cartes en trois dimensions.

2 Simulation sur grille

Un premier simulateur à deux dimensions sur grilles de cellules a été développé pour reproduire différents algorithmes de l'état de l'art ainsi que ceux proposés. Ce simulateur a été développé en Python et implémente les algorithmes suivants :

1. MDFS [60]
2. Frontières globales [27]
   - Assignation Greedy
   - Assignation MinPos [89]
3. BMILRV [102]
4. Frontières locales [134]

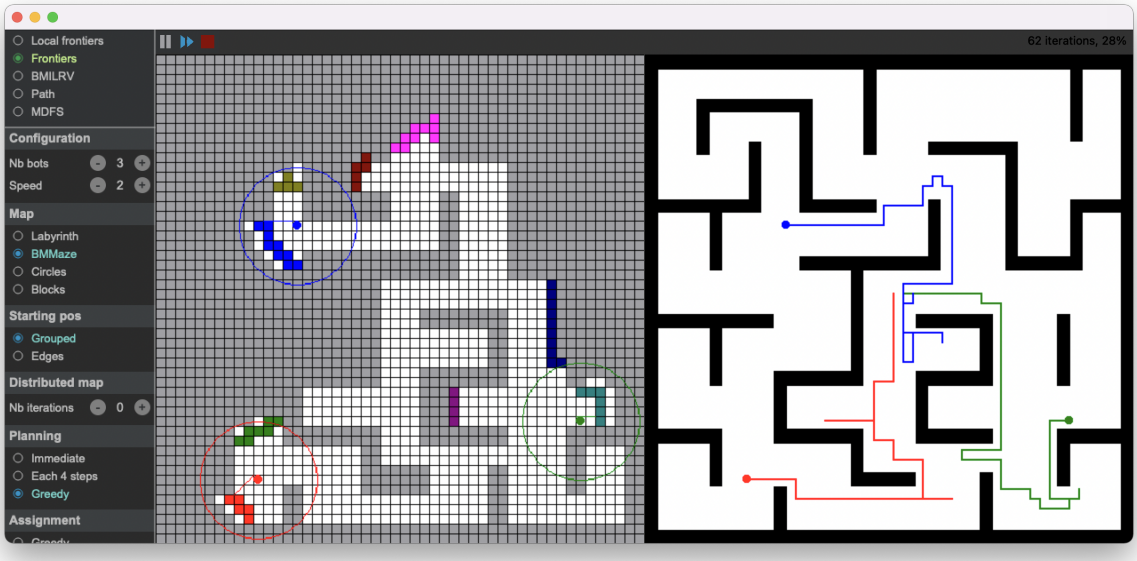


FIGURE 6.1 – Simulateur sur grille développé en Python au début de la thèse.

Il est possible de choisir le nombre de robots, leur rayon de perception, leurs positions de départ (groupés au centre ou séparés aux bords de la carte) et de tester les algorithmes sur quatre labyrinthes. L'intervalle entre deux partages d'informations entre les robots peut également être choisi. Le simulateur intègre également des fonctions de calcul et de sauvegarde de la progression de l'exploration qui a permis de réaliser une partie des expériences de comparaison de l'approche par frontières locales avec les approches de l'état de l'art.

L'objectif de ce simulateur était triple : mieux comprendre les approches de l'état de l'art, tester de nouvelles approches, et comparer les approches proposées aux approches de l'état de l'art. Il a été programmé de façon à être léger et facile à utiliser.

Ce simulateur a été utilisé pour produire les résultats de l'article «Exploration et couverture par stigmergie d'un environnement inconnu avec une flotte de robots autonomes réactifs »[134] ainsi que les vidéos utilisées pour sa présentation à la conférence *JFSMA 2019* (27e Journées Francophones sur les Systèmes Multi-Agents).

3 Expérimentation sur robot réel

Pour explorer un environnement avec une flotte de robots réels, l'équipe Larsen dispose de plusieurs Turtlebots 2 [1] équipés d'un LIDAR. Ces derniers utilisent ROS [2], et j'ai donc adapté l'approche que j'ai proposé sur ROS.

Le passage à ROS n'a pas été aisé. L'installation de la version de ROS spécifique des robots demande une configuration précise (différentes versions de Python, de nombreux paquets spécifiques, etc), et l'installation sur MacOS a nécessité une compilation complète de ROS à partir des sources. De plus, les différents outils de ROS utilisés (movebase, gmapping, etc) n'ont pas toujours une documentation à jour, et dans de nombreux cas trouver les bonnes fonctions à utiliser a nécessité de nombreuses heures de recherche sur internet. J'ai cependant pu compter sur l'aide d'Olivier Rochel pour différents aspects de la configuration et l'utilisation des robots avec ROS.

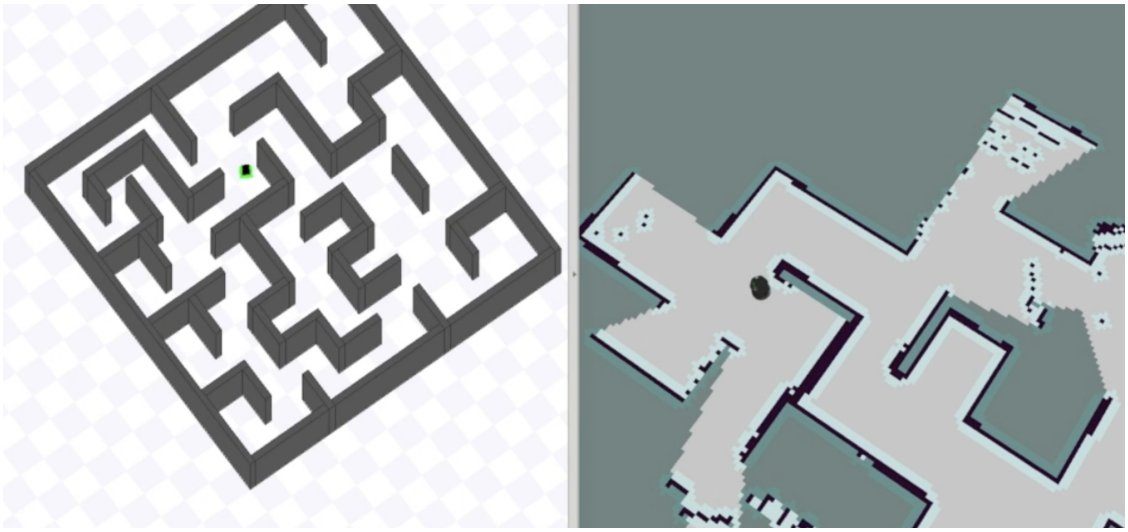


FIGURE 6.2 – Exécution de l'algorithme de frontières locales sur ROS (Robot Operating System). Vue de la carte sur Stage à gauche, et de la grille d'occupation du robot sur RViz à droite.

La figure 6.2 montre la simulation de l'algorithme de frontières locales que j'ai proposé sur un Turtlebot 2 avec *Stage* et *RViz*. L'implémentation de l'algorithme a cependant été mise en pause au moment du premier confinement, laissant certains problèmes non résolus : un seul robot est pris en charge, et il arrive que l'exploration ne soit pas complète à cause d'un problème de synchronisation de la grille d'occupation entre les données reçues par mon script python et des données effectives calculées par *gmapping* [3].

1. Turtlebot 2 https ://www.turtlebot.com/turtlebot2/
2. Robot Operating System https ://www.ros.org
3. Gmapping http ://wiki.ros.org/gmapping

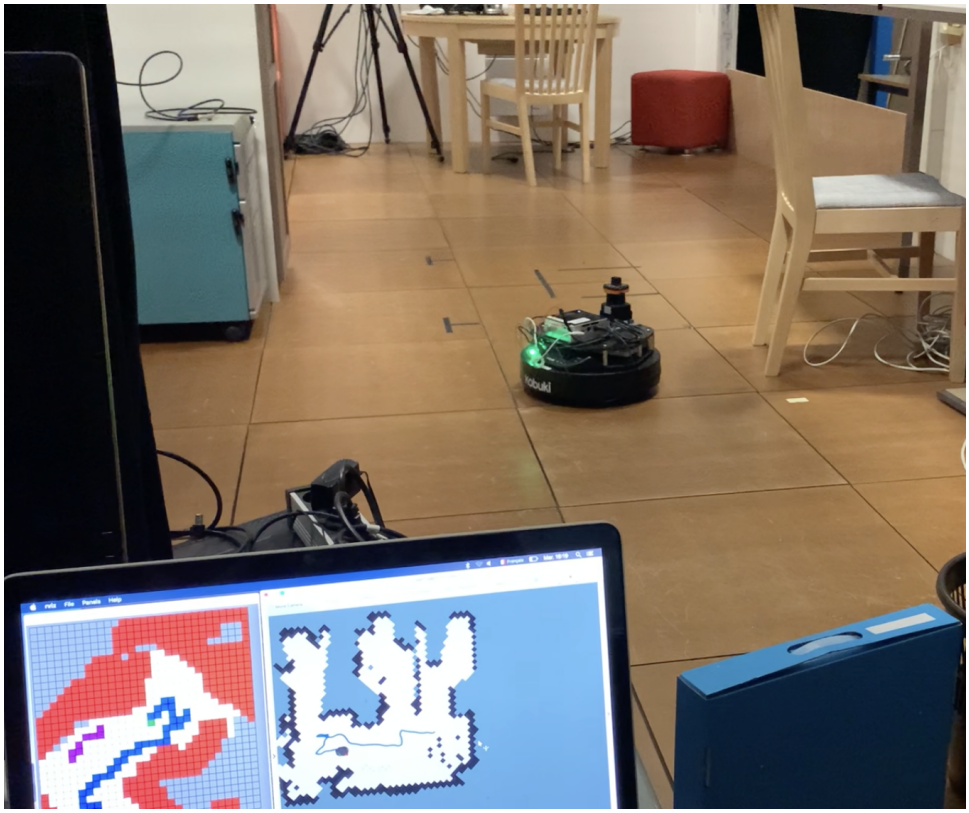


FIGURE 6.3 – Exploration de l'appartement intelligent au Loria par un Turtlebot 2 avec l'approche par frontières locales lors des expérimentations menées au cours de cette thèse.

Les tests effectués avec ROS ont cependant permis d'explorer efficacement l'appartement intelligent du Loria avec un Turtlebot 2 (voir figure 6.3). Ce travail a également été rendu disponible sur le Gitlab de l'Inria, pour pouvoir être éventuellement repris et continué si besoin.

4 Simulation avec LiDAR

De façon à simuler les algorithmes dans un environnement plus réaliste avec un LiDAR, j'ai développé un nouveau simulateur en 3D sans grille. Celui-ci à été programmé en Javascript avec le moteur de rendu 3D opensource Three.JS [4].

L'utilisation du Javascript et d'un moteur compatible avec les navigateurs web a permis de faciliter les échanges avec mes encadrants, leur permettant de voir et tester les simulations en un clic. En effet, l'utilisation du simulateur utilisant ROS nécessitais une très longues installation et configuration (ROS, Stage, RViz, ...), représentant souvent plusieurs journées de travail, tandis que le simulateur Python demandait d'installer Python ainsi que plusieurs bibliothèques, ce qui n'est pas toujours aisé sur des ordinateurs d'entreprise, dont les autorisations sont souvent restreintes.

En dehors des cartes les plus lourdes provenant d'*Habitat Matterport Dataset*, le simulateur fonctionne également sur smartphone avec un navigateur web.

Ce simulateur utilise les quatre labyrinthes utilisés dans le simulateur Python et les simulations ROS, en convertissant ces derniers en cartes 3D à l'execution. De plus, dif-

4. Moteur de rendu ThreeJS https ://threejs.org

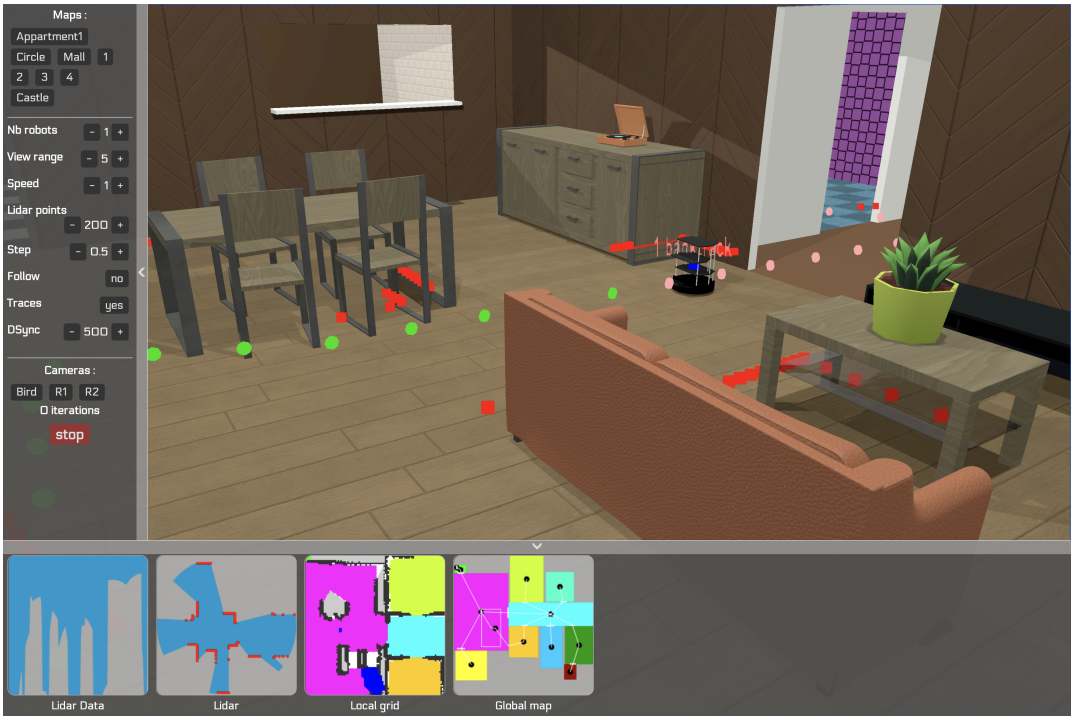


FIGURE 6.4 – Capture d'écran du simulateur 3D développé pour les algorithmes proposés.

férents environnements ont été modélisés avec le logiciel opensource Blender [5] pour se rapprocher de situations réelles, notamment un centre commercial et l'appartement visible sur la capture d'écran de la figure 6.4. Le simulateur est également compatible avec les datasets couramment utilisés avec Habitat [146, 137], tels que *Habitat Matterport Dataset* [145].

Le menu du simulateur permet de choisir le nombre de robots, leur rayon de perception, leur vitesse, le nombre de points du LiDAR, l'intervalle entre deux exécutions de l'algorithme de décision, et l'intervalle entre deux communications entre les robots.

Un menu en bas de l'écran permet également d'afficher différentes informations récoltées et déterminées localement par l'un des robots, notamment les données du LiDAR, la grille d'occupation locale et la carte des pièces et couloirs.

Tout comme les deux autres simulateurs, le code source a été rendu disponible sur le Gitlab de l'Inria, pour permettre de le réutiliser pour d'autres expérimentations à l'avenir.

4.1 Simulation de capteurs

Lors de l'utilisation d'un simulateur, l'intégration de certains algorithmes, par exemple pour la reconnaissance d'objets, peut s'avérer coûteuse. Une alternative à l'utilisation de ce type d'algorithme est de simuler la reconnaissance d'objets, en indiquant préalablement des informations sémantiques sur les objets de la scène, puis en considérant une probabilité qu'un robot identifie un objet donné en fonction, par exemple, de sa distance, du champ de vision du robot, etc.

La figure 6.5 donne un exemple d'objet identifié par le robot lorsqu'il se trouve dans son champ de vision grâce à des informations sémantiques préalablement inscrites dans la scène.

5. Blender https ://www.blender.org

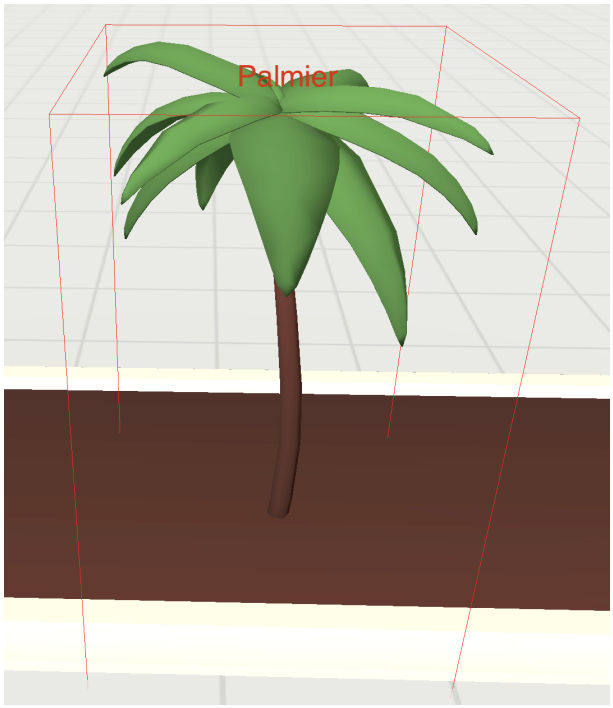


FIGURE 6.5 – Simulation d'un objet (palmier) reconnu par le robot, dans le simulateur 3D développé au cours de cette thèse.

Dans les jeux vidéos, identifier par exemple un objet se trouvant sous le pointeur de la souris du joueur peut être coûteux en calcul, nécessitant un lancer de rayon dans la scène. Ce calcul sera réalisé en testant si le rayon (une droite) traverse les boites englobantes des objets présents, puis pour chacun des objets traversés, en testant si l'un des triangles le composant a une intersection avec la droite donnée. Une alternative est alors d'utiliser une mémoire tampon spécifique, non rendue à l'écran, dans lequel chaque type d'objet a une couleur spécifique, unique et identifiable. Il suffira alors de regarder la couleur du pixel dans la mémoire tampon à la position du curseur de la souris pour connaître le type d'objet survolé. Le rendu de plusieurs images (mémoires tampons) est utilisé de manière systématique par les moteurs de jeux actuels, notamment pour permettre aux différents algorithmes de rendus de prendre en compte différentes informations sur la scène (telles que la normale des faces, la profondeur (distance à la caméra), où des matériaux présents (la façon dont ils renvoient la lumière, etc.).

La figure 6.6 montre différents buffers calculés par le moteur Unreal Engine 5 [6] utilisés pour le rendu de la scène située au centre de l'image. Différents buffers enregistrent les informations de *shading* spécifiques des matériaux, comme la *metalness* (à quel point le matériau se comporte comme un métal, renvoyant les couleurs autour de lui), la *specular* (à quel point le matériau renvoie la lumière, à la façon d'une surface brillante et non d'un miroir, correspondant à la *metalness*), où encore là *roughness* (dureté, correspondant à la façon dont la lumière renvoyée selon les paramètres *roughness* et *metalness* sera diffuse, « floutée »). D'autres buffers incluent la *subsurface color*, utilisée pour les matériaux absorbant la lumière (comme la cire d'une bougie allumée, ou l'herbe de la figure 6.6), la normale (où chaque composante des couleurs RGB correspond à l'un des axes XYZ pour encoder un vecteur normal, permettant de savoir dans quelle direction renvoyer la lumière), la profondeur ou l'opacité. Les moteurs de jeux récents tels qu'Unreal Engine intègrent ainsi de nombreux buffers rendus simultanément, puis composés

6. Moteur gratuit et opensource Unreal Engine https ://www.unrealengine.com

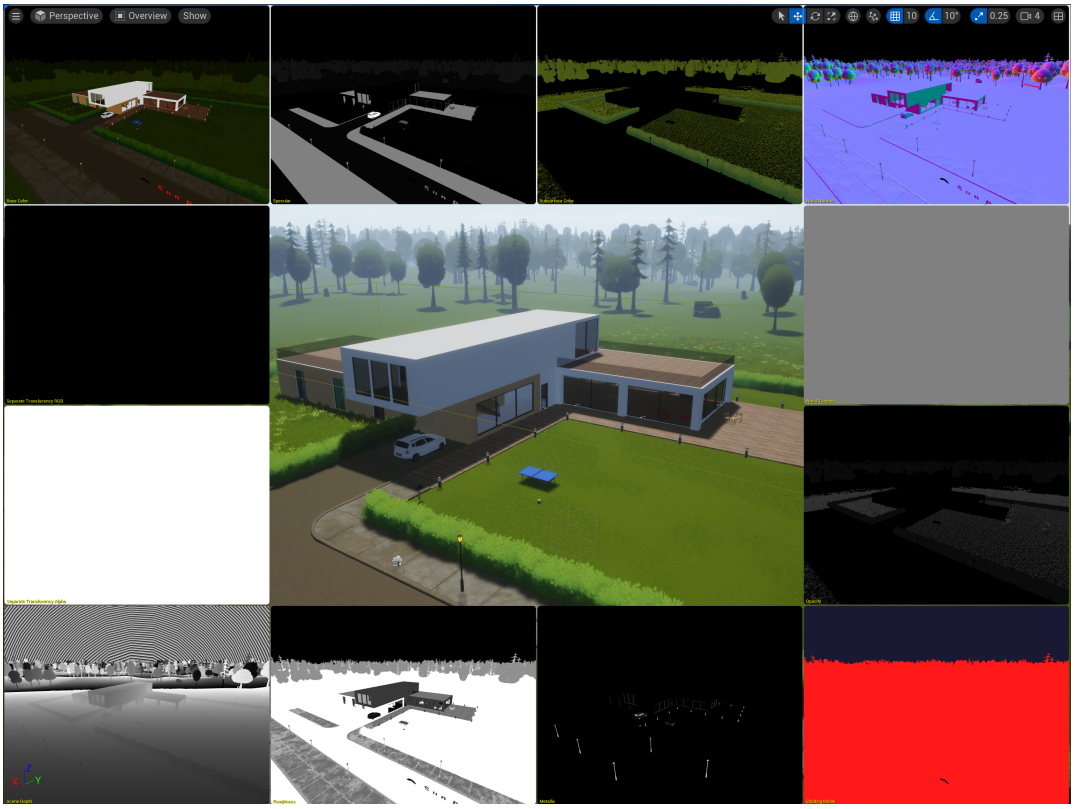


FIGURE 6.6 – Visualisation de différents buffers d'une scène (rendu final au centre) avec le moteur Unreal Engine 5

ensemble pour produire le rendu final. Une part significative des algorithmes de rendu, utilisant ces différents buffers pour produire l'image finale sont issus de la recherche [7, 24, 86].

La méthode décrite ci-dessus permet un gain de temps significatif pour l'exécution de nombreuses simulations sur le simulateur 3D, accélérant ainsi le test des différents algorithmes. Dans le cas de l'utilisation d'une caméra couleur et distance (RGB-D), deux buffeurs (un pour les couleurs, un pour la distance) peuvent ainsi être remplis simultanément dans une seule phase de rendu de la scène 3D.

5 Hypothèses générales

Dans toutes les expérimentations réalisées dans cette thèse, différentes hypothèses ont été posées de façon à simplifier le cadre des approches testées. Notamment :

- les robots sont équipés de LiDARs à une couche percevant les obstacles autour d'eux à 360 degrés à une distance maximale paramétrable (les robots réels dans les simulations sur ROS subissent cependant un angle mort),
- les robots disposent d'un module *SLAM* leur permettant une localisation précise (gslam sur les robots réels sur ROS),
- les déplacements des robots sont supposés holonomes, leur permettant une rotation sur eux-mêmes et ainsi un déplacement dans toutes les directions,
- le sol est supposé plat, réduisant ainsi la carte des robots à une projection sur un plan.

6 Bilan des simulateurs et expérimentations

Les trois modes de simulation utilisés au cours de cette thèse ont été complémentaires, permettant des simulations plus ou moins proches de la réalité avec différents algorithmes. Le simulateur développé en Python permet de tester efficacement des algorithmes sur un carte préalablement définie sous forme d'une grille. Le simulateur *ROS* permet à la fois de simuler l'exploration et de l'exécuter sur des robots réels. Enfin, le simulateur 3D ne nécessite aucune installation, et permet de réaliser des simulations plus proches de la réalité, en simulant un *LIDAR* dans un environnement 3D.

	Simulateur Python	**ROS**	**Simulateur 3D**
Type	Simulateur executable	Simulation et utilisation sur robot réél (Turtlebot 2)	Simulateur web
Langage	Python 3	Python 2	Javascript, TypeScript
Logiciels et bibliothèques	PyQt5	ROS, RViz, Gmapping, Stage	ThreeJS, Webpack, EaselJS
Algorithmes implémentés	MDFS, Frontières globales (Greedy, MinPos), BMILRV, Frontières locales	Frontières locales	Frontières locales, segmentation de l'environnement
Type de monde	Grille d'occupation (avec coordonnées entières pour les robots)	Monde continu + grille d'occupation générée par les robots	Monde continu + grille d'occupation générée par les robots
Capteurs	Simulation du lidar avec lancé de rayon (non bruité) sur une grille d'occupation	Simulation de LIDAR, utilisation de lidar réél	Simulation du lidar avec lancé de rayon (bruité) sur un monde continu 3D, caméra couleur et distance
Cartes utilisées	Quatre labyrinthes, cartes aléatoires générées par automates cellulaires	Quatre labyrinthes, appartement intelligent du Loria (réél)	Quatre labyrinthes, cartes issues de nuages de points d'explorations rééles, appartements modélisés en 3D

Chapitre 7

Exploration par stigmergie

Sommaire

1 Introduction

L'objectif de l'algorithme proposé dans cette partie est de permettre une approche par frontières *locales* pour aborder le problème de l'exploration multirobot. Par frontières locales, nous entendons un algorithme par frontières dans lequel un robot prend une décision en ne tenant compte que des informations à l'intérieur de son champ de perception (y compris des traces laissées au sol par d'autres robots). La finalité de ce travail est de proposer une approche évolutive et efficace, même dans un environnement où la communication est restreinte, pour résoudre le problème de l'exploration avec un grand nombre de robots. L'algorithme proposé se veut aussi simple que possible, exécutable avec des ressources et communications limitées, tout en garantissant la couverture.

L'algorithme proposé est un compromis entre les approches locales, qui nécessitent peu de calculs et communications, et les approches globales, qui intègrent la carte globale partielle et les marquages associés (visité et non visité). Cette approche permet également de regrouper les robots à un point de rendez-vous une fois l'exploration terminée, tout en s'assurant que l'exploration a été exhaustive, s'assurant ainsi qu'aucune zone pouvant être explorée n'a été oubliée.

Des algorithmes comme *BMILRV* [102] ont essayé d'adapter des approches locales (ou les robots ne voient qu'une cellule et ses voisins immédiats dans une grille d'occupation) pour considérer l'ensemble du champ de vision des robots, mais ces algorithmes sont beaucoup plus lents que les approches globales [134].

Bien qu'inspirée des approches par frontières globales [26, 61], l'approche que nous proposons aborde différemment le problème de l'exploration : dans le cas de l'approche globale, c'est avant tout un problème de planification, où l'algorithme doit décider de l'affectation des robots aux frontières connues. Dans cette version locale, les robots ne connaissent pas le nombre ou la position actuelle des autres robots et ne s'en soucient pas. Ils ne font que réagir aux informations locales. La collaboration entre robots se fait par stigmergie, dans notre cas simulée à l'aide d'un ensemble de positions occupées et d'une pile de positions successives (historique des positions).

L'approche présentée dans cette partie est également inspirée des algorithmes de *backtracking* (comme l'algorithme *BoB* [108]) où les robots reviennent à leur position précédente lorsqu'il n'y a plus de zone non visitée dans leur voisinage.

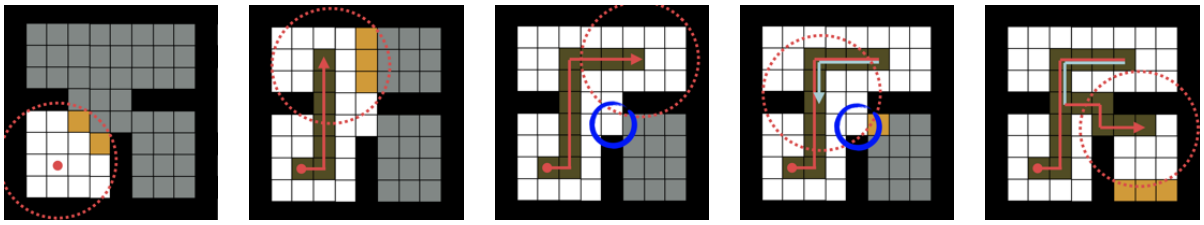
Nous présenterons dans ce chapitre une première version simple de notre approche par frontières locales fonctionnant sur une grille d'occupation (section 2. Suivi de frontières locales sur grille). Améliorations). Enfin, une version sans grille est présentée section 5 avec pour objectif de limiter la quantité d'informations utilisée par l'algorithme et transmise entre les robots. Plusieurs améliorations seront ensuite présentées pour améliorer les performances de l'exploration (section 6. Améliorations).

2 Suivi de frontières locales sur grille

2.1 Exploration exhaustive

De façon informelle, l'algorithme local consiste à mémoriser les chemins parcourus, à se déplacer le plus loin possible vers des zones inconnues, et revenir sur ses pas lorsqu'il n'y a plus de zone inconnue visible, jusqu'à ce que l'on en retrouve une. Lorsque le robot est revenu à son point de départ sans avoir vu aucune zone inexplorée, toute la carte a été explorée. La carte peut être vue comme un graphe, dont l'algorithme effectue

un parcours exhaustif en profondeur d'abord.



a b c d e

FIGURE 7.1 – Lorsqu'une zone est laissée inexplorée, comme c'est le cas à partir de l'image *b* (zone inexplorée entourée en bleu dans les images *c* et *d*), le robot explorera cette zone lors du *backtracking*, effectué lorsque le robot est arrivé au bout d'une zone à explorer (le robot *backtrack* dans l'image *d*).

Dans le cas de plusieurs robots, selon les conditions initiales et la topologie de la carte, un ou plusieurs robots peuvent être revenus à leur point de départ avant la fin de l'exploration, ce qui entraîne une mauvaise distribution de l'exploration et réduit la performance de l'algorithme. Pour réduire cette perte de performance, on peut envisager d'utiliser dans ce cas les traces des autres robots pour reprendre l'exploration une fois qu'ils sont revenus au point de départ, si tous les autres robots ne se sont pas arrêtés. Cette piste d'amélioration est implémentée dans la partie 4.

L'idée générale de l'algorithme proposé est qu'une zone inexplorée est explorée immédiatement, où le sera lors d'un des retour en arrière du robot. Lors d'un retour en arrière, un robot revient au moins une fois à chacune des positions visitées. Ce principe garanti l'exhaustivité de l'exploration (voir figure 7.1).

Bien que cet algorithme soit simple, différentes améliorations peuvent augmenter son efficacité. Nous avons proposé plusieurs pistes d'amélioration dans la partie 6.

2.2 Formalisation

Nous considérons une flotte homogène de m robots mobiles $R = \{r_1, \dots r_m\}$ équipés de capteurs leur permettant de détecter des obstacles dans un rayon de perception ρ. Les robots utilisent une grille d'occupation $\mathcal{O}$ représentant la carte, où chaque cellule peut être dans trois états : *inexploré*, *obstacle* ou *exploré*, ainsi qu'une grille $\mathcal{P}$ pour assigner un booléen à chaque cellule, indiquant si une cellule est visitée ou non (c'est à dire si un robot a été sur cette cellule), initialisée à *faux*. Ces cartes sont échangées à intervalles réguliers entre les robots (voir section 2.3. Procédure de partage de la carte). Enfin, chaque robot peut définir une cellule objectif (représentant un point à atteindre sur la carte), et possède une grille $\mathcal{G}r_i$ indiquant pour chaque cellule visitée la distance maximale parcourue depuis le point de départ.

La grille d'occupation $\mathcal{O}$ correspond à la carte produite par le robot. Les cellules *explorées* correspondent au cellules vues et accessibles par les robots, les cellules *inexplorées* n'ont pas été vues par le robot, et les cellules *obstacle* correspondent aux obstacles identifiés par le robot. La grille $\mathcal{P}$ permet de savoir si une cellule à déjà été occupée par un robot.

Pour chaque déplacement (d'une cellule), chaque robot connaît l'état des cellules dans son rayon de perception et sa position sur la grille (x, y). On notera $\mathcal{V}_{r_i}^t$ l'ensemble des cellules visibles perçues par le robot à un moment donné t qui ne sont pas des obstacles.

Les cartes locales ($\mathcal{G}$ et $\mathcal{P}$) sont partagées entre les robots toutes les d_{sync}itérations avec la procédure décrite dans la partie 2.3.

Le paramètre d_{max} est utilisé pour forcer les robots à réassigner la cellule objectif à intervalles réguliers, évitant ainsi de choisir une cible maintenue trop longtemps si le rayon de perception est grand.

Chaque robot r_i applique l'algorithme suivant :

1. Marquer toutes les cellules visibles comme explorées :
   $\forall (x, y) \in \mathcal{V}_{r_i}^t, \mathcal{O}_{r_i}(x, y) = \textit{exploré}$
2. Si aucune cellule objectif n'est définie, déterminez les frontières locales $\mathcal{F}$, c'est-à-dire toutes les cellule de $\mathcal{V}_{r_i}^t$ ayant des voisins inexplorés.
   (a) Si $|\mathcal{F}| > 0$, choisir la cellule frontière la plus éloignée de toute cellule visitée (dans la grille $\mathcal{P}_{r_i}$) à une distance maximale d_{max} de la cellule comme nouvel objectif. Si plusieurs cellules sont possibles, choisir celle qui nécessite le moins de rotation du robot.
   (b) Si $|\mathcal{F}| = 0$ et si on est de retour à la cellule de départ, s'arrêter.
   (c) Sinon, retournez un pas en arrière sur la trace G_{r_i} : choisir la cellule, parmi les 4 cellules adjacentes, ayant la valeur $\mathcal{G}_{r_i}(x, y)$ la plus basse.
3. Sinon, si une cellule objectif est définie
   (a) Si $\mathcal{G}_{r_i}(x, y) = 0$, mettre à jour la trace : $\mathcal{G}_{r_i}(x, y) = 1 + \max(\mathcal{G}_{r_i}(x-1, y), \mathcal{G}_{r_i}(x+1, y), \mathcal{G}_{r_i}(x, y-1), \mathcal{G}_{r_i}(x, y-1), \mathcal{G}_{r_i}(x, y+1))$.
   (b) Mettre à jour la grille : $\mathcal{P}_{r_i}(x, y) = 1$
   (c) Se rapprocher de l'objectif (à l'aide d'un algorithme de *pathfinding* parmi les cellules visibles), puis revenir à l'étape 1.
4. Si plus de d_{sync}itérations ont été effectuées depuis le dernier partage de carte, partagez la carte $(\mathcal{O}_{r_i}, \mathcal{P}_{r_i})$ avec les autres robots (voir section 2.3).

Dans nos expériences, nous avons utilisé l'algorithme *A** pour rechercher le chemin optimal entre les robots et leurs cellules objectifs.

2.3 Procédure de partage de carte

Le partage de la carte est nécessaire pour deux raisons : la grille d'occupation permet aux robots de définir les frontières locales ; si elle n'est pas partagée, les robots ne peuvent pas savoir si autre un robot a déjà exploré une partie donnée de la carte. De plus, la trace est également partagée pour permettre une meilleure répartition des robots, car elle est utilisée dans le choix de la frontière à définir comme objectif.

Toutes les d_{sync}itérations, chaque robot peut envoyer sa grille d'occupation $\mathcal{O}$ et ses cellules visitées $\mathcal{P}$ à tous les autres robots. Lorsqu'un robot r_i reçoit des données d'un autre robot r_j (grille d'occupation $\mathcal{O}_{r_j}$ et cellules visitées $\mathcal{P}_{r_j}$), il met à jour sa grille d'occupation de la façon suivante :

$$\mathcal{P}_{r_i}(x, y) = \mathcal{P}_{r_i}(x, y) \vee \mathcal{P}_{r_j}(x, y)$$

Un *ou* logique est appliqué entre la valeur du robot et la valeur reçue : une cellule est visitée si le robot ou tout autre robot l'a déjà marquée comme visitée.

Il n'est pas nécessaire que les robots soient synchronisés pour partager leurs cartes en même temps, ils peuvent les envoyer ou les recevoir à tout moment.

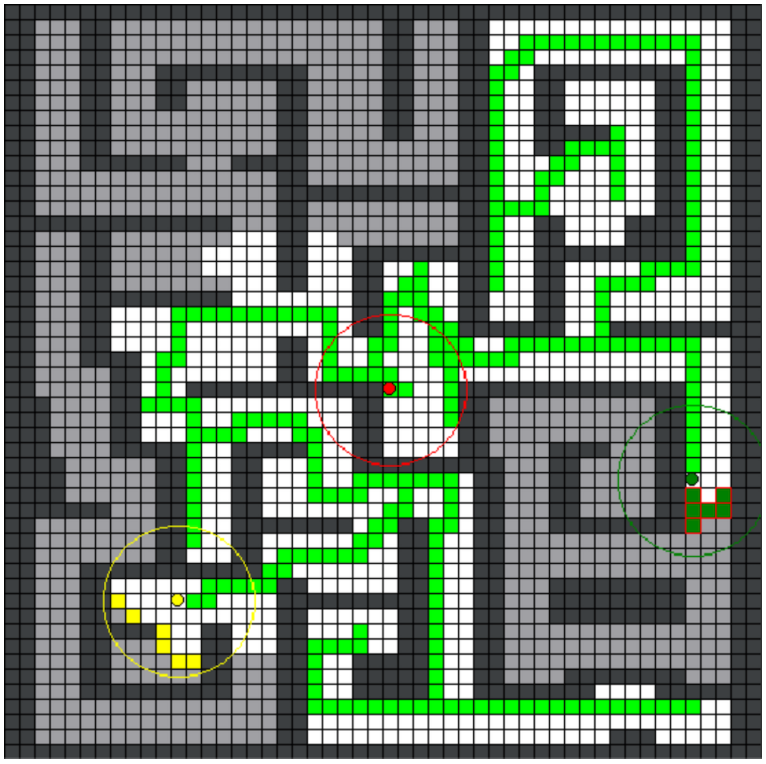
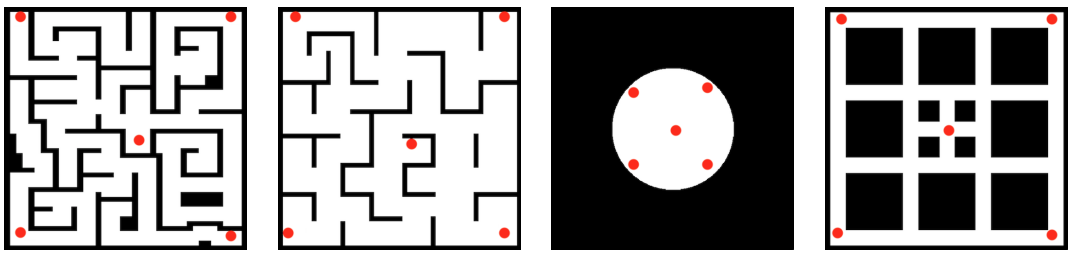


FIGURE 7.2 – Représentation de la grille lors d'une exploration par l'algorithme par frontières locales (cellules explorées en blanc, traces en vert ; les cellules vert foncé et jaune correspondent aux frontières locales des robots vert foncé et jaune).

3 Résultats expérimentaux

Nous avons comparé trois approches différentes : l'approche frontière reproduit l'expérience décrite dans [106] avec une réaffectation de l'objectif tous les $d_{\max}$ pas de temps, en utilisant la stratégie d'affectation *MinPos*. Comme cette approche ne fournit pas de point de rendez-vous, nous considérerons le nombre total d'itérations jusqu'à ce que toute la carte soit visitée (c'est-à-dire avant la fin de la mission). L'approche *BMILRV* est une reproduction de l'algorithme décrit dans [102]. Il s'agit d'une approche locale (*Brick & Mortar*) utilisant uniquement les cellules marquées dans le champ de perception des robots, comme dans notre algorithme.



(a) Labyrinthe 1 (b) Labyrinthe 2 (c) Cercle (d) Blocs

FIGURE 7.3 – Cartes testées

Cette partie présente les résultats expérimentaux obtenus en simulation pour 4 cartes différentes présentées dans la figure 7.3. Les points de départ des robots sont représentés par des cercles rouges. Dans ces expériences, la carte est partagée à chaque pas de

temps ($d_{\text{sync}} = 1$, voir la section 3.5 pour la version distribuée), la plage de perception ρ est de 5 cellules, et $d_{\text{max}} = 2$ (les objectifs sont mis à jour tous les 2 pas). Le labyrinthe 2 est celui utilisé dans [102], le labyrinthe 1 est une variante avec des couloirs plus serrés. La carte Cercles permet de tester un environnement libre (avec peu d'obstacles), et la carte Blocs correspond à un environnement avec de nombreux obstacles isolés.

Pour obtenir une première comparaison expérimentale des performances des différents algorithmes, nous avons développé un simulateur codé en Python, qui permet d'exécuter les algorithmes dans des conditions identiques sur les 4 cartes présentées dans la Figure 7.3. Notre simulateur permet une exécution décentralisée des algorithmes pour les robots. Dans nos expériences, nous supposons une localisation parfaite, une détection déterministe des obstacles. Nous supposons également que les cellules de la grille d'occupation permettent à plusieurs robots de partager la même cellule.

Nous avons évalué le temps requis par notre algorithme pour explorer la totalité d'une carte donnée, comme son évolution lorsque les communications entre robots sont désactivées une partie du temps.

3.1 Trajectoires des robots

Chaque algorithme produit des trajectoires de robots caractéristiques. La figure 7.4 présente les trajectoires obtenues sur les cartes Cercle et Blocs avec notre approche par frontières locales avec deux robots, placés initialement au centre des deux cartes. Sur la carte cercle, le rayon de perception relativement faible du robot produit une trajectoire particulière où les traces du robot seront généralement éloignées d'un peu moins de deux fois le rayon de perception du robot.

La Figure 7.5 montre les trajectoires obtenues avec l'approche frontière et l'algorithme proposé sur la carte *Labyrinthe 2*. Plusieurs différences sont visibles : le retour en arrière se fait en suivant la trace du robot dans l'algorithme proposé, alors que les retours arrière diffèrent des trajectoires en avant dans le cas de l'approche frontière (voir la section 6.1. Retours en arrière).

L'approche locale donne une distribution légèrement moins bonne des robots sur la carte, puisqu'un robot s'arrête lorsqu'il est revenu au point de départ et qu'aucune zone inexplorée n'est visible dans son champ de perception, mais le temps total d'exploration de la carte entière est le même pour les deux algorithmes.

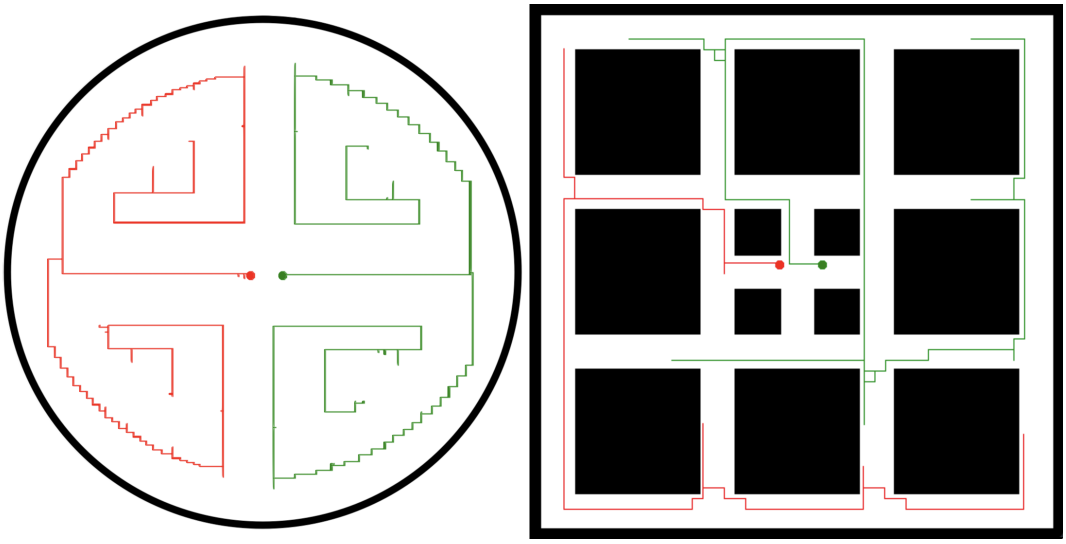
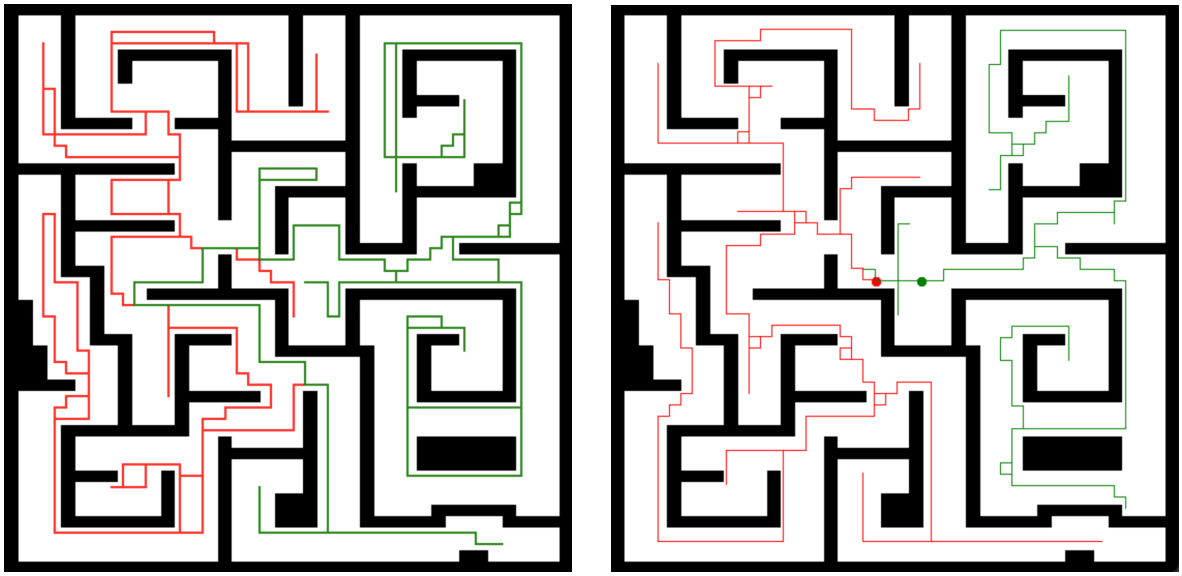


FIGURE 7.4 – Trajectoires obtenues avec notre algorithme sur les cartes Cercle et Blocs avec deux robots.



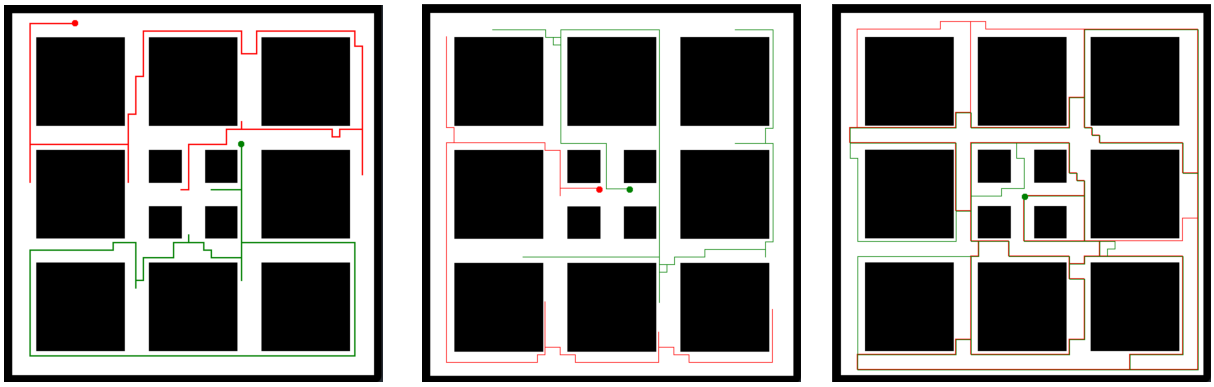
Approche globale Algorithme proposé

FIGURE 7.5 – Comparaison des trajectoires obtenues avec l'approche par frontières globales (avec MinPos) et l'algorithme proposé pour deux robots.

La comparaison des trajectoires sur une carte avec de nombreux obstacles isolés illustre clairement la meilleure répartition des robots dans notre approche locale face aux autres approches locales de l'état de l'art telles que *BMILRV*. La figure 7.6 montre les trajectoires obtenues sur la carte Blocs pour les trois approches.

Dans le cas de l'approche *BMILRV*, la majorité des couloirs ont été traversés plusieurs fois par les deux robots plusieurs fois avant la fin de l'exploration. Notons cependant ici que l'approche globale ne regroupe pas les robots aux points de départ, contrairement aux deux autres approches.

Notre approche fournit quant à elle une répartition des robots similaire à l'approche globale sur ce type de carte. Nous allons maintenant comparer la durée de l'exploration sur les différentes cartes par les trois approches.



Approche globale | Algorithme proposé | Algorithme BMILRV

FIGURE 7.6 – Comparaison des trajectoires obtenues par les approches frontières globales (MinPos), frontières locales et BMILRV sur la carte Blocs.

3.2 Durée de l'exploration

L'approche globale ne regroupant pas les robots en fin d'exploration, nous allons ici considérer le nombre d'itérations nécessaire pour que l'ensemble des cellules de la carte aient été vues.

La Figure 7.7 montre le nombre d'itérations nécessaires pour explorer la carte entière avec $m = 3$ robots pour les différentes approches. L'approche *BMILRV* est beaucoup plus lente que les approches frontières locales ou globales, surtout lorsqu'il y a des obstacles isolés (à cause du système de contrôle de boucles), ce qui est le cas dans les cartes Labyrinthe 2 et Blocs. L'approche frontière et notre algorithme donnent des résultats comparables, l'approche frontière globale restant légèrement meilleure.

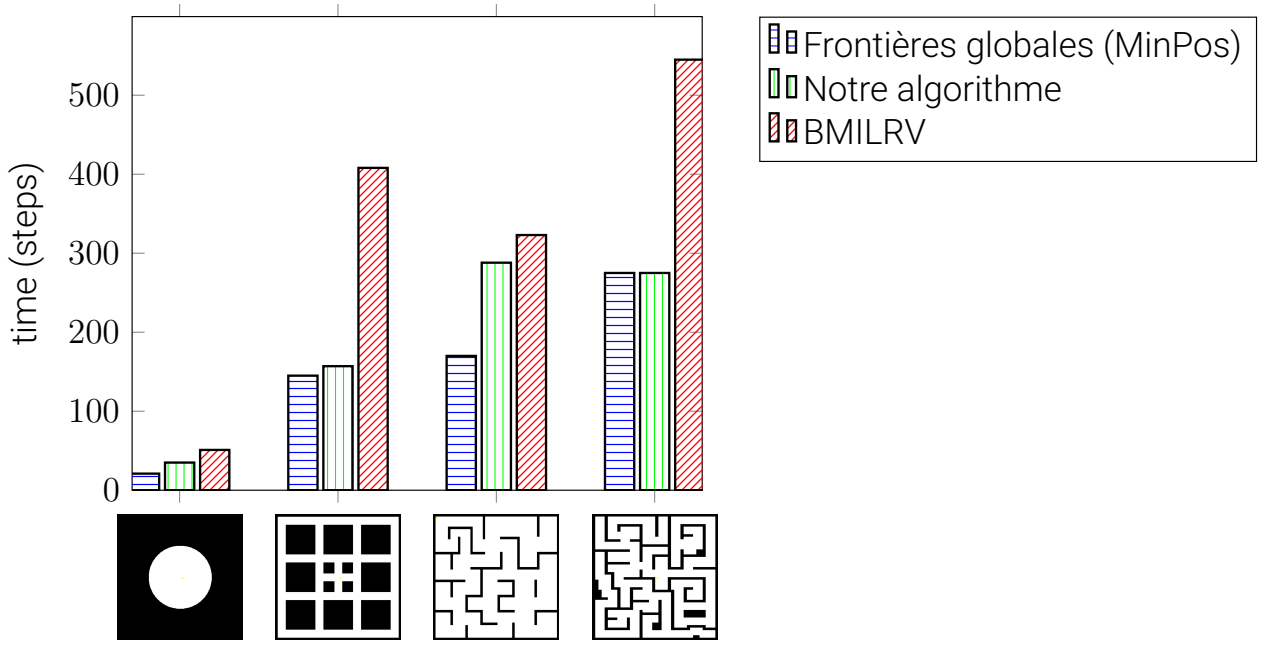


FIGURE 7.7 – Temps total d'exploration pour chaque approche sur les cartes testées avec $m = 3$ robots. Les robots commencent l'exploration au centre de la carte.

L'algorithme proposé donne parfois une moins bonne répartition des robots, et donc de moins bonnes performances, notamment sur la carte *Labyrinthe 1*. Plusieurs ap-

proches peuvent être envisagées pour rapprocher les performances de l'approche globale (voir les sections 4. Seconde chance et 6. Améliorations).

3.3 Progression de l'exploration

De façon à mieux comparer les performances des différents algorithmes, nous pouvons également nous intéresser à la progression de l'exploration, c'est-à-dire à la proportion de la carte qui a été explorée à un instant donné.

La Figure 7.8 montre le pourcentage de la carte exploré au fil du temps (en itérations) pour les approche par frontières locales et globales. Le pourcentage de la carte visité correspond au nombre de cellules de la grille d'occupation $\mathcal{O}$ marquées comme explorées à l'itération t divisée par le nombre total de cellules explorées à la fin de l'exploration.

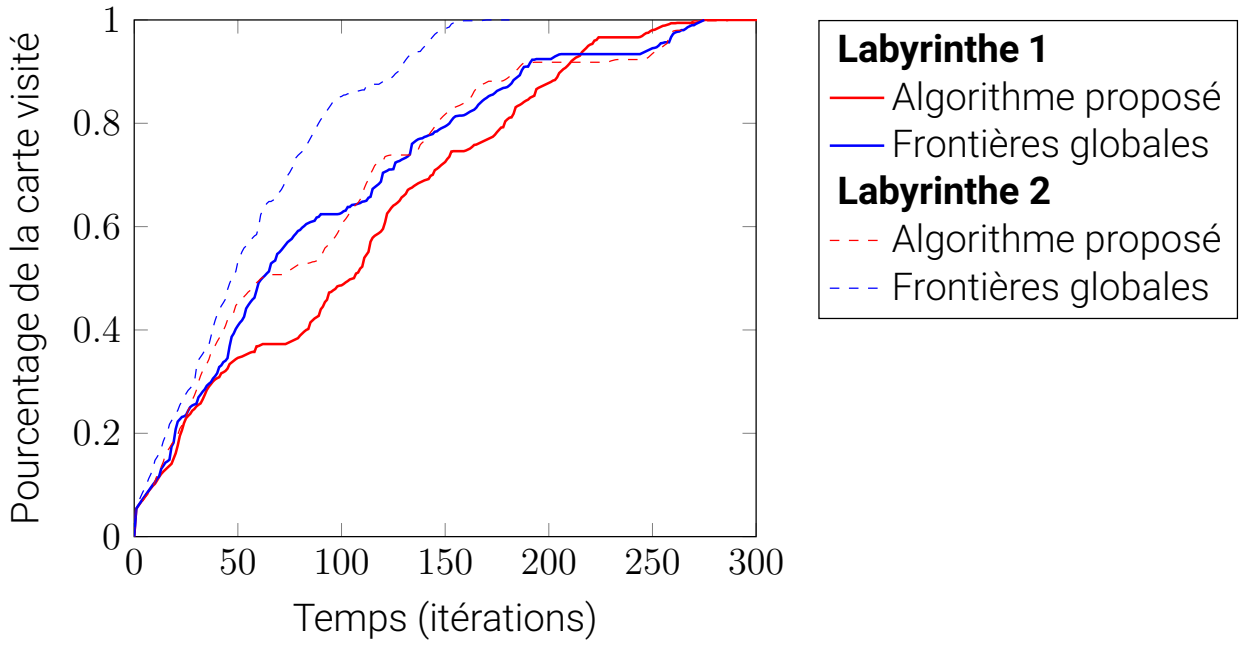


FIGURE 7.8 – Évolution du pourcentage de la carte explorée au fil du temps ($m = 3$ bot) sur les labyrinthes 1 et 2.

L'algorithme local perd facilement du temps lorsque les robots reviennent sur leurs pas car ils empruntent le même chemin que lors de leur premier passage, contrairement à l'algorithme global qui calcule le meilleur chemin pour atteindre une frontière. Cependant, les deux algorithmes montrent des résultats similaires, les deux graphiques se croisent plusieurs fois pour le labyrinthe 1.

3.4 Nombre de robots

Nous allons maintenant comparer le nombre d'itérations nécessaires pour explorer les quatre avec 1 à 8 robots. La figure 7.9 montre le nombre d'itérations nécessaires pour explorer les 4 cartes pour différents nombres de robots. Sur ces 4 cartes, les approches par frontières locales et globales donnent des résultats comparables.

Lorsque le nombre de robots augmente, l'algorithme par frontières globales prend l'avantage en offrant une meilleure répartition des robots sur la carte.

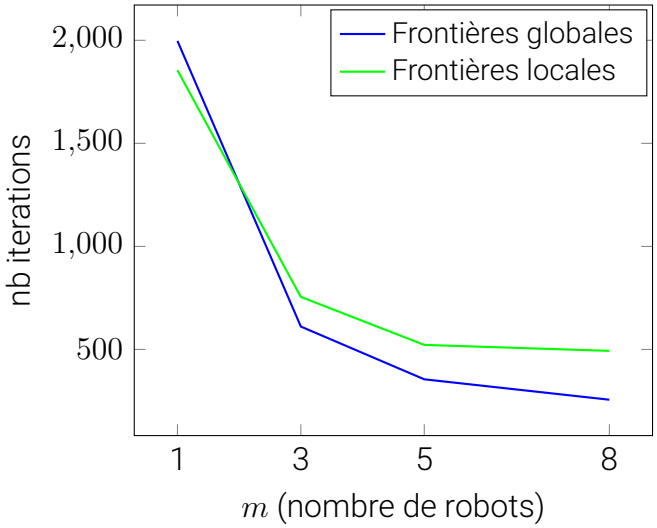


FIGURE 7.9 – Nombre d'itérations nécessaires pour explorer toutes les cartes pour différents nombres de robots.

3.5 Perte de communication

Pour évaluer notre algorithme dans des conditions de perte de communication, nous supposons maintenant que les robots ne peuvent communiquer qu'une fois toutes les d_{sync}itérations, et comparer la durée de l'exploration selon ce paramètre. La Figure 7.10 montre le nombre d'itérations nécessaires pour explorer les 4 cartes avec notre algorithme, en utilisant 3 robots pour différentes valeurs de d_{sync}.

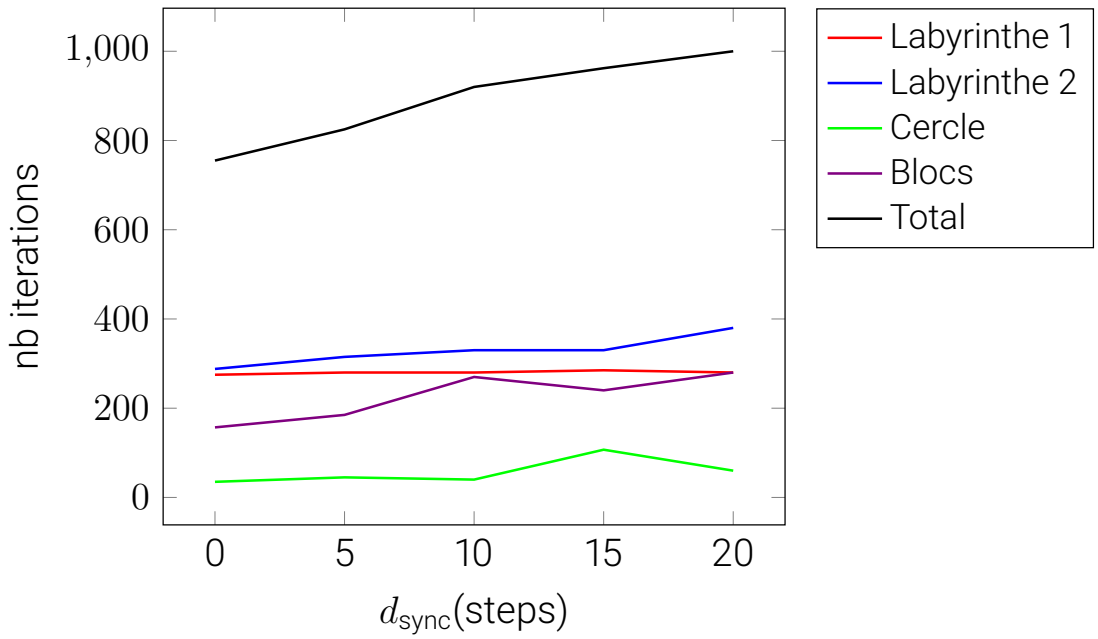


FIGURE 7.10 – Évolution de la durée d'exploration pour différentes valeurs de d_{sync}sur les 4 cartes.

Nous observons une augmentation du temps d'exploration lorsque d_{sync} augmente. Ceci est dû à une moins bonne dispersion lorsque plusieurs robots sont proches les uns des autres : ils explorent alors la même zone simultanément. Si l'on suppose que les robots partagent plus souvent leur carte lorsqu'ils sont proches les uns des autres,

cette diminution est limitée la plupart du temps (la présence de cycles peut cependant amener deux robots éloignés à explorer la même zone).

Bien qu'augmenter d_{sync} diminue les performances de l'exploration, une perte de communication, ou une communication éparse ne causera pas de problème dans l'exploration (« oubli » d'une zone, robot bloqué,...) et ne peut, dans le pire des cas, que prolonger la durée de l'exploration. Dans des applications concrètes, le partage de cartes entre robots peut être adapté en fonction de leur vitesse et de leur rayon de perception, éventuellement en fonction de la distance entre les robots.

4 Arrêt de l'exploration et deuxième chance

Il peut arriver lors de l'exploration qu'un ou plusieurs robots retournent à leur point de départ et ne trouvent plus de frontières visibles avant que l'exploration soit terminée, laissant les robots encore actifs terminer seuls. Lorsque cela se produit, l'exploration reste complète, mais les robots revenus à leur point de départ ne participent plus, ce qui peut augmenter la durée totale de l'exploration. Dans certains cas, la portion explorée par l'un des robots peut être très faible comparée aux autres, rendant l'apport de l'un des robots négligeables, comme dans l'expérimentation figure 7.11.

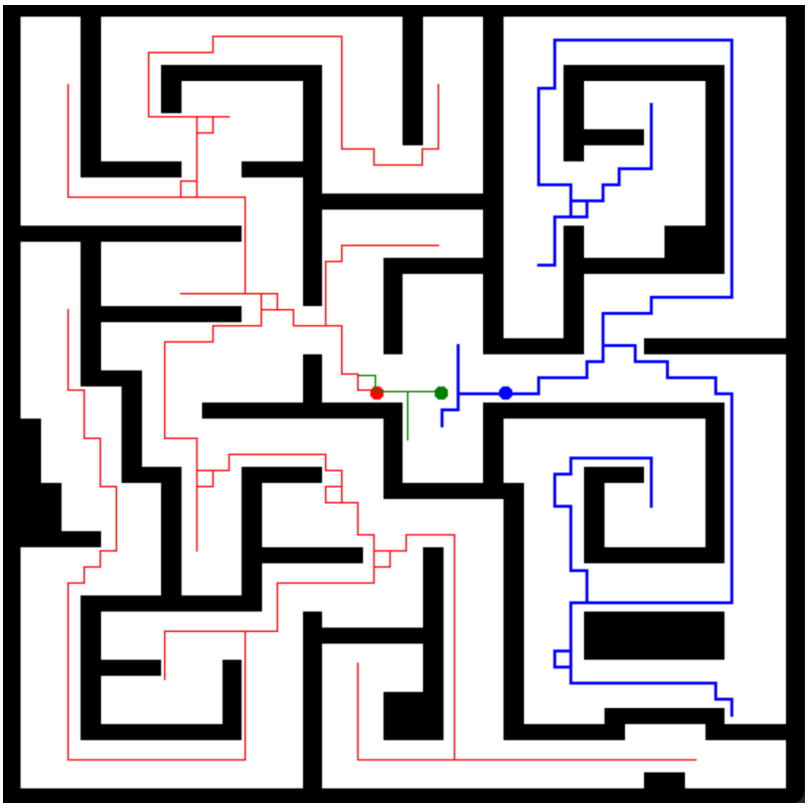


FIGURE 7.11 – Exemple de simulation ou le robot vert (au centre) n'explore qu'une portion infime de la carte, se retrouvant bloqué par les traces des robots à sa droite et sa gauche.

Pour éviter cette perte de performances, nous pouvons envisager un système de « seconde chance », autorisant ces robots à repartir explorer les zones restantes en suivant les traces des autres robots.

Pour permettre aux robots de repartir, nous allons ajouter une variable *mode*, assigné à « *normal* » par défaut, et qui vaudra « *secondeChance* » lorsqu'un robot repart en suivant les traces des autres robots.

Nous allons maintenant comparer trois algorithmes : frontières globales (avec *MinPos*), frontières locales sans seconde chance, et frontières locales avec seconde chance. Cette fois, nous choisissons $m = 10$ robots qui partirons tous du centre du labyrinthe.

4.1 Progression de l'exploration

La figure 7.12 présente l'évolution de l'exploration pour les trois algorithmes sur le premier labyrinthe. Sur cette carte, où les approches globales et locales donnaient des résultats très proches pour $m = 3$ robots, l'algorithme local prend l'avantage avec la seconde chance, finissant l'exploration avant l'algorithme global.

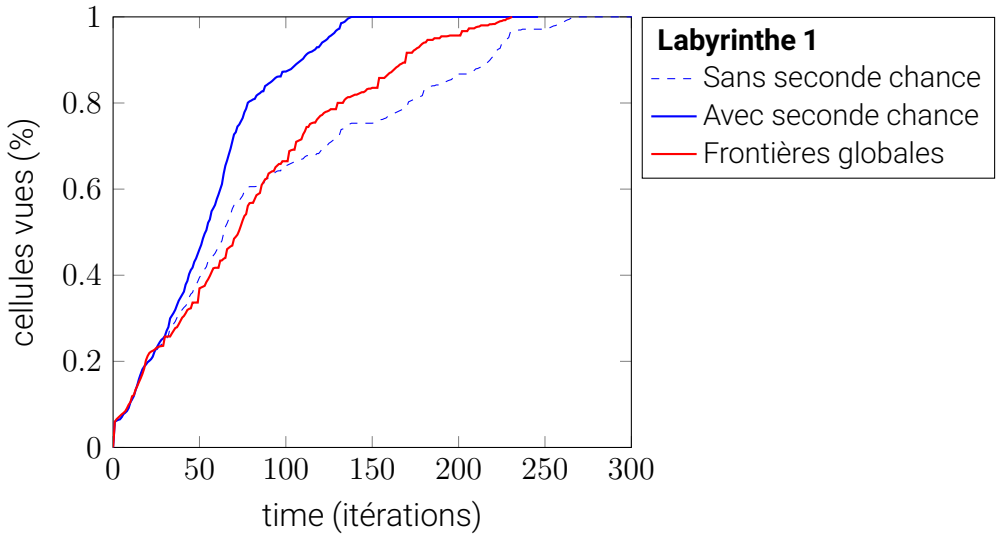


FIGURE 7.12 – Évolution du pourcentage de cellules vues au cours du temps ($m = 10$ robots) avec et sans seconde chance.

4.2 Durée de l'exploration

Nous allons maintenant nous intéresser à la durée totale de l'exploration des quatre cartes par les trois algorithmes, pour $m \in \{3, 5, 8\}$.

La figure 7.13 présente le nombre d'itérations nécessaires pour explorer les quatre cartes avec 3, 5 ou 8 robots. L'approche locale s'avère nettement meilleure avec la seconde chance pour 3 ou 8 robots et s'approche des performances de l'approche globale. Notons que nous mesurons ici la durée nécessaire pour parcourir tout le labyrinthe. Si nous prenons en compte le temps de retour au point de départ des robots (leur permettant notamment de savoir que l'exploration est terminée), la seconde chance peut dans certains cas allonger l'exploration.

4.3 Trajectoires obtenues

La figure 7.14 présente les trajectoires obtenues pour $m = 3$ sur la carte Labyrinthe 1 avec et sans seconde chance.

Avec la seconde chance, le robot vert au centre explore une plus grande partie de la carte et évite ainsi de rester bloqué au centre, en attendant la fin de l'exploration sans y contribuer.

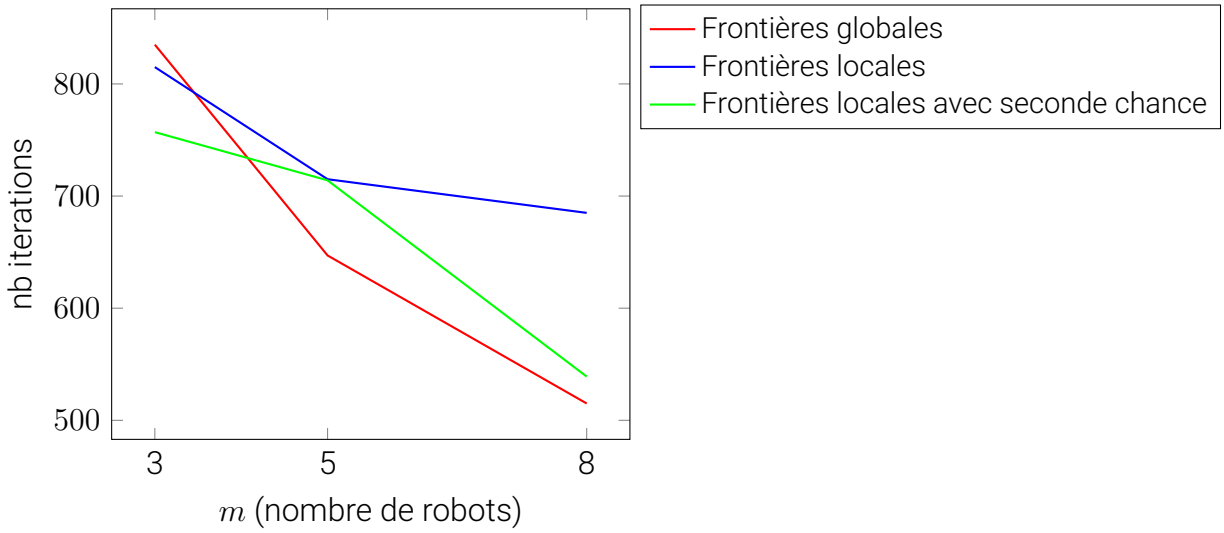


FIGURE 7.13 – Nombre d'itérations nécessaires pour explorer toutes les cartes pour différents nombres de robots. Les robots partent ici groupés au centre des cartes à explorer.

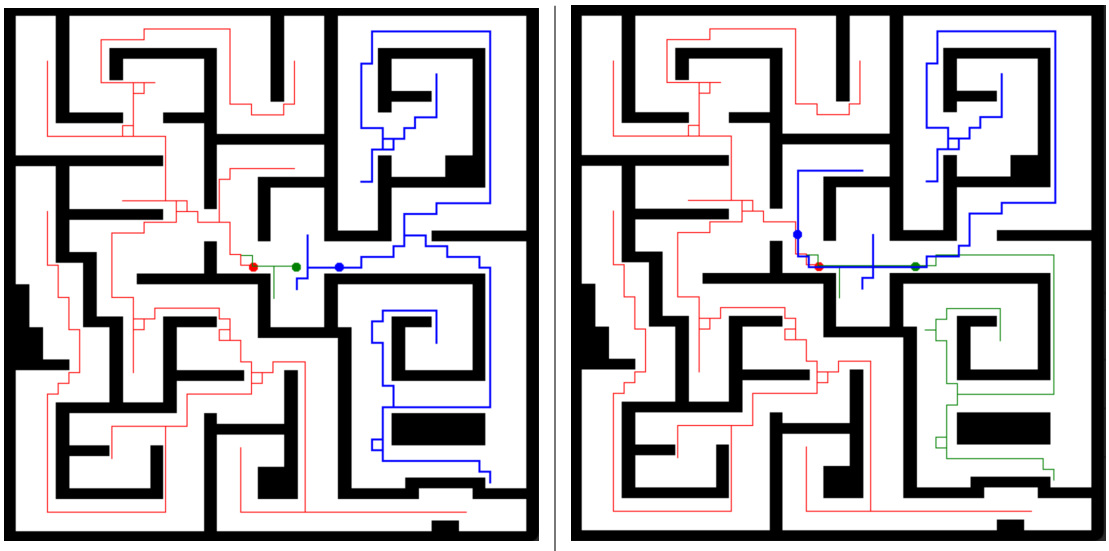


FIGURE 7.14 – Comparaison des trajectoires obtenues avec et sans seconde chance sur la carte Labyrinthe avec trois robots.

5 Frontières locales sans grille

L'approche proposée jusqu'ici se rapproche des algorithmes globaux en terme de performances tout en restreignant les calculs à une zone réduite aux perceptions des robots. De plus, notre algorithme de décision ne nécessite aucune délibération entre les robots, est ne prend en compte que les traces perçues localement.

Cependant, l'algorithme décrit jusqu'ici nécessite de partager l'ensemble de la grille d'occupation $\mathcal{O}$ ainsi que les traces $\mathcal{P}$. Nous allons ici proposer une version beaucoup plus légère, ne nécessitant de partager aucune grille, mais uniquement une et une seule liste de positions successives.

Chaque robot mémorisera maintenant un historique de ses positions successives à un intervalle de distance donné. Cet historique sera partagé avec les autres robots dans un ensemble commun de positions qui ont déjà été occupées par un robot quelconque.

Notons qu'une grille d'occupation est généralement utilisée de façon à cartographier un environnement. Notre algorithme ayant pour objectif de guider les robots de manière à ce qu'ils soient passés par tous les endroits accessibles d'une carte, nous ne considérerons pas de grille d'occupation ici. Si les robots doivent produire une carte de la zone explorée, nous supposerons qu'un autre programme enregistre la grille d'occupation au fil de l'exploration, tandis que notre algorithme se contentera de la prise de décisions.

5.1 Formalisation

Considérons un groupe de m robots $R = \{r_0 \dots r_m\}$, équipés de LiDARs qui perçoivent les obstacles à une distance maximale d. Nous noterons $x_{r_r,t}$ la position du robot r_i au temps t, et h_{r_i} l'historique des positions de ce robot (à un intervalle de distance l).

Tous les robots partagent leur historique de positions régulièrement, et constituent ainsi un ensemble des positions précédemment occupés par tous les robots, noté H :

$$H = \bigcup_{i=0}^{m} h_{r_i}$$

Dans cette nouvelle version, l'historique h_{r_i} est donc la seule donnée émise et reçue par les robots. Nous utiliserons également la distance euclidienne entre deux points notée *dist* :

$$dist(a, b) : \mathbb{R}^2 \times \mathbb{R}^2 \longrightarrow \mathbb{R}.$$

Enfin, nous noterons $\mathcal{P}_{r_i,t}$ l'ensemble des points visibles par le robot r_i au temps t.

Chaque robot choisit un objectif utilisant l'algorithme décrit ci-dessous, se déplace ensuite en direction de cet objectif jusqu'à avoir parcouru une distance l. À chaque fois que la distance l à été parcourue l'algorithme de décision est exécuté à nouveau. La distance l doit être inférieure au rayon de perception d ($l < d$).

L'opération *push* ajoute un élément à la fin d'une liste (notamment l'historique h_{r_i}), tandis que l'opération *pop* récupère et enlève le dernier élément d'une liste.

L'intérêt de cette nouvelle version est qu'elle fourni un résultat équivalent à l'approche sur grille, tout en mémorisant et partageant beaucoup moins d'informations. Dans la version sur grille, on partage la grille d'occupation $\mathcal{O}$, qui est une matrice à trois dimensions *Largeur de la carte* $\times$ *Hauteur de la carte* $\times$ 3. Dans cette version, seuls l'historique des positions des robots et l'ensemble des positions occupées (à un intervalle de distance l) sont mémorisés.

Notons que l'amélioration de seconde chance décrite pour l'approche sur grille peut également facilement être reprise ici. Nous allons maintenant aborder deux autres améliorations de notre approche de façon à améliorer les performances de l'exploration.

5.2 Comparaison des approches

L'approche sans grille fonctionne sur les mêmes principes que l'approche avec grille : dans les deux cas, il s'agit d'avancer dans une direction tant qu'il y a des zones inexplorées visibles depuis notre position, et revenir en arrière lorsqu'il n'y en a plus. La version sans grille évite cependant le stockage inutile d'une grande quantité d'informations, puisque seuls l'historique des positions d'un robot, et le partage de l'historique de toutes les positions précédemment occupés sont nécessaires.

Algorithm 1 Choix de l'objectif du robot r_i au temps t

Les frontières sont constituées des points à la limite du champ de perception du robot qui sont au moins à une distance d de toute trace laissée par un robot quelconque.
$\mathcal{F} \longleftarrow \{p \mid p \in \mathcal{P}_{r_i,t}, \mathit{dist}(p,q) \leq d \; \forall q \in H\}$

S'il y a au moins une frontière disponible
if $\mathcal{F} \neq \emptyset$ **then**
  On ajoute la position actuelle à l'historique des positions
  h_{r_i}.push(x_{r_i})
  H.push(x_{r_i})
  retourEnArrière $\longleftarrow$ faux

  On choisit la frontière la plus éloignée de toute trace du robot.
  objectif $\longleftarrow \underset{p \in \mathcal{F}}{\operatorname{argmax}} \mathit{dist}(p,q) \; \forall q \in H$
  return objectif
else if $x_{r_i,t} \neq x_{r_i,0}$ **then**
  if retourEnArrière $=$ faux **then**
    h_{r_i}.push(x_{r_i})
    H.push(x_{r_i})
    retourEnArrière $\longleftarrow$ vrai
  end if
  S'il n'y a aucune frontière inexplorée, mais que l'on est pas au point de départ, on revient sur la position précédente
  return h_{r_i}.pop()
else
  Sinon, l'exploration est terminée
  return $\emptyset$
end if

Ces approches restent proches des algorithmes très simples tels que *MDFS* d'un point de vue logique, mais tirent parti de la totalité du champ de perception du robot, et non d'une seule cellule d'une grille.

Cette innovation permet de parcourir la carte à la façon d'un algorithme de parcours d'un arbre en profondeur d'abord, mais en construisant dynamiquement cet arbre au fil de l'exploration dans un monde continu. Dans cette représentation, les positions de l'historique du robot constituent les noeuds de l'arbre, et chaque arête correspond à un déplacement de la distance l.

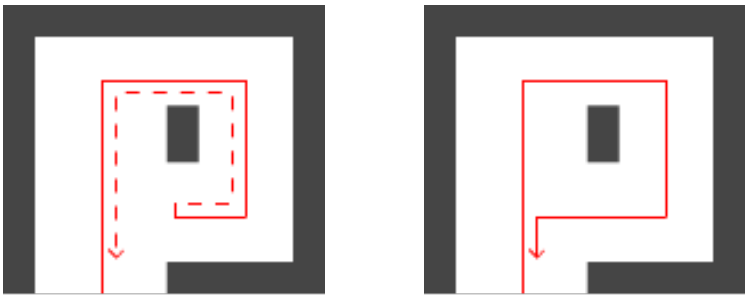
6 Améliorations

Différentes améliorations peuvent améliorer significativement les résultats obtenus par notre approche locale, sans trop complexifier celle-ci pour autant. Dans de nombreux cas, les retours en arrières des robots provoquent des chemins non-optimaux, empêchant les robots de prendre des raccourcis pourtant connus. Un mauvais placement des robots par rapport aux obstacles peut également ajouter des aller-retours inutiles. Les prochaines sous-sections visent à répondre à ces problèmes.

6.1 Retours en arrière

Lorsqu'un robot retourne à une position précédente, il peut revenir sur ses pas pendant un certain temps sans trouver de nouvel objectif, et le chemin vers le prochain objectif peut ne pas être optimal, car le chemin est contraint par ses positions précédentes.

La Fig. 7.15 montre un exemple où le robot, après avoir exploré toute la zone, revient sur ses pas (chemin en pointillés sur la Fig. 7.15 a), alors qu'un chemin plus court était disponible (Fig. 7.15 b).



(a) Chemin de l'algorithme actuel N

(b) Chemin souhaité

FIGURE 7.15 – Comparaison du comportement actuel de l'algorithme lors du retour en arrière avec le comportement souhaité.

Nous aimerions améliorer ce comportement de retour en arrière sans perdre la garantie d'une exploration complète. Si nous comptons à chaque pas de temps le nombre de zones contiguës qui contiennent des points éligibles en tant qu'objectifs, nous pouvons observer que si le robot passe en mode *backtrack*, il restera dans ce mode jusqu'à ce qu'il atteigne la dernière position pour laquelle le nombre de zones contiguës avec des points éligibles a diminué. Nous pouvons stocker la dernière position pour laquelle le nombre de frontières a diminué et la définir comme un objectif lorsqu'il n'y a plus de

frontières disponibles et que le robot n'est pas encore passé par cette position. Cela donne un chemin plus court en revenant en arrière, avec la même idée que dans l'algorithme BoB [108].

Modifications de l'algorithme

Nous allons devoir compter le nombre de zones contiguës de $\mathcal{F}$. Dans notre implémentation, nous utilisons l'algorithme décrit dans [42] sur un espace discret pour déterminer le nombre de zones. Nous supposerons ici que la fonction *compterLesZonesContiguës* calcule ce nombre. Nous noterons $c_{r_i,t}$ le nombre de zones contiguës de $\mathcal{F}$ pour le robot r_i au temps t.

L'idée générale est donc de mémoriser les positions ou le robot à du choisir un passage parmi plusieurs possibilités, de façon à retourner à ces positions lorsqu'il doit revenir sur ces pas.

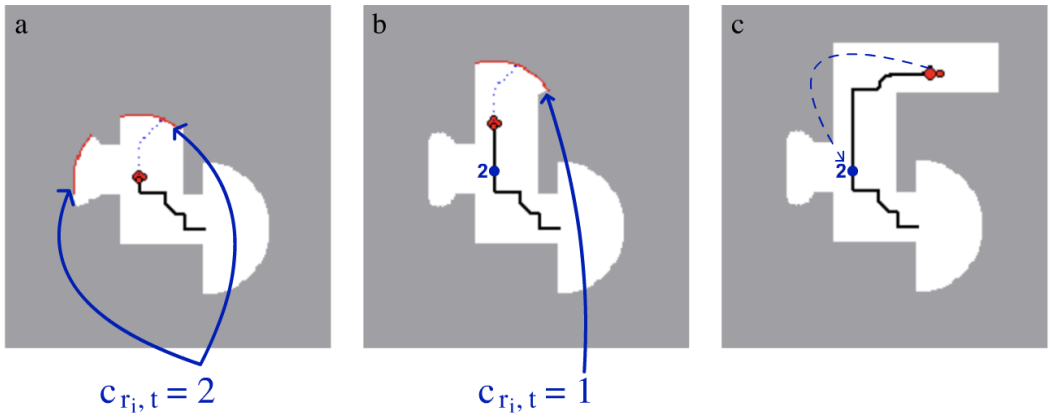


FIGURE 7.16 – Illustration de l'algorithme de *backtrack* optimisé avec l'enregistrement des variations des valeurs de c_{r_i}

Nous allons maintenant considérer une pile q_{r_i} des positions sur lesquelles le robot fera un retour en arrière lorsqu'il n'y a plus de frontière à explorer dans le champ de perception. Une position sera ajoutée à la pile à chaque fois que le nombre de frontières contiguës passe d'un nombre supérieur à un à une seule frontière. Chaque position de q_{r_i} correspond donc à une position ou le robot avait le choix entre plusieurs zones à explorer, et est parti explorer l'une d'entre-elles, ce qui signifie que le robot devra revenir à cette même position pour explorer les zones restantes.

La figure 7.16 illustre le fonctionnement du nouvel algorithme. Dans la vignette *a*, le robot est face à deux frontières contiguës distinctes, donc $c_{r_i} = 2$. Dans la vignette *b*, il n'y a plus qu'une frontière contiguë ($c_{r_i} = 1$), la dernière position ayant eu plusieurs frontières est donc ajoutée à la pile q_{r_i} (représentée par le point « 2 »en bleu sur la vignette *b*). Dans la vignette *c*, le robot n'ayant plus de zone à explorer autour de lui, peut revenir en arrière jusqu'au dernier point de la pile q_{r_i}, en ignorant les traces intermédiaires. De cette façon, un éventuel raccourci vers cette position peut être emprunté, la pile q_{r_i} garantissant qu'aucun point intermédiaire ne menait à une zone inexplorée.

L'algorithme ci-dessous présente la nouvelle fonction de choix d'un objectif, mémorisant les positions où le nombre de frontières à diminué dans la pile q_{r_i}, et utilisant directement cette pile pour déterminer les retours en arrière, plutôt que la trace du robot h_{r_i}.

Algorithm 2 Choix de l'objectif du robot r_i au temps t

Les frontières sont constituées des points à la limite du champ de perception du robot qui sont au moins à une distance d de toute trace laissée par un robot quelconque.
$\mathcal{F} \longleftarrow \{p \mid p \in \mathcal{P}_{r_i,t}, \mathit{dist}(p,q) \leq d \; \forall q \in H\}$

On compte le nombre de frontières contiguës à la position actuelle
$c_{r_i,t} \longleftarrow \mathit{compterLesZonesContiguës}(\mathcal{F})$

S'il n'y a qu'une seule frontière contiguë alors qu'il y en avait plusieurs à l'itération précédente, on l'ajoute comme position de backtrack.
if $c_{r_i,t} = 1$ **and** $c_{r_i,t-1} > 1$ **then**
  q_{r_i}.push($x_{r_i,t-1}$)
end if

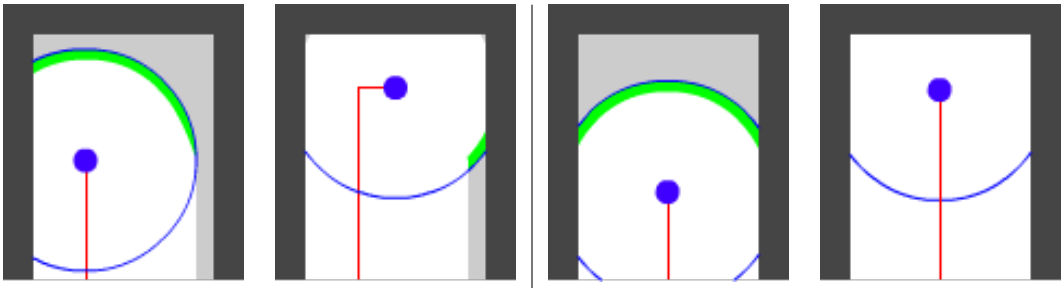
S'il y a au moins une frontière disponible
if $\mathcal{F} \neq \emptyset$ **then**
  On ajoute la position actuelle à l'historique des positions
  h_{r_i}.push(x_{r_i})

  On choisit la frontière la plus éloignée de toute trace du robot.
  objectif $\longleftarrow \underset{p \in \mathcal{F}}{\operatorname{argmax}} \; \mathit{dist}(p,q) \; \forall q \in H$
  return objectif
  Sinon, si la pile q_{r_i} n'est pas vide, on revient à la dernière zone où il y avait plusieurs frontières contigues
else if $|q_{r_i}| > 0$ **then**
  return q_{r_i}.pop()
else
  Sinon, l'exploration est terminée, on peut revenir au point de départ
  return $x_{r_i,0}$
end if

Dans cette nouvelle version, les traces h_{r_i} servent toujours à éviter d'explorer plusieurs fois la même zone et disperser les robots, tandis que la nouvelle pile q_{r_i}, non partagée, sert à revenir en arrière de façon optimale.

6.2 Distance aux obstacles

Lorsqu'un robot est trop proche ou trop loin d'un mur par rapport à son champ de perception, il peut être amené à faire plusieurs allers-retours dans un couloir alors qu'un seul passage aurait pu couvrir toute la zone. La Fig.7.17 montre le cas d'un robot trop proche du mur de gauche, laissant une zone inexplorée lors de son premier passage *(a, b)*, alors qu'un seul passage aurait suffi s'il était à une distance mieux choisie *(c, d)*. Évidemment, il est préférable de positionner le robot à une distance des murs proche mais inférieure à la distance de perception maximale.



(a) t = 5 (b) t = 8 (c) t = 5 (d) t = 8

FIGURE 7.17 – Exploration d'un couloir avec un robot près d'un mur (a, b) ou à égale distance des deux murs latéraux (c, d)

Pour optimiser la trajectoire des robots, nous allons préférer, lors du choix d'un but, les points à une distance idéale des murs. Nous choisirons arbitrairement l'intervalle $[\lambda d, d[$ pour définir une distance « proche mais inférieure »aux murs. On note $\mathcal{W}_{r_i,t}$ l'ensemble de toutes les positions d'obstacles visibles par le robot r_i au temps t.

Algorithme mis à jour

Pour maintenir les robots à une distance optimale des murs, nous pouvons modifier la première règle de notre algorithme, qui consiste à préférer les points qui sont à une distance idéale des murs, en préférant les points à une distance des murs dans l'intervalle idéal $[\lambda d, d[$ (avec λ proche de 1, $\lambda < 1$).

On notera $\mathcal{G}$ les frontières à une distance idéale des murs, c'est à dire à une distance d'un obstacle dans l'intervalle $[\lambda d, d[$.

Algorithm 3 Choix de l'objectif du robot r_i au temps t

Les frontières sont constituées des points à la limite du champ de perception du robot qui sont au moins à une distance d de toute trace laissée par un robot quelconque.
$\mathcal{F} \longleftarrow \{p \mid p \in \mathcal{P}_{r_i,t}, \mathit{dist}(p,q) \leq d \; \forall q \in H\}$

On compte le nombre de frontières contiguës à la position actuelle
$c_{r_i,t} \longleftarrow \mathit{compterLesZonesContiguës}(\mathcal{F})$

S'il n'y a qu'une seule frontière contiguë alors qu'il y en avait plusieurs à l'itération précédente, on l'ajoute comme position de backtrack.
if $c_{r_i,t} = 1$ **and** $c_{r_i,t-1} > 1$ **then**
  q_{r_i}.push($x_{r_i,t-1}$)
end if

S'il y a au moins une frontière disponible
if $\mathcal{F} \neq \emptyset$ **then**
  On calcule les frontières prioritaires, c'est à dire à une distance idéale d'un obstacle
  $\mathcal{G} \longleftarrow \{p \mid p \in \mathcal{F}, \mathit{dist}(p,w) \in [\lambda d, d[\; \forall w \in \mathcal{W}_{r_i,t}\}$
  if $\mathcal{G} \neq \emptyset$ **then**
    Si des frontières prioritaires existent, on choisit parmi elles
    objectif $\longleftarrow \underset{p \in \mathcal{G}}{\operatorname{argmax}} \; \mathit{dist}(p,q) \; \forall q \in G$
  else
    Sinon, on choisit parmi toutes les frontières
    objectif $\longleftarrow \underset{p \in \mathcal{F}}{\operatorname{argmax}} \; \mathit{dist}(p,q) \; \forall q \in H$
  end if
  On ajoute la position actuelle à l'historique des positions
  h_{r_i}.push(x_{r_i})
  return objectif
  Sinon, si la pile q_{r_i} n'est pas vide, on revient à la dernière zone où il y avait plusieurs frontières contigues
else if $|q_{r_i}| > 0$ **then**
  return q_{r_i}.pop()
else
  Sinon, l'exploration est terminée, on peut revenir au point de départ
  return $x_{r_i,0}$
end if

7 Comparaison des version sur grille et continue

La nouvelle version de l'algorithme (continue) nécessite un partage d'informations beaucoup moins important que la version initiale tout en proposant une efficacité équivalente. L'amélioration des retours en arrière et le choix d'une distance aux obstacles mieux adaptée permettent également une amélioration de la vitesse de l'exploration.

La figure 7.18 compare les trajectoires obtenues par les versions discrètes et continues de l'algorithme proposé dans ce chapitre.

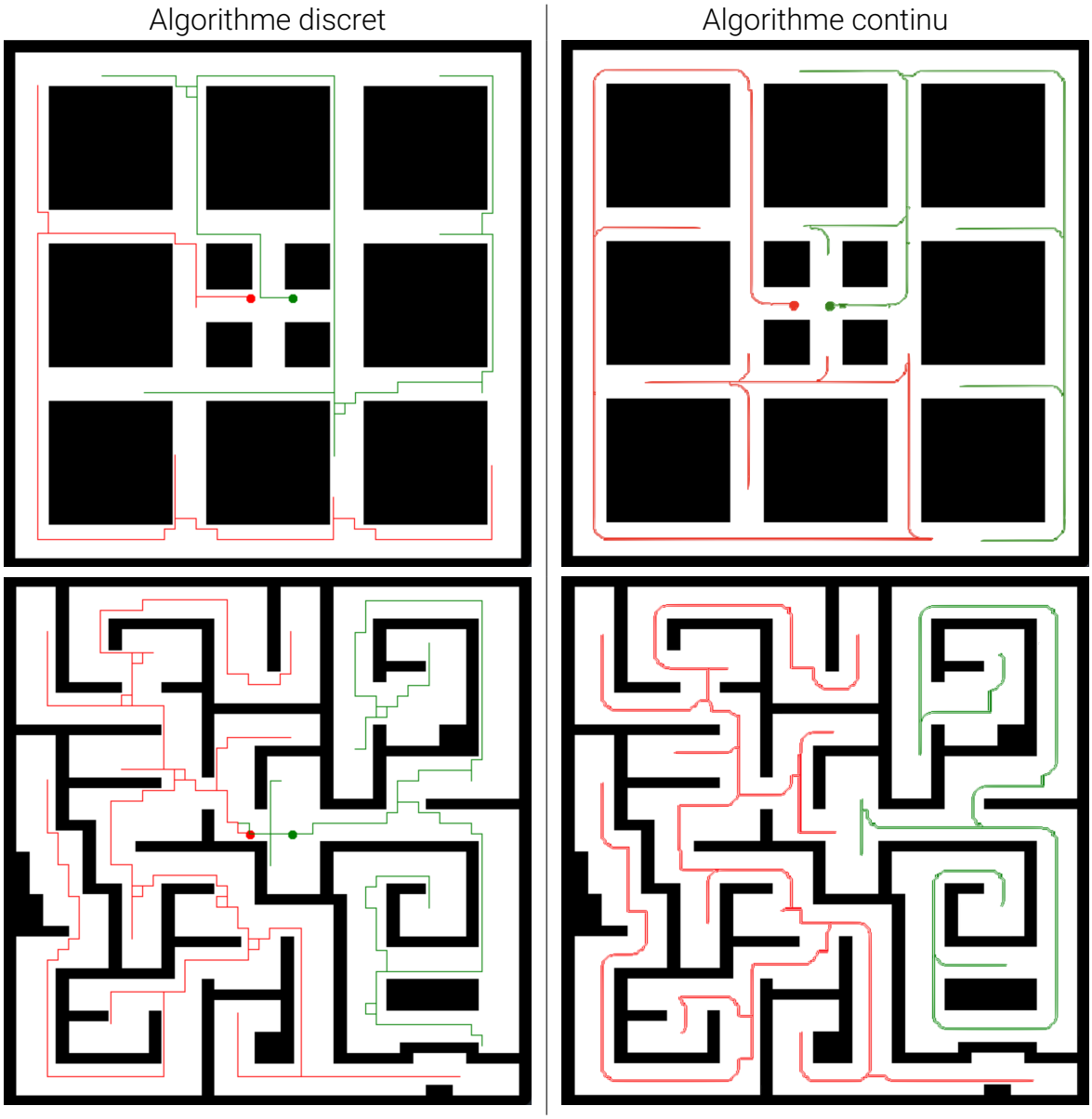


FIGURE 7.18 – Comparaison des trajectoires obtenues avec les approches discrètes et continues de l'algorithme « Frontières Locales »proposé dans ce chapitre.

8 Conclusion

8.1 Forces et faiblesses des différents algorithmes

L'approche *BMILRV* est locale, mais a un fonctionnement complexe (de nombreux états pour les robots, la fermeture des cellules nécessite de nombreux calculs [102]) et donne de mauvais résultats lorsque des obstacles sont isolés ou qu'il y a des boucles sur la carte, forçant les robots à faire plusieurs passages pour marquer les cellules successivement comme contrôlées, fermées et finalement nettoyées (comme le confirment les expériences sur la figure 7.7). A notre connaissance, cette approche n'a pas été adaptée pour une mise en œuvre décentralisée.

L'approche par frontières (globales) est à la fois simple et efficace. Elle peut être distribué facilement, mais nécessite une synchronisation fréquente de la carte et des positions du robot. Avec la stratégie d'affectation *MinPos*[89], chaque robot prend en compte la position de tous les autres robots pendant chaque affectation, ce qui implique que la carte des frontières globales et la position du robot sont fréquemment partagés. De plus, le temps de calcul de cette stratégie d'affectation dépend du nombre de robots et du nombre de frontières (les distances entre toutes les paires frontière/robot sont calculées à chaque itération), ce qui peut rendre cette méthode inutilisable si le nombre de robots est trop grand, ou si la carte est trop grande. Des stratégies sans affectations optimisées existent également, mais nécessitent toujours le partage d'un grand nombre d'informations entre les robots (notamment la grille d'occupation).

Enfin, l'approche par frontières locales présente des résultats proches de l'approche globale dans la plupart des cas, tout en ayant une complexité constante, ne dépendant ni du nombre total de robots ni de la taille de la carte. Cette approche est également facilement distribuable sans contrainte sur la fréquence de partage des cartes, et les robots n'ont pas besoin de connaître le nombre ou la position actuelle des autres robots. Sans l'ajout de la « seconde chance », cependant, certains robots peuvent s'arrêter avant la fin de l'exploration, réduisant ainsi la performance de l'exploration.

La version continue de cette approche permet d'économiser plus encore sur les informations partagées entre les robots, réduisant celles-ci à une simple liste de positions précédemment occupées à un intervalle de distance régulier. L'amélioration apportée sur la distance aux obstacles permet d'améliorer la vitesse de l'exploration en évitant les allers-retours inutiles. Enfin, l'amélioration des retours en arrière permet de se rapprocher encore un peu plus des performances des approches globales utilisant efficacement d'éventuels raccourcis lors des retours en arrière.

8.2 Possibilités d'amélioration

La version de l'algorithme décrite ici ramène au point de départ des robots qui ne trouvent plus de frontières à explorer. Selon la carte et les conditions initiales, un robot peut revenir à son point de départ et s'arrêter avant que la carte entière ne soit explorée (laissant les autres robots terminer l'exploration). La distribution entre les robots est alors moins équilibrée et les performances de l'algorithme sont réduites. Pour éviter cela, nous pouvons envisager un système de « seconde chance » (voir partie 4), permettant aux robots qui sont revenus à leur point de départ avant la fin de l'exploration de recommencer en suivant la trace de l'un des autres robots jusqu'à ce qu'il trouve une frontière.

La dispersion des robots est également rudimentaire et ne repose que sur le choix

d'une cible aussi loin que possible de toute trace laissée. Lorsque plusieurs robots sont proches (dans leurs rayons de perception respectifs), et en particulier si d_{sync}est élevé, l'ajout d'un système de dispersion supplémentaire pourrait également augmenter de manière significative les performances de l'algorithme sans augmenter considérablement sa complexité.

8.3 Conclusion

Dans la version proposée ici, l'algorithme donne de bonnes performances (nombre d'étapes pour compléter la couverture) par rapport aux deux autres approches, et au bénéfice d'une complexité constante (en temps et en mémoire) pour un robot donné, ce qui permet d'utiliser un très grand nombre de robots. En effet, dans le cas de l'approche par frontières globales, les stratégies les plus efficaces, telles que *MinPos*, nécessitent de calculer autant de distances que de paires robot/frontière, ce qui peut poser problème s'il y a un trop grand nombre de robots ou si la carte est trop grande. De plus, notre approche locale est plus facilement distribuable au prix d'une perte de performance possible lorsque les robots sont proches, et ne nécessite pas de synchronisation temporelle entre les robots.

Ces résultats montrent également qu'il existe des approches intermédiaires entre les algorithmes locaux et les algorithmes globaux, pouvant prendre en compte les cellules de la grille dans un rayon de perception déterminé et garantissant une exploration complète.

Notre approche est également très économe en calculs, mémoire et partage d'informations, répondant ainsi parfaitement à nos objectifs de développer une approche efficace, mais très peu coûteuse en calculs et communications.

Cependant, les algorithmes décrits dans ce chapitre ne tiennent compte que des obstacles les entourant, sans chercher à exploiter l'architecture des bâtiments qu'ils doivent explorer. Le prochain chapitre s'intéressera à d'autres approches visant à donner un sens aux obstacles perçus pour mieux diriger l'exploration.

Chapitre 8

Exploitation de données sémantiques

Sommaire

1 Introduction

Les algorithmes proposés jusqu'ici ont exploité les données récoltées par les LiDARs pour établir la position des obstacles dans les environnements des robots et prendre des décisions en tenant compte de ces derniers.

Cette approche possède cependant plusieurs lacunes. Pour commencer, ces données sont exclusivement métriques, et donc sensibles à de nombreuses erreurs, notamment lors de la dérive des capteurs et d'erreurs de localisation. Comme nous l'avons vu, il peut être difficile pour un robot équipé d'un LiDAR de bien identifier sa progression dans un couloir sans repère fixe lui permettant d'estimer précisément une éventuelle dérive de capteurs. Dans ce type de situation, une commande du type « aller au fond du couloir puis tourner à droite »serait ainsi moins propice aux erreurs qu'un ordre du type « avancer de 10,5m puis tourner de 90 degrés en sens horaire ». Comprendre ce qu'est un couloir ou une salle est alors un atout précieux, autant pour se localiser que pour communiquer ou diriger un robot vers un objectif. Une représentation plus abstraite du monde, bien que moins détaillé qu'une carte métrique, peut ainsi offrir une meilleure robustesse aux variations du monde (robot glissant sur une surface, obstacle imprévu, etc.), tout en étant plus propice aux échanges avec les êtres humains.

De plus, dans le cadre de l'exploration d'environnement intérieur ou urbain, la structure des obstacles n'est pas aléatoire, et obéit à un ensemble de règles structurelles spécifiques au type de lieu et à l'architecture. Par exemple, un grand bâtiment possède généralement de grands couloirs permettant de parcourir les grands axes de ce dernier. Dans une mission de recherche et de sauvetage, tirer parti de cette particularité peut ainsi permettre un gain de temps précieux. Si un robot doit aller à l'autre bout d'un grand bâtiment, il sera ainsi plus avisé de chercher de grands couloirs allant dans la bonne direction, que de visiter une à une les petites salles pouvant se trouver sur son passage. Comprendre la structure d'un bâtiment ou des rues d'une ville peut ainsi permettre d'améliorer significativement la prise de décision en tenant compte des règles générales de conception de ces derniers.

Enfin, l'exploration multirobots est également souvent soumise à de fortes contraintes, notamment en terme de bande passante pour la communication entre les robots du système. Dans cette optique, le partage de données de plus haut niveau, par exemple l'identification de salles, avec les portes les reliant, leur forme et leur superficie, permet d'économiser significativement la quantité de données à échanger, par rapport à un partage de grille d'occupation ou de données métriques. Le travail réalisé jusqu'ici a été motivé par ce type de contrainte, notamment en proposant un algorithme (présenté chapitre 7. Exploration par stigmergie) échangeant uniquement un historique de positions, et non une grille d'occupation. Cependant, restreindre les communications entre les robots n'est pas la seule contrainte à laquelle doivent faire face des entreprises comme Safran dans le cadre de missions d'exploration en situation réelle. En effet, géolocaliser les robots de manière précise et dans un référentiel global est également difficile à mettre en œuvre, et les algorithmes proposés jusqu'ici nécessitent justement une localisation de tous les robots dans un repère commun pour simuler correctement le dépôt et la perception de traces par les différents robots. Construire une carte sémantique de données de plus haut niveau peut ainsi permettre de mieux situer la position des robots, en s'abstrayant des données métriques et des obstacles.

Les objectifs de ce chapitre sont ainsi multiples : inférer des données sémantiques de plus haut niveau, et exploiter ces dernières de façon à améliorer la prise de décision, réduire la quantité de données partagées entre les robots, permettre de prendre des

décisions en tenant compte de la structure des lieux explorés, et limiter les problèmes de localisation et la nécessité d'un géoréférencement global des robots.

1.1 Contraintes opérationnelles

Les approches proposées jusqu'ici sont totalement autonomes et non guidées, dans le sens où les robots choisissent des objectifs selon des critères algorithmiques qui leur sont propre, sans prendre en compte d'éventuelles contraintes où indications extérieures. Dans de nombreuses situations, telles que la recherche de victimes dans un bâtiment en feu où l'appui d'un groupe de combattants, avoir la possibilité de contraindre les robots peut être primordial. Par exemple dans un incendie, certaines zones peuvent être à explorer en priorité, tandis que d'autres pourraient être à éviter ; dans un cadre militaire, des contraintes d'observabilité, de navigation où encore de discretion peuvent entrer en jeu. Il pourrait par exemple être souhaitable que certaines régions soient toujours observées par au moins un robot, où encore de limiter le nombre de robots passant par des zones trop visibles pour assurer leur discrétion.

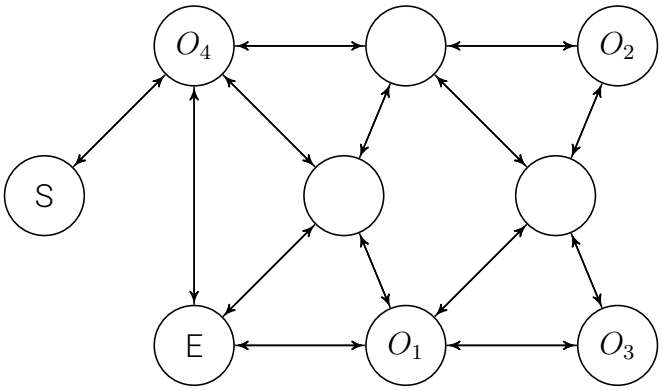


FIGURE 8.1 – Exemple de plan de navigation et demandes d'observation des utilisateurs. Les robots doivent maximiser les observations en limitant la durée de l'exploration et l'énergie consommée, inspirée de [110].

Supposons par exemple un contexte milliaire, où des robots doivent explorer différentes zones dans une mission de reconnaissance. Ces derniers sont à la fois limités en ressources énergétiques et par le temps de mission et la structure du terrain. La figure 8.1 donne un ensemble de points à observer $\{O_1, O_2, O_3, O_4\}$. Ces derniers sont définis au cours de la préparation de la mission par l'analyse du terrain, l'évaluation de la situation et les objectifs de la mission. Des contraintes de navigation sont également définies par les passages possibles, déterminés par l'ordre du commandement. Sur cet exemple de plan, les robots partent d'une position initiale S (contexte de défense, le robot s'expose à la visibilité de l'adversaire et l'on cherche à minimiser l'observabilité de la mission du robot) et doivent être récupérés en fin de mission (en E). De façon à minimiser l'énergie et l'exposition du robot, l'utilisateur (humain) décide de planifier des actions d'observation en $\{O_1, O_2, O_4\}$. L'objectif est alors d'effectuer l'exploration en respectant les différentes contraintes données par les opérateurs humains préalablement à la mission.

1.2 Assignation sémantique

Nous allons proposer une nouvelle approche visant à produire une carte sémantique de l'environnement au fil de l'exploration, et déterminer à partir de celle-ci une assignation pour les différents robots selon différents critères sémantiques. Ainsi, nous pourrons par exemple privilégier les couloirs aux salles closes.

Nous verrons ensuite comment adapter l'approche par frontières locales vue dans le chapitre précédent pour prendre en compte la carte sémantique et utiliser la nouvelle méthode d'assignation sémantique.

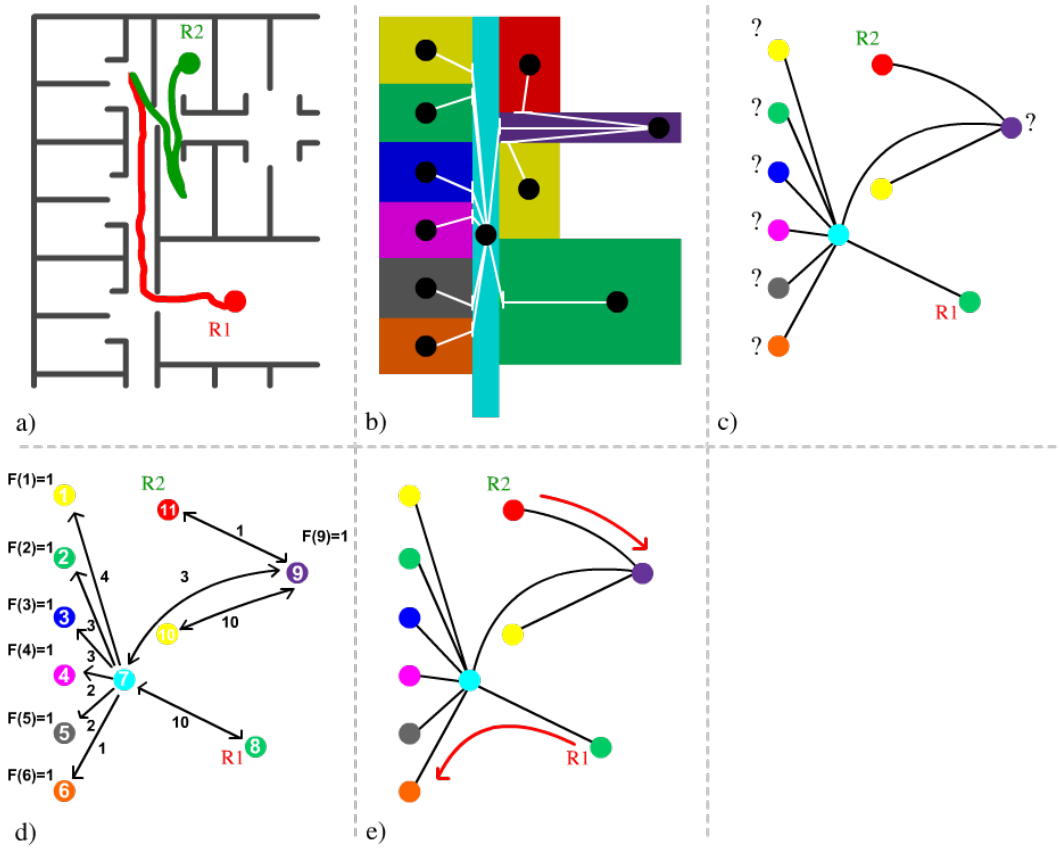


FIGURE 8.2 – Étapes des l'approche développée dans ce chapitre.

Pour assigner les robots à partir d'une carte sémantique, nous procéderons en 4 étapes illustrées figure 8.2 :

1. Au fil de l'exploration, on produit une carte sémantique discriminant les salles et couloirs. La figure 8.2.a présente les trajectoires des robots dans la carte et la figure 8.2.b la carte sémantique, qui segmente les salles et identifies les portes qui les relient. Notons que cette carte est mise à jour au fil de l'exploration, et que les salles identifiées sont ainsi susceptibles de changer au fil de l'exploration.
2. Nous construirons alors un graphe à partir de la carte sémantique, dans lequel les robots sont localisés dans cette représentation sémantique (b, c)
3. Nous pourrons alors définir des contraintes sur ce graphe. L'objectif est ici de définir là où il faudra des robots (zones inexplorées), un coût sur les différentes arrête selon l'affectation souhaitée (par exemple, privilégier les couloirs, réduire les distances parcourues, etc), etc (c, d).

4. À partir de ce graphe sous contraintes, on produit enfin une affectation des robots. Ces derniers pourront alors naviguer vers leur objectif jusqu'à ré-assignation (d, e).

L'intérêt de ce nouveau découpage est de pouvoir choisir différentes contraintes à l'étape 3 selon le résultat visé, sans avoir à changer le reste de l'algorithme. Ainsi, si d'autres informations sémantiques sont à prendre en compte, comme la présence d'objets dans certaines salles, où si des missions spécifiques doivent être menées, comme regrouper les robots à un point donné, seule l'étape 3 sera modifiée.

Nous verrons comment produire la carte sémantique (étape 1) dans la partie 2. Identification des pièces et portes. La partie 4.3. Données locales et globales introduira la modélisation du problème et sa résolution (étapes 2, 3 et 4). Nous adapterons la méthode par frontières locales à cette nouvelle assignation dans la partie 5 Frontières locales avec contraintes sémantiques. Enfin, nous verrons différents choix de contraintes (pour l'étape 3) dans la partie 6 Objectifs et missions.

2 Identification des pièces et portes

De façon à pouvoir exploiter et échanger des données de plus haut niveau, il est avant tout nécessaire d'inférer et déterminer ces données. Le premier objectif est donc de construire une carte sémantique au fil de l'exploration, de façon à identifier les salles, les portes et les couloirs, à partir des données métriques perçues par les robots. Dans cette première partie, nous allons donc construire un plan sémantique à partir d'une grille d'occupation construite au cours de l'exploration à l'aide du LiDAR d'un robot.

2.1 Découpage de l'environnement

Pour découper l'environnement en pièces et identifier les portes ou passages permettant de naviguer entre les différentes salles, j'ai développé un algorithme simple utilisant comme entrée une grille d'occupation. Cet algorithme procède en plusieurs étapes présentées figure 8.3 et décrites ci-dessus.

À partir des données locales du LiDAR, le robot construit une grille d'occupation de l'environnement, puis procède ainsi :

- Dilatation des obstacles (cellules occupées) de la grille d'occupation. La taille de la dilatation est choisie de façon à «fermer »les portes (images 2 et 3 de la figure 8.3).
- On utilise ensuite un algorithme de remplissage par diffusion («flood-fill ») [8] [1] de la façon suivante :
    1. Choisir une nouvelle couleur
    2. Appliquer l'algorithme de remplissage par diffusion sur une cellule inoccupée de la grille (choisie au hasard).
    3. Recommencer tant qu'il existe au moins une cellule libre non-colorée dans la grille d'occupation avec obstacles dilatés.
- Pour toutes les cellules dilatées (c'est à dire occupées dans la grille d'obstacles dilatés (3), mais libres dans la grille d'occupation non dilatées (2), les réaffecter à la couleur la plus proche (parmi les couleurs de l'étape précédente).

1. Page Wikipedia https ://fr.wikipedia.org/wiki/Algorithme_de_remplissage_par_diffusion

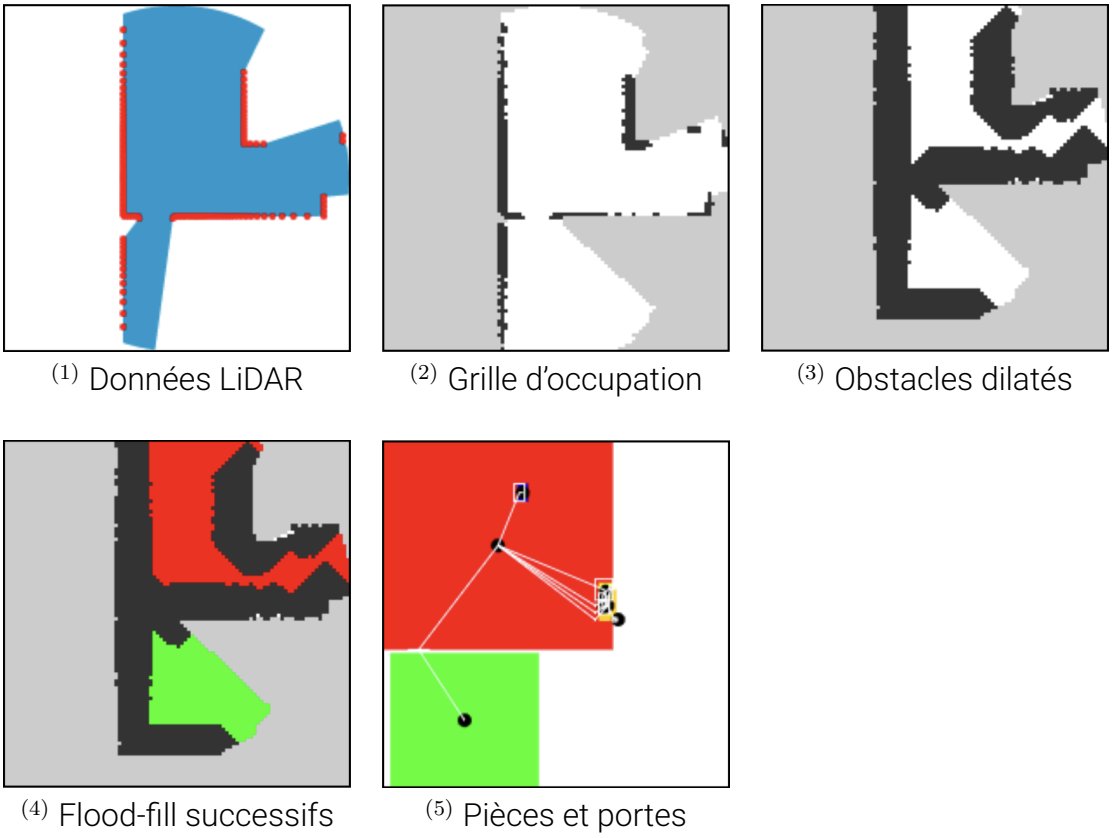


FIGURE 8.3 – Étapes de l'algorithme de segmentation des pièces

— Toutes les cellules d'une couleur donnée ayant une voisine d'une autre couleur (hors obstacles) sont des «portes »(au sens de passage d'une pièce à l'autre).

2.2 Extraction de la structure

L'algorithme décrit ci-dessus permet ainsi d'identifier les différentes salles, ainsi que les portes entre celles-ci. Une fois les salles identifiées, différentes informations peuvent alors être déterminées, telles que les dimensions et la superficie des différentes salles. De façon à mieux comprendre l'environnement, il est aussi possible de classifier les salles en différents types.

2.3 Discrimination des salles

Pour exploiter au mieux les informations sémantiques, nous regrouperons les salles identifiées en différents types selon plusieurs critères, incluant leurs dimensions, le nombre de portes et leur superficie.

Nous considérerons les types de salles suivants :

— Inconnue, valeur par défaut, correspondant à une zone partiellement explorée

— Couloir, correspondant à une salle avec plusieurs portes, pouvant être subdivisé en plusieurs sous-types :

  — Couloir strict, contenant plusieurs portes, dont la longueur est au moins égale à deux fois la largeur, et la largeur inférieure à 3 mètres

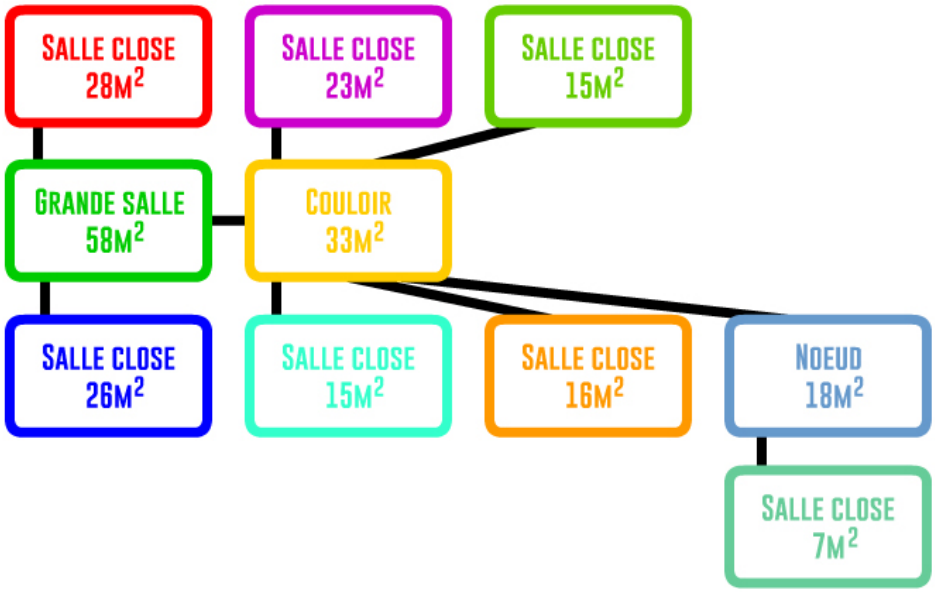


FIGURE 8.4 – Représentation du schéma de salles extrait dans la carte de l'appartement. Notons que les superficies indiquées ici correspondent aux surfaces accessibles au sol, et non aux superficies réelles des pièces.

  - nœud, correspondant à une salle ayant deux portes, permettant le passage entre deux salles
  - Grande salle, soit une salle contenant plusieurs portes et ne répondant pas à la définition d'un nœud ou d'un couloir strict
- Salle close, salle n'ayant qu'une seule porte

Les algorithmes précédemment expliqués permettent de connaître les dimensions, la superficie et le nombre de portes de chaque salle, discriminer ces dernières dans les types et sous-types est ainsi aisé. Cependant, il est important de noter que toutes ces étapes sont réitérées au fil de l'exploration, et donc que les salles seront d'abord étiquetées comme « inconnue », et pourront changer de type lorsque leur exploration sera plus avancée.

2.4 Expérimentation sur simulateur

L'algorithme de segmentation de l'environnement a été implémenté sur le simulateur 3D. Certains paramètres, tels que la distance de dilatation des obstacles doivent être adaptés selon la taille des portes et les obstacles rencontrés. Dans la plupart des cas, la carte et le graphe de navigation obtenus correspondent bien aux résultats attendus. La figure 8.5 montre la carte obtenue sur un appartement.

Les informations ainsi obtenues peuvent permettre d'améliorer l'algorithme d'exploration de nombreuses façon, en tenant compte de la structure du bâtiment qu'il est en-train d'explorer. Identifier les couloirs est également aisé, en considérant les dimensions d'une pièce (écart significatif entre la longueur et la largeur) et du nombre de portes dans une salle donnée.

Notons cependant que la carte obtenue est généralement incorrecte à de nombreux instants au fil de l'exploration : une pièce partiellement explorée peut être identifiée à tort comme plusieurs salles lorsque des obstacles importants s'y trouvent et que ces

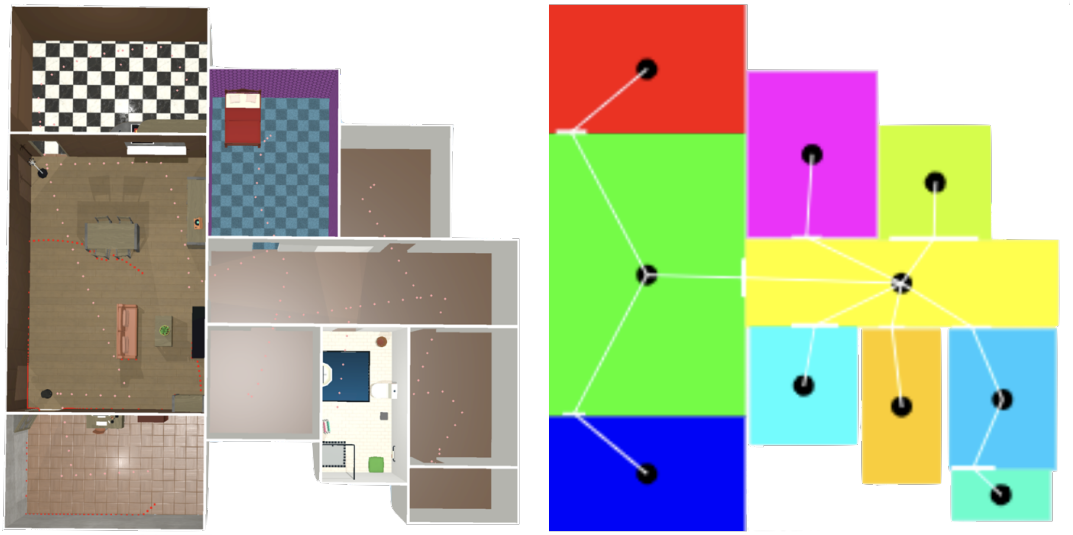


FIGURE 8.5 – Résultat obtenu sur le simulateur 3D développé en utilisant la technique de segmentation de l'environnement décrite précédemment. L'appartement est présenté à gauche, et la carte obtenue à droite. Chaque couleur correspond à une pièce, et les traits blancs les relient en passant par les portes.

derniers n'ont pas été contournés. La carte sémantique (des salles et portes) se corrige ainsi au fil de l'exploration jusqu'à correspondre aux données réelles.

3 Partage des informations sémantiques

De façon à pouvoir regrouper les robots sur une représentation sémantique globale de l'environnement, nous supposerons ici que les robots sont équipés d'une boussole ou d'une centrale inertielles suffisamment précise pour pouvoir orienter correctement les différentes cartes dans un repère commun. Différentes façons de partager la carte sémantique entre les robots sont possibles, avec différentes facilités d'implémentations et différents avantages. Nous verrons ici deux façons de partager la carte sémantique entre les robots.

3.1 Partage de la grille d'occupation

Une des façons les plus simples de procéder est de partager la grille d'occupation, ce qui permet alors à chaque robot de construire la même carte sémantique, tout en ayant la totalité de la carte représentée.

Cette approche nécessite de partager une grande quantité d'informations, la totalité de la grille d'occupation, comme dans le cas des approches par frontières classiques. Dans le cadre de cette thèse, nous tentons de minimiser la quantité de données échangées entre les robots, et nous allons donc éviter cette approche.

3.2 Partage de la carte sémantique

Il est également possible de ne partager que la carte sémantique, permettant ainsi une communication très réduite (il ne s'agit alors que d'une liste de salles correspondant

à un type, une superficie et des dimensions, et une liste de portes, avec les salles reliées et la position relative de la porte).

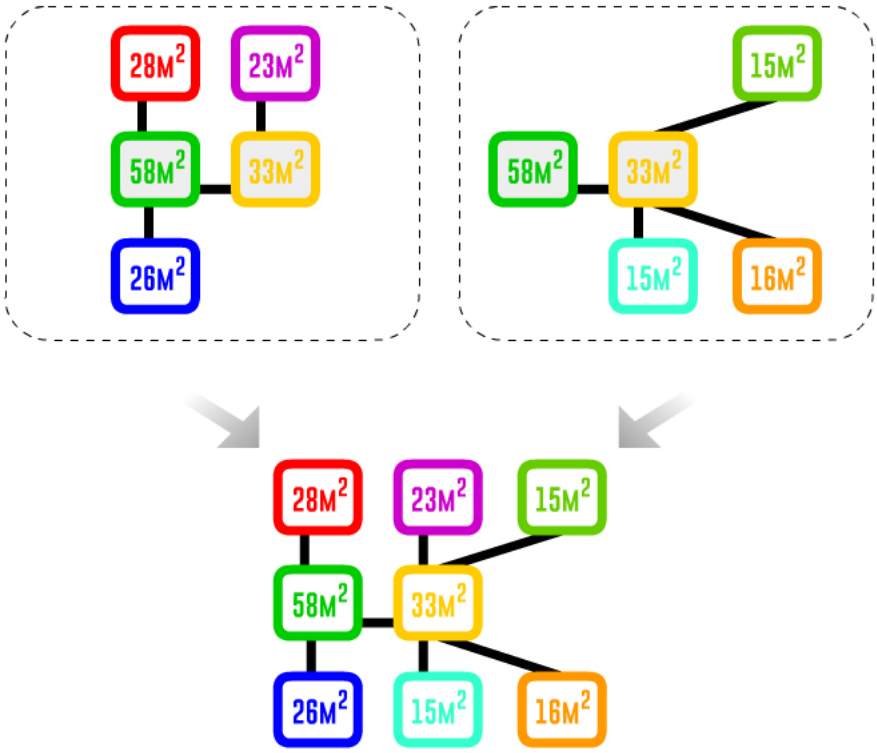


FIGURE 8.6 – Exemple de fusion de cartes de deux robots (en haut), de façon à ne former qu'une seule carte (en bas), en fusionnant les cartes à partir des informations communes.

Dans ce cas, il est nécessaire de fusionner les différentes cartes sémantiques en une seule carte. Si les robots connaissent leurs positions respectives dans un référentiel commun, la fusion est alors triviale (ils connaissent leur salle de départ commune, à partir de laquelle ils peuvent replacer l'ensemble des autres selon les positions des portes).

3.3 Partage sans géoréférencement global

Dans de nombreux cas, il n'est pas possible d'avoir une localisation précise des différents robots dans un référentiel global. Supposons maintenant que chaque robot connaît sa position dans la carte sémantique (salle dans laquelle il se trouve et position dans cette salle), et partage ces informations avec les autres robots.

Dans cette hypothèse, si l'on parviens à fusionner correctement les différentes cartes sémantiques, les robots peuvent alors déterminer efficacement la position des autres robots, sans données métriques précises échangées, ni quantité importante d'informations partagées. La figure 8.6 donne un exemple de fusions de deux cartes sémantiques contenant deux salles communes.

4 Assignation sémantique

Nous allons maintenant voir comment assigner les robots à différentes salles de la carte sémantique en tenant compte de différentes contraintes. Pour cela, nous allons utiliser une représentation par réseau de flot.

4.1 Réseau de flot

Les réseaux de flots, issus de la théorie des graphes, correspondent à des graphes orientés où chaque arête possède une capacité et peut recevoir un flot. [111] Les réseaux de flots peuvent notamment être utilisés pour modéliser le trafic dans un réseau routier, des flux de fluides dans des conduits, etc. Ces graphes sont souvent utilisés en recherche opérationnelle, permettant de modéliser efficacement différents problèmes à l'aide d'un ensemble de contraintes [12, 110]. Pour qu'un flot soit valide, une règle de conservation du flot permet de contraindre ce dernier, de façon à ce que la somme des flots sortants d'un nœud soit égale à la somme des flots entrants. Il est possible de déroger à cette règle avec la notion de nœud source (sans flot entrant, induisant un flot sur le graphe) et de nœud puits (sans flot sortant, absorbant le flot). Il est également possible de contrainte de flot du graphe avec une notion de capacité, de façon à définir la quantité maximale pouvant traverser une arête. Nous aborderons également une notion de coût, permettant à un flot de traverser différentes arêtes, mais incluant un coût selon la quantité traversant chaque arête, et permettant ainsi de calculer un coût total du flot, discriminant alors les différents flots possibles pour un même graphe.

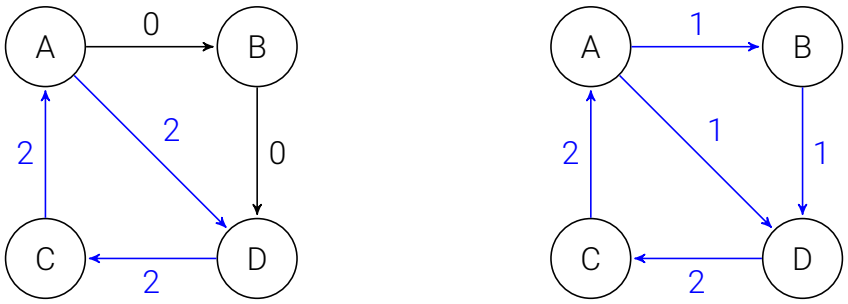


FIGURE 8.7 – Exemples de réseaux de flot avec un flot (en bleu) circulant sur le réseau.

La figure 8.7 présente deux exemples de flots pouvant circuler dans un même réseau de quatre nœuds (A, B, C, D). Dans l'exemple de gauche, le nœud B n'est pas traversé, tandis que l'ensemble des nœuds sont traversés dans l'exemple de droite. Dans les deux cas, la règle de conservation du flot est respectée, c'est-à-dire que la somme des valeurs des arêtes entrantes est égale à la somme des valeurs des arêtes sortantes. Par exemple, dans l'exemple de gauche, un flot de 2 entre via l'arête venant de C, et un flot de 2 sort via l'arête allant vers D. Dans l'exemple de droite, le flot entrant de 2 provenant de C, sort vers B et D.

Notons que le flot est défini à la fois pour les arêtes (comme représenté dans la figure 8.7) et pour les nœuds. Dans l'exemple de la figure 8.7, le flot des nœud $\{A, D, C\}$ vaut 2 dans les deux cas, tandis que le flot du nœud B vaut 0 à gauche et 1 à droite.

Nous modéliserons le graphe $\mathcal{G}$ du réseau de flot par un couple (V, E) où V et l'ensemble des sommets et $E \in V \times V$ l'ensemble des arêtes avec n sommets et m arêtes. Le flot s'exprime alors par une fonction $\mathcal{F} : V \times V \longrightarrow \mathbb{R}$, et vérifie les propriétés suivantes :

- *Conservation du flot :* $\sum_{v \in V} \mathcal{F}(u, v) = 0$, sauf s'il s'agit d'une source ou d'un puits.
- *Anti-symétrie :* $\mathcal{F}(u, v) = -\mathcal{F}(v, u)$
- *Contrainte de capacité :* $\mathcal{F}(u, v) < c(u, v)$ où $c : V \times V \longrightarrow \mathbb{R}$ est la fonction de capacité.
- *Coût du flot :* CoûtTotal $= \sum_{v \in V}$ *coût*$(u, v) \times \mathcal{F}(u, v)$

Dans notre approche, nous utiliserons un flot pour modéliser les robots parcourant la carte. Le coût du flot correspond ici à notre heuristique, permettant de choisir, parmi plusieurs flots possibles, celui à utiliser pour répartir les robots.

4.2 Flot de robots

Les flots se prêtent bien aux problèmes d'assignation. Il est par exemple possible d'affecter différentes personnes à différentes tâches à l'aide d'un réseau de flot, sachant que certaines tâches ne peuvent être effectuées que par certaines personnes. La figure 8.7 montre un exemple de réseau de flot, avec pour objectif d'assigner les tâches $\{T_1, T_2, T_3, T_4, T_5\}$ aux personnes $\{P_1, P_2, P_3\}$ à l'aide du flot. Le nœud S est une source et le nœud P un puits. L'objectif est alors de déterminer le flot de valeur maximal, et ainsi obtenir la meilleure affectation possible des tâches aux personnes.

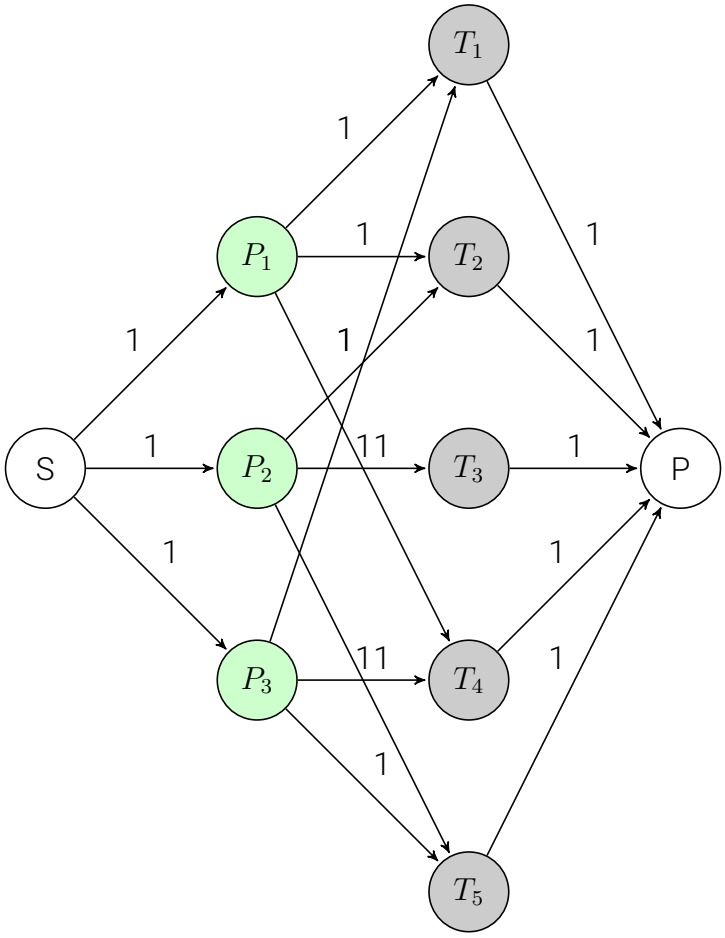


FIGURE 8.8 – Réseau de flot pour assigner différentes personnes à différentes tâches.

Nous allons utiliser une approche similaire pour affecter les robots aux zones inexplorées. Le coût des arêtes correspond à un coût heuristique de l'exploration, pouvant

être une métrique de distance (entre les deux nœuds d'une arête), une préférence sémantique (par exemple, préférer les couloirs aux salles), ou encore des commandes spécifiques définies par les opérateurs humains (zones à éviter ou privilégier).

Dans notre cas, le réseau correspond à la carte sémantique des robots, et le flot aux robots circulant sur les nœuds et arêtes de cette carte. Notons que cette représentation n'est pas temporelle (le flot circulant « en continu » sur le graphe), et ne différencie pas les robots les uns des autres, ne considérant que le nombre de robots passant par un endroit donné.

4.3 Données locales et globales

Dans cette section, nous verrons comment produire une assignation sémantique à partir d'un réseau de flot, produit à partir de la carte sémantique des robots. Cette carte sémantique et l'assignation des robots sont globales, et considèrent ainsi l'ensemble des robots et la totalité de la carte sémantique. Dans la section 5 Frontières locales avec contraintes sémantiques, nous verrons une adaptation de l'approche par « frontières locales » utilisant en partie des données globales de l'assignation sémantique.

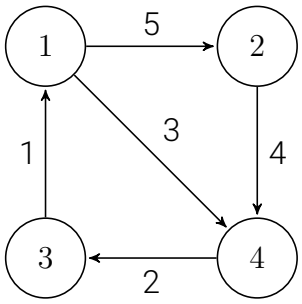
4.4 Représentation de la navigabilité

La carte sémantique est modélisée par un graphe $\mathcal{G}$, composé d'un couple (V, E) où V et l'ensemble des sommets et $E \in V \times V$ l'ensemble des arêtes avec n sommets et m arêtes.

Nous représenterons cette carte à l'aide du nombre de nœuds n, du nombre d'arêtes m, et d'une matrice d'incidence $M : n \times m$ dont chaque valeur est définie par :

$$M_{n,m} = \begin{cases} -1 \text{ si l'arête } m \text{ part de } n \\ 0 \text{ si l'arête } m \text{ ne concerne pas le nœud } m \\ 1 \text{ si l'arête } m \text{ arrive sur } n \end{cases}$$

Le coût d'une arrête est représenté par une fonction coût$(V) \longrightarrow \mathbb{N}$.



$$M = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & -1 \\ -1 & 1 & 0 & 0 & 0 \\ 0 & -1 & -1 & -1 & 0 \end{pmatrix}$$

FIGURE 8.9 – Représentation d'un graphe avec $n = 4$ nœuds et $m = 5$ arêtes (à gauche), et de la matrice d'incidence associée (à droite).

La figure 8.9 montre un exemple de graphe et la matrice d'incidence de ce graphe. Nous allons maintenant voir comment produire un graphe à partir d'une carte sémantique (déterminée à l'aide de la méthode vue dans la partie 2. Identification des pièces et portes.

4.5 Construction du graphe

L'objectif est maintenant de produire un graphe orienté à l'aide de la carte sémantique produite par les robots au cours de l'exploration. Pour cela, les arêtes entre deux salles seront déterminées de la manière suivante :

1. On part des feuilles de l'arbre, et on crée une arête orientée vers la feuille partant de toute salle (nœud) menant à cette feuille.
2. On prend les nœuds depuis lesquels sont partis les nouvelles arrêtent. Pour chacun de ces nœuds, on considère le groupe de nœud relié, et tout autre groupe relié dans la carte sémantique :
   - Si l'un des groupes contient un ou plusieurs robots et l'autre non, alors l'arête part du groupe contenant un ou plusieurs robots.
   - Si les deux groupes contiennent des robots, alors l'arête est bidirectionnelle.
   - Sinon, le groupe le plus petit (ayant le moins de nœuds) reçoit l'arête.
3. On réitère l'étape 2 avec les nœuds pour lesquels on a créé une arête sortante dans l'étape 2

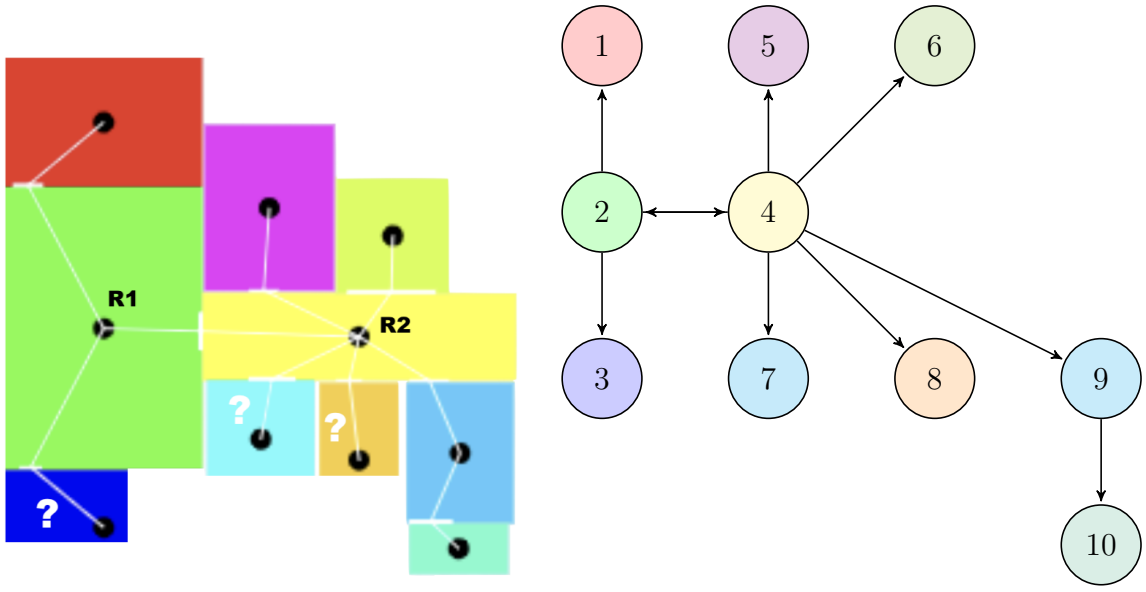


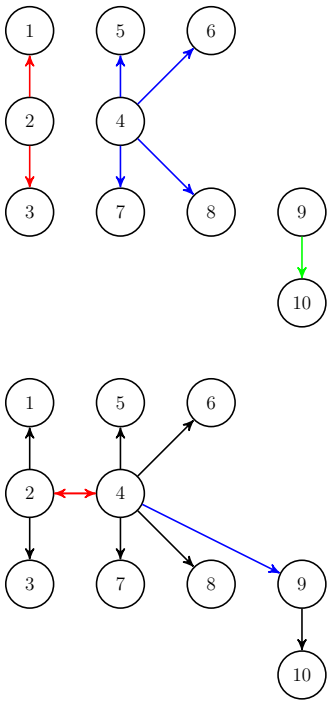
FIGURE 8.10 – Carte sémantique produite par les robots (à gauche) et graphe associé (à droite)

La figure 8.10 présente la carte sémantique au cours de l'exploration (à gauche), avec deux robots *R1* et *R2* et trois salles inexplorées identifiées par un point d'interrogation. La partie droite présente le graphe obtenu à partir de cette carte et de la position des robots.

La figure 8.11 présente la construction itérative du graphe suivant l'algorithme décrit ci-dessus. Dans le cas de ce graphe, un seul passage par les étapes 1 et 2 est nécessaire pour créer toutes les arêtes.

4.6 Définition des contraintes

Une fois le graphe réalisé, nous allons devoir définir la fonction de coût des arêtes, ainsi que les contraintes sur les positions des robots.



Étape 1

- Les nœuds $\{1, 3\}$ sont des feuilles, et reçoivent donc des arêtes du nœud **2** (en rouge).
- Les nœud $\{5, 6, 7, 8\}$ sont des feuilles et reçoivent donc des arêtes du nœud **4** (en bleu).
- Le nœud 10 est une feuille et reçoit une arête du nœud **9** (en vert).
- Les nœuds **2, 4** et **9** sont donc à traiter ensuite.

Étape 2

- Le nœud 2 est relié au nœud 4. Un robot se trouve dans le groupe $\{1, 2, 3\}$ du nœud 2, et un robot se trouve dans le groupe $\{4, 5, 6, 7, 8\}$ du nœud 4. On crée donc une arête bidirectionelle entre 2 et 4 (en rouge).
- Le nœud 9 est relié au nœud 4. Le groupe $\{9, 10\}$ du nœud 9 ne contient pas de robot, contrairement à celui du nœud 4, l'arête part donc de 4 (en bleu).

FIGURE 8.11 – Construction itérative du graphe

Fonction de coût

Pour la fonction de coût, nous pouvons attribuer un coût aux arêtes selon une métrique de distance, en tenant compte des données sémantiques (couloirs, pièces closes, etc), ou selon d'autres critères suivant l'objectif recherché. La fonction de coût déterminera ainsi les choix réalisés par les robots pour leur assignation. Nous verrons différentes assignations possibles dans la partie 6 *Objectifs et missions*. Dans un premier temps, nous allons nous contenter d'assigner un coût uniforme de 1 à chaque arrête.

$$\text{coût}(V) = 1 \quad \forall V$$

Propagation du flot

Pour diriger les robots, nous allons simuler un flot $\mathcal{F}$ parcourant le graphe, correspondant au nombre de robots passant par chaque arrête, et présent sur chaque nœud.

$$\begin{array}{rccl} \mathcal{F}: & V & \longrightarrow & \mathbb{N} \\ & v & \longmapsto & \text{nombre de robot passant par l'arête } v. \end{array}$$

$$\begin{array}{rccl} \mathcal{F}: & E & \longrightarrow & \mathbb{N} \\ & e & \longmapsto & \text{nombre de robot présents sur le nœud } e \end{array}$$

Le flot d'un nœud correspond au nombre de robots passés sur ce nœud. Le flot d'une arête correspond au nombre de robots ayant emprunté cette arête.

4.7 Résolution

De façon à simuler le flot des robots sur le graphe, nous commencerons par relier les feuilles du graphe aux positions où se trouvent les robots. Ces arêtes supplémentaires

permettent au flot d'être cyclique. En soi, cela revient à transformer les nœuds contenant des robots en sources, et les nœuds inexplorés en puits.

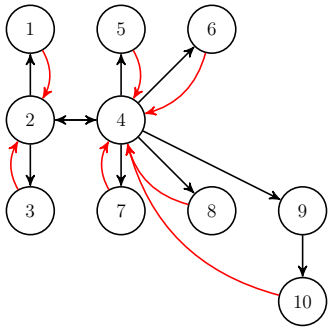


FIGURE 8.12 – Arêtes ajoutées au graphe pour la résolution (en rouge)

On définit ensuite le flot des nœuds :

- Le flot initial de tous les nœuds est 0.
- Les nœuds contenant des robots ont un flot égal au nombre de robots présents. Ici, nous avons un robot en 2 et un robot en 4, nous définissons donc $F(2) = 1$ et $F(4) = 1$.

Pour résoudre le système de contrainte, nous utilisons le résolveur de contraintes MiniZinc.

4.8 Résolution de contraintes avec MiniZinc

MiniZinc[2] est un langage de modélisation de contraintes gratuit et opensource, développé à l'Université Monash en collaboration avec *Data61 Decision* Sciences et l'Université de Melbourne.

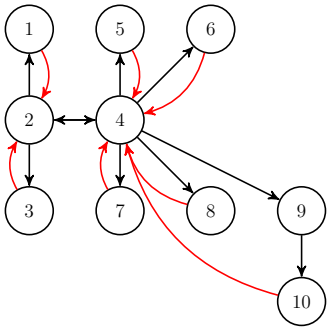
MiniZinc est disponible avec un IDE permettant de tester et exécuter des systèmes de contraintes, où sans IDE pour simplement exécuter un code MiniZinc. Des bibliothèques logicielles permettent également de résoudre des systèmes définis avec MiniZinc depuis différents langages de programmation, donc *Javascript*, utilisé par le simulateur 3D développé au cours de cette thèse.

4.9 Implémentation avec MiniZinc

Pour résoudre le problème avec MiniZinc, il faut commencer par lui fournir le graphe de la carte dans un format adapté. La première étape consiste donc à convertir la carte sémantique en graphe suivant le formalisme décrit dans la section précédente. La figure 8.13 présente le graphe de la carte sémantique vue précédemment et la matrice d'incidence correspondante.

Nous produisons ainsi un ensemble de données au format *JSON* pouvant être lues par MiniZinc et le simulateur (voir ci-dessous).

2. Site web de MiniZinc : https://www.minizinc.org



$$M = \begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & -1 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & 1 & -1 & -1 & -1 & -1 & -1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1
\end{pmatrix}$$

FIGURE 8.13 – Matrice d'incidence (à droite) résultat du graphe (à gauche).

```
{
  "n_nodes": 10,
  "n_edges":17,
  "n_rob":2,
  "incidence": [
    [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, -1, 0, 0, 0, 0, 0, 0],
    [-1, -1, 1, -1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0],
    [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, -1, 0,0, 0, 0, 0],
    [0, 0, -1, 1, -1, -1,-1, -1, -1, 0, 0, 0, 1, 1, 1, 1, 1],
    [0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, -1, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, -1, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, -1, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,-1, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 1, -1, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, -1]
  ],

  "cost": [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1]
}
```

Enfin nous résolvons le passage du flot sur le graphe à l'aide d'un script MiniZinc.

```
% Parametres principaux
int: n_edges; % Nombre d'aretes
int: n_nodes; % Nombre de noeuds

% Matrice d'incidence et vecteur de couts
array[1..n_nodes, 1..n_edges] of int:incidence;
array[1..n_edges] of int:cost;

var 0..1000: total_cost;

array[1..n_edges] of var 0..n_rob:flow;

% Initialisation des contraintes dut flot
constraint flow[2]>=1; % il y a un robot en 2
constraint flow[4]>=1; % il y a un robot en 4

constraint flow[3]>=1;  % on veut au moins un robot en 3
constraint flow[8]>=1;  % on veut au moins un robot en 8
constraint flow[7]>=1;  % on veut au moins un robot en 7

% Definition du cout de l'exploration
constraint
total_cost = sum(edge in 1..n_edges) (cost[edge]*flow[edge]);

% Conservation du flot
constraint
  forall (
    node in 1..n_nodes
  ) (
      0 = sum(edge in 1..n_edges) (incidence[node,edge]*flow[edge])
    );

solve minimize total_cost;
```

Script MiniZinc

Nous obtenons ainsi le flot des arêtes, c'est-à-dire combien de robots doivent passer par chaque arête, nous permettant alors de déplacer les robots.

Le résolveur de contraintes MiniZinc nous renvoie alors le flot de chaque arête et le coût total pour chaque solution trouvée :

```
total_cost = 8;
flow = [0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0];
```

Résultat

La figure 8.14 montre le résultat donné par MiniZinc sur le graphe (flot de chaque arête, indiqué par le nombre donné par MiniZinc lorsqu'il est non nul. Le flot correspond au nombre de robots passant par chaque arête. À partir de ce flot, nous pouvons déterminer plusieurs assignations possibles.

Notons que dans cet exemple, plusieurs assignations sont possibles : d'une part le flot n'est pas spécifique à un robot, et d'autre part le flot ne donne pas d'ordonnancement. Sur l'exemple de la figure 8.14, le robot en 2 peut explorer le nœud 3, puis revenir au nœud 2, et explorer le nœud 7 ou 8 en passant par le nœud 4. Une façon de limiter les choix possibles est de contrainte davantage le système, par exemple en ajoutant un coût supplémentaire pour chaque déplacement de façon à limiter le nombre d'arêtes traversées pour une assignation (permettant de limiter les choix à long terme). Une fois un

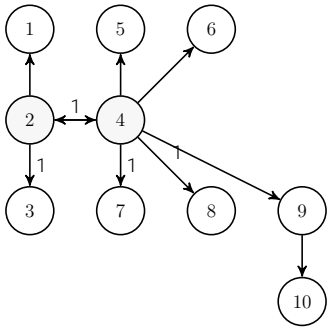


FIGURE 8.14 – Graphe avec les résultats obtenus avec MiniZinc

graphe avec flot obtenu, nous pouvons définir une assignation, par exemple en prenant la première arête disponible pour chaque robot. L'algorithme 4 définit une liste d'objectifs pour un robot r_i à partir du flot, de la matrice d'incidence et de la position initiale du robot.

Algorithm 4 Choix des objectifs pour le robot r_i

Entrées :
e_{r_i} *(position du robot* r_i*)*
$\mathcal{F} : E \longrightarrow \mathbb{N}$ *(flot obtenu avec MiniZinc)*
$M : n \times m$ *(matrice d'incidence)*

Sortie :
$O_{r_i} \longleftarrow []$ *(liste des objectifs du robot* r_i*)*

for all e_i **do**
  Pour toute arête du graphe
  if $v_{e_{r_i}, e_i} > 0$ **then**
    Si au moins un robot passe par l'arête menant à e_i
    On ajoute le nœud à la liste d'objectif
    o_{r_i}.push(e_i)
    On se déplace à e_i
    $e_{r_i} \longleftarrow e_i$
    Et on recommence
    break for
  end if
end for
Enfin, on renvoie la liste d'objectifs
return o_{r_i}

Avec cet algorithme, nous pouvons ainsi définir les objectifs des robots à l'aide du flot obtenu avec MiniZinc. Cette approche est proche des approches globales, permettant aux robots de choisir des objectifs sur l'ensemble de la carte (au-delà de leur champ de perception), et nécessite un partage de la carte sémantique globale.

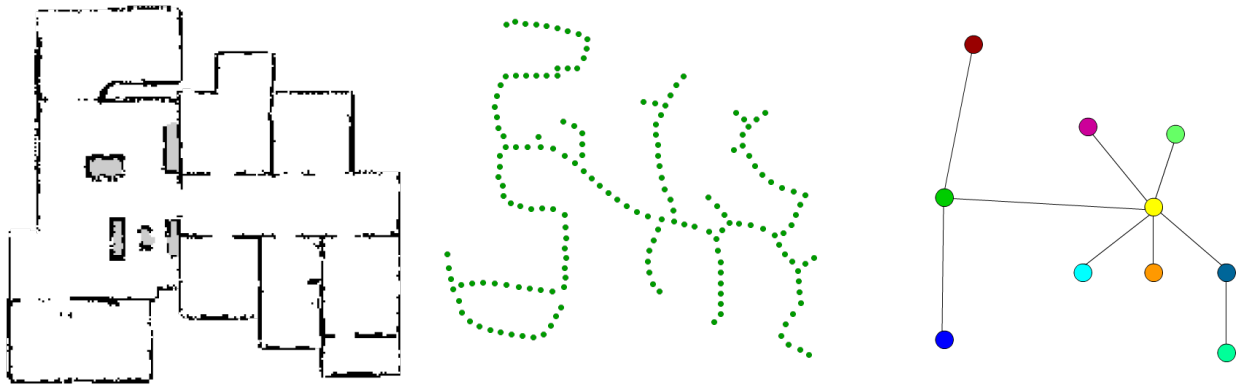
4.10 Intérêt d'une assignation par flot

Ce type d'affectation est facilement utilisable dans une approche par frontières globales (voir chapitre 4 Approches de l'exploration, 4 Approches par frontières).

Avec une assignation par flot, il est facile de calculer le coût heuristique d'une assignation donnée, et ainsi de déterminer la meilleure affectation possible à l'aide du flot maximal, suivant des critères externes à la représentation des données. Le coût des arêtes peut ainsi être déterminé par de nombreux critères locaux (métriques) ou globaux (sémantiques, commandes d'utilisateurs humains).

5 Frontières locales avec contraintes sémantiques

Avec l'approche par frontières locales, les robots mémorisent leur historique de positions, et partagent entre eux l'ensemble de leurs positions précédemment occupées (figure 8.15 b). Pour construire parallèlement la carte sémantique, ils utilisent leur grille d'occupation (figure 8.15 a), et partagent un graphe sémantique, identifiant les salles et portes (figure 8.15 c).



a) Grille d'occupation b) Historique des positions c) Graphe sémantique

FIGURE 8.15 – Informations locales et partagées des robots.

Si nous voulons utiliser une assignation sémantique pour définir l'objectif des robots, il faut pouvoir tirer parti des informations globales partagées (jusqu'ici, la carte sémantique figure 8.15 c) pour pouvoir assigner à un robot un objectif au-delà de son rayon de perception, et lui permettre de déterminer comment se rendre à son objectif.

Supposons par exemple que le robot de la figure 8.16 soit assigné à la salle identifiée avec le « A » rouge, qui aurait été vue par une autre robot plus tôt dans l'exploration. Dans cette configuration, le robot n'étant pas passé devant cette salle, ne peut trouver un chemin pour y accéder à l'aide de ses traces. L'historique des positions partagé entre les robots n'indique jusqu'ici aucune information vis-à-vis du graphe sémantique, et ne permet donc pas au robot de trouver son chemin jusqu'à la salle indiquée.

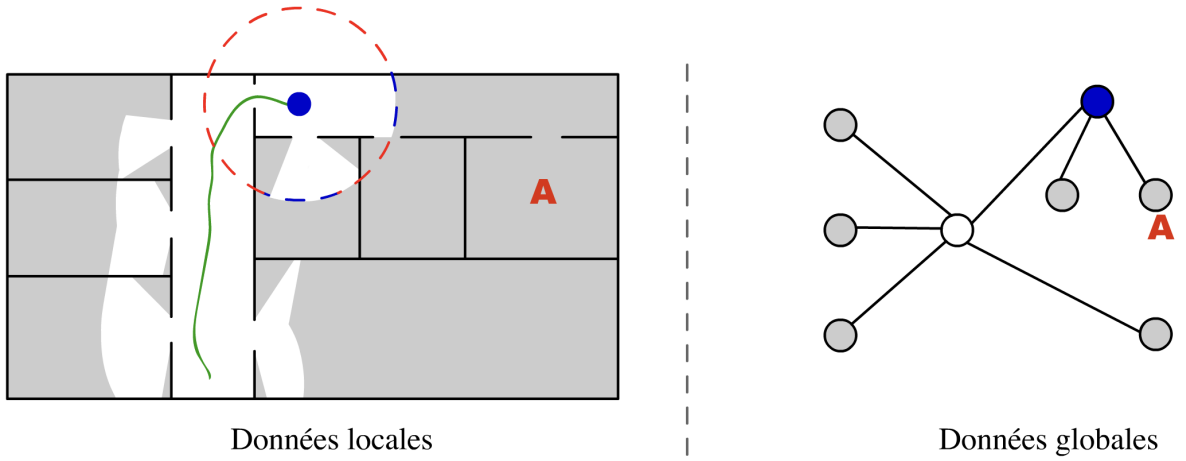
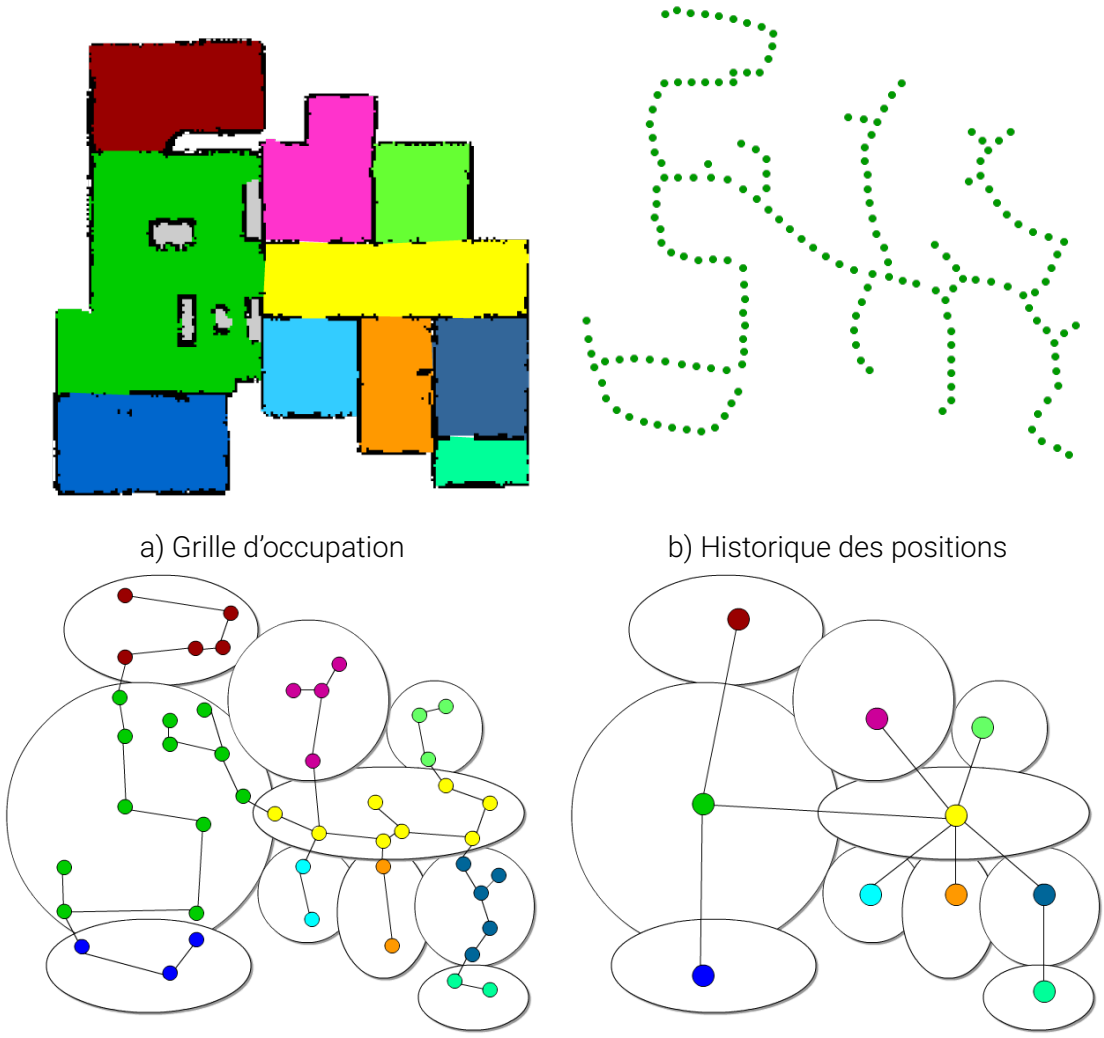


FIGURE 8.16 – Représentation des données locales du robot (à gauche) et globales (à droite) dans l'approche par frontières locales avec contraintes sémantiques.

Pour faciliter l'assignation sémantique, nous allons construire un graphe intermédiaire, plus proche des données déjà identifiées par les robots dans l'approche locale (historique des positions représenté figure 8.15 b) et des informations sémantiques (figure 8.15 c). L'idée est de pouvoir utiliser l'algorithme d'assignation sémantique décrit dans les sections précédentes sur ce nouveau graphe intermédiaire, tout en pouvant extraire les données de navigation utiles pour déplacer les robots jusqu'à leur objectif.



a) Grille d'occupation

b) Historique des positions

c) Graphe intermédiaire

d) Carte sémantique

FIGURE 8.17 – Données locales et partagées avec le nouveau graphe intermédiaire (c), sur lequel nous pouvons exécuter une assignation sémantique, et produire un chemin pour déplacer les robots jusqu'à leur objectif.

La figure 8.17 présente les nouvelles données locales et partagées entre les robots, incluant le nouveau graphe intermédiaire, incluant les données sémantiques tout en étant plus proche de l'historique des positions des robots pour permettre de déterminer les chemins pour déplacer les robots à leur objectif.

Pour construire ce graphe, nous devons suivre plusieurs principes :

— *Économie des nœuds* : de façon à pouvoir exécuter régulièrement et rapidement des assignations sémantiques, il est préférable de limiter le nombre de nœuds dans le graphe intermédiaire. Certains points de l'historique ne sont pas utiles dans le graphe intermédiaire, notamment lorsque le robot s'est déplacé en ligne droite.

— *Identifier les raccourcis* : lorsqu'un nœud du graphe est visible depuis un autre nœud sans obstacles sur le passage, il est possible de déduire des passages supplémentaires (arêtes du graphe), même lorsque le robot n'a pas emprunté ce chemin. Ces arêtes supplémentaires pourront ensuite permettre aux robots d'utiliser des raccourcis lors du calcul de leur trajectoire suite aux assignations séman-

tiques.

— *Identifier toutes les salles connues* : le graphe intermédiaire doit contenir au moins un nœud pour chaque salle identifiée sur la carte sémantique. En effet, si une salle est identifiée sur la carte sémantique, mais ne contient aucun nœud sur le graphe intermédiaire, il ne sera pas possible d'assigner un robot à cette salle. Il est donc nécessaire d'ajouter un nœud au graphe pour toute nouvelle salle identifiée (à l'entrée de la salle).

5.1 Algorithme général

Le nouvel algorithme fonctionne de la manière suivante :

— Les robots construisent une carte sémantique locale à l'aide de leur grille d'occupation en identifiant les salles et portes avec la méthode décrite dans la section 2 Identification des pièces et portes.

— Les cartes sémantiques sont fusionnées (voir section 3 Partage des informations sémantiques) pour produire une carte sémantique partagée.

— Les robots ajoutent les éventuels nouveaux noeuds utiles à l'historique des positions utiles (déterminés selon les trois critères de la section précédente, avec l'algorithme décrit dans la section suivante 5.2 Construction du graphe intermédiaire).

— Le graphe intermédiaire est produit à partir des noeuds utiles partagés par les robots et de la carte sémantique partagée (voir section 5.2 Construction du graphe intermédiaire).

— On assigne les robots à un objectif dans le graphe sémantique (avec la méthode décrite dans 4 Assignation sémantique).

— On transpose les trajectoires dans le graphe intermédiaire

— Les robots se déplacent vers leur objectif jusqu'à réassignation.

La figure 8.18 présente les différentes étapes de l'algorithme.

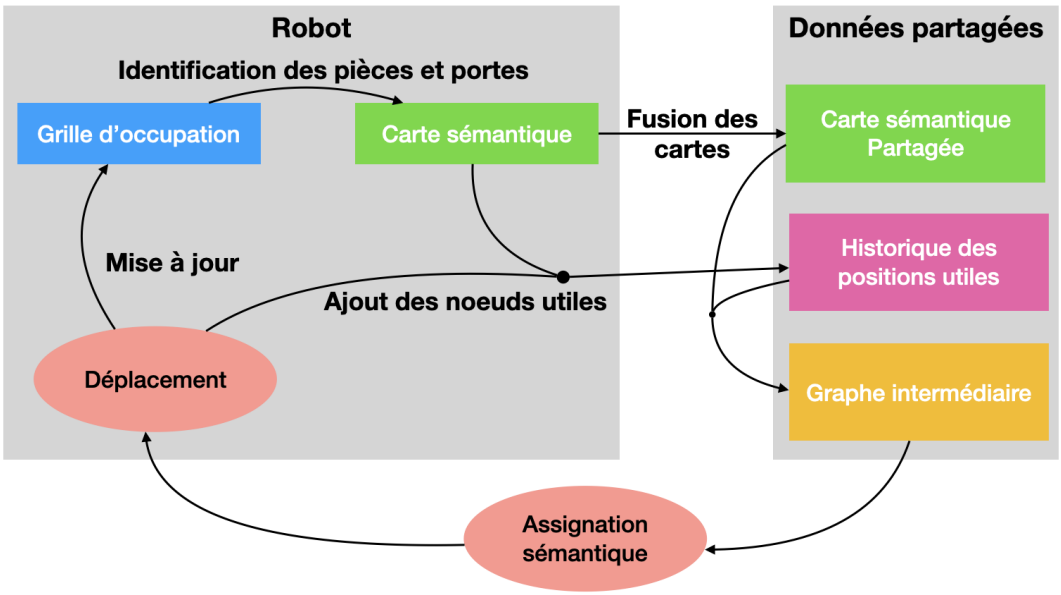


FIGURE 8.18 – Présentation générale du fonctionnement de l'algorithme local sémantique.

5.2 Construction du graphe intermédiaire

Lors de chaque déplacement, un robot peut ajouter un ou plusieurs nœuds au graphe intermédiaire, en suivant les règles suivantes :

— Si le robot a effectué une rotation, il ajoute sa position actuelle au graphe.

— Pour toute nouvelle salle découverte dans le graphe sémantique, ajouter un nœud à l'entrée de la salle.

— Ne pas ajouter de nœud sinon.

De plus, si un nœud du graphe intermédiaire est visible depuis la position actuelle, que la position actuelle a été ajoutée au graphe intermédiaire, et qu'il n'y a aucune arête entre le nœud actuel et le nœud vu, ajouter l'arête.

En suivant ces règles, nous limitons ainsi le nombre de nœuds du graphe intermédiaire tout en nous assurant qu'il y a au moins un nœud par salle du graphe sémantique et en intégrant les éventuels raccourcis identifiés.

5.3 Correspondance entre les représentations

Trois niveaux de représentations de la navigabilité existent ainsi pour les robots : la carte sémantique permet d'assigner les robots aux différentes salles tandis que le graphe intermédiaire permet de calculer les trajectoires. Pour passer aisément d'une représentation à l'autre, chaque nœud du graphe intermédiaire doit être associé à une salle (un nœud) de la carte sémantique. Lors d'une assignation sémantique, les trajectoires obtenues pour chacun des robots sont ainsi développées en trajectoires correspondantes dans le graphe intermédiaire pour calculer aisément une trajectoire dans le monde réel.

6 Objectifs et missions

L'approche proposée dans cette partie a pour objectif de permettre de modifier le comportement des robots aisément selon le résultat attendu. Dans cette partie, nous allons adapter les contraintes de l'algorithme de façon à adapter les choix effectués par les robots.

6.1 Conditions expérimentales

De façon à tester plus efficacement les diverses améliorations que nous allons apporté, nous avons conçu et ajouté de nouvelles cartes au simulateur, de façon à disposer de lieux plus grands, avec de nombreux couloirs et salles.

La figure 8.19 montre la carte hôpital ajoutée au simulateur, nettement plus grande que les cartes utilisées jusqu'ici, avec plus de couloirs et de pièces. La figure 8.20 présente la grille d'occupation résultant de l'exploration de cette carte.

6.2 Privilégier les couloirs

Nous allons maintenant adapter les contraintes de façon à privilégier les couloirs face aux salles. Pour cela, nous allons simplement augmenter le coût des arêtes orientées vers un couloir.

FIGURE 8.19 – Nouvelle carte inspirée d'un hôpital ajoutée au simulateur, avec de très nombreuses salles et couloirs plus complexes.

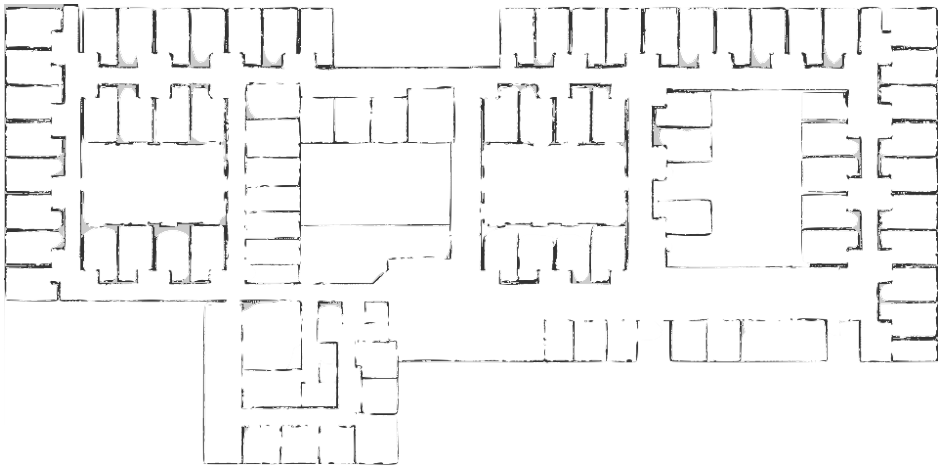


FIGURE 8.20 – Grille d'occupation de la carte hôpital après l'exploration

La figure 8.21 montre un exemple de simulation ou le robot (sa trajectoire est indiquée par un trait bleu) explore la carte hôpital en privilégiant les couloirs, n'entrant donc pas dans les pièces closes.

Pour comparer les approches, nous pouvons utiliser différentes métriques :

- Salles découvertes : le nombre de salles qui ont été identifiées au fil de l'exploration,
- Espace exploré : aire du carré englobant l'historique des positions des robots au fil de l'exploration.

La figure 8.22 montre l'évolution de l'espace exploré au fil des itérations, avec et sans critère de sélection de frontières, tandis que la figure 8.23 montre l'évolution du nombre de salles découvertes. L'intérêt de cette approche n'est pas de maximiser l'espace exploré ni les salles découvertes, mais de contraindre les robots à différents choix (ici, privilégier les couloirs). Ces mesures montrent cependant que la zone dans laquelle le robot se déplace croît plus rapidement lorsque les couloirs sont privilégiés, ainsi que le

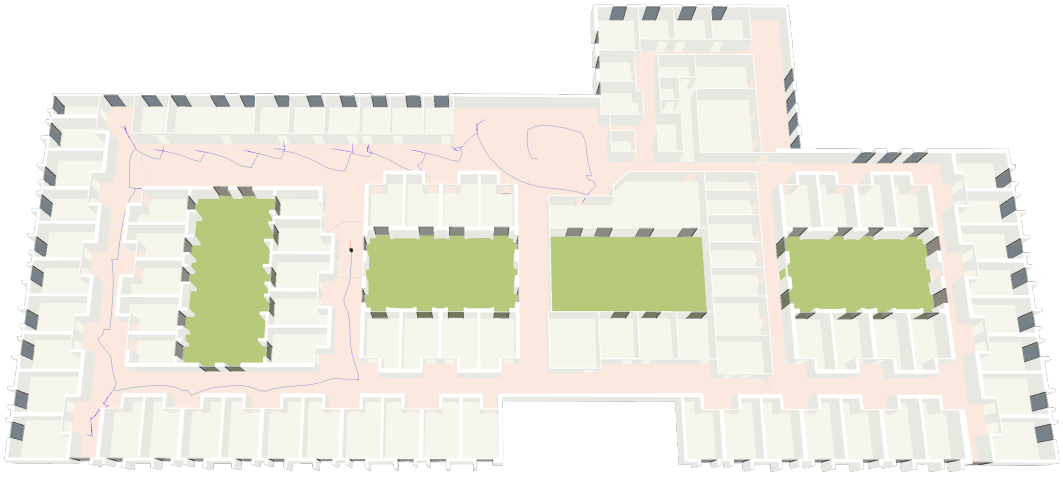


FIGURE 8.21 – Exploration de la carte hôpital en privilégiant les couloirs aux salles, permettant de faire le tour du bâtiment avant d'explorer chacune de ses salles.

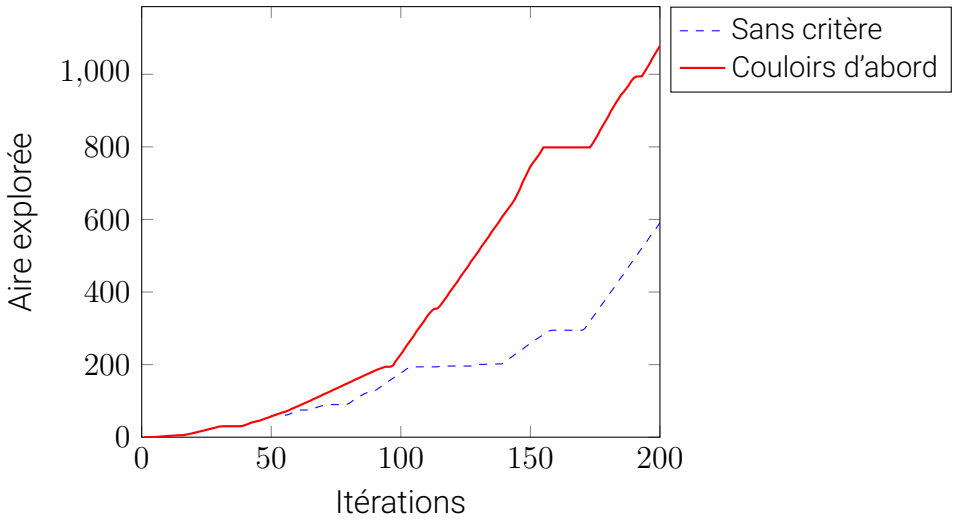


FIGURE 8.22 – Comparaison de l'aire du carré englobant les positions du robot au fil des itérations avec et sans critères de sélection.

nombre de salles découvertes, permettant ainsi au robot de « faire le tour du bâtiment » avant d'explorer plus minutieusement chaque pièce.

7 Intérêt d'une approche par contraintes

De nombreuses situations nécessitent d'adapter les algorithmes selon le contexte. Par exemple, lors d'un incendie, certaines zones seront à privilégier, tandis que d'autres pourront être trop dangereuses pour que les robots s'y aventurent. De la même façon, l'utilisation de robots dans un cadre militaire doit souvent répondre à un certain nombre de contraintes opérationnelles dépendantes de la mission.

De fait, l'exploration d'un environnement par une flotte de robots autonomes avec les algorithmes décrits jusqu'ici peut sembler aléatoire d'un point de vue humain, et ainsi ne pas s'adapter suffisamment à la situation, selon le contexte opérationnel souvent

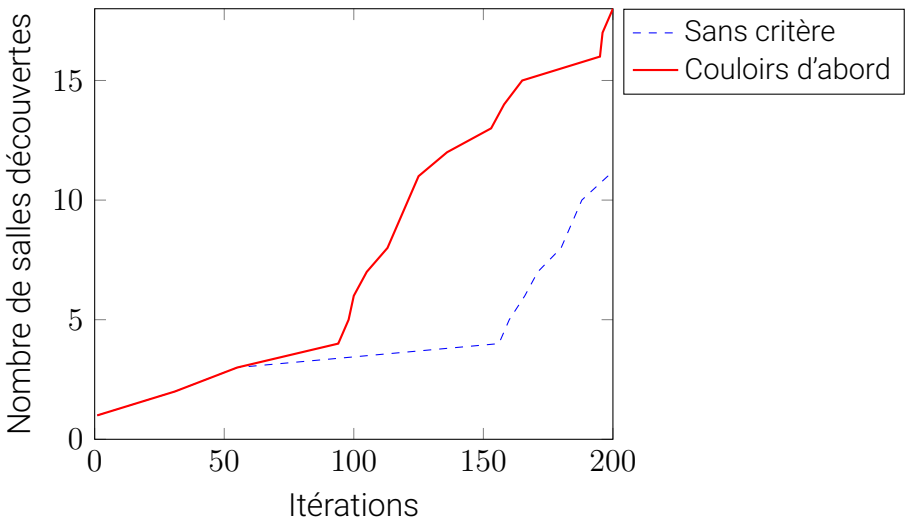


FIGURE 8.23 – Comparaison de l'aire du carré englobant les positions du robot au fil des itérations avec et sans critères de sélection.

mieux compris et déterminé par les agents humains. Les approches décrites dans ce chapitre visent ainsi à permettre aux opérateurs de mieux cadrer les objectifs des robots de façon à les adapter à la situation.

L'utilisation d'un résolveur de contraintes (dans notre cas, *MiniZinc*, mais d'autres pourraient être utilisés) nous permet ainsi d'aborder le problème de façon modulaire et paramétrable. Nous emprunterons ici des méthodes issues de la recherche opérationnelle avec diffusion de flots [148] pour modéliser le problème (là encore, d'autres modélisations, par exemple à base de TSP, problème du sac à dos, etc. sont également possibles).

8 Conclusion

Le chapitre précédent a abordé l'exploration autonome avec une flotte de robots autonomes réactifs. Dans ce chapitre, nous avons essayé d'adapter cette approche de façon à tirer parti d'informations sémantiques tout en permettant aux opérateurs de mieux définir les objectifs des robots.

La construction d'une carte sémantique au fil de l'exploration permet aux robots de mieux connaître leur environnement, en identifiant les différents couloirs, pièces et portes. Ce type de carte peut également permettre de réduire encore la quantité de données échangées entre les robots dans certains contextes.

L'utilisation de résolveur de contraintes permet d'avoir une approche modulaire et facilement paramétrable, de façon à s'adapter aux différentes situations concrètes, qu'il s'agisse de trouver d'éventuelles personnes à secourir dans un bâtiment en flammes où d'appuyer des combattants dans une stratégie militaire. Cette façon de procéder permet ainsi d'aborder une multitude de scénarios très différents en s'adaptant aux contraintes opérationnelles de chacun d'eux.

Nous avons vu comment adapter l'approche par « frontières locales » avec un système de contraintes pour prioriser certains types de pièces grâce à la carte sémantique construite au fil de l'opération. Bien sûr, de nombreuses autres possibilités peuvent être envisagées en adaptant les contraintes, de façon à répondre aux besoins de différentes missions, par exemple pour éviter ou privilégier certaines zones de la carte, regrouper les robots, etc.

Notre approche est basée sur la diffusion d'un flot, représentant le nombre de robots passant par chaque chemin. Là encore, d'autres modélisations du problème sont possibles, par exemple sous forme du problème du voyageur de commerce (*Travelling salesman problem*), du problème du sac à dos, etc. L'idée générale est celle d'un coût heuristique, définissant les actions des robots à partir de règles générales définies pour l'ensemble de l'exploration.

Troisième partie

Conclusions

Chapitre 9

Conclusion générale

Cette thèse a pour objectif de permettre à une flotte de robots autonomes d'explorer efficacement un environnement inconnu en limitant les informations échangées et les calculs effectués. En effet, les calculs nécessitent une grande quantité d'énergie, réduisant ainsi l'autonomie des robots tout en augmentant le coût des robots. Les communications, en plus de nécessiter également de l'énergie particulièrement sur des transmissions à longue distance, peuvent également informer de la présence des robots, ce qui peut être problématique dans des situations militaires.

De très nombreuses approches ont été proposées ces dernières décennies pour permettre à un groupe de robots autonomes d'explorer un environnement inconnu. Parmi elles, l'approche par frontières donne de bons résultats tout en permettant d'utiliser différents algorithmes d'assignation. Cette approche nécessite cependant un consensus des robots pour chaque assignation aux frontières. De plus, la totalité de la grille d'occupation des robots est partagée régulièrement, de façon à ce que tous les robots aient connaissance de l'ensemble des frontières actuelles. Ces contraintes impliquent alors un partage d'informations conséquent tout au long de l'exploration, et une quantité de calculs importante lors de l'assignation, d'autant plus grande que le nombre de robots et la taille de la carte sont grands.

D'autres approches permettent d'explorer un environnement inconnu avec une flotte de robots autonomes sans nécessiter de consensus global entre eux tout en limitant les calculs. Nous pouvons notamment citer des approches fourmis et Brick & Mortar telles que *MDFS*, *BoB* où encore *BMILRV*. Ces dernières présentent cependant de nombreuses limitations. Les algorithmes tels que *BoB* et *MDFS* travaillent sur une grille où le robot ne perçoit que les quatre cellules adjacentes, sans tenir compte du rayon de perception de celui-ci. Les approches qui incluent la totalité du champ de perception, comme *BMILRV*, présentent quant à elles des performances significativement inférieures aux approches globales.

La première contribution de cette thèse est l'approche par frontières locales. Cette nouvelle approche est locale, ne tenant compte que des informations à l'intérieur du champ de perception des robots, tout en permettant des performances proches des algorithmes par frontières globales. La version continue de cette approche réduit également significativement les données échangées, passant de la totalité de la grille d'occupation à une simple liste de positions précédemment occupées. La version discrète de cette approche à été publiée aux 27èmes Journées Francophones sur les Systèmes Multi-Agents (*JFSMA*) en 2019 [134] et a reçu le prix du meilleur article de cette conférence, et est présentée dans le chapitre 7 Exploration par stigmergie. D'un point de vue théorique, cette approche produit un arbre couvrant l'environnement au fil de l'exploration dans un monde continu, permettant de garantir l'exhaustivité de l'exploration.

La présence d'obstacles n'est pas la seule information pouvant être traitée par les robots pour explorer efficacement un bâtiment. D'une part, de nombreux capteurs sont disponibles, telles que les caméras RGB-D, et différents algorithmes existent aujourd'hui pour identifier rapidement différents objets, permettant ainsi aux robots d'obtenir de nombreuses informations supplémentaires sur leur environnement. D'autre part, l'architecture humaine obéit à de nombreuses règles générales, pouvant permettre de mieux prédire ce qui pourra être trouvé dans un lieu donné. En utilisant ce type d'informations, un robot devant trouver une table de nuit pourra identifier des zones où la probabilité d'en trouver une sera plus importante, par exemple proche d'un lit ; un robot devant aller à l'autre bout d'un grand bâtiment pourra tenir compte du fait que des couloirs permettent généralement de traverser les grands bâtiments, et évitera de visiter chaque pièce une à une.

Dans le chapitre 8 Exploitation de données sémantiques, nous avons proposé une approche permettant de segmenter les salles d'un bâtiment et d'identifier les pièces et couloirs de celui-ci au cours d'une exploration autonome. Cette segmentation permet de limiter encore le partage d'informations entre robots, tout en comprenant mieux l'environnement exploré.

Nous avons ensuite utilisé une méthode de propagation de flots, de façon à orienter l'exploration suivant des critères définis à partir des informations sémantiques récoltées. Cette approche permet une grande flexibilité, permettant ainsi d'adapter l'algorithme pour privilégier certains types de pièces, modifier la répartition des robots, ou privilégier ou interdire des zones spécifiques.

Les possibilités ainsi obtenues sont nombreuses, permettant de s'adapter aux contraintes opérationnelles de différentes missions d'exploration, dont les objectifs et conditions peuvent être très différents.

Le travail proposé dans cette thèse permet ainsi d'explorer efficacement des environnements variés à l'aide d'une flotte de robots autonomes, tout en minimisant les communications et les calculs, et en étant capable d'adapter facilement l'exploration à différentes contraintes. Les simulations nous ont permis de tester les différentes approches sur plusieurs cartes inspirées de lieux réels dans différentes conditions. Il serait intéressant d'approfondir ce travail en collaboration avec différents acteurs souhaitant utiliser une flotte de robots dans des missions d'exploration réelles (pompiers, militaires, etc.), pour définir des scénarios concrets et adapter les algorithmes proposés à ces situations.

Résumés

Résumé en français

Cette thèse s'intéresse à l'exploration d'environnements inconnus à l'aide d'une flotte de robots autonomes réactifs. L'exploration autonome est utilisée dans différents domaines, allant des robots aspirateurs aux robots de recherche et de sauvetage utilisés lors de catastrophes naturelles (incendies, éboulements) ou encore dans des contextes militaires.

Le travail réalisé au cours de cette thèse a été financé par Safran Electronics & Defense en soutient du projet FURIOUS (FUturs systèmes Robotiques Innovants en tant qu'OUtilS au profit du combattant embarqué et débarqué) de la Direction Générale de l'Armement . Il fait suite au projet Cart-O-Matic, qui était l'un des cinq projets fondés par l'Agence Nationale de la Recherche (ANR) pour sa participation au concours de robotique « Défi CAROTTE » organisé par la *Délégation générale pour l'armement*, et qui a remporté ce concours.

De très nombreuses approches de l'exploration autonome existent dans l'état de l'art. Dans cette thèse, nous avons cherché à explorer efficacement un environnement intérieur en limitant les calculs et communications. Réduire la quantité de calculs nécessaires permet d'économiser les batteries des robots et d'utiliser plus facilement un grand nombre de robots. Réduire les communications permet une économie d'énergie, mais est également intéressant dans un cadre militaire, de façon à limiter les risques de compromettre la présence des robots. Un algorithme d'exploration local a été proposé, permettant de réduire significativement communications et calculs tout en maintenant un haut niveau de performances et publié aux 27èmes Journées Francophones sur les Systèmes Multi-Agents.

Nous avons ensuite proposé une nouvelle approche par carte sémantique, permettant de segmenter l'environnement en pièces et couloirs, ainsi qu'un nouvel algorithme d'exploration par contraintes. Cette nouvelle approche permet aux robots de mieux comprendre l'environnement dans lequel ils évoluent, et de réduire les erreurs de localisation et de perception propres aux capteurs qui les équipent. Notre approche par contrainte permet aux opérateurs de mieux définir les objectifs et priorités des robots, répondant ainsi aux besoins opérationnels de différentes missions, notamment dans le cadre de missions de recherche et de sauvetage ou de soutien militaire.

Résumé en anglais

This thesis focuses on exploring unknown environments using a group of reactive autonomous robots. Autonomous exploration is used in different domains, from vacuum robots to search and rescue robots used in natural disasters (fires, landslides) or military contexts.

The work achieved during this thesis was financed by Safran Electronics & Defense in support of the FURIOUS project (FUturs Innovative Robotic Systems as Tools for the benefit of the embarked and disembarked fighter) of the French General Direction of Armaments. It follows the Cart-O-Matic project, which was one of the five projects founded by the French National Research Agency (ANR) for its participation in the robotics competition "Défi CAROTTE" organized by the "Direction Générale pour l'Armement," and which won this competition.

Many approaches to autonomous exploration exist in the state of the art. In this thesis, we have tried to efficiently explore an indoor environment while limiting the amount of computation and communication. Reducing the amount of computation required saves robot batteries and makes it easier to use a large number of robots. Reducing communications saves energy but is also interesting in a military setting to limit the risks of compromising the presence of robots. A local exploration algorithm has been proposed to significantly reduce communications and computations while maintaining a high level of performance and was published at the *27th Journées Francophones sur les Systèmes Multi-Agents*.

We then proposed a new semantic map approach, allowing us to segment the environment into rooms and corridors, and a new constraint-based exploration algorithm. This new approach allows robots to understand better the environment in which they evolve and to reduce localization and perception errors specific to the sensors which equip them. Our constraint-based approach allows operators to define robot objectives and priorities better, thus meeting the operational needs of various missions, including search and rescue and military support.

Bibliographie

[1] Thomas Bayes. « LII. An essay towards solving a problem in the doctrine of chances. By the late Rev. Mr. Bayes, FRS communicated by Mr. Price, in a letter to John Canton, AMFR S ». In : *Philosophical transactions of the Royal Society of London* 53 (1763), p. 370-418.

[2] Nicholas Metropolis et Stanislas Ulam. *The Monte Carlo Method Journal of the American Statistical Association, vol. 44, no 247*. 1949.

[3] Richard Bellman. « A Markovian decision process ». In : *Journal of mathematics and mechanics* (1957), p. 679-684.

[4] Rudolph Emil Kalman. « A new approach to linear filtering and prediction problems ». In : (1960).

[5] Gene M Amdahl. « Validity of the single processor approach to achieving large scale computing capabilities ». In : *Proceedings of the April 18-20, 1967, spring joint computer conference*. 1967, p. 483-485.

[6] Kenneth E Batcher. « Sorting networks and their applications ». In : *Proceedings of the April 30–May 2, 1968, spring joint computer conference*. 1968, p. 307-314.

[7] James F Blinn. « Models of light reflection for computer synthesized pictures ». In : *Proceedings of the 4th annual conference on Computer graphics and interactive techniques*. 1977, p. 192-198.

[8] Alvy Ray Smith. « Tint fill ». In : *Proceedings of the 6th annual conference on Computer graphics and interactive techniques*. 1979, p. 276-283.

[9] D Meagher. « Octree Encoding : a new technique for the representation, manipulation and display of arbitrary 3-D objects ». In : *Computer. Rensselaer Polytechnic Institute* (1980).

[10] Donald Meagher. « Geometric modeling using octree encoding ». In : *Computer graphics and image processing* 19.2 (1982), p. 129-147.

[11] Hanan Samet. « The quadtree and related hierarchical data structures ». In : *ACM Computing Surveys (CSUR)* 16.2 (1984), p. 187-260.

[12] M Gondran et M Minoux. « Graphes et Algorithmes, chapitre 3 ». In : *Eyrolles. Les Algebres de Chemins* (1985).

[13] D Lee et Arthurk Lin. « Computational complexity of art gallery problems ». In : *IEEE Transactions on Information Theory* 32.2 (1986), p. 276-282.

[14] Lawrence Rabiner et Biinghwang Juang. « An introduction to hidden Markov models ». In : *ieee assp magazine* 3.1 (1986), p. 4-16.

[15] Alberto Elfes. « Sonar-based real-world mapping and navigation ». In : *IEEE Journal on Robotics and Automation* 3.3 (1987), p. 249-265.

[16] Richard E KORF. « Real-Time Heuristic Search : First Results. » In : *AAAI*. 1987, p. 133-138.

[17] Wei-pang CHIN et Simeon NTAFOS. « Optimum watchman routes ». In : *Information Processing Letters* 28.1 (1988), p. 39-44. ISSN : 0020-0190. DOI : https://doi.org/10.1016/0020-0190(88)90141-X. URL : https://www.sciencedirect.com/science/article/pii/002001908890141X.

[18] James L CROWLEY. « World modeling and position estimation for a mobile robot using ultrasonic ranging. » In : *ICRA*. T. 89. 1989, p. 674-680.

[19] Alberto ELFES et al. « Occupancy grids : A stochastic spatial representation for active robot perception ». In : *Proceedings of the Sixth Conference on Uncertainty in AI*. T. 2929. Morgan Kaufmann San Mateo, CA. 1990, p. 6.

[20] Bernard CHAZELLE. « Triangulating a simple polygon in linear time ». In : *Discrete & Computational Geometry* 6.3 (1991), p. 485-524.

[21] E. ANGELOPOULOU, T. H. HONG et A. Y. WU. « World model representations for mobile robots ». In : *Proceedings of the Intelligent Vehicles '92 Symposium*. Juin 1992, p. 293-297. DOI : 10.1109/IVS.1992.252274.

[22] Sven KOENIG et Reid G SIMMONS. « Real-time search in non-deterministic domains ». In : *IJCAI*. Citeseer. 1995, p. 1660-1669.

[23] Johann BORENSTEIN et Liqiang FENG. « Measurement and correction of systematic odometry errors in mobile robots ». In : *IEEE Transactions on robotics and automation* 12.6 (1996), p. 869-880.

[24] Philippe DECAUDIN. « Cartoon-looking rendering of 3D-scenes ». In : *Syntim Project Inria* 6 (1996).

[25] Sebastian THRUN et Arno BÜCKEN. « Integrating grid-based and topological maps for mobile robot navigation ». In : *Proceedings of the National Conference on Artificial Intelligence*. 1996, p. 944-951.

[26] B. YAMAUCHI. « A frontier-based approach for autonomous exploration ». In : *Proceedings 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA'97. 'Towards New Computational Principles for Robotics and Automation'*. 1997, p. 146-151.

[27] Brian YAMAUCHI. « A frontier-based approach for autonomous exploration ». In : *Proceedings 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA'97. 'Towards New Computational Principles for Robotics and Automation'* (1997), p. 146-151. DOI : 10.1109/CIRA.1997.613851.

[28] Mehran SAHAMI et al. « A Bayesian approach to filtering junk e-mail ». In : *Learning for Text Categorization : Papers from the 1998 workshop*. T. 62. Madison, Wisconsin. 1998, p. 98-105.

[29] Brian YAMAUCHI et al. « Frontier-based exploration using multiple robots ». In : *Agents*. T. 98. 1998, p. 47-53.

[30] Israel A WAGNER, Michael LINDENBAUM et Alfred M BRUCKSTEIN. « Distributed covering by ant-robots using evaporating traces ». In : *IEEE Transactions on Robotics and Automation* 15.5 (1999), p. 918-933.

[31] Brian YAMAUCHI. « Decentralized coordination for multirobot exploration ». In : *Robotics and Autonomous Systems* 29.2-3 (1999), p. 111-118.

[32] Surachai SUKSAKULCHAI et al. « Mobile robot localization using an electronic compass for corridor environment ». In : *Smc 2000 conference proceedings. 2000 ieee international conference on systems, man and cybernetics.'cybernetics evolving to systems, humans, organizations, and their complex interactions'(cat. no. 0*. T. 5. IEEE. 2000, p. 3354-3359.

[33] B. ALLO, C. GUETTIER et N. LECUBIN. « A demonstration of dedicated constraint-based planning within agent-based architectures for autonomous aircraft ». In : *Proceeding of the 2001 IEEE International Symposium on Intelligent Control (ISIC '01) (Cat. No.01CH37206)*. Sept. 2001, p. 31-38. DOI : 10.1109/ISIC.2001.971480.

[34] D. S. APOSTOLOPOULOS et al. « Robotic Antarctic meteorite search : outcomes ». In : *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*. T. 4. Mai 2001, 4174-4179 vol.4. DOI : 10.1109/ROBOT.2001.933270.

[35] Yoav GABRIELY et Elon RIMON. « Spanning-tree based coverage of continuous areas by a mobile robot ». In : *Annals of mathematics and artificial intelligence* 31.1 (2001), p. 77-98.

[36] Nicholas ROY et Gregory DUDEK. « Collaborative robot exploration and rendezvous : Algorithms, performance bounds and observations ». In : *Autonomous Robots* 11.2 (2001), p. 117-136.

[37] Vladimir YANOVSKI, Israel A WAGNER et Alfred M BRUCKSTEIN. « Vertex-Ant-Walk–A robust method for efficient exploration of faulty graphs ». In : *Annals of Mathematics and Artificial Intelligence* 31.1 (2001), p. 99-112.

[38] Sarah F FRISKEN et Ronald N PERRY. « Simple and efficient traversal methods for quadtrees and octrees ». In : *Journal of Graphics Tools* 7.3 (2002), p. 1-11.

[39] M. BERHAULT et al. « Robot exploration with combinatorial auctions ». In : *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*. T. 2. Oct. 2003, 1957-1962 vol.2. DOI : 10.1109/IROS.2003.1248932.

[40] François SEMPÉ et Alexis DROGOUL. « Adaptive patrol for a group of robots ». In : *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)(Cat. No. 03CH37453)*. T. 3. IEEE. 2003, p. 2865-2869.

[41] Krzysztof DIKS et al. « Tree exploration with little memory ». In : *Journal of Algorithms* 51.1 (2004), p. 38-63. ISSN : 0196-6774. DOI : https://doi.org/10.1016/j.jalgor.2003.10.002.

[42] W.G.M GERAETS, A.N van DAATSELAAR et J.G.C VERHEIJ. « An efficient filling algorithm for counting regions ». In : *Computer Methods and Programs in Biomedicine* 76.1 (2004), p. 1-11. ISSN : 0169-2607. DOI : https://doi.org/10.1016/j.cmpb.2003.09.004.

[43] Victoria HODGE et Jim AUSTIN. « A survey of outlier detection methodologies ». In : *Artificial intelligence review* 22.2 (2004), p. 85-126.

[44] Gerhard K KRAETZSCHMAR, Guillem Pages GASSULL et Klaus UHL. « Probabilistic quadtrees for variable-resolution mapping of large environments ». In : *IFAC Proceedings Volumes* 37.8 (2004), p. 675-680.

[45] Frank LOSASSO, Frédéric GIBOU et Ron FEDKIW. « Simulating water and smoke with an octree data structure ». In : *ACM SIGGRAPH 2004 Papers*. 2004, p. 457-462.

[46] Michael MONTEMERLO et Sebastian THRUN. « A multi-resolution pyramid for outdoor robot terrain perception ». In : *AAAI*. T. 4. 2004, p. 464-469.

[47] Carlos HERNÁNDEZ et Pedro MESEGUER. « LRTA*(k) ». In : *Proceedings of the 19th international joint conference on Artificial intelligence*. 2005, p. 1238-1243.

[48] Agostino MARTINELLI, Frederic PONT et Roland SIEGWART. « Multi-robot localization using relative observations ». In : *Proceedings of the 2005 IEEE international conference on robotics and automation*. IEEE. 2005, p. 2797-2802.

[49] Sebastian THRUN et Yufeng LIU. « Multi-robot SLAM with sparse extended information filers ». In : *Robotics Research. The Eleventh International Symposium*. Springer. 2005, p. 254-266.

[50] Noa AGMON, Noam HAZON et Gal A KAMINKA. « Constructing spanning trees for efficient multi-robot coverage ». In : *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.* IEEE. 2006, p. 1698-1703.

[51] Hugh DURRANT-WHYTE et Tim BAILEY. « Simultaneous localization and mapping : part I ». In : *IEEE robotics & automation magazine* 13.2 (2006), p. 99-110.

[52] Noam HAZON, Fabrizio MIELI et Gal A KAMINKA. « Towards robust on-line multi-robot coverage ». In : *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.* IEEE. 2006, p. 1710-1715.

[53] Talita MENEZES, Patricia TEDESCO et Geber RAMALHO. « Negotiator agents for the patrolling task ». In : *Advances in Artificial Intelligence-IBERAMIA-SBIA 2006*. Springer, 2006, p. 48-57.

[54] Ibrahim ZUNAIDI et al. « Positioning system for 4-wheel mobile robot : encoder, gyro and accelerometer data fusion with error model method ». In : *CMU. Journal* 5.1 (2006), p. 1.

[55] H Jacky CHANG et al. « Multi-robot SLAM with topological/metric maps ». In : *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2007, p. 1467-1472.

[56] H. N. CHU et al. « Swarm Approaches for the Patrolling Problem, Information Propagation vs. Pheromone Evaporation ». In : 1 (oct. 2007), p. 442-449. ISSN : 2375-0197. DOI : 10.1109/ICTAI.2007.80.

[57] Nathaniel FAIRFIELD, George KANTOR et David WETTERGREEN. « Real-time SLAM with octree evidence grids for exploration in underwater tunnels ». In : *Journal of Field Robotics* 24.1-2 (2007), p. 03-21.

[58] Nazim FATÈS. « Decentralised gathering on a discrete field : coupling reaction-diffusion and chemotaxis as social amoebae do ». In : (2007).

[59] E. FERRANTI, N. TRIGONI et M. LEVENE. « Brick Mortar : an on-line multi-agent exploration algorithm ». In : *Proceedings 2007 IEEE International Conference on Robotics and Automation*. Avr. 2007, p. 761-767. DOI : 10.1109/ROBOT.2007.363078.

[60] Ettore FERRANTI, Niki TRIGONI et Mark LEVENE. « Brick & Mortar : an on-line multi-agent exploration algorithm. » In : *ICRA*. 2007, p. 761-767.

[61] S. GARNIER et al. « Alice in Pheromone Land : An Experimental Setup for the Study of Ant-like Robots ». In : *2007 IEEE Swarm Intelligence Symposium*. Avr. 2007, p. 37-44. DOI : 10.1109/SIS.2007.368024.

[62] Simon GARNIER et al. « Alice in Pheromone Land : Collective Decision Making by a Group of Ant-like Robots ». In : *regulation* 2 (2007), p. 3.

[63] Martijn N Rooker et Andreas Birk. « Multi-robot exploration under the constraints of wireless networking ». In : *Control Engineering Practice* 15.4 (2007), p. 435-445.

[64] Jo-Anne Ting, Evangelos Theodorou et Stefan Schaal. « A Kalman filter for robust outlier detection ». In : *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2007, p. 1514-1519.

[65] Adrien Angeli. « Détection visuelle de fermeture de boucle et applications à la localisation et cartographie simultanées ». Thèse de doct. Université Pierre et Marie Curie-Paris VI, 2008.

[66] Michael Mossinghoff, Jeffry L Hirst et John Harris. *Combinatorics and Graph Theory*. Springer-Verlag New York, 2008.

[67] Denis G Pelli et Katharine A Tillman. « The uncrowded window of object recognition ». In : *Nature neuroscience* 11.10 (2008), p. 1129-1135.

[68] Peter M Roth et Martin Winter. « Survey of appearance-based methods for object recognition ». In : *Inst. for computer graphics and vision, Graz University of Technology, Austria, technical report ICGTR0108 (ICG-TR-01/08)* (2008).

[69] KS Senthilkumar et KK Bharadwaj. « Spanning tree based terrain coverage by multi robots in unknown environments ». In : *2008 Annual IEEE India Conference*. T. 1. IEEE. 2008, p. 120-125.

[70] Kai M Wurm, Cyrill Stachniss et Wolfram Burgard. « Coordinated multi-robot exploration using a segmentation of the environment ». In : *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2008, p. 1160-1165.

[71] Antoni Burguera, Yolanda González et Gabriel Oliver. « Sonar sensor models and their application to mobile robot localization ». In : *Sensors* 9.12 (2009), p. 10217-10243.

[72] Arnaud Glad et al. « Self-organization of patrolling-ant algorithms ». In : *2009 Third IEEE International Conference on Self-Adaptive and Self-Organizing Systems*. IEEE. 2009, p. 61-70.

[73] Christophe Guettier, Francois Lucas et Patrick Siarry. « Hybridisation of Constraint Solving with an Ant Colony Algorithm for Vehicle On-Line Path Planning ». In : 2009.

[74] Jaewoong Kim et Sukhan Lee. « Fast neighbor cells finding method for multiple octree representation ». In : *2009 IEEE International Symposium on Computational Intelligence in Robotics and Automation-(CIRA)*. IEEE. 2009, p. 540-545.

[75] Rainer Kümmerle et al. « On measuring the accuracy of SLAM algorithms ». In : *Autonomous Robots* 27.4 (2009), p. 387-407.

[76] A. Marjovi et al. « Multi-robot exploration and fire searching ». In : *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Oct. 2009, p. 1929-1934. doi : 10.1109/IROS.2009.5354598.

[77] T. -. N. Nguyen et al. « Optimized grid-based environment perception in advanced driver assistance systems ». In : *2009 IEEE Intelligent Vehicles Symposium*. Juin 2009, p. 425-430. doi : 10.1109/IVS.2009.5164315.

[78] Brian Williams et al. « A comparison of loop closing techniques in monocular SLAM ». In : *Robotics and Autonomous Systems* 57.12 (2009), p. 1188-1197.

[79] Dorigo. *Ant colony optimization*. Springer, 2010.

[80] Marco Dorigo et Mauro Birattari. *Ant colony optimization*. Springer, 2010.

[81] Nazim Fatès. « Solving the decentralised gathering problem with a reaction–diffusion–chemotaxis scheme ». In : *Swarm Intelligence* 4.2 (2010), p. 91-115.

[82] David Portugal et Rui Rocha. « Msp algorithm : multi-robot patrolling based on territory allocation using balanced graph partitioning ». In : *Proceedings of the 2010 ACM symposium on applied computing*. 2010, p. 1271-1276.

[83] Antoine Bautin, Olivier Simonin et François Charpillet. « Towards a communication free coordination for multi-robot exploration ». In : *6th National conference on control architectures of robots*. 2011, 8-p.

[84] Nicolas Beaufort et al. « Interactive table to study interactions between swarm of robots and active environments ». In : (2011).

[85] Peter Brass et al. « Multirobot tree and graph exploration ». In : *IEEE Transactions on Robotics* 27.4 (2011), p. 707-717.

[86] Eric Heitz et Fabrice Neyret. « A Visibility-Aware Model for Pre-Filtering and Rendering Surfaces in Real-Time ». Thèse de doct. INRIA, 2011.

[87] Miroslav Kulich, Jan Faigl et Libor Přeučil. « On distance utility in the exploration task ». In : *2011 IEEE International Conference on Robotics and Automation*. 2011, p. 4455-4460.

[88] David Portugal et Rui Rocha. « A Survey on Multi-robot Patrolling Algorithms ». In : *Technological Innovation for Sustainability*. Sous la dir. de Luis M. Camarinha-Matos. Berlin, Heidelberg : Springer Berlin Heidelberg, 2011, p. 139-146. isbn : 978-3-642-19170-1.

[89] Antoine Bautin, Olivier Simonin et François Charpillet. « Minpos : A novel frontier allocation algorithm for multi-robot exploration ». In : *International conference on intelligent robotics and applications*. Springer. 2012, p. 496-508.

[90] J. Faigl, M. Kulich et L. Přeučil. « Goal assignment using distance cost in multi-robot exploration ». In : *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Oct. 2012, p. 3741-3746. doi : 10.1109/IROS.2012.6385660.

[91] Laëtitia Matignon, Laurent Jeanpierre et Abdel-Illah Mouaddib. « Coordinated multi-robot exploration under communication constraints using decentralized markov decision processes ». In : *Twenty-sixth AAAI conference on artificial intelligence*. 2012.

[92] KS Senthilkumar et Kamal Kant Bharadwaj. « Multi-robot exploration and terrain coverage in an unknown environment ». In : *Robotics and Autonomous Systems* 60.1 (2012), p. 123-132.

[93] Antoine Bautin et al. « Cart-O-matic project : autonomous and collaborative multi-robot localization, exploration and mapping. 5th Workshop on Planning, Perception and Navigation for Intelligent Vehicles. » In : *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Tokyo, Japan, nov. 2013, p. x.

[94] Enric Galceran et Marc Carreras. « A survey on coverage path planning for robotics ». In : *Robotics and Autonomous Systems* 61.12 (2013), p. 1258-1276. issn : 0921-8890. doi : 10.1016/j.robot.2013.09.004.

[95] Armin Hornung et al. « OctoMap : An efficient probabilistic 3D mapping framework based on octrees ». In : *Autonomous robots* 34.3 (2013), p. 189-206.

[96] Maria Teresa LAZARO et al. « Multi-robot SLAM using condensed measurements ». In : *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2013, p. 1069-1076.

[97] Hoang Huu VIET et al. « BA* : an online complete coverage algorithm for cleaning robots ». In : *Applied Intelligence* 39.2 (sept. 2013), p. 217-235. ISSN : 1573-7497. DOI : 10.1007/s10489-012-0406-4.

[98] Jan FAIGL, Olivier SIMONIN et Francois CHARPILLET. « Comparison of task-allocation algorithms in frontier-based multi-robot exploration ». In : *European Conference on Multi-Agent Systems*. Springer. 2014, p. 101-110.

[99] J JESSUP, Sidney N GIVIGI et Alain BEAULIEU. « Merging of octree based 3d occupancy grid maps ». In : *2014 IEEE International Systems Conference Proceedings*. IEEE. 2014, p. 371-377.

[100] J JESSUP, Sidney Nascimento GIVIGI et Alain BEAULIEU. « Robust and efficient multi-robot 3D mapping with octree based occupancy grids ». In : *2014 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE. 2014, p. 3996-4001.

[101] Yan Yan ZHANG, Yan Chun SHEN et Li Ni MA. « Pathfinding algorithm of 3D scene based on navigation mesh ». In : *Advanced materials research*. T. 1030. Trans Tech Publ. 2014, p. 1745-1750.

[102] M. ANDRIES et F. CHARPILLET. « Multi-robot taboo-list exploration of unknown structured environments ». In : *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Sept. 2015, p. 5195-5201. DOI : 10.1109/IROS.2015.7354109.

[103] M. ANDRIES et F. CHARPILLET. « Multi-robot taboo-list exploration of unknown structured environments ». In : *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Sept. 2015, p. 5195-5201. DOI : 10.1109/IROS.2015.7354109.

[104] Tsz-Chiu AU et Ty NGUYEN. « Augmented Motion Plans for Planning in Uncertain Terrains ». In : (2015).

[105] J. FAIGL et M. KULICH. « On benchmarking of frontier-based multi-robot exploration strategies ». In : *2015 European Conference on Mobile Robots (ECMR)*. Sept. 2015, p. 1-8. DOI : 10.1109/ECMR.2015.7324183.

[106] Jan FAIGL, Olivier SIMONIN et Francois CHARPILLET. « Comparison of Task-Allocation Algorithms in Frontier-Based Multi-robot Exploration ». In : *Multi-Agent Systems*. Sous la dir. de Nils BULLING. Cham : Springer International Publishing, 2015, p. 101-110. ISBN : 978-3-319-17130-2.

[107] Jorge FUENTES-PACHECO, José RUIZ-ASCENCIO et Juan Manuel RENDÓN-MANCHA. « Visual simultaneous localization and mapping : a survey ». In : *Artificial intelligence review* 43.1 (2015), p. 55-81.

[108] Hoang Huu VIET et al. « BoB : an online coverage approach for multi-robot systems ». In : *Applied Intelligence* 42.2 (mars 2015), p. 157-173. ISSN : 1573-7497. DOI : 10.1007/s10489-014-0571-8.

[109] E. VINCENT, A. SINI et F. CHARPILLET. « Audio source localization by optimal control of a mobile robot ». In : *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. Avr. 2015, p. 5630-5634. DOI : 10.1109/ICASSP.2015.7179049.

[110] Christophe GUETTIER et François LUCAS. « A constraint-based approach for planning unmanned aerial vehicle activities ». In : *The Knowledge Engineering Review* 31.5 (2016), p. 486-497.

[111] George T HEINEMAN, Gary POLLICE et Stanley SELKOW. *Algorithms in a nutshell : A practical guide*. " O'Reilly Media, Inc.", 2016.

[112] Max JADERBERG et al. « Reinforcement learning with unsupervised auxiliary tasks ». In : *arXiv preprint arXiv :1611.05397* (2016).

[113] Mehdi KHAMASSI et Stéphane DONCIEUX. « Nouvelles approches en robotique cognitive ». In : *Intellectica-La revue de l'Association pour la Recherche sur les sciences de la Cognition (ARCo)* 2016.65 (2016), p. 7-25.

[114] Piotr MIROWSKI et al. « Learning to navigate in complex environments ». In : *arXiv preprint arXiv :1611.03673* (2016).

[115] Q. V. NGUYEN et al. « Localizing an intermittent and moving sound source using a mobile robot ». In : *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Oct. 2016, p. 1986-1991. DOI : 10.1109/IROS.2016.7759313.

[116] Quan V NGUYEN et al. « Localizing an intermittent and moving sound source using a mobile robot ». In : *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2016, p. 1986-1991.

[117] Joseph REDMON et al. *You Only Look Once : Unified, Real-Time Object Detection*. 2016. arXiv : 1506.02640 [cs.CV].

[118] Cyrill STACHNISS, John J LEONARD et Sebastian THRUN. « Simultaneous localization and mapping ». In : *Springer Handbook of Robotics*. Springer, 2016, p. 1153-1176.

[119] CM SUKANYA, Roopa GOKUL et Vince PAUL. « A survey on object recognition methods ». In : *International Journal of Science, Engineering and Computer Technology* 6.1 (2016), p. 48.

[120] Renaud DUBÉ et al. « An online multi-robot SLAM system for 3D LiDARs ». In : *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2017, p. 1004-1011.

[121] L. FERMIN-LEON, J. NEIRA et J. A. CASTELLANOS. « TIGRE : Topological graph based robotic exploration ». In : *2017 European Conference on Mobile Robots (ECMR)*. 2017, p. 1-6. DOI : 10.1109/ECMR.2017.8098718.

[122] U. JAIN, R. TIWARI et W. W. GODFREY. « Comparative study of frontier based exploration methods ». In : *2017 Conference on Information and Communication Technology (CICT)*. Nov. 2017, p. 1-5. DOI : 10.1109/INFOCOMTECH.2017.8340589.

[123] Sergiu NEDEVSCHI et al. « Online cross-calibration of camera and lidar ». In : *2017 13th IEEE International Conference on Intelligent Computer Communication and Processing (ICCP)*. IEEE. 2017, p. 295-301.

[124] Łukasz WIŚNIEWSKI et al. « Mobility and terrain accessibility analysis for Hopter—An underactuated mobile robot for planetary exploration ». In : *Proceedings of the ESA Workshop on Advanced Space Technologies for Robotics and Automation (ASTRA), Leiden, The Netherlands*. 2017, p. 20-22.

[125] Haiming GAO et al. « Autonomous indoor exploration via polygon map construction and graph-based SLAM using directional endpoint features ». In : *IEEE Transactions on Automation Science and Engineering* 16.4 (2018), p. 1531-1542.

[126] Jose J Lopez-Perez et al. « Distributed Multirobot Exploration Based on Scene Partitioning and Frontier Selection ». In : *Mathematical Problems in Engineering* 2018 (2018).

[127] Kevin Osanlou et al. « Constrained Shortest Path Search with Graph Convolutional Neural Networks ». In : *Workshop on Planning and Learning (PAL-18) Stockholm, Sweden*. 2018.

[128] Abhishek Kadian* et al. « Are We Making Real Progress in Simulated Environments ? Measuring the Sim2Real Gap in Embodied Visual Navigation ». In : *arXiv :1912.06321*. 2019.

[129] K. Albina et S. G. Lee. « Hybrid Stochastic Exploration Using Grey Wolf Optimizer and Coordinated Multi-Robot Exploration Algorithms ». In : *IEEE Access* 7 (2019), p. 14246-14255. ISSN : 2169-3536. DOI : 10.1109/ACCESS.2019.2894524.

[130] Joseph Azeta et al. « Obstacle detection using ultrasonic sensor for a mobile robot ». In : *IOP Conference Series : Materials Science and Engineering*. T. 707. 1. IOP Publishing. 2019, p. 012012.

[131] Edward Beeching et al. « Deep reinforcement learning on a budget : 3d control and reasoning without a supercomputer ». In : *arXiv preprint arXiv :1904.01806* (2019).

[132] Facundo Benavides Olivera. « Multi-robot exploration under non-ideal communication conditions ». 2019ESAE0006. Thèse de doct. 2019.

[133] Marco Dorigo et Thomas Stützle. « Ant colony optimization : overview and recent advances ». In : *Handbook of metaheuristics*. Springer, 2019, p. 311-351.

[134] Nicolas Gauville et François Charpillet. « Exploration et couverture par stigmergie d'un environnement inconnu avec une flotte de robots autonomes réactifs ». In : 27 (juill. 2019), p. 77-86.

[135] Elizabeth A Jensen et Maria Gini. « Effects of communication restriction on online multi-robot exploration in bounded environments ». In : *Distributed Autonomous Robotic Systems*. Springer, 2019, p. 469-483.

[136] Philippe Lambert, Lionel Lapierre et Didier Crestani. « An approach for Fault Tolerant and Performance Guarantee Autonomous Robotic Mission ». In : *AHS : Adaptive Hardware and Systems*. Colchester, United Kingdom, 2019, p. 87-94. DOI : 10.1109/AHS.2019.00009. URL : https://hal.archives-ouvertes.fr/hal-02160494.

[137] Manolis Savva* et al. « Habitat : A Platform for Embodied AI Research ». In : *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 2019.

[138] Michael Stradner et Gerald Steinbauer. « Lifting robot exploration to 3d environments ». In : *2019 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*. IEEE. 2019, p. 1-8.

[139] Devendra Singh Chaplot et al. « Object goal navigation using goal-oriented semantic exploration ». In : *Advances in Neural Information Processing Systems* 33 (2020), p. 4247-4258.

[140] Di Deng et al. « Frontier-based automatic-differentiable information gain measure for robotic exploration of unknown 3D environments ». In : *arXiv preprint arXiv :2011.05288* (2020).

[141] William Qi et al. « Learning to move with affordance maps ». In : *arXiv preprint arXiv :2001.02364* (2020).

[142] Zhuping Wang et al. « Intelligent vehicle self-localization based on double-layer features and multilayer LIDAR ». In : *IEEE Transactions on Intelligent Vehicles* 5.4 (2020), p. 616-625.

[143] Raj Korpan et Susan L Epstein. « Hierarchical freespace planning for navigation in unfamiliar worlds ». In : *Proceedings of the International Conference on Automated Planning and Scheduling*. T. 31. 2021, p. 663-672.

[144] Quentin Massone, Sebastien Druon et Jean Triboulet. « An original 3D reconstruction method using a conical light and a camera in underwater caves ». In : *2021 4th International Conference on Control and Computer Vision*. 2021, p. 126-134.

[145] Santhosh Kumar Ramakrishnan et al. « Habitat-Matterport 3D Dataset (HM3D) : 1000 Large-scale 3D Environments for Embodied AI ». In : *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*. 2021. URL : https://openreview.net/forum?id=-v4OuqNs5P.

[146] Andrew Szot et al. « Habitat 2.0 : Training Home Assistants to Rearrange their Habitat ». In : *arXiv preprint arXiv :2106.14405* (2021).

[147] Tien Quang Tran, Andreas Becker et Damian Grzechca. « Environment Mapping Using Sensor Fusion of 2D Laser Scanner and 3D Ultrasonic Sensor for a Real Mobile Robot ». In : *Sensors* 21.9 (2021), p. 3184.

[148] Thomas H Cormen et al. *Introduction to algorithms*. MIT press, 2022, p. 561-579.

[149] Panagiotis Rousseas et al. « Indoor Visual Exploration with Multi-Rotor Aerial Robotic Vehicles ». In : *Sensors* 22.14 (2022), p. 5194.

[150] Tomáš Juchelka. « Exploration algorithms in a polygonal domain ». Thèse de doct.