

RAPPORT DE PROJET DE FIN D'ÉTUDES

GRENOBLE INP Esisar 2022/2023

Titre du projet

Navigation Autonome en Environnement Inconnu

Nom et adresse de l'entreprise

Laboratoire de Conception et d'Intégration des Systèmes

**50 rue Barthélémy de Laffemas,
CS 10054
26902 Valence Cedex 9**

Nom et prénom de l'étudiant

Loïs RAJAONSON

Dates du stage	01/02/2023 - 30/06/2023
Spécialité	IR&C - SCI
Tuteur Entreprise	Simon GAY, Ionela PRODAN
Tuteur ESISAR	Quentin GIORGIO

RAPPORT DE PROJET DE FIN D'ETUDES

GRENOBLE INP Esisar 2022/2023

Mots clés : Robot mobile, Algorithme bio-inspiré, SLAM (Simultaneous Localization And Mapping), courbes B-Splines, Navigation autonome

Résumé :

Au cours de ce stage de recherche au laboratoire LCIS, j'ai travaillé sur des algorithmes de cartographie et de localisation, en vue de permettre la navigation autonome d'un robot mobile. J'ai exploré en détail deux algorithmes, l'un bio-inspiré, l'autre basé sur la représentation d'obstacle par des courbes appelées B-Splines. J'ai implémenté l'algorithme bio-inspiré en simulation, puis directement sur le robot Turtlebot3 Burger. Enfin, j'ai identifié des pistes de recherche possibles pour arriver à intégrer une génération de trajectoires plus avancée dans le modèle bio-inspiré.

Keywords : Mobile robot, Bio-inspired algorithm, SLAM (Simultaneous Localization And Mapping), B-Spline curves, Autonomous navigation

Abstract :

During this internship at the LCIS Laboratory, I worked on localization and mapping algorithms for autonomous navigation of mobile robots. I studied two algorithms in detail: one bio-inspired, the other based on B-Spline curves as a means of representing obstacles. I have implemented the bio-inspired algorithm in simulation, then directly on the TurtleBot3 robot itself. Finally, I have identified possible ways to achieve advanced trajectory generation in the bio-inspired model, improving its navigation capabilities.

Remerciements

J'aimerais exprimer mes sincères remerciements à Simon Gay et Ionela Prodan, pour m'avoir offert l'opportunité d'effectuer ce stage au sein du LCIS, et pour leur accompagnement précieux tout au long de cette expérience.

Je tiens également à remercier l'équipe CO4SYS, ainsi que tous les autres membres du laboratoire, pour leur accueil chaleureux, et l'ambiance de travail agréable qu'ils ont su créer.

Je remercie aussi l'équipe pédagogique de l'ESISAR pour ces trois années de formation. Leur engagement et leur enseignement ont nourri ma curiosité et ont été une source d'inspiration qui m'a encouragé à entreprendre ce stage avec confiance..

Je remercie enfin ma famille et mes proches, qui m'ont toujours été un soutien inconditionnel.

I. Sommaire

Remerciements.....	2
I. Sommaire.....	3
II. Introduction.....	4
III. Contexte et enjeux.....	5
Le laboratoire.....	5
Motivation.....	5
Problème de recherche: SLAM.....	6
Le sujet.....	6
IV. Cahier des charges.....	8
V. Présentation du déroulement.....	9
1. Etat de l'art et choix techniques.....	9
Revue des méthodes de SLAM.....	9
Etude détaillée: Modèle de navigation bio-inspiré prédictif.....	13
Parenthèse mathématique: les courbes B-splines.....	16
Etude détaillée: B-Spline Curves SLAM.....	17
Etude détaillée: Comparaison des approches.....	19
Outils logiciels et matériels.....	22
2. Conception/Développement.....	25
Modèle Bio-inspiré.....	25
B-Splines SLAM.....	28
Piste explorée: Intersection de courbes.....	29
VI. Bilan.....	32
Coût global.....	32
Résultats obtenus, impact.....	32
VII. Conclusion.....	33
VIII. Références.....	34

II. Introduction

Dans le cadre de ma formation d'ingénieur à l'ESISAR, j'ai réalisé mon stage de fin d'études au Laboratoire de Conception et d'Intégration des Systèmes (LCIS), de février à juin 2023. Au travers de ce rapport de stage, je partage mon expérience au sein de l'équipe CO4SYS, qui s'intéresse à plusieurs aspects de la robotique mobile.

Mon projet de fin d'études concerne les thématiques de recherche de l'équipe, particulièrement de Simon GAY, chercheur en apprentissage développemental sur des robots mobiles, et de Ionela PRODAN, chercheuse en théorie du contrôle. Ce stage vise à apporter une contribution aux travaux de l'équipe.

Ce document présente les thématiques étudiées, et retrace les travaux effectués au cours de ce stage.

III. Contexte et enjeux

Le laboratoire



Le Laboratoire de Conception et d'Intégration des Systèmes est un laboratoire de recherche public situé à Valence. Il est rattaché à l'Institut polytechnique de Grenoble (Grenoble INP), lui-même une composante de l'Université Grenoble-Alpes. L'activité du laboratoire s'étend sur de nombreux domaines de recherche: technologies sans fil, théorie du contrôle, systèmes multi-agents, sécurité des systèmes embarqués... Le laboratoire compte plus d'une soixantaine de chercheurs.

Mon stage se situe au sein de l'équipe CO4SYS, qui s'intéresse aux robots mobiles,, à leurs interactions entre eux et avec l'environnement. L'équipe peut être divisée en deux "branches": d'un côté, une partie de l'équipe travaille sur les systèmes multi-agents, l'apprentissage développemental et l'intelligence artificielle distribuée. De l'autre, le restant de l'équipe est issu de la théorie du contrôle.

Motivation

La robotique englobe un large champ d'applications, dans l'industrie, l'armée, les transports, les usages domestiques. Le domaine de la recherche en robotique peut être subdivisé en deux catégories très distinctes: D'un côté, la robotique industrielle, qui concerne principalement des robots fixes, utilisés dans des environnements de production. Ces robots industriels sont conçus pour effectuer des tâches répétitives à une cadence et une précision considérables. Utilisés dans de nombreuses tâches sur les lignes de production (soudure, assemblage, peinture...), ils dominent le marché de l'automatisation industrielle. Leur utilisation est généralement limitée à des environnements contrôlés et bien connus. D'un autre côté, la robotique mobile, qui se concentre sur des robots capables de se déplacer et d'interagir avec un environnement incertain, voire inconnu. Les

robots mobiles sont conçus pour s'adapter à des situations changeantes, réagir à des perturbations extérieures, ce qui nécessite de mettre en place des stratégies sophistiquées de localisation, de planification de trajectoire, de cartographie. Le champ d'applications de la robotique mobile est vaste, de la logistique à l'agriculture, en passant par les transports ou même l'exploration spatiale. Selon les prévisions de nombreux experts, le marché de la robotique mobile est en passe de connaître une expansion significative dans la décennie à venir[1].

Problème de recherche: SLAM

Simultaneous Localization And Mapping, ou SLAM, est un des problèmes de recherche centraux pour les robots mobiles autonomes. Comme son nom l'indique, le but est de simultanément construire une carte de l'environnement inconnu, et de pouvoir s'y localiser. Pour cela, le robot autonome ne dispose que des données provenant des capteurs qu'il embarque. Un algorithme pour SLAM est nécessaire dès qu'on souhaite déployer un robot dans un milieu inconnu où utiliser un système de localisation externe (ex: GPS) est peu précis, voire impossible.

SLAM est un problème difficile, à cause de la dépendance circulaire forte entre localisation et cartographie. En essence, il s'agit d'un problème de l'œuf et de la poule: D'un côté, pour se localiser dans l'environnement à partir de ses capteurs, l'agent a besoin d'une carte. Mais de l'autre côté, lorsque le robot détecte un obstacle avec ses capteurs, il a besoin de sa position globale pour l'ajouter à la carte. Ainsi, les erreurs s'accumulent et s'aggravent très vite: une petite erreur de localisation causera une erreur dans la construction de la carte. A l'étape suivante, le robot utilisera la carte légèrement erronée pour se localiser, ce qui causera une erreur plus importante dans la localisation, et ainsi de suite, jusqu'à rendre la carte inutilisable.

Le sujet

Le sujet de mon stage porte sur la navigation dans un environnement inconnu. En particulier, le stage permet d'amorcer une liaison entre les deux parties de l'équipe CO4SYS (Algorithmes bio-inspirés et théorie du contrôle). Dans cette perspective, mon travail consiste à étudier plusieurs approches au problème SLAM[2][3] issues de ces deux

domaines de recherche, dans le but d'explorer de nouvelles pistes et de favoriser une meilleure compréhension du sujet. Bien qu'un but final précis soit difficile à fixer dans un stage de recherche, l'accent est mis sur l'identification et l'exploration de quelques idées prometteuses.

IV. Cahier des charges

Attentes différentes d'un stage de recherche

Étant un projet de recherche, mon stage n'est pas encadré par un cahier des charges fixe, contrairement à l'industrie, où les contraintes et objectifs sont souvent définis en amont. La recherche offre une flexibilité, et permet des évolutions et ajustements constants.

Plutôt que de suivre un cahier des charges rigide, mon stage a débuté par une série de pistes prometteuses à explorer. Au fil de l'avancement, de nouvelles idées émergent, de nouvelles pistes se dessinent, d'autres peuvent être abandonnées, et la destination précise du projet soit connue.

Pistes de recherche

Parmi les pistes initiales figuraient:

- L'études d'approches de localisation et de cartographie, et en particulier d'une approche bio-inspirée proposée par Simon GAY, chercheur de l'équipe et encadrant de ce stage.
- L'implémentation de telles approches en simulation, puis sur un vrai robot
- L'étude de la possibilité d'ajouter des capacités de navigation autonome à ces approches, qui ne sont pour l'instant capables que de construire une carte et de se localiser.

V. Présentation du déroulement

1. Etat de l'art et choix techniques

Revue des méthodes de SLAM

Le problème SLAM a été l'objet de nombreuses publications au fil des années. Il existe une multitude de techniques adaptées à des cas d'usages variés, (environnement terrestre, aérien ou même marin, différents capteurs, différentes échelles...). Les principales façons de représenter un environnement inconnu peuvent être divisées en deux catégories: Les cartes dites "métriques", et les cartes dites "topologiques".

Cartes métriques et fermeture de boucle

Dans une carte métrique, la représentation est construite de manière à refléter la géométrie de l'environnement. Dans une telle carte, les obstacles sont positionnés dans un référentiel global, et la distance estimée entre les différents éléments est sauvegardée. Les cartes métriques peuvent prendre de nombreuses formes, notamment:

Carte par points de repère (landmarks): Les cartes par points de repère stockent des caractéristiques de l'environnement détectées par les capteurs du robot. Ces caractéristiques peuvent être artificielles (tags, réflecteurs, placés par un humain) ou extraites directement des acquisitions du capteur (détection d'un coin, de lignes verticales, etc.). Ces cartes sont compactes car elles représentent peu d'éléments, mais elles nécessitent un environnement "dense" en points de repères pour permettre une localisation efficace.

Carte discrète (discrete grid): Les cartes par grille discrètes décomposent le monde en petites subdivisions. Les approches les plus utilisées représentent dans chaque subdivision la probabilité que l'espace soit occupé. La probabilité d'occupation augmente si plusieurs acquisitions du capteur indiquent la présence d'un obstacle à la position correspondante. Ces cartes se présentent simplement comme des matrices, donc rapides à modifier et à lire. Cet avantage en performances est sûrement la raison pour laquelle

cette méthode est la plus répandue. Cependant, la discréétisation des mesures du capteur peuvent être à l'origine de faibles erreurs de localisation. De plus, cette méthode n'est pas adaptée à la représentation de grands espaces, car elle demanderait trop de mémoire, ou une réduction importante de la résolution.

Carte continue: dans des représentations continues, ou géométriques, les obstacles sont représentés par des fonctions continues ou des objets géométriques (lignes, polygones). Ces cartes sont à la fois compactes et précises, mais sont difficiles à exploiter car leur construction en temps réel requiert une puissance de calcul élevée, et il est complexe de déterminer si un point appartient à une zone libre ou une zone occupée de l'espace.

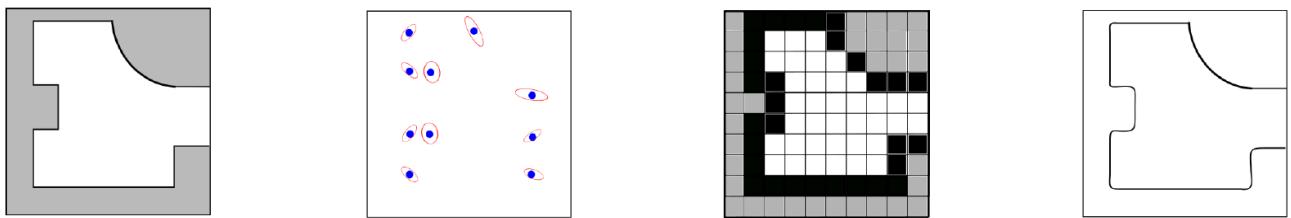


Figure 1. De gauche à droite: L'environnement réel, une carte par points de repère, une grille discrète, et une carte continue

Il faut également noter que dans le cas des cartes métriques, les méthodes de SLAM les plus robustes et les plus utilisées reposent sur une architecture en deux composantes: le front-end (ou online SLAM) et le back end (offline SLAM). Le *front-end* fonctionne en temps réel, il construit la carte et estime ses déplacements d'une acquisition à l'autre, à la même cadence que les acquisitions du capteur. En différé, le *back-end* exécute des algorithmes plus lents et plus haut-niveau sur l'ensemble des données collectées. Il permet d'améliorer la carte et l'estimation de position. Une des opérations cruciales effectuées par le back-end est la **détection de fermeture de boucle (loop closure detection)**. Cette opération consiste à détecter que le robot arrive dans un lieu déjà cartographié, et ajuster la carte en fonction de cette information. Sans ce mécanisme, le front-end peut rapidement rencontrer des problèmes et rendre la carte inexploitable.

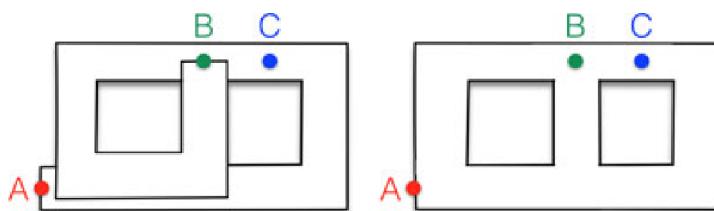


Figure 2. A gauche, la carte construite par le front-end seul, à droite la carte après la détection de fermeture de boucle. On voit que la deuxième carte permet de calculer des chemins entre A, B et C.

Cartes topologiques et approches bio-inspirées

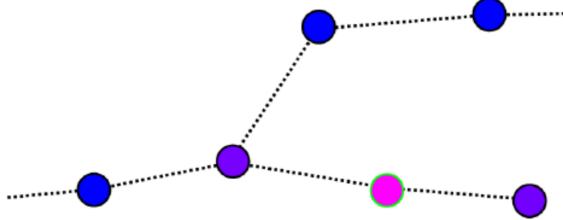


Figure 3. Graphe de cellules de lieu colorées selon leur activité

Les cartes topologiques fonctionnent différemment. Une carte topologique est une représentation haut-niveau de l'environnement. Elle encode des relations topologiques entre les endroits ou objets du monde, sans nécessairement préserver la géométrie de l'environnement. Généralement, ces cartes se présentent sous la forme d'un graphe où les nœuds représentent un élément de l'environnement (ex: un lieu), et les arêtes représentent une relation topologique (ex: adjacence, accessibilité...). Une carte topologique permet de planifier la navigation à un haut niveau: rechercher des chemins, trouver des raccourcis, ou encore optimiser une boucle passant par des positions données. Elle ne permet pas de générer une trajectoire globale précise, évitant tous les obstacles, car la géométrie (forme, échelle,...) n'est pas nécessairement encodée.

Parmi les cartes topologiques, un grand nombre provient d'approches bio-inspirées pour la navigation autonome. Ces approches s'inspirent de la structure du système de navigation des cerveaux des mammifères. Plus particulièrement, ces approches visent à modéliser certains types de neurones trouvés dans l'hippocampe, une région du cerveau responsable de la mémoire et de la navigation spatiale. Parmi les neurones étudiés, on trouve entre-autres:

Les **cellules de lieu (Place cells)**, découvertes chez le rat en 1971 puis chez l'homme en 2003, dont l'activité est corrélée à la position de l'individu. Chaque cellule de lieu est associée à une position physique, formant ensemble une carte de l'environnement exploré. Il est à noter que cette carte est définie dans un repère global, et pas par rapport à l'individu (carte allocentrique vs égocentrique).

Les **cellules de grille (Grid cells)**, dont la découverte en 2005 fait l'objet de l'attribution d'un Prix Nobel en 2014. Leur activité suit un motif particulier, une forme de grille triangulaire, qui réagit à des petits déplacements d'un individu. Il est supposé qu'elles

encodent des informations sur la position et les déplacements de l'individu dans un repère local.

Les **cellules de direction de tête (*Head direction cells*)**, découvertes en 1984, qui encodent l'orientation de l'individu.

Les **cellules de vision spatiale (*Spatial view cells*)**, qui interviennent dans la reconnaissance de lieux à partir d'informations purement visuelles. Note: ces dernières n'apparaissent pas dans le modèle bio-inspiré spécifiquement étudié dans le cadre de ce stage.

Etude détaillée: Modèle de navigation bio-inspiré prédictif

Le modèle de navigation bio-inspiré étudié ici est un modèle proposé dans un papier de recherche par Simon Gay, Kévin Le Run, Edwige Pissaloux, Katerine Romeo et Christèle Lecomte. Cette approche propose de construire un graphe de navigation basé sur la modélisation des cellules de lieu, cellules de grilles et cellules de direction de la tête.

Motivation

Dans le papier qui l'introduit, ce modèle est comparé à d'autres solutions bio-inspirées. Les avantages des solutions bio-inspirées en général sont la robustesse face à l'accumulation d'erreurs et la capacité à modéliser de grands environnements. Ce modèle se distingue des autres solutions bio-inspirées notamment par sa capacité à fonctionner avec un champ de vision réduit (90° contre 360° pour beaucoup d'autres approches), et indépendamment de la direction du mouvement de l'agent (contrairement notamment à RatSLAM[4] et ses améliorations).

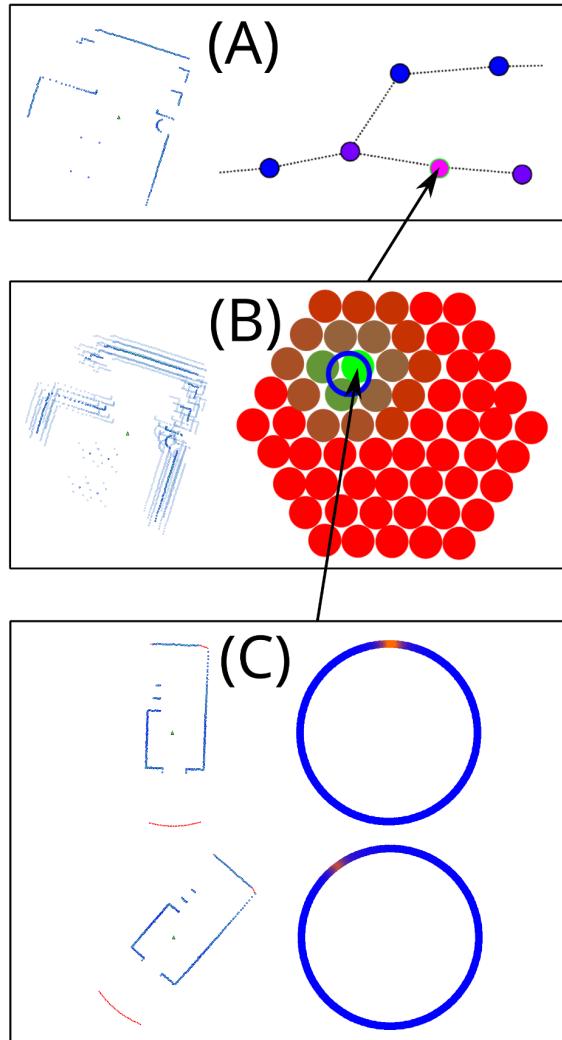


Figure 4. Illustration des différentes composantes du modèle bio-inspiré. De haut en bas et de droite à gauche: Graphe de cellules de lieu, contexte d'une cellule; Cellules de grille, contextes déplacés des cellules de grille; Cellule de rotation de la tête, pour deux contextes observés.

Explication

Au plus haut niveau, il y a les **cellules de lieu (A)**. Les cellules de lieu sont représentées comme des nœuds d'un graphe qui encodent la topologie de l'environnement. Chaque nœud représente une position accessible dans l'espace. Chaque arête indique qu'il est possible de passer d'une position à l'autre. Ainsi, un chemin peut être représenté par une suite de nœuds (= cellules de lieu) à suivre. Le graphe de cellules de lieu agit comme un repère global qui représente tout l'espace exploré. Chaque cellule de lieu enregistre également un contexte. En comparant leur contexte au contexte observé par l'agent, les cellules de lieu calculent leur activité. La cellule la plus active correspond à la position globale estimée du robot.

A un niveau plus fin, on trouve les **cellules de grille (B)**. Les cellules de grille sont organisées sous forme d'une grille (hexagonale dans la nature, carrée dans le modèle proposé). Dans le modèle proposé, chaque cellule de la grille décale le contexte de la cellule de lieu active. En somme, les cellules de la grille "prédisent" ce que le robot devrait voir après un certain déplacement, et calculent leur activité en fonction. L'activité de la grille permet de donner une estimation locale précise de la position de l'agent, relativement à la cellule de lieu active. Lorsque la cellule de lieu active change, les cellules de grilles mettent à jour leur contexte, de façon à toujours donner la position locale vis à vis de la cellule de lieu la plus proche.

Enfin, les **cellules de direction de la tête (C)** permettent d'estimer la rotation de l'agent. Elles opèrent de la même manière que les cellules de grille, en prédisant les observations du robot pour certains angles de rotation. Ces cellules interviennent dans le calcul de l'activité des cellules de grille et de lieu.

Ensemble, ces composants forment un modèle complet, qui permet à un agent de construire un graphe de cellules de lieu. Pour naviguer dans cette carte, il peut se déplacer de cellule de lieu en cellule de lieu, étant donné que les cellules de lieu connaissent la position de leurs voisines, et que le robot peut se localiser précisément relativement à une cellule de lieu.

Parenthèse mathématique: les courbes B-splines

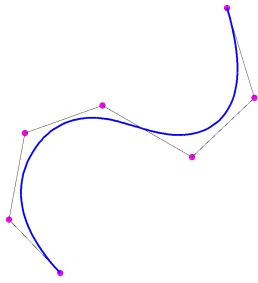


Figure 5. Une courbe B-spline. Les points de contrôle (rose) dictent la forme de la courbe (bleue)

Les courbes B-splines (ou simplement B-splines) sont des courbes paramétriques fréquemment utilisées en théorie du contrôle pour définir des trajectoires. Elles sont définies notamment par leurs points de contrôle, qui dictent la forme de la courbe.

Il existe une grande variété de courbes pouvant être définies à partir de points de contrôle (Courbes de Bézier, Splines d'Hermite, etc.), cependant les B-splines possèdent des propriétés particulières qui expliquent leur importance pour la définition de trajectoires:

- Le **support local** est une propriété qui garantit que chaque point de contrôle n'a d'influence que sur une partie de la courbe. Ainsi, déplacer un point de contrôle n'affecte qu'une portion de la courbe. Cela permet de contrôler facilement la forme de la courbe.
- L'**enveloppe convexe**, qui garantit qu'une portion de la courbe est contenue dans l'enveloppe convexe de certains points de contrôle. En d'autres termes, il est facile de contenir la courbe dans un ou plusieurs polygones convexes, ce qui peut être utile pour garantir l'absence de collision pour une trajectoire.
- La **continuité C^{n-1}** est garantie pour une courbe de degré n . Ainsi, il est possible de garantir la continuité de la vitesse et de l'accélération le long de la trajectoire, ce qui est utile pour des systèmes sujets à des contraintes dynamiques importantes.

Si l'utilisation des B-splines dans ce stage se limite à deux dimensions, il faut noter que les B-splines peuvent être définies dans de nombreux espaces, même abstraits. (2D, 3D, espace d'articulations 6D, espace de couleurs pour la création de gradients).

Etude détaillée: B-Spline Curves SLAM¹

B-Spline Curve SLAM est une approche proposée dans un papier de recherche par Rômulo T. Rodrigues, A. Pedro Aguiar and António Pascoal. Cette approche propose de construire une carte métrique, dans laquelle les obstacles sont représentés par des courbes continues, appelées B-Splines. L'algorithme est conçu spécifiquement pour des capteurs de proximité en 2D tels que le LiDAR du TurtleBot3. Il faut également noter que cette approche n'est que la partie *front-end* de l'algorithme, un système complet aurait aussi une partie *back-end* exécutée en différé pour effectuer la détection de fermeture de boucle, comme expliqué dans la section [Cartes métriques et fermeture de boucle](#).

Motivation

Dans le papier de recherche qui l'introduit, B-Spline Curves SLAM est comparé à d'autres approches de SLAM construisant des cartes métriques. Les avantages avancés pour la représentation d'obstacles par des courbes continues par rapport aux autres méthodes de SLAM métrique sont:

- Une utilisation réduite de la mémoire. La carte étant composée exclusivement de quelques points de contrôle encodant les obstacles, elle est bien plus compacte en mémoire que les cartes discrètes classiques, qui gaspillent de l'espace notamment en représentant des zones vides ou inaccessibles.
- Une précision accrue, car les obstacles sont représentés de manière continue. Bien-sûr, il y a déjà une conversion analogique-numérique lorsque les acquisitions du capteur sont transmises à l'ordinateur qui exécute SLAM. Mais dans un algorithme SLAM classique (comme Google Cartographer), on enregistre les obstacles dans une grille discrète, dont la résolution est en général fixe. Cela peut conduire à des imprécisions imprévisibles, qui dépendent de l'alignement entre les acquisitions du capteur et la grille discrète.
- Un lissage automatique du bruit du capteur.

¹ A B-spline Mapping Framework for Long-Term Autonomous Operations - Rômulo T. Rodrigues , A. Pedro Aguiar, and António Pascoal

Explication

Cet algorithme traite les entrées du capteur, mettant à jour la carte et produisant une estimation de position à chaque itération. Les différentes étapes parcourues pour chaque acquisition sont:

- *pre-processing*
- *tracking*
- *fitting*
- *map update and localization*

Le **pré-processing** est la première étape de chaque itération. Dans cette étape, les acquisitions sont d'abord passées en coordonnées cartésiennes. Puis, les points sont segmentés en "clusters". Cette segmentation permet de grouper les points appartenant à un seul et même obstacle, qui sera modélisé par une seule courbe. Cette segmentation élimine également les points isolés, susceptibles d'être une anomalie de mesure.

L'étape de **tracking** permet de reconnaître un obstacle d'une itération à l'autre. La reconnaissance intervient dans la mise à jour de la carte, elle permet de savoir quelles courbes sont parfaitement nouvelles et quelles courbes sont à fusionner avec une courbe existante de la carte.

Le **fitting** est une étape cruciale de l'algorithme. Cette étape transforme chaque cluster en une courbe B-Spline. Pour ce faire, il faut résoudre un problème d'optimisation de manière à minimiser la distance entre la courbe et l'ensemble de points qu'elle modélise. La courbe est définie de manière à ce que son nombre de points de contrôle soit faible, et reflète la taille de l'obstacle. Le nombre réduit de points de contrôle permet également d'assurer que la courbe soit relativement lisse.

Enfin, la **mise à jour de la carte et la localisation** sont les dernières étapes de l'algorithme. Ces deux étapes sont faites ensemble, par la résolution d'un nouveau problème d'optimisation. Cette fois-ci, le problème d'optimisation consiste à trouver le déplacement du robot qui permettrait de minimiser la distance entre les courbes (= obstacles) observées et les courbes (=obstacles) enregistrées dans la carte. Ce problème d'optimisation donne également des informations permettant la mise à jour de la carte, en fusionnant les courbes observées avec les courbes identifiées dans la carte.

Etude détaillée: Comparaison des approches

Lorsqu'elles sont proposées, ces approches sont comparées aux approches de la même classe (les approches bio-inspirées sont comparées entre elles, et vice-versa). Il convient de comparer les deux classes d'approches entre elles.

Avantages des modèles bio-inspirés et des cartes topologiques

L'intérêt principal de construire une carte topologique est que cela permet de rendre le modèle robuste. Comme expliqué dans la [partie III \(Contexte et enjeux\)](#), une des préoccupations principales pour les algorithmes qui tentent de résoudre SLAM est l'accumulation d'erreurs. Dans une carte métrique, de faibles erreurs peuvent avoir de grandes répercussions au fur et à mesure que l'agent explore. Si une carte métrique est erronée et ne reflète pas précisément la géométrie de l'environnement, il est impossible de l'exploiter pour la navigation.

Ce modèle bio-inspiré n'élimine pas l'accumulation d'erreurs, on peut le voir en plaçant les nœuds du graphe de façon à représenter leur position estimée par le modèle. Si le modèle bio-inspiré est plus robuste, c'est plutôt parce que cette erreur accumulée n'affecte pas la capacité de l'agent à naviguer. En effet, même si la position absolue (position des nœuds du graphe) est fausse, la position locale de l'agent vis-à-vis du nœud actif reste précise. La position de chaque nœud vis à vis de ses voisins est connue avec suffisamment de précision pour que le robot puisse passer d'un nœud à l'autre en se servant de ces estimations locales de position.

Un autre avantage des cartes topologiques est que leur construction implique automatiquement la détection de fermeture de boucles. Dans le modèle bio-inspiré étudié au cours du stage, la reconnaissance d'un lieu déjà visité est assurée par le calcul de l'activité des place cells. Lorsqu'une place cell dépasse un seuil d'activité, on considère que l'agent a reconnu le lieu correspondant, et on lie les place cells entre elles, sans se soucier de si l'estimation de leur position absolue ne correspond pas à leurs distances observées. De plus, lorsqu'une boucle est détectée, il n'y a pas de modification à apporter à la carte, si ce n'est l'ajout d'une arête dans le graphe. En comparaison, les approches de SLAM par cartes métriques nécessitent un algorithme complexe qui n'opère pas en temps réel pour détecter la fermeture d'une boucle. De plus, lorsqu'une boucle est détectée, la carte entière doit être reconstruite pour être cohérente.

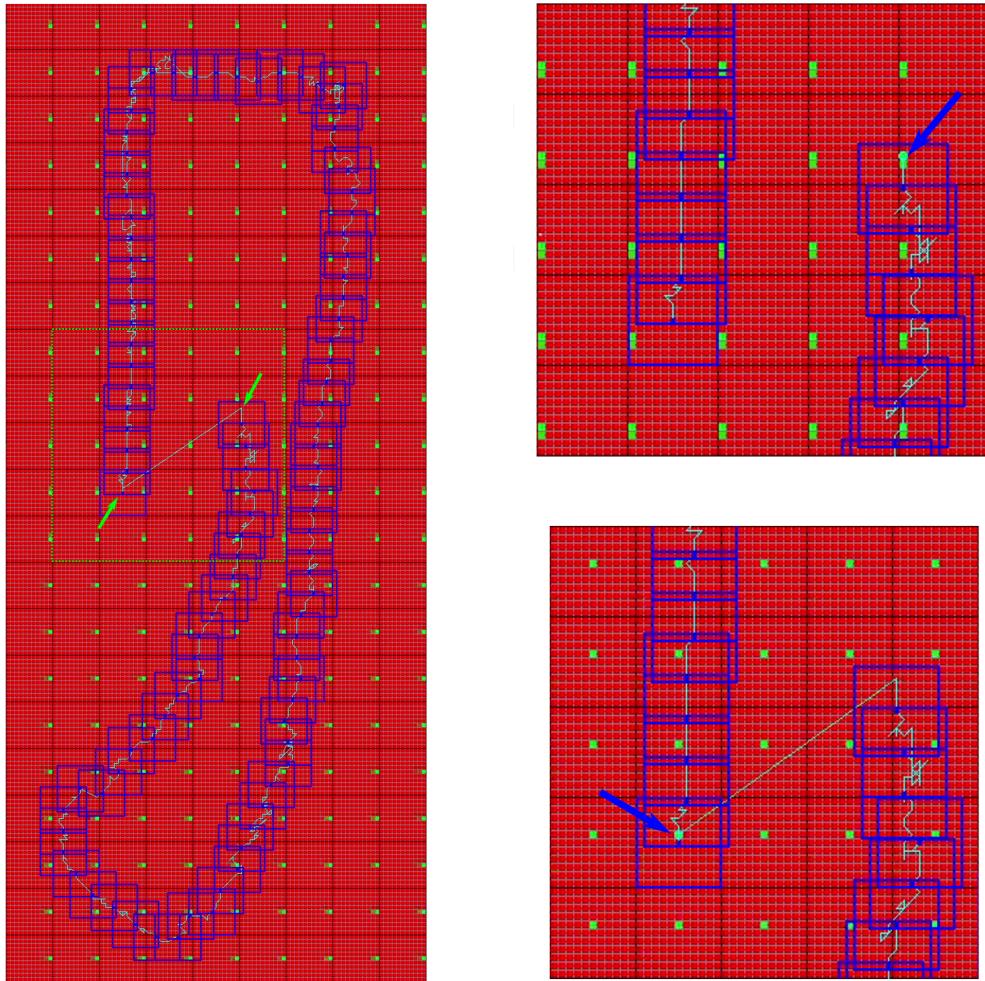


Figure 6. Test du modèle bio-inspiré en se déplaçant sur une boucle

L'illustration ci-contre provient du modèle préexistant en Java, testé avec une caméra stéréo[5]. Chaque point bleu représente une place cell, encadrée d'un carré bleu qui indique la zone de l'espace dans laquelle l'agent peut se repérer localement. Malgré l'accumulation d'erreurs, qui fait que les place cells ne forment pas une boucle, l'agent parvient à reconnaître un lieu déjà exploré, et revient bien au départ, sans se soucier de la discontinuité de sa position globale. L'agent peut naviguer dans le graphe des place cells, même aux endroits présentant une discontinuité, car il ne se soucie que de sa position relative à une cellule.

Avantages des cartes métriques

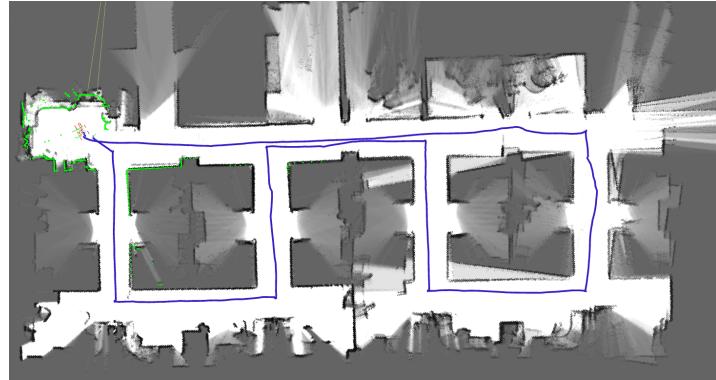


Figure 7. Carte générée par Google Cartographer, une approche “standard” de SLAM

Les cartes métriques ayant de nombreuses variantes, il est difficile de citer des avantages qui s'appliquent à toutes. Il existe de nombreuses cartes métriques, adaptées à beaucoup de cas d'usage différents. Certaines sont compactes, permettant de représenter de larges environnements, alors que d'autres sont précises. Certaines sont facilement exploitable pour la navigation autonome, d'autres non. On peut tout de même noter que la recherche sur des méthodes de SLAM produisant des cartes métriques est plus mature que pour les modèles bio-inspirés. Les algorithmes construisant une grille discrète probabiliste représentant l'occupation de l'espace sont généralement considérés standards, entre-autres parce que la carte qu'ils produisent est facile à interpréter et à exploiter.

Outils logiciels et matériels

ROS

Robot Operating System (ROS) est un middleware libre et open source utilisé dans la robotique. Contrairement à ce que son nom peut laisser penser, il ne s'agit pas d'un système d'exploitation, mais plutôt d'un ensemble d'outils logiciels pour le développement de logiciels et d'algorithmes de robotique [MAL DIT]. ROS permet la communication entre des processus hétérogènes, sur une ou entre plusieurs machines connectées à un même réseau. Son intérêt est de faciliter le développement de logiciels interagissant avec des systèmes robotiques, ou éventuellement avec d'autres logiciels. Pour cela, ROS fournit divers services: couche d'abstraction matérielle, communication inter-processus, système de *build* et de gestion de paquets...

Les processus ROS sont organisés en nœuds dans un graphe, connectés par des arêtes, appelées “topics”. Ces topics constituent des canaux de communication, que les nœuds ROS utilisent en suivant un modèle publisher-subscriber. Par exemple, un nœud peut être un robot mobile équipé d'un capteur. Ce nœud peut publier les acquisitions du capteur sur un topic (par exemple “/scan”), mais aussi s'abonner à un topic (“/cmd_vel”), qui lui donne des consignes pour se déplacer. Ainsi, on peut lancer divers noeuds pour interagir avec le robot: Un noeud sur un ordinateur distant qui lit les entrées clavier et les retranscrit en commandes envoyées sur “/cmd_vel” pour déplacer le robot, un ou plusieurs autres noeuds qui écoutent le topic “/scan” et visualisent les sorties du capteur. ROS met en place la communication entre les nœuds, se charge de gérer le format des données envoyées, et fournit un API pour récupérer ces données et les utiliser directement dans les langages de programmation supportés (C++ et Python). Enfin, il existe un écosystème de paquets riche, qui offre de nombreux outils de simulation, de visualisation, des démonstrations d'algorithmes “classiques”, etc.

TurtleBot

Les TurtleBot sont une série de robots mobiles terrestres créés à l'origine par deux chercheurs américains, Melonee Wise et Wose et Tully Foote². Ces robots sont conçus spécifiquement pour la recherche et l'éducation. Il existe à ce jour quatre familles de TurtleBot (TurtleBot1, 2, 3 et 4), elles-mêmes composées de plusieurs modèles.

Dans ce stage, on utilise un TurtleBot3 Burger, fabriqué par ROBOTIS. Il s'agit d'un petit³ robot sur roues. Le robot embarque un LiDAR permettant une acquisition à 360°, ainsi que deux moteurs équipés de codeurs rotatifs absous. Enfin, le Raspberry Pi 4B fournit la puissance de calcul nécessaire pour piloter l'ensemble du robot et exécuter les différentes tâches requises.

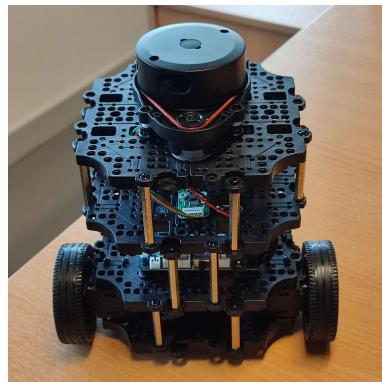


Figure 8. le TurtleBot3 Burger

Le TurtleBot3 est particulièrement bien adapté à notre cas d'utilisation pour plusieurs raisons. D'abord, le fabricant ROBOTIS met à disposition un guide officiel pour installer et configurer le robot avec ROS. De plus, grâce à sa popularité, le TurtleBot bénéficie d'une abondance de packages préexistants pour faciliter son utilisation. Enfin, le capteur LiDAR est particulièrement adapté au cadre de ce projet car il permet d'acquérir directement un nuage de points relativement précis et facilement exploitable, là où l'utilisation d'une caméra nécessiterait d'avoir recours à des techniques de vision par ordinateur.

² La conception et la fabrication de nouveaux TurtleBot continue, malgré la fermeture du laboratoire d'origine (Willow Garage, Californie) en 2014. Les robots sont conçus en collaboration avec de nombreuses entreprises de robotique, qui apparaissent sur le site officiel (<https://www.turtlebot.com>).

³ 13.8cm*17.8cm*19.2cm

Simulation

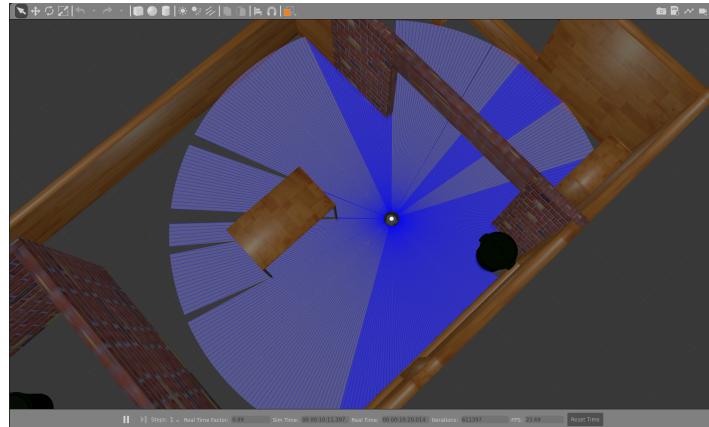


Figure 9. Gazebo. Les traits bleus proviennent du LiDAR du robot et représentent son champ de détection

Tester des algorithmes de navigation directement sur un robot physique introduit une multitude de variables qui peuvent ralentir considérablement le développement: incertitudes, niveau de la batterie, capacité de calcul, mise en place du matériel, etc. Ainsi, il convient d'utiliser un simulateur, qui permet de tester et d'itérer sur les algorithmes, avant de passer sur un robot réel.

Parmi les nombreux simulateurs existants, j'ai choisi Gazebo, pour son intégration complète avec ROS et ses packages qui permettent directement la simulation d'un TurtleBot3. Grâce au système de nœuds de ROS, il est possible de développer et tester un algorithme sur le simulateur, puis de le déployer sur le robot réel sans changements majeurs.

Autres outils

L'installation de ROS est connue pour pouvoir poser de nombreux problèmes. Pour les contourner, il est courant de passer par des conteneurs. L'utilisation de conteneurs permet également d'utiliser plusieurs versions de ROS différentes. J'ai choisi le gestionnaire de conteneurs LXD pour travailler avec ROS tout en l'isolant de mon système.

Le code produit au cours de ce stage a été écrit avec la dernière version de Python à ce jour (3.11).

2. Conception/Développement

Modèle Bio-inspiré

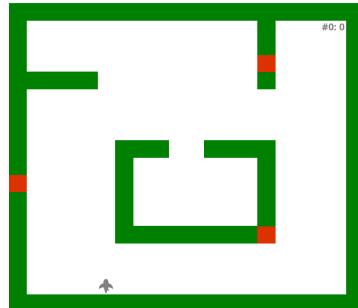


Figure 10. Environnement de test du modèle bio-inspiré, codé en Java

Le modèle bio-inspiré existant est écrit en Java. Un simulateur est codé entièrement en java, dans lequel un agent peut se déplacer de façon omnidirectionnelle. L'agent perçoit les obstacles dans un champ de vision réduit devant lui, et peut distinguer deux types d'obstacle, représentés par différentes couleurs (vert et rouge)

Pour pouvoir l'implémenter sur le TurtleBot3, il a fallu extraire du code de base la partie touchant au modèle de navigation, et la réécrire entièrement en un langage supporté par ROS (C++ ou Python). Python a été choisi, car il est mieux adapté aux activités de recherche, permettant d'expérimenter plus rapidement différentes idées et de faire des ajustements fréquents.

Performances

Une des difficultés rencontrées lors de la réécriture du code est le coût en performances lié à l'utilisation de Python. En effet, même si Python est un langage compilé en bytecode et exécuté par une machine virtuelle (de la même manière que Java et sa JVM) il reste plus lent, notamment à cause de l'overhead introduit par le typage dynamique. Dans la première implémentation "naïve", le calcul de l'activité des cellules pour une acquisition pouvait prendre 1-2 minutes. Pour référence, le capteur LiDAR LDS-02 dont le TurtleBot est équipé fait plus de 5 acquisitions par seconde. En utilisant cProfile pour diagnostiquer les problèmes de performance, je me suis rendu compte que les points les plus intensifs

étaient le calcul de l'activité des cellules, tâche qui fait intervenir de nombreuses opérations sur des matrices.

Pour pallier ce problème, il existe de nombreuses bibliothèques, écrites dans des langages de plus bas niveau, qui mettent à disposition des outils plus efficaces pour manipuler des données.

Numpy est une librairie open source très répandue, qui permet d'accélérer de nombreuses opérations sur des vecteurs ou des matrices. Comme le modèle bio-inspiré représente les lieux explorés sous forme de matrices, et calcule l'activité des cellules par des opérations matricielles, utiliser numpy apparaît comme une solution évidente. Après de nombreuses itérations, réécrire les opérations les plus lentes avec numpy, combiné à quelques simplifications, a permis d'améliorer la performance de plusieurs ordres de grandeur, jusqu'à environ ~500ms par itération. Cela reste trop lent pour pouvoir traiter 5 acquisitions par seconde. Je me suis donc tourné vers d'autres librairies.

Numba est une autre librairie open source d'optimisation pour Python. Elle permet d'indiquer des portions de code à optimiser par l'ajout de décorateurs. Numba compile à la volée les portions de code indiquées en langage machine (par l'intermédiaire de LLVM), permettant en théorie d'atteindre une vitesse proche des langages compilés comme C. Numba est compatible avec un sous ensemble de Python, ainsi que quelques librairies populaires, dont numpy. En isolant la partie la plus lente de l'algorithme, la version accélérée par numba s'exécute environ 100x plus vite que celle réécrite avec numpy. En utilisant numba à quelques moments clés du code, l'algorithme devient suffisamment rapide pour être capable de traiter toutes les acquisitions envoyées par le capteur.

Au niveau du code, l'optimisation en utilisant Numba nécessite peu de modifications, pourvu que la portion de code optimisée se limite au sous-ensemble de Python supporté par Numba.

```

#1 Native implementation in vanilla python
def vanilla(cues):
    """
    compute the context matrix from the list of context cues
    """

    result= np.zeros((360,100,2))

    # Spread_size: the size of the "impact" that each cue makes in the matrix
    spread_size = 4

    for cue in cues:
        angle_int = int(cue[0])
        d_int = int(cue[1])
        cue_type = int(cue[2])

        for t in range(-spread_size, spread_size + 1):
            for j in range(-spread_size, spread_size + 1):
                dist_as_index = d_int + i
                angle_as_index = (angle_int + j) % 360
                if dist_as_index >= 0 and dist_as_index < 100 and angle_as_index >=0 and angle_as_index < 360:
                    val = 1 - max(abs(t), abs(j))/spread_size + 1
                    if val > result[angle_as_index][dist_as_index][cue_type]:
                        result[angle_as_index][dist_as_index][cue_type] = val

#2 Faster Implementation with Numba JIT compilation (Several thousand times faster)
@jit(nopython=True)
def vanilla(cues):
    """
    compute the context matrix from the list of context cues
    """

    result= np.zeros((360,100,2))

    # Spread_size: the size of the "impact" that each cue makes in the matrix
    spread_size = 4

    for cue in cues:
        angle_int = int(cue[0])
        d_int = int(cue[1])
        cue_type = int(cue[2])
        for t in range(-spread_size, spread_size + 1):
            for j in range(-spread_size, spread_size + 1):
                dist_as_index = d_int + i
                angle_as_index = (angle_int + j) % 360
                if dist_as_index >= 0 and dist_as_index < 100 and angle_as_index >=0 and angle_as_index < 360:
                    val = 1 - max(abs(t), abs(j))/spread_size + 1
                    if val > result[angle_as_index][dist_as_index][cue_type]:
                        result[angle_as_index][dist_as_index][cue_type] = val

```

Figure 12. Code non optimisé (gauche), puis optimisé par numba (droite). L'image n'est pas lisible, mais on voit bien qu'il s'agit du même code, à l'exception de l'ajout d'un décorateur au tout début.

La réécriture du code a également permis une restructuration, rendant le code produit plus flexible et supprimant des redondances. Parmi les améliorations notables, l'ajout de tests unitaires avec la librairie standard unittest a permis d'accélérer grandement le développement de cette nouvelle version du modèle.

Au final, le modèle produit permet de suivre les déplacements du robot dans un simulateur, en utilisant ROS.

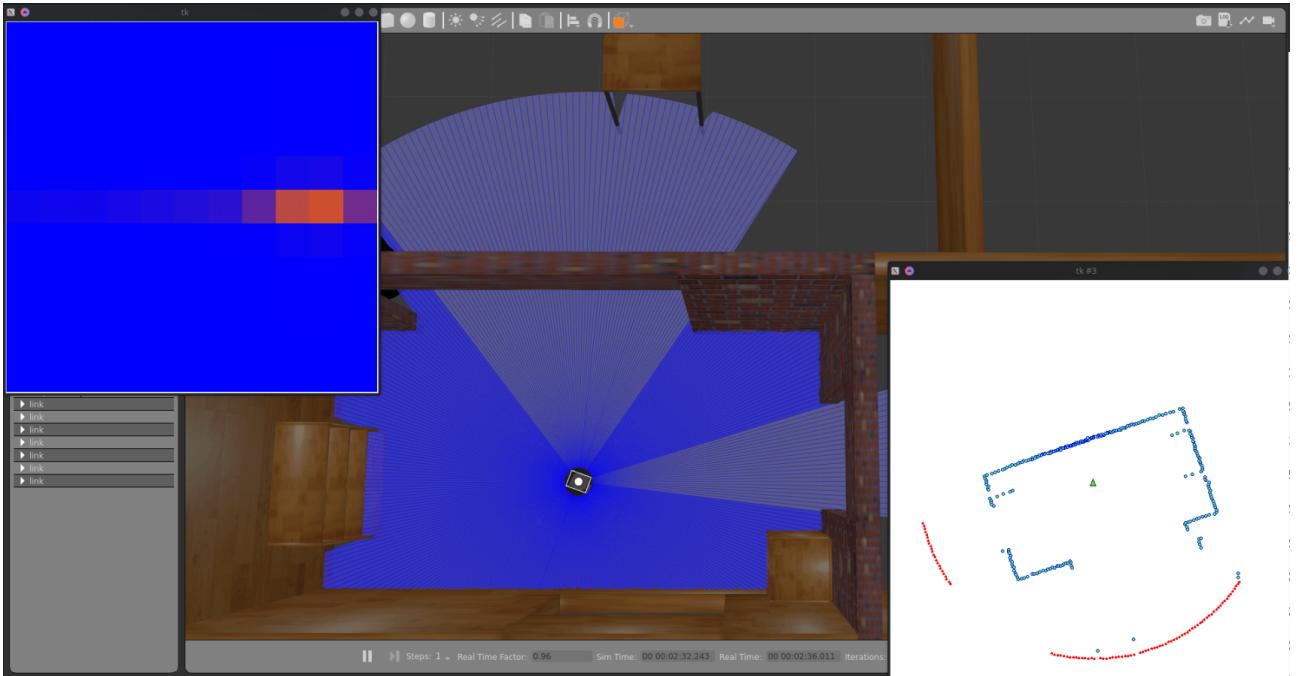


Figure 13. En haut à gauche, le motif d'activation des cellules de grille.

Le passage sur un modèle réel a posé plusieurs difficultés. Notamment, le simulateur modélise une ancienne version du robot, avec un capteur différent. Il a fallu adapter les algorithmes en conséquence. Plus embêtant, les mesures réelles sont très bruitées, ce qui a compliqué la mise en oeuvre du modèle dans le monde réel.

B-Splines SLAM

Il n'y a aucun code disponible pour l'implémentation de l'algorithme B-Spline Curves SLAM. J'ai donc tenté de l'implémenter moi-même, à partir des détails décrits dans le papier.

Les étapes clés de l'algorithme B-Spline Curves SLAM consistent à résoudre des problèmes d'optimisation. Pour résoudre ces problèmes d'optimisation dans Python, on utilise CasADi, un outil open source dédié à l'optimisation numérique en théorie du contrôle. Il permet de définir des problèmes d'optimisation et de les résoudre avec un solveur configurable et performant.

Pré-traitement et segmentation

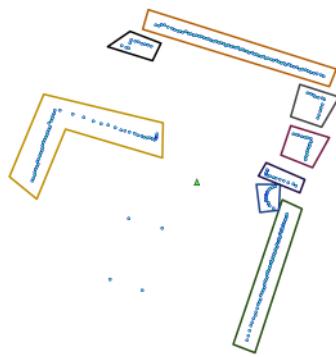


Figure 14. Le nuage de points est divisé en clusters

Cette étape n'a pas posé de souci d'implémentation particulier. On voit ci-contre une acquisition LiDAR, formée d'un nuage de points bleus. L'algorithme fait correctement des groupes de points qui représentent des obstacles individuels

Fitting

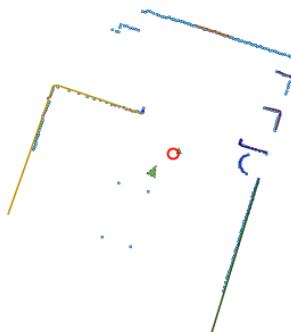


Figure 15. Chaque cluster devient une courbe

Cette étape a été plus difficile. Il a fallu résoudre un problème d'optimisation linéaire à l'aide de CasADi. Au final, j'ai pu transformer les groupes de points en différentes courbes qui correspondent aux obstacles, comme voulu. Cependant, la construction des B-splines est légèrement différente de celle proposée dans la thèse, qui donnait des résultats trop instables.

Localisation et mise à jour de la carte

Malgré beaucoup d'efforts, je n'ai pas réussi à faire fonctionner cette étape. Estimer la localisation implique de résoudre un problème d'optimisation non linéaire, dont l'implémentation dans CasADi est très complexe.

Piste explorée: Intersection de courbes

L'idée d'une carte où les obstacles sont représentés par des courbes soulève une question importante: Comment faire pour exploiter cette carte?

Lorsqu'on veut générer une trajectoire évitant des collisions, la toute première étape consiste à pouvoir détecter ces collisions. Pour des cartes classiques sous forme de grille d'occupation, il est facile d'évaluer numériquement si une courbe traverse un obstacle ou une zone non accessible. Pour une carte faite de B-splines, en revanche, c'est plus compliqué. J'ai étudié les méthodes existantes pour détecter une intersection entre deux courbes, et j'en ai retenu une que j'ai implémenté:

Cet algorithme est générique, il ne dépend pas du type de courbes utilisé. Pour pouvoir l'utiliser, il faut que les courbes satisfassent deux hypothèses:

- Il doit être possible d'englober les courbes dans un volume de contrôle (*bounding box*). Pour une B-spline, c'est possible grâce à la propriété d'enveloppe convexe.
- Il doit être possible de découper la courbe en deux indéfiniment. Pour une B-spline, il est possible d'insérer des points de contrôle sans affecter la courbe. Ensuite, par la propriété de contrôle local, il est possible de supprimer des points de contrôle tout en gardant une partie de la courbe non affectée. Il est donc possible de diviser la courbe.

L'algorithme est récursif et fonctionne de la manière suivante:

- On vérifie que les bounding boxes des deux courbes se touchent.
- Si ce n'est pas le cas, il n'y a pas d'intersection.
- Si c'est le cas, alors on divise chaque courbe en deux, et on teste l'intersection entre chaque nouvelle paire de courbes.
- Si la taille des courbes est de moins d'un pixel, on arrête l'algorithme et il y a intersection.

J'ai implémenté cet algorithme en Python, que j'ai testé avec des courbes de Bézier, étant donné qu'elles sont nettement plus faciles à découper que les B-splines.

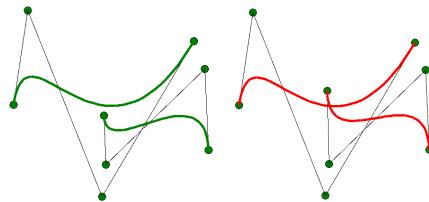


Figure 17. L'algorithme détecte bien l'intersection (les courbes passent du vert au rouge en cas d'intersection)

Piste de continuation proposée

- D'un côté, le modèle de navigation bio-inspiré ne permet que de suivre un chemin de point à point, sans faire d'évitement d'obstacles. Il est cependant capable de se repérer et de construire une carte en temps réel. De l'autre côté, la mise en place de la localisation et du calcul de carte se sont révélés difficiles à mettre en œuvre pour le modèle B-Splines SLAM. En revanche, on a pu représenter des obstacles avec des B-splines, et on a montré qu'il est possible de détecter si une trajectoire cause une collision avec une B-spline. Serait-il alors possible de combiner les deux approches, c'est-à-dire utiliser le modèle bio inspiré pour la localisation et la construction de la carte, mais en y intégrant des B-splines pour permettre la génération autonome de trajectoires évitant des obstacles ? L'introduction de B-splines pourrait également éviter la perte de précision causée par la discréétisation des mesures dans le modèle bio-inspiré.

VI. Bilan

Étant un projet de recherche, la façon de formuler des objectifs et les impacts du stages sont très différents de l'industrie. Le contenu de cette section est adapté en conséquence.

Coût global

Le projet n'a pas entraîné de dépenses pour le laboratoire, excepté l'achat du TurtleBot3 (~800€) et les coûts humains.

Résultats obtenus, impact

Le projet a permis d'explorer différentes facettes du problème SLAM et de mettre en regard des approches provenant de domaines de recherches très différents. Il a permis également le développement d'une version alternative du modèle bio-inspiré du chercheur encadrant le stage. Cette version alternative, écrite en Python pour l'écosystème ROS peut permettre d'ouvrir de nouvelles portes, pour appliquer le système sur d'autres robots utilisant ROS.

Il est également notable qu'un nouveau stagiaire est arrivé en juin pour travailler sur le projet. Son sujet porte sur le contrôle du TurtleBot, de façon à lui permettre de suivre une trajectoire donnée. Les travaux effectués dans mon stage pourront lui être utiles dans son projet. Les connaissances que j'ai acquises sur le robot et l'écosystème ROS m'ont déjà permis de l'aider plusieurs fois dans le début de son stage.

Enfin, ce stage a permis d'identifier des pistes de recherche intéressantes pour lier les différentes disciplines des chercheurs de l'équipe CO4SYS, et proposer de nouvelles façons pour un robot mobile de naviguer dans un environnement inconnu.

VII. Conclusion

Ce stage m'a permis d'avoir un premier contact avec le monde académique. Ayant déjà eu l'occasion de travailler pour l'industrie dans le cadre de ma formation à l'ESISAR, cette expérience complète parfaitement ma formation. Elle m'a permis d'appréhender le travail de chercheur de manière concrète, en côtoyant plusieurs doctorants et en observant leur quotidien.

La recherche s'est révélée être une discipline fascinante, où l'imprévu et les défis sont omniprésents. J'ai particulièrement apprécié la liberté et la créativité qu'elle offre, me permettant d'explorer de nouvelles idées et d'expérimenter différentes approches, en me laissant guider par ma curiosité.

Ce stage est également ma deuxième expérience dans le domaine de la robotique (cette fois-ci du côté de la robotique mobile, et non industrielle). Acquérir de nouvelles connaissances sur des algorithmes phares de la robotique n'a fait que renforcer mon intérêt pour ce domaine passionnant.

Ce stage me permettra de faire un choix informé dans la suite de mon projet professionnel.

VIII. Références

[1]Etude par ABI Research, fin 2019

<https://industryeurope.com/sectors/technology-innovation/abi-robotics-market-to-shift-from-fixed-automation-to-mobile-systems/>

[2] Towards a Predictive Bio-Inspired Navigation Model - Simon Gay 1 , Kévin Le Run 2, *, Edwige Pissaloux 2 , Katerine Romeo 2 and Christèle Lecomte 2

[3] A B-spline Mapping Framework for Long-Term Autonomous Operations - Rômulo T. Rodrigues , A. Pedro Aguiar, and António Pascoal

[4] RatSLAM: a hippocampal model for simultaneous localization and mapping - M.J. Milford; G.F. Wyeth; D. Prasse

[5] A bio-Inspired Model for Robust Navigation Assistive Devices - Simon L. Gay , Edwige Pissaloux and Jean-Paul Jamont