UNIVERSITÉ DU QUÉBEC EN OUTAOUAIS

NAVIGATION AUTONOME DE ROBOTS MOBILES DANS DES ENVIRONNEMENTS INCONNUS

MÉMOIRE

PRÉSENTÉ

COMME EXIGENCE PARTIELLE

DE LA MAÎTRISE EN SCIENCES ET TECHNOLOGIES DE L'INFORMATION

PAR

ABDESSAMED GUERRAICHE

MAI 2025

Ce mémoire a été évalué par un jury composé des personnes suivantes :

Prof. Larbi Talbi ..... Président du jury

Dr. Nadia Baaziz ..... Membre du jury

Dr. Soulaimane Berkane ..... Directeur de recherche

Dernière mise à jour le : **31 juillet 2025**

À ma femme, Hanifa

À mon fils, Saned Abdelbasset

À ma famille

Remerciements

Avant tout, je rends grâce à ALLAH, le Tout-Puissant, pour m'avoir accordé la force, la patience et la persévérance qui m'ont permis d'achever ce mémoire.

Je tiens à exprimer ma profonde gratitude à ma femme et à ma famille, dont le soutien moral et financier a été indispensable tout au long de mon parcours.

Je remercie sincèrement mon directeur de recherche, Dr. Soulaimane Berkane, pour son encadrement, sa disponibilité et ses conseils.

Je remercie également les membres du jury de l'Université du Québec en Outaouais pour le temps consacré à l'évaluation de ce travail et pour leurs retours enrichissants.

Table des matières

Liste des figures

Liste des tableaux

Liste des abréviations, sigles et acronymes

APF Artificial Potential Field

ROS Robot Operating System

RRT Rapidly-exploring Random Tree

RRT* Optimal Rapidly-exploring Random Tree

SVC Safety Velocity Cones

Tbug Tangent Bug

VCP Vehicle Control Position

VFH Vector Field Histogram

Résumé

Ce mémoire traite de la problématique de la navigation autonome de robots mobiles dans des environnements inconnus et potentiellement encombrés. L'objectif principal de cette étude est de concevoir, comparer et valider différents algorithmes de navigation réactive capables d'assurer à la fois l'évitement d'obstacles en temps réel et la convergence vers une cible spécifique, sans recourir à une cartographie préalable.

Une première phase exploratoire a consisté à implémenter plusieurs approches de navigation globale et locale dans MATLAB, dans le but de sélectionner des méthodes pertinentes pour une étude comparative. Deux méthodes classiques et bien établies ont été retenues : l'histogramme de champ vectoriel modifié (VFH) et l'algorithme T-Bug, en raison de leur large adoption et de leur efficacité éprouvée. Une troisième approche, le contrôleur à vitesse de sécurité (SVC), développée récemment au sein du laboratoire de robotique et systèmes autonomes (LaRSA), a été sélectionnée pour son potentiel prometteur et sa simplicité de mise en œuvre, reposant sur une modélisation issue de la théorie du contrôle.

Ces trois algorithmes ont ensuite été implantés dans l'environnement ROS/Gazebo en vue d'une évaluation plus réaliste. Une architecture hybride combinant un régulateur PID au contrôleur SVC a également été développée, afin d'améliorer la robustesse du comportement face aux perturbations. Plusieurs scénarios de simulation ont permis de comparer les performances des différentes approches dans des environnements variés.

L'étude conclut que chaque méthode présente des avantages propres selon le contexte d'application. Enfin, des perspectives sont proposées, notamment en vue d'une expé-

rimentation sur plateforme réelle et de l'exploration d'approches hybrides combinant planification locale et globale, ou encore basées sur l'apprentissage automatique.

Mots-clés : robot mobile, navigation réactive, ROS, Gazebo, SVC, VFH, T-Bug, évitement d'obstacles.

Abstract

This thesis addresses the problem of autonomous navigation for mobile robots in unknown and potentially cluttered environments. The main objective of the study is to design, compare, and validate different reactive navigation algorithms capable of ensuring both real-time obstacle avoidance and convergence to a specified target, without relying on prior mapping.

An initial exploratory phase involved implementing various global and local navigation approaches in MATLAB, with the aim of identifying suitable methods for a comparative study. Two well-established and widely used algorithms were selected : the modified Vector Field Histogram (VFH) and the T-Bug algorithm, due to their proven effectiveness and extensive use in the literature. A third method, the Safety Velocity Cone (SVC) controller, recently developed within the *laboratoire de robotique et systèmes autonomes (LaRSA)*, was also included for its promising potential and ease of implementation based on control theory.

These three algorithms were then implemented in the ROS/Gazebo simulation environment for more realistic evaluation. In addition, a hybrid architecture combining a PID controller with the SVC approach was developed to improve robustness against disturbances. Several simulated scenarios were used to compare the performance of the different methods across a variety of environments.

The study concludes that each method offers specific advantages depending on the application context. Finally, future directions are proposed, including experimentation on real robotic platforms and the exploration of hybrid strategies combining global and local planning or incorporating machine learning techniques.

Keywords : mobile robot, reactive navigation, ROS, Gazebo, SVC, VFH, T-Bug, obstacle avoidance.

1

Introduction générale et revue de la littérature

1.1 Introduction et motivation

Le domaine de la robotique mobile a connu une croissance significative ces dernières années, en grande partie grâce aux avancées technologiques réalisées dans les domaines de l'électronique, de l'informatique et de l'intelligence artificielle. Ce développement a favorisé l'avantage de robots autonomes capables de naviguer dans divers environnements, qu'ils soient intérieurs ou extérieurs, afin de répondre à une gamme d'applications allant de l'exploration à la logistique, en passant par la surveillance et les interventions de santé. Au cœur de cette autonomie se trouve une compétition fondamentale : la navigation [46].

La navigation autonome englobe l'ensemble des procédés qui permettent à un robot mobile de se déplacer de manière efficace d'un point initial à une destination particulière, tout en contournant les obstacles présents dans son environnement. En général, ce mécanisme repose sur trois éléments essentiels : la localisation, qui permet l'identification de la position du robot dans l'espace ; la cartographie, qui implique la création d'une représentation de l'environnement ; et la planification de trajectoire, qui vise à estimer le meilleur chemin à suivre en considérant les contraintes environnementales.

Cette étude se focalise sur la planification de trajectoire, en mettant l'accent sur les stratégies d'évitement d'obstacles. Diverses méthodes ont été suggérées pour traiter cette problématique, lesquelles peuvent être classées en deux principales catégories : la navigation globale et la navigation locale. Les algorithmes de planification de trajectoire globale, tels que A*, D* et RRT* [32], requièrent fréquemment une connaissance préalable ou partielle de l'environnement. Ces approches sont capables de produire des

trajectoires intégrales jusqu'à la destination ; cependant, elles rencontrent des obstacles dans des environnements changeants ou non familiers. En revanche, les méthodes de navigation locale telles que les algorithmes Bug1 et Bug2 [34] ou le champ de potentiel artificiel (APF) [26] se basent exclusivement sur des données sensorielles locales afin de répondre aux obstacles en temps réel. Malgré leur adaptation aux situations évolutives ou partiellement connues, ces approches peuvent conduire à des trajectoires sous-optimales voire à l'échec lorsqu'elles sont exposées à des configurations compliquées.

Malgré les avancées dans le domaine de la navigation robotique, les robots autonomes demeurent confrontés à des défis considérables lorsqu'ils évoluent dans des environnements complexes ou inconnus. Ces défis portent sur la capacité du robot à éviter de façon dynamique les obstacles, à s'ajuster à des changements imprévus dans l'environnement, ainsi qu'à tenir compte des contraintes physiques qui lui sont imposées. Les récentes recherches se focalisent sur l'élaboration de solutions hybrides combinant réactivité locale et planification globale pour renforcer la robustesse et la sécurité des systèmes de navigation. Cette étude s'inscrit dans ce cadre en menant une analyse comparative de différentes méthodes pour évaluer leur efficacité et leurs limites dans des contextes tant simulés que réels.

1.2 Revue de la littérature

L'évitement d'obstacles a motivé l'intérêt de différentes approches, lesquelles peuvent être classées en deux catégories principales : la navigation globale, qui suppose une représentation explicite (partielle ou complète) de l'environnement afin de planifier un trajet vers la destination, et la navigation locale, qui se base sur les données sensorielles pour réagir de manière active aux obstacles.

Cette section examine en détail les principales techniques appartenant à ces deux classifications. Pour la navigation globale, nous traitons les algorithmes de recherche conventionnels tels que A*, les approches probabilistes telles que RRT, ainsi que les méthodes fondées sur des représentations géométriques comme les diagrammes de Voronoï ou les graphes de visibilité. Pour la navigation locale, nous montrons les méthodes établies sur les champs de potentiel, les algorithmes Bug, ainsi que des stratégies avancées de contrôle par retour d'état.

1.2.1 Navigation globale

Les techniques de navigation globale reposent sur une illustration partielle ou totale de l'environnement. Ils souhaitent prévoir un itinéraire optimal menant à la cible tout en contournant les obstacles identifiés, de manière anticipée. Cette catégorie englobe principalement les algorithmes de graphes tels que Dijkstra et A*, les méthodes probabilistes comme RRT et PRM, ainsi que les méthodes basées sur la géométrie telles que Voronoï et le graphe de visibilité.

1.2.1.1 Algorithmes de recherche de chemin (A* et Dijkstra)

En ce qui se rapporte à la navigation autonome, divers algorithmes classiques sont utilisés pour trouver le trajet le plus court dans une carte, notamment Dijkstra [14], A* [21], Bellman-Ford [3], Floyd-Warshall [17], ainsi que D* et D*-Lite [49, 27]. Dijkstra et A* se distinguent comme les algorithmes les plus courants en raison de leur facilité d'utilisation, de leur efficacité et de leur promesse d'optimalité.

Algorithme de Dijkstra

Dans le cadre d'une problématique de planification, il est possible de représenter l'environnement du robot de manière discrète grâce à un graphe pondéré. Chaque nœud du graphe équivaut à une position possible, tandis que chaque segment représente un déplacement associé à un coût spécifique (tel que la distance parcourue, l'énergie consommée, etc.). Cette idée permet l'application d'algorithmes provenant de la théorie des graphes afin de calculer un chemin optimal menant à la cible.

L'algorithme de Dijkstra représente une approche traditionnelle pour trouver le chemin le plus court dans un graphe donné. L'algorithme consiste à examiner de manière progressive les nœuds accessibles à partir du nœud source, en considérant le coût cumulé depuis le point de départ. Cette méthode assure l'optimalité du chemin identifié, à condition que les poids possèdent des valeurs positives.

Dans le domaine de la navigation automatique, on utilise généralement cet algorithme sur des graphes issus d'illustrations de l'environnement telles que les grilles d'occupation, les cartes PRM (Probabilistic Roadmaps) ou les diagrammes de Voronoï. Cependant, sans heuristique, ses performances peuvent être restreintes dans des environnements étendus ou fortement interconnectés [14].

L'exemple présenté dans la figure 1.1 met en lumière un graphe élémentaire où un robot devrait effectuer un déplacement du nœud de départ **S** (en vert) vers le nœud cible **C** (en bleu). Chaque arête est pourvue d'un coût de déplacement. L'utilisation de l'algorithme de Dijkstra permet de déterminer le chemin idéal, indiqué en couleur rouge, de la manière suivante :

$$S \rightarrow F \rightarrow D \rightarrow C \quad \text{(coût total : 7)}$$

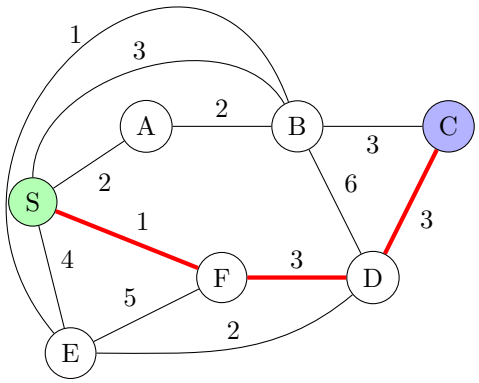

FIGURE 1.1 – Utilisation de l'algorithme de Dijkstra sur un graphe pondéré.

Algorithme A* (A Star)

L'algorithme A* représente une approche de planification globale couramment appliquée afin d'identifier le trajet optimal reliant un point initial à une destination, en considérant les éventuels obstacles présents sur le parcours. À la différence de Dijkstra, l'algorithme intègre une évaluation de la distance restante jusqu'à la cible, ce qui améliore l'orientation de la recherche tout en assurant l'optimalité du chemin découvert [21]. À chaque itération, l'algorithme A* sélectionne les régions les plus prometteuses, ce qui en fait une méthode équilibrée alliant efficacité et exactitude. L'exemple présenté dans la figure 1.2 démontre l'application caractéristique de cette technique à une carte binaire, où le point de départ est symbolisé par un **X** et le point d'arrivée par un **O**.

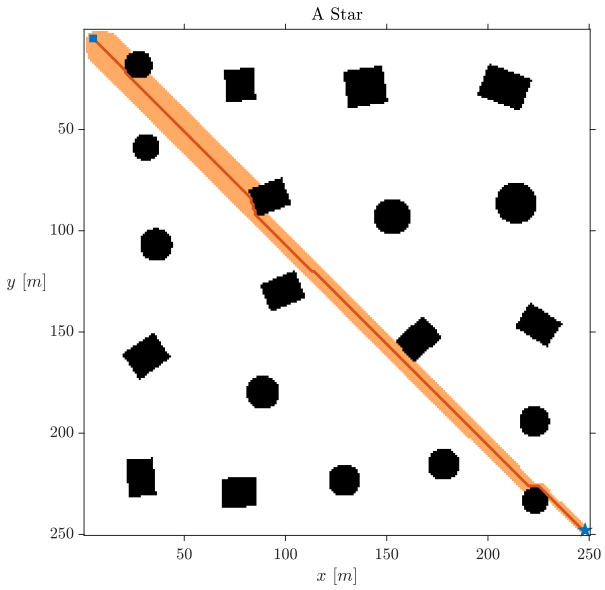

FIGURE 1.2 – Planification de trajectoire à l'aide de l'algorithme A* sur une carte binaire.

1.2.1.2 Algorithmes basés sur l'échantillonnage probabiliste

Les techniques de planification probabilistes se sont avérées efficaces dans l'exploration d'espaces de configuration complexes, en particulier dans le domaine de la robotique mobile [32, 12]. Diverses méthodes sont disponibles, cependant, cette recherche se focalise sur les approches les plus couramment employées : PRM, RRT [31] et RRT* [24], du fait de leur forte présence dans la littérature théorique et les applications robotiques actualisées [56].

Rapidly-exploring Random Tree (RRT)

L'algorithme *RRT*, tel qu'introduit par [31], représente une approche probabiliste performante en matière de planification de trajectoire au regard d'espaces de configuration compliqués. Il forme un arbre d'exploration en effectuant un échantillonnage aléatoire de l'espace dégagé, puis en reliant chaque nouvel point au nœud le plus proche. Cette tactique simplifie une exploration efficace, même en existence d'obstacles ou dans des terrains de vaste dimension.

L'algorithme RRT débute en initialisant un arbre de manière aléatoire. À chaque cycle, un point est échantillonné, puis relié au nœud le plus proche tout en examinant

les collisions, avant d'être ajouté à l'arbre si la connexion est jugée valide. Lorsque la cible est atteinte, une méthode classique telle que Dijkstra est utilisée pour calculer un chemin réalisable.

La trajectoire générée par l'algorithme *RRT* dans un environnement contenant des obstacles convexes et non convexes est illustrée dans la figure 1.3. Ce résultat met en lumière la capacité de l'algorithme à découvrir efficacement l'espace de configuration et à produire des trajectoires réalisables, même dans des environnements géométriquement complexes [1].

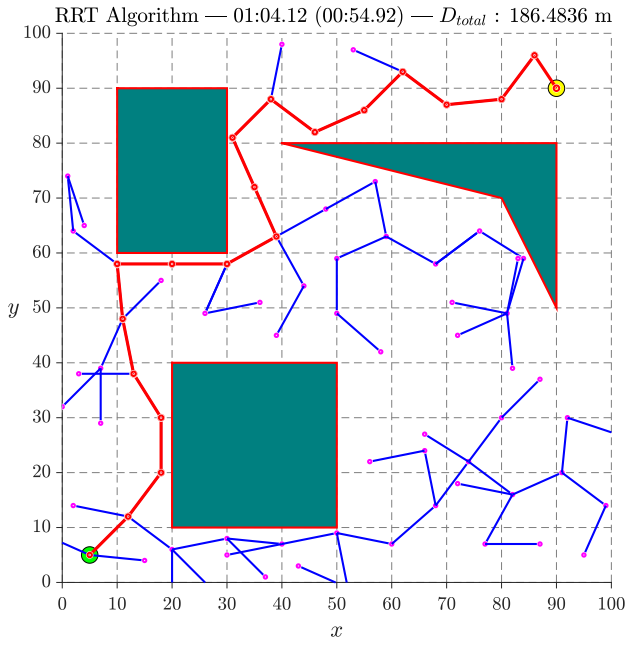

FIGURE 1.3 – Chemin tracé par l'algorithme RRT dans un environnement complexe.

RRT* (Optimal Rapidly-exploring Random Tree)

L'algorithme *RRT**, développé par [24], s'inspire du RRT en y ajoutant une technique d'optimisation locale nommée "réoptimisation". À la suite de l'ajout de chaque nouvel point, l'algorithme analyse ses voisins immédiats afin de déterminer la possibilité de les connecter et ainsi réduire le coût total du chemin.

Cette procédure permet à l'arbre de progresser de manière graduelle vers une solution optimale à mesure que le nombre de points augmente. Le compromis se traduit par une

1. Vidéo illustrative de la méthode *RRT* : https://youtu.be/AMWRkRIAPag

durée de calcul plus longue, ce qui est justifié dans les situations où la précision de la trajectoire est essentielle.

L'exemple de trajectoire optimisée générée par l'algorithme *RRT** dans un environnement contenant des obstacles convexes et non convexes est présenté dans la figure 1.4, mettant en évidence l'amélioration notable de la trajectoire par rapport à l'algorithme *RRT*. Cette amélioration est importante en termes de qualité du chemin généré. Une vidéo illustrant la méthode *RRT** est disponible à l'adresse suivante : https://youtu.be/gzTkD00EUKE.

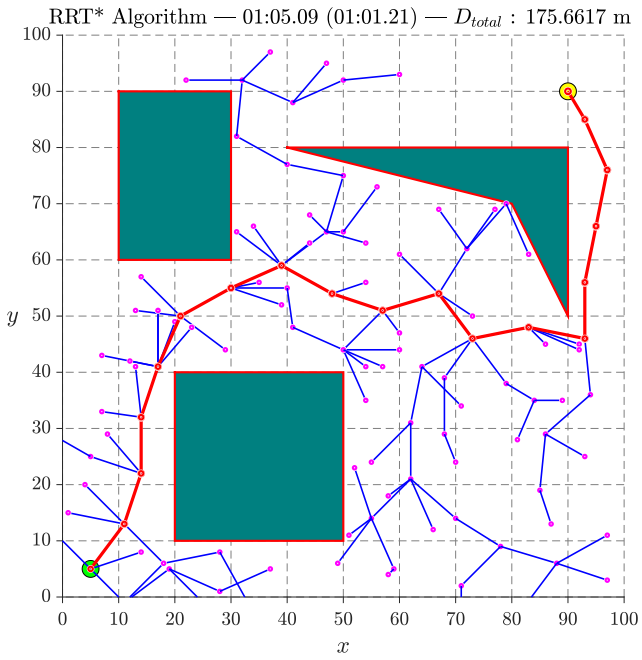

FIGURE 1.4 – Ttrajectoire fluide et optimisée générée par RRT^*.

Méthode PRM (Probabilistic Roadmap)

La technique de planification Probabilité Roadmap (PRM) est une méthode globale qui repose sur l'échantillonnage aléatoire de l'espace d'occupations. Le fonctionnement de l'algorithme se divise en deux phases distinctes : tout d'abord, il crée une série de points de manière aléatoire dans les zones non occupées de la carte, puis il cherche à établir des connexions directes entre ces points en s'assurant qu'ils ne contournent aucun obstacle. Après avoir établi ce graphe, également connu sous le nom de roadmap, les points de départ et d'arrivée sont reliés au réseau, puis un algorithme de recherche de chemin le plus court tel que A* est appliqué pour déterminer le trajet final [25].

L'évaluation MATLAB représentée dans la figure 1.5 met en lumière ce processus, avec les positions initiale et finale symbolisées respectivement par les symboles **X** et **O**.

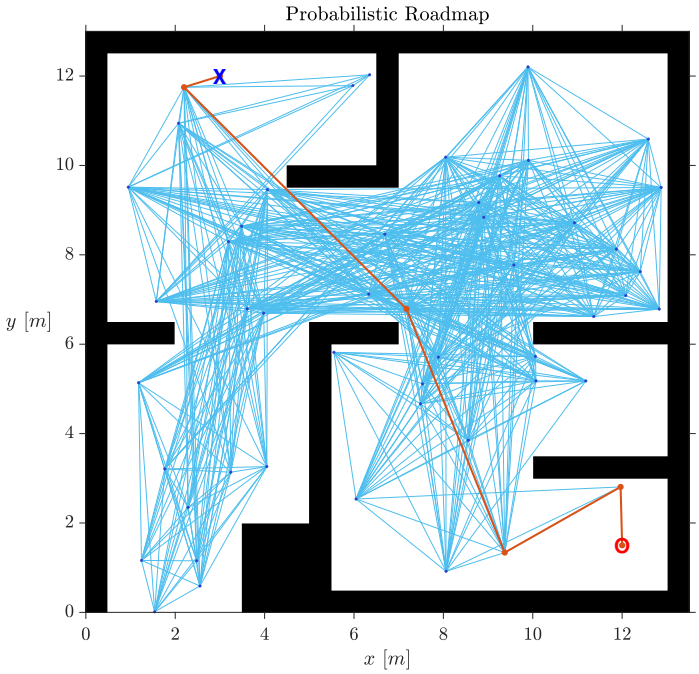

FIGURE 1.5 – Résultat de la planification globale par PRM simulée sous MATLAB

1.2.1.3 Méthode du diagramme de Voronoï

La technique du diagramme de Voronoï consiste à partitionner l'espace dégagé en zones évitant des obstacles, créant ainsi une structure utilisée dans le processus de planification de trajectoires sécurisées [11]. Une variante plus large, connue sous le nom de *Generalized Voronoi Graph* (GVG), a été proposée par Choset et Burdick en tant qu'approche de planification reposant sur les capteurs [10].

Le mécanisme est représenté dans la figure 1.6 : le diagramme est créé grâce à la fonction `voronoi` de MATLAB (à gauche), puis il est filtré afin de conserver uniquement les segments valides qui n'interfèrent avec aucun obstacle (à droite).

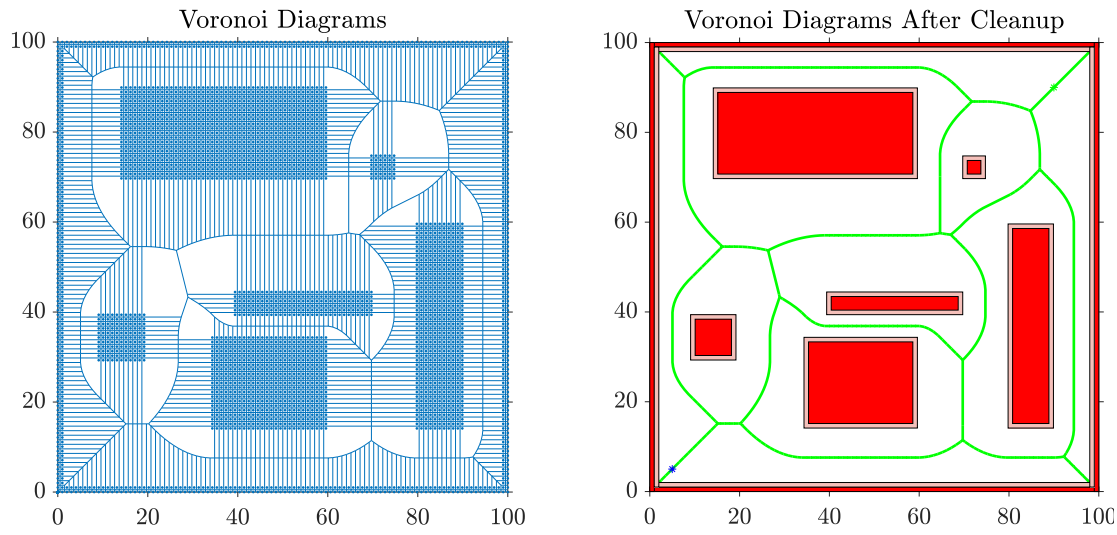

FIGURE 1.6 – Diagramme de Voronoï avant et après nettoyage.

Par la suite, on établit une matrice d'adjacence à partir du graphe obtenu. L'utilisation d'un algorithme de recherche tel que `A*` ou `Dijkstra`, à travers l'application de la fonction `shortestpath`, permet d'obtenir un itinéraire optimal et sécurisé entre un lieu de départ et une destination. L'exemple de trajectoire simplement obtenue après cette étape est illustré dans la figure 1.7 [52]. Un outil de simulation en ligne est aussi accessible pour permettre de mettre en pratique cette approche et observer la création dynamique du modèle de Voronoï [a].

a. https://voronoi-editor.web.app/

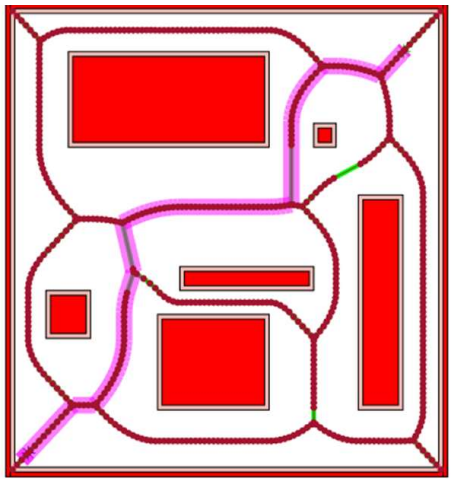

FIGURE 1.7 – Trajectoire optimale calculée à partir du diagramme de Voronoï nettoyé.

1.2.1.4 Méthode du graphe de visibilité

La méthode du graphe de visibilité est une approche de planification globale fréquemment employée dans les contextes polygonaux. Cette méthode implique la représentation des éléments essentiels de l'environnement, tels que les pics des obstacles, ainsi que les points de début et d'arrivée, selon la configuration de nœuds dans un graphe. Deux

nœuds sont liés par une arête lorsqu'il n'existe aucun obstacle sur la ligne droite les reliant, permettant ainsi une visibilité directe entre ces points.

Ce graphe représente de manière exhaustive toutes les liaisons sécurisées réalisables dans l'espace disponible. Après avoir établi le graphe de visibilité, il est possible d'utiliser un algorithme de recherche de chemin optimal tel que *Dijkstra* ou $A*$ pour calculer un itinéraire optimal reliant le point de départ à la destination [33].

Cette méthode est géométriquement optimale, car elle garantit la distance minimale dans un contexte sans incertitude, comme mentionné par Choset [10]. Un exemple de trajectoire générée par cette méthode est illustré dans la figure 1.8, accompagné d'une vidéo de démonstration accessible en ligne [2].

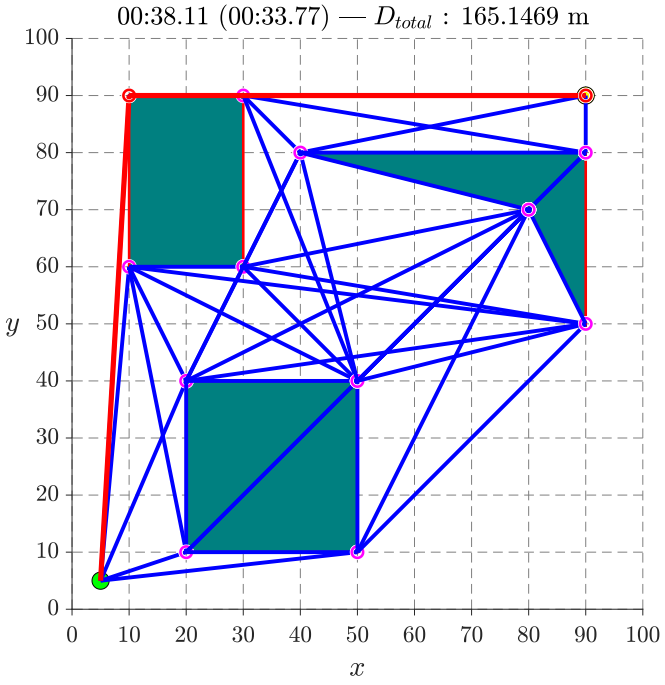

FIGURE 1.8 – Trajectoire optimale obtenue par la méthode du graphe de visibilité.

Résumé

Les techniques de navigation globale assurent une certitude appropriée en termes d'optimalité et de fiabilité dans le cas où une carte est accessible. Toutefois, leur performance diminue dans des contextes mouvementés ou partiellement connus. Il est souvent

2. Démo du graphe de visibilité : https://youtu.be/rs1t8LxjEW4

nécessaire de combiner ces technologies avec des approches locales afin de garantir une navigation fiable dans des conditions réelles.

1.2.2 Navigation locale

Les méthodes locales favorisent au robot de naviguer en temps réel en correspondant à son environnement actuel, sans recourir à une carte globale. Ces méthodes se révèlent particulièrement bénéfiques dans des contextes inexplorés ou changeants. Ces éléments comprennent les champs de potentiels, les algorithmes Bug et les contrôleurs réactifs.

1.2.2.1 Méthode du champ de potentiel (APF)

La technique du champ de potentiel artificiel (APF), développée par Khatib [26], s'établit sur la représentation du robot exposé à des forces virtuelles : une force attractive dirigée vers la cible et des forces répulsives générées à proximité des obstacles. Le robot suit la direction du gradient du potentiel total afin d'atteindre son objectif. Cette approche se distingue par sa simplicité, sa rapidité et son intuitivité ; elle est sujette à une limitation significative : la probabilité de convergence vers un minimum local, caractérisé par un champ nul sans parvenir à l'objectif.

La figure 1.9 met en lumière deux scénarios caractéristiques observés lors des simulations : À gauche, un itinéraire couronné de succès dans un environnement élémentaire ; à droite, une situation où le robot se retrouve coincé dans un minimum local en raison de la présence d'obstacles non convexes.

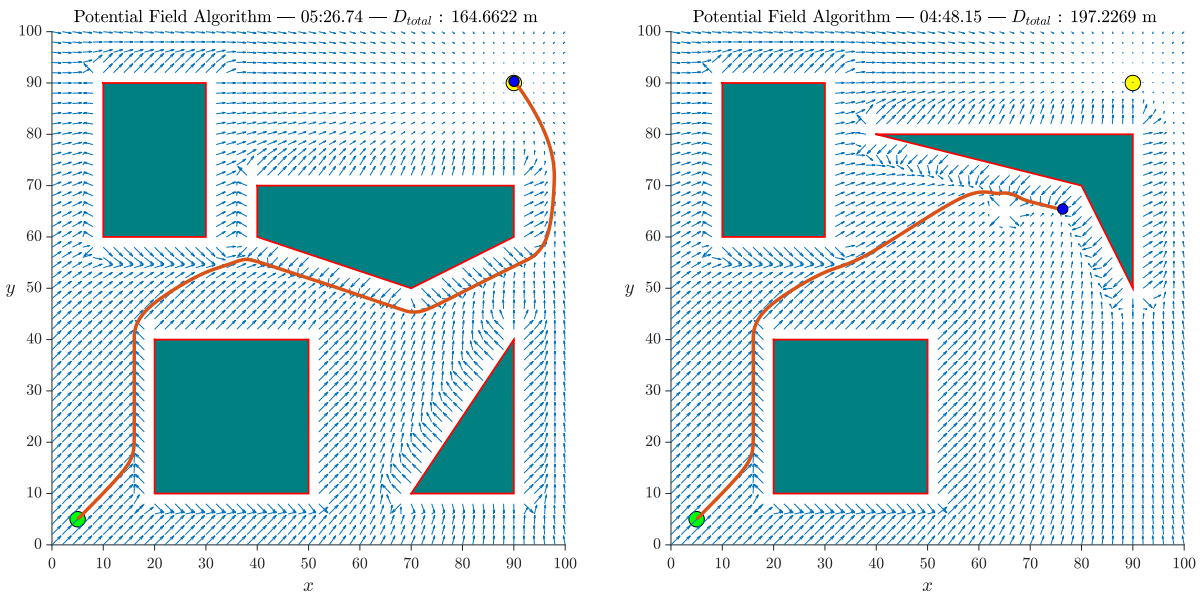

FIGURE 1.9 – Simulations de trajectoires produites à l'aide de la méthode APF. Sur la gauche : comportement nominal non problématique. Sur la droite : présence d'un minimum local. [3]

Plusieurs solutions ont été proposées pour contourner ce problème, notamment l'ajustement des paramètres du champ [19], la visualisation du gradient pour anticiper les pièges [28], ou l'intégration d'une replanification locale comme RRT ou DWA [31, 18].

1.2.3 La méthode VFH : champ vectoriel basé sur histogramme

L'algorithme VFH (Vector Field Histogram), présenté par Borenstein et Koren [7], se base sur la transformation des données de distance provenant de capteurs (tel qu'un LiDAR 2D) en un histogramme polaire illustrant la densité d'obstacles autour du robot. Cette illustration permet de repérer les zones dégagées, désignées sous le terme de « vallées », vers lesquelles le robot peut se mouvoir sans risque. Par la suite, l'algorithme choisit le meilleur chemin à travers trois paramètres : éviter les zones à forte densité, minimiser l'écart angulaire par rapport à la cible et maintenir un trajet fluide. La transformation des données locales en une représentation directionnelle utilisable est démontrée à la figure 2.11.

3. Vidéos de démonstration : https://youtu.be/_fqGW3XQyrI (cas sans minimum) et https://youtu.be/MdkDHRhuuzk (cas avec minimum local).

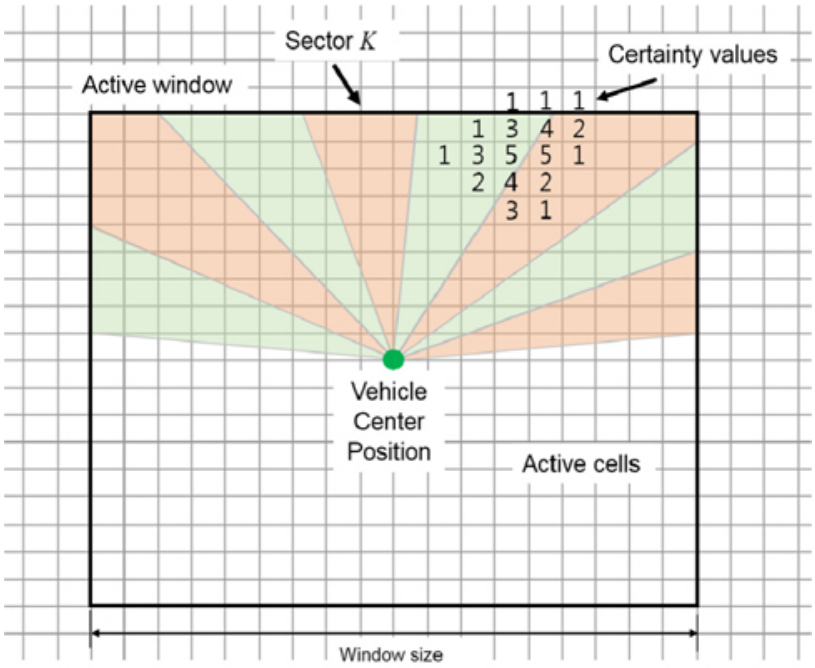

FIGURE 1.10 – Projection des cellules actives sur l'histogramme polaire dans l'approche VFH [7].

1.2.3.1 Famille des algorithmes Bug (Bug1, Bug2 et Tbug)

Les algorithmes Bug sont des techniques de navigation locale qui reposent sur la vision, permettant au robot de réagir en temps réel aux obstacles sans avoir d'évaluation préalable de l'environnement. L'algorithme Bug1 [34] contourne intégralement les obstacles avant de reprendre sa trajectoire, alors que l'algorithme Bug2 [34] tente de rejoindre la ligne directe le plus rapidement possible. Tbug [23] renforce cette approche en recourant à des calculs de distance afin de créer une carte locale et d'optimiser le trajet menant à la cible.

Algorithme Bug-1

L'algorithme Bug-1, tel que présenté par Lumelsky [34], permet au robot de se diriger en ligne droite vers sa cible, en contournant les obstacles rencontrés sur son chemin. Ensuite, le robot suit l'intégralité du contour, enregistre les points visités, et ensuite reprend son avancée à partir du point le plus proche de l'objectif. Cette approche, bien que simple et exhaustive d'un point de vue géométrique, conduit à des trajectoires étendues lorsqu'elle est appliquée dans des environnements compliqués ou comportant

des obstacles concaves [23]. Les contraintes dynamiques du robot ne sont pas prises en considération par elle-même.

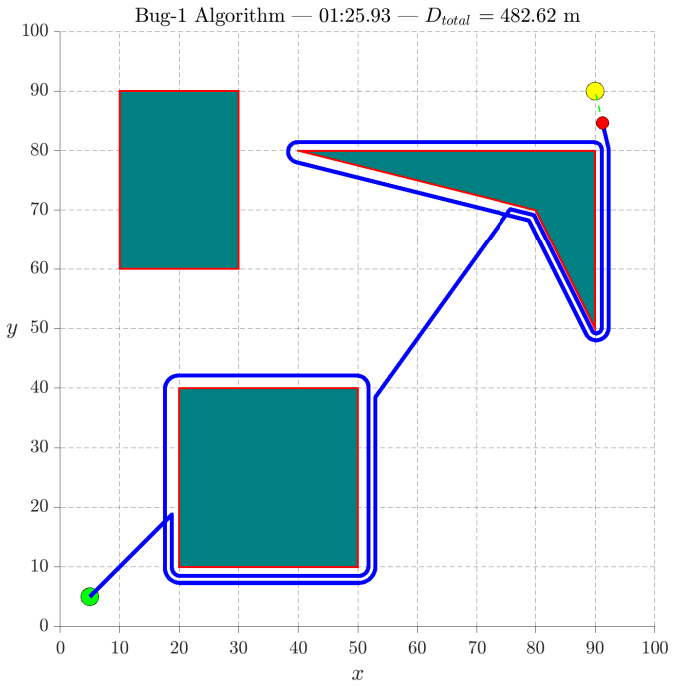

FIGURE 1.11 – Trajectoire typique de Bug-1 en présence d'obstacles de géométrie mixte. [4]

Algorithme Bug-2

L'algorithme Bug-2, décrit par Lumelsky et Stepanov [34], apporte une amélioration à l'algorithme Bug-1 en intégrant une trajectoire de référence nommée *m-ligne*, qui relie le point initial au point cible. Le robot suit la trajectoire spécifiée jusqu'à ce qu'il soit confronté à un obstacle, qu'il contourne en restant en contact, puis il reprend le suivi de la trajectoire dès qu'il atteint un point plus proche de la cible. Cette approche permet de minimiser la distance de déplacement ; cependant, elle peut être vulnérable aux surfaces concaves ou aux erreurs de mesure des capteurs.

Le schéma 1.12 expose une trajectoire réussie démontrant la fonctionnalité de Bug-2 dans un environnement comprenant divers obstacles de formes.

4. Démo de Bug-1 : https://youtu.be/9xhEM0F08dY

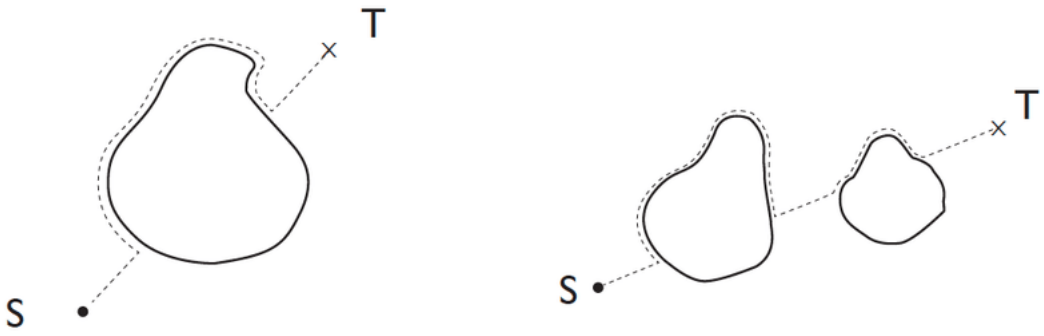

FIGURE 1.12 – Illustration du comportement de l'algorithme Bug-2. : à gauche, un obstacle unique ; à droite, plusieurs obstacles successifs [30].

Algorithme Tbug

L'approche Tbug, développée par Kamon en 1996 [23], représente une évolution sophistiquée de la catégorie d'algorithmes Bug. À la différence de Bug-1 et Bug-2, ce robot utilise de manière proactive les données de distances en temps réel provenant d'un télémètre ou d'un capteur laser pour créer une carte locale et évaluer les directions tangentes aux obstacles. Le robot sélectionne ensuite le trajet le plus optimal en réduisant un coût qui prend en compte à la fois la distance jusqu'à la cible et les obstacles à contourner.

Cette approche contribue de manière significative à l'optimisation de la navigation, en particulier en réduisant la distance parcourue et en évitant les itinéraires redondants. L'exemple représenté par la figure 1.13 met en lumière une trajectoire caractéristique produite par Tbug dans un environnement difficile.

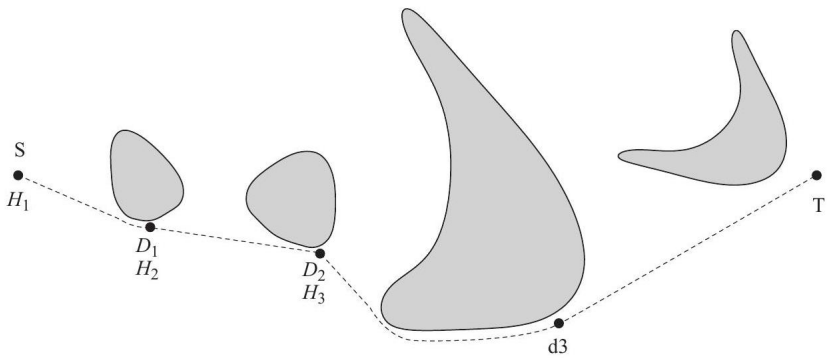

FIGURE 1.13 – Exemple de trajectoire produite par l'algorithme Tbug dans un environnement comportant plusieurs obstacles.

1.2.3.2 Nouvelles approches dans le domaine du contrôle par retour d'état

Au cours des dernières années, diverses méthodes efficaces ont été mises au point pour pallier les lacunes des approches traditionnelles, en particulier pour la sécurité, la robustesse et la stabilité de la dynamique du robot. Les stratégies de stabilisation établies sur les fonctions de navigation développées par Rimon et Koditschek [42] ont été pionnières, mais leur efficacité est réduite dans des environnements comportant des obstacles complexes. Simultanément, les fonctions barrières de contrôle (CBF) ont été introduites par [1] pour fournir un cadre mathématique visant à assurer le respect des contraintes de sécurité en temps réel. Cependant, certaines études ont révélé des comportements non souhaités dans des configurations spécifiques [50].

Pour améliorer sa performance, de nouvelles approches hybrides ont été développées, combinant des techniques de commande continue et de logique discrète. C'est le cas, par exemple, de l'étude menée par Berkane et ses collègues (2022), qui ont développé une solution stable à l'échelle mondiale, ainsi que de la recherche réalisée par Cheniouni et son équipe [9], qui ont appliqué cette approche à des environnements non familiers. D'autres recherches, telles que celles menées par Sawant [44], utilisent des champs vectoriels continus afin de produire des trajectoires réactives et fluides, tandis qu'Arslan [2] étudie l'emploi de modèles de puissance pour assurer une navigation précise et sans collision.

En outre, des travaux tels que les cônes de vitesse sécurisée (SVC) [4] ou les stratégies de commande dissipative pour les systèmes à dynamique d'ordre supérieur [47] considèrent de manière explicite les contraintes physiques du robot lors du processus de décision. De manière concomitante, l'incorporation de la perception logique dans le processus de navigation [55]facilite une connexion plus précise avec des contextes complexes et dynamiques.

Les approches locales permettent une navigation réactive, sans nécessité de carte préalable, ce qui les rend idéales pour les environnements inconnus. Bien que les méthodes simples et efficaces puissent être utilisées, elles ne garantissent pas l'optimalité et peuvent entraîner des pièges pour le robot, notamment en présence de minima locaux dans la fonction d'objectif. Les approches avancées offrent une plus grande robustesse, tout en étant additionnelles aux stratégies de planification globale.

Objectifs du mémoire

Ce mémoire a pour objectif d'étudier la mise en œuvre et l'évaluation comparative d'algorithmes de navigation réactive basés sur des commandes de vitesse, dans un contexte de robot mobile évoluant dans un environnement inconnu.

Les objectifs spécifiques sont les suivants :

1. **Implémenter plusieurs algorithmes de navigation locale réactive** dans un environnement de simulation MATLAB, en formulant le problème à l'aide de commandes de vitesse. Les algorithmes choisis incluent deux approches classiques (T-Bug et VFH) ainsi qu'un contrôleur (SVC) développé dans le laboratoire de robotique et systèmes autonomes.
2. **Adapter ces algorithmes à un robot différentiel non-holonome** en les intégrant dans l'environnement ROS/Gazebo à l'aide de la plateforme TurtleBot3. Une étape de conversion des commandes de vitesse cartésiennes vers des commandes admissibles (vitesse linéaire et angulaire) a été réalisée pour tenir compte des contraintes cinématiques du robot.
3. **Évaluer quantitativement les performances des trois approches** selon des critères objectifs : (i) la distance totale parcourue jusqu'à la cible, (ii) le temps moyen d'exécution par itération de planification, et (iii) la distance moyenne de dégagement par rapport aux obstacles. Cette évaluation permet de comparer les avantages, limitations et comportements typiques de chaque méthode dans divers scénarios simulés.
4. **Renforcement de la robustesse via l'ajout d'un PID au contrôleur SVC** : bien que le contrôleur SVC soit formulé de manière pour garantir la sécurité, des perturbations pratiques telles qu'un biais dans les actionneurs, du glissement au sol ou des erreurs de modélisation peuvent compromettre ses garanties de sécurité. L'intégration d'un régulateur PID permet de compenser ces effets en boucle fermée. Le terme proportionnel assure un retour rapide vers la consigne, le terme intégral corrige les erreurs systématiques, et le terme dérivatif anticipe les changements brusques. Cette correction dynamique améliore la robustesse face aux perturbations externes, tout en produisant des trajectoires plus stables et visuellement plus fluides.

Organisation du mémoire

Ce mémoire est structuré en quatre chapitres principaux, chacun correspondant à une étape de la démarche expérimentale :

- **Chapitre 1 – Contexte et état de l'art :** Ce chapitre présente les notions fondamentales liées à la navigation autonome des robots mobiles, en mettant l'accent sur les approches réactives. Une revue critique des méthodes existantes de planification globale et locale y est proposée, ainsi que les motivations ayant conduit à la sélection des trois algorithmes étudiés.
- **Chapitre 2 – Implémentation dans MATLAB :** Ce chapitre décrit la mise en œuvre initiale des algorithmes T-Bug, VFH et SVC dans un environnement simulé en 2D basé sur des commandes de vitesse. Cette étape a permis une première exploration comparative afin de comprendre les caractéristiques et les limitations de chaque méthode.
- **Chapitre 3 – Simulation avancée sous ROS/Gazebo :** Les algorithmes sont ensuite adaptés et testés dans un environnement de simulation 3D plus réaliste, en tenant compte des contraintes non-holonomes d'un robot différentiel. Une version améliorée du SVC, intégrant un régulateur PID, est également proposée afin d'accroître la robustesse aux perturbations. Les performances sont évaluées selon trois critères quantitatifs : la distance parcourue, le temps moyen par itération de planification et la distance moyenne de dégagement par rapport aux obstacles.
- **Chapitre 4 – Conclusion générale et perspectives :** Ce dernier chapitre résume les principaux résultats obtenus, met en évidence les apports et les limites de l'étude, et propose des orientations pour des travaux futurs, notamment l'implémentation sur plateforme réelle et l'intégration d'approches hybrides ou apprenantes.

2 Navigation réactive

2.1 Introduction

La navigation réactive est une technique qui permet à un robot de circuler de manière indépendante dans des environnements inconnus en se basant exclusivement sur des informations sensorielles locales. À la différence des approches globales, cette méthode se focalise sur la prise de décisions en temps réel afin de contourner les obstacles et d'atteindre sa cible sans recourir à une planification préalable.

Ce chapitre examine trois méthodes de navigation locale : l'approche établie sur les cônes de vitesse de sécurité (SVC), l'histogramme du champ vectoriel (VFH) et l'algorithme Tangent Bug (T-Bug). Le contrôle de vitesse proportionnelle (SVC) adopte une méthode géométrique afin de garantir la sécurité en temps réel du robot. D'autre part, le VFH génère un histogramme polaire établi sur les données LiDAR pour déterminer la meilleure orientation à suivre, tandis que le T-Bug intègre le suivi des contours et l'exploitation d'informations tangentielles pour améliorer la planification des trajectoires.

Chaque méthode est étudiée de manière approfondie à travers des mises en œuvre sous MATLAB, accompagnées d'une comparaison détaillée établie sur divers critères pour évaluer ses performances. Des suggestions d'amélioration seront présentées dans le chapitre d'implémentation sur ROS/Gazebo.

2.2 Approche des cônes de vitesse de sécurité (SVC)

La méthode des cônes de vitesse de sécurité (SVC), développée par Berkane [6], s'inscrit dans le cadre des techniques de navigation locale, impliquant la prise de décisions

en temps réel établie sur les informations sensorielles. L'objectif est d'assurer la sécurité du robot sans avoir de connaissance initiale de l'environnement, tout en le guidant vers la position souhaitée. L'un des avantages majeurs de cette méthode réside dans sa capacité à personnaliser tout contrôle nominal et à le protéger contre les obstacles dans un milieu inconnu. Cette méthode se distingue par sa similarité de complexité entre les dimensions 2D et 3D, une caractéristique peu répandue dans les méthodes de navigation locale.

La méthode SVC repose sur un modèle géométrique fondé sur le théorème d'invariance de Nagumo [38] et sur la géométrie des cônes tangents de Bouligand [8]. Cette conception garantit que le robot demeure dans un espace dégagé, même en présence d'obstacles, tout en continuant à poursuivre son objectif. Deux catégories de contrôleurs ont été suggérées dans la conception initiale de l'approche SVC. Il s'agit d'une variante discrète visant à garantir la sûreté et la convergence dans un contexte théorique, et d'une variante continue développée pour une mise en œuvre plus aisée sur le terrain. Cette thèse se focalisera sur la version continue, qui est plus appropriée pour les utilisations robotiques tout en préservant les assurances de sécurité.

Nous allons désormais exposer formellement le problème en question, ainsi que la modélisation géométrique de l'environnement dans lequel le robot opère.

2.2.1 Formulation du problème

Selon Smaili et Berkane [47], on considère des environnements n-dimensionnels contenant des obstacles convexes ou non convexes. L'environnement de travail est défini de manière formelle comme un sous-ensemble fermé $\mathcal{W} \subset \mathbb{R}^n$ qui inclut M des obstacles ouverts $\mathcal{O}_i \subset \mathbb{R}^n$, où :

$$\mathcal{O}_0 := \partial\mathcal{W}, \quad \mathbb{M} := \{0, 1, \ldots, M\}.$$

L'espace libre, noté $\mathcal{X}$, est défini par :

$$\mathcal{X} := \mathcal{W} \setminus \bigcup_{i=1}^{M} \mathcal{O}_i, \tag{2.1}$$

Il s'agit de la région abordable par le robot.

Pour garantir le déplacement dans l'espace libre $\mathcal{X}$, il est nécessaire de respecter deux exigences géométriques. La première attention concerne la distance minimale entre les

obstacles, définie par la condition suivante :

$$\mathbf{d}_{\mathcal{O}_i,\mathcal{O}_j} > 2h, \quad \forall i \neq j \in \mathbb{M}, \tag{2.2}$$

Lorsque $h > 0$ représente l'**atteignabilité normale** de $\mathcal{X}$, il s'agit de la distance maximale pour laquelle la projection d'un point sur $\partial\mathcal{X}$ demeure unique [13].

La deuxième condition porte sur les dimensions du robot et sa marge de sécurité. Elle établit la condition suivante :

$$0 < R < \varepsilon < h, \tag{2.3}$$

où R représente la portée du robot et ε la distance minimale désirée entre le robot et les obstacles. Ce critère assure que le robot est capable de se déplacer dans un environnement réalisable tout en observant les normes de sécurité requises.

L'espace de travail faisable $\mathcal{X}_\varepsilon$ est défini comme l'ensemble des configurations du robot qui assurent une distance minimale ε par rapport à la zone occupée par les obstacles. Selon la définition de [47], il est caractérisé par :

$$\mathcal{X}_\varepsilon := \{x \in \mathbb{R}^n \mid \mathbf{b}_{\complement\mathcal{X}}(x) \geq \varepsilon\}, \tag{2.4}$$

La fonction $\mathbf{b}_{\complement\mathcal{X}}(x)$ représente la distance dirigée à la frontière de l'espace libre. Cette équation garantit que la position du robot demeure à l'intérieur d'une zone sécurisée, sans entrer en contact avec les obstacles.

Le domaine $\mathcal{X}_\varepsilon$ est employé comme domaine d'application pour le contrôleur réactif suggéré, assurant à la fois la sécurité du mouvement et l'unicité de la projection sur la frontière.

Pour chaque point x appartenant à l'ensemble $\mathcal{X}_\varepsilon$ pour lequel la projection $\mathbf{P}_{\partial\mathcal{X}}(x)$ est unique, le vecteur normal unitaire sortant de la frontière est défini comme suit :

$$\nabla\mathbf{b}_{\complement\mathcal{X}}(x) := \frac{x - \mathbf{P}_{\partial\mathcal{X}}(x)}{\|x - \mathbf{P}_{\partial\mathcal{X}}(x)\|}, \tag{2.5}$$

Le vecteur normal est essentiel dans la création du cône tangent de sécurité, car il détermine la direction d'évitement en projetant le champ de vitesse nominal sur l'espace praticable.

La commande nominale $\kappa_0(x)$ est définie comme une loi proportionnelle orientée vers la cible x_d.

$$\kappa_0(x) = -k(x - x_d), \quad k > 0, \tag{2.6}$$

où k est un coefficient positif. En l'absence d'obstacles, cette commande permet au robot de se diriger directement vers sa but. Le robot est représenté par une dynamique de type intégrateur :

$$\dot{x} = u, \tag{2.7}$$

où $x \in \mathbb{R}^n$ est la position du robot, et $u \in \mathbb{R}^n$ la commande de vitesse.

En nous basant sur ce modèle, nous allons maintenant détailler la conception du contrôleur SVC, en débutant par la commande nominale, puis en ajoutant graduellement les dispositifs de sécurité.

2.2.2 Construction de la commande sécurisée

La commande u est calculée en temps réel à partir des informations des capteurs, selon la relation $u = \kappa_{\mathcal{X}}(x)$. Elle est définie de la manière suivante :

$$\dot{x} = \kappa_{\mathcal{X}}(x) \in \mathbf{P}_{\mathbf{T}_{\mathcal{X}}(x)}(\kappa_0(x)), \tag{2.8}$$

où $\mathbf{T}_{\mathcal{X}}(x)$ désigne le cône tangent local, défini par :

$$\mathbf{T}_{\mathcal{X}}(x) := \left\{ z \in \mathbb{R}^n : \nu(x)^\top z \leq 0 \right\}, \tag{2.9}$$

Nous présentons dans ce contexte la variante discontinue du contrôleur, qui repose sur une projection géométrique rigoureuse.

Lorsque la frontière $\partial\mathcal{X}$ est une hypersurface régulière, la projection sur le plan tangent est parfaitement définie. La commande sécurisée est déterminée par la projection orthogonale sur ledit hyperplan :

$$\Pi(\nu(x))\kappa_0(x) = \left(I_n - \nu(x)\nu(x)^\top\right) \kappa_0(x), \tag{2.10}$$

Cette projection permet de maintenir le vecteur vitesse dans une direction sécurisée, en respectant les conditions d'invariance établies par Nagumo.

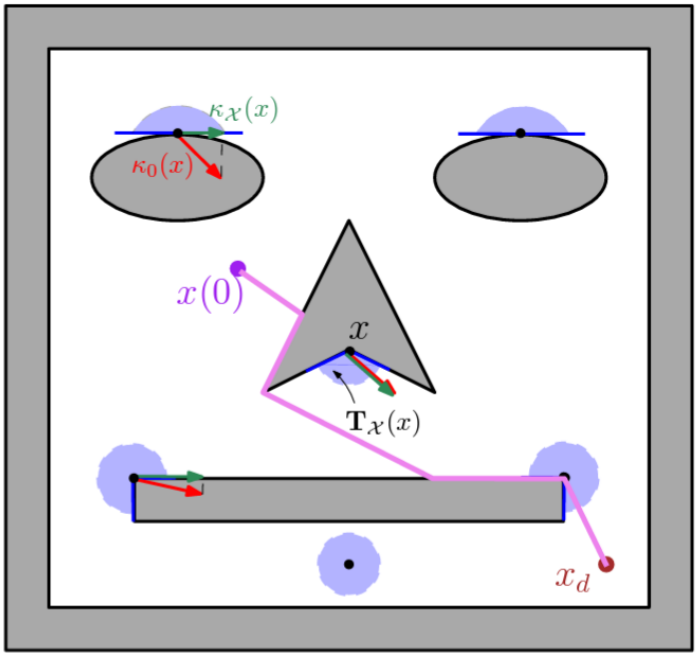

FIGURE 2.1 – Principe de projection sur les cônes tangents dans un espace libre $\mathcal{X}$.

Comme le montre la figure 2.1, cette projection ajuste de manière dynamique la direction de la commande en fonction de la configuration locale, garantissant ainsi une transition en douceur entre les modes de stabilisation et d'évitement.

La loi de commande discontinue est définie comme suite :

$$\kappa_{\mathcal{X}}(x) = \begin{cases} \kappa_0(x), & \text{si } x \in \text{int}(\mathcal{X}) \text{ ou } \nu(x)^\top \kappa_0(x) \leq 0, \\ \Pi(\nu(x))\kappa_0(x), & \text{sinon.} \end{cases} \tag{2.11}$$

Ce dispositif de contrôle assure le respect des contraintes de sécurité à la frontière par le mouvement. Il facilite la transition naturelle entre les phases de stabilisation et d'évitement.

Version continue du contrôleur SVC

La variante discontinue du régulateur SVC, malgré sa robustesse théorique, peut provoquer des réactions abruptes, notamment en cas de perturbation ou de changements fréquents de modes opératoires. Afin d'améliorer la fluidité et la robustesse de la navigation, une version continue du contrôleur a été présentée dans l'étude de [4].

En premier lieu, il est supposé que la région complémentaire $\complement\mathcal{X}$ a une mesure strictement positive. Cette caractéristique assure la singularité de la projection orthogonale d'un point voisin sur la frontière $\partial\mathcal{X}$ [13]. En se basant sur cette considération, on définit un ensemble sûr $\mathcal{X}_\varepsilon$ de la manière suivante :

$$\mathcal{X}_\varepsilon = \{x \in \mathbb{R}^n \mid d_\mathcal{X}(x) \geq \varepsilon\}, \tag{2.12}$$

où $d_\mathcal{X}(x)$ est la distance à la frontière $\partial\mathcal{X}$, et $\varepsilon > 0$ une marge de sécurité minimale.

Le contrôle continu s'appuie sur un opérateur de projection lisse qui n'est activé que quand la distance par rapport au bord atteint un niveau critique. La définition de la loi de commande est la suivante :

$$\hat{\kappa}_\mathcal{X}(x) = \begin{cases} \kappa_0(x), & \text{si } d_\mathcal{X}(x) > \varepsilon' \text{ ou } \kappa_0(x)^\top \nabla \mathbf{b}_{\complement\mathcal{X}}(x) \geq 0, \\ \hat{\Pi}(x)\,\kappa_0(x), & \text{sinon,} \end{cases} \tag{2.13}$$

où $\varepsilon' > \varepsilon$ définit une zone de transition.

L'opérateur de projection continue $\hat{\Pi}(x)$ est défini à partir du gradient de la fonction distance :

$$\hat{\Pi}(x) = I_n - \phi(x)\,\nabla \mathbf{b}_{\complement\mathcal{X}}(x)\,\nabla \mathbf{b}_{\complement\mathcal{X}}(x)^\top, \tag{2.14}$$

En présence d'un facteur d'atténuation $\phi(x)$ qui module l'intensité de la correction :

$$\phi(x) = \min\left(1, \frac{\varepsilon' - d_\mathcal{X}(x)}{\varepsilon' - \varepsilon}\right), \tag{2.15}$$

Cela assure que la fonction $\phi(x)$ est comprise dans l'intervalle $[0, 1]$. Quand le robot se trouve à une distance de la frontière, la commande demeure constante ($\phi(x) = 0$). À mesure que la distance $d_\mathcal{X}(x)$ tend vers ε dans la zone limite, la commande est progressivement ramenée à l'intérieur de l'ensemble $\mathcal{X}$.

L'itération continue du régulateur SVC améliore significativement la robustesse dans des situations réelles. Elle inclut de manière explicite une marge de sécurité ε ainsi qu'une transition progressive entre les modes de stabilisation et d'évitement, ce qui la rend particulièrement appropriée pour les environnements bruyants et imprévus.

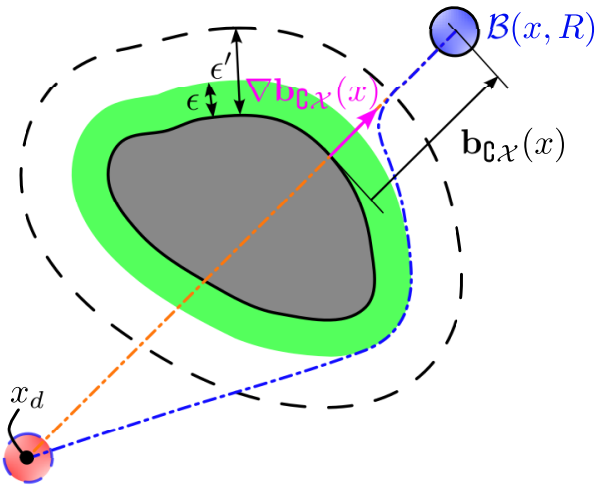

FIGURE 2.2 – Le comportement du régulateur SVC continu consiste à entourer l'obstacle (en gris) d'une marge de sécurité (en vert) d'épaisseur ε. Le vecteur $\nabla \mathbf{b}_{\complement\mathcal{X}}(x)$ de couleur magenta représente la direction normale sortie de l'obstacle. La fonction $\mathbf{b}_{\complement\mathcal{X}}(x)$ évalue la distance orientée par rapport à l'espace complémentaire libre. Afin d'assurer la sécurité, la fonction $\kappa_0(x)$ est progressivement intégrée dans l'espace libre $\mathcal{X}$ [47].

Comme le montre la figure 2.2, cette itération garantit une transition en douceur entre la stabilisation et l'évitement, tout en préservant les garanties théoriques de sécurité. Le champ de commande s'ajuste en permanence en fonction de la proximité des obstacles, sans avoir besoin de changements brusques. Ceci en fait une solution spécialement appropriée pour des environnements réels, dans lesquels les capteurs peuvent être affectés par du bruit et des incertitudes.

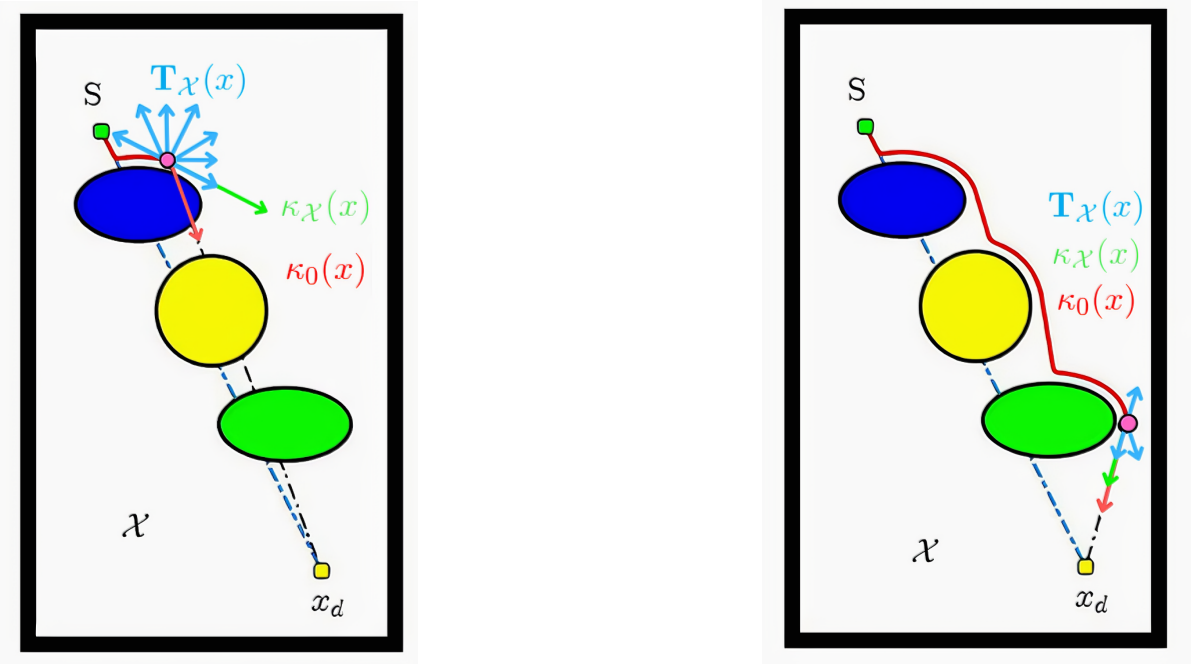

FIGURE 2.3 – Illustration des régimes de fonctionnement : évitement (gauche) et stabilisation (droite).

Comme illustré dans la figure 2.3, cette approche traite de manière efficace les passages d'un mode de commande à un autre. Toutefois, l'interruption du contrôleur peut causer des comportements non souhaités dans des environnements réels bruyants. La section suivante montre une formulation continue de cette loi.

Une fois la conception du contrôleur déterminée, nous abordons désormais les contraintes de cette méthode, notamment les structures géométriques difficiles qui provoquent des effets non souhaités.

2.2.3 Limites du contrôleur SVC

Points d'équilibre indésirables

Dans certaines circonstances, le robot peut converger vers un point stable non souhaité, localiser à la frontière de l'espace opérationnel, sans parvenir à atteindre sa destination. Ces circonstances sont en adéquation avec le groupe mentionné par Smaili [47] :

$$\mathcal{E} := \{x \in \partial\mathcal{X}_\varepsilon \mid x - x_d = \lambda \nabla \mathbf{b}_{\complement\mathcal{X}}(x),\ \lambda > 0\}\,, \tag{2.16}$$

Lorsque x_d représente la cible, $\nabla \mathbf{b}_{\complement\mathcal{X}}(x)$ est le vecteur normal dirigé vers l'intérieur de la frontière. Ces équilibres se manifestent lorsque la trajectoire menant à la cible est en totale opposition avec l'obstacle le plus près. La stabilité de ces équilibres est étroitement liée à la courbure géométrique des obstacles, comme discuté dans le point suivant.

Limites géométriques du contrôleur SVC

Malgré les promesses théoriques fournies par l'approche SVC, certaines configurations géométriques spécifiques peuvent entraîner des comportements non souhaités. Ces limitations trouvent leur origine principalement dans deux facteurs : la non-convexité des obstacles et la faible courbure locale de leur frontière.

Effet de la convexité : Comme le montre la figure 2.4, un obstacle non convexe peut engendrer des équilibres indésirables. Dans cette situation, le robot tend vers un point stable $\bar{x}$ localisé sur l'obstacle sans parvenir à atteindre la cible x_d. En revanche, un obstacle convexe favorise une trajectoire fluide vers la cible, mettant en évidence l'importance de cette caractéristique géométrique.

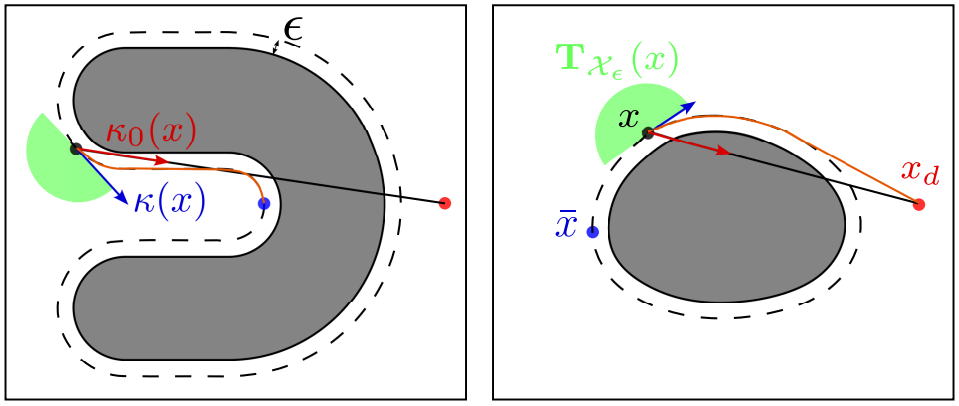

FIGURE 2.4 – L'impact de la convexité sur les itinéraires est illustré dans cette figure : du côté gauche, la présence d'un obstacle non convexe entraîne un point d'équilibre indésirable $\bar{x}$; tandis que, du côté droit, un obstacle convexe favorise la convergence vers la cible x_d [47].

Effet de la courbure : Néanmoins, même en présence d'obstacles convexes, il est possible d'observer l'apparition de points d'équilibre localement stables. La figure 2.5

met en évidence que l'augmentation de la courbure peut supprimer ces points d'équilibre indésirables en rendant instables les zones d'instabilité.

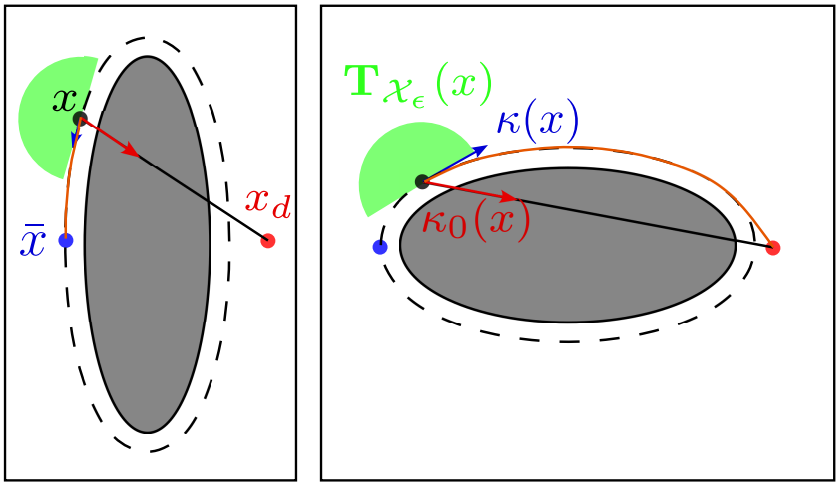

FIGURE 2.5 – Effet de la courbure sur la stabilité des trajectoires : à gauche, une courbure faible favorise un équilibre parasite ; à droite, une courbure suffisante assure la convergence vers la cible x_d [47].

Ces observations mettent en évidence l'importance de conditions géométriques supplémentaires telles que la stricte convexité ou la courbure minimale afin d'assurer la convergence presque globale du robot. Il est possible de considérer des approches hybrides ou correctives afin de surmonter ces limitations [5].

Ces limites motivent l'usage d'une perception fiable de l'environnement. Dans ce contexte, la section suivante est consacrée à la modélisation mathématique du capteur LiDAR, utilisé pour détecter les obstacles en temps réel.

2.2.4 Navigation par LiDAR : principe de fonctionnement

Les capteurs LiDAR 2D occupent une place essentielle dans le cadre de la navigation locale réactive en ce qui concerne la détection des obstacles. Ils opèrent en scannant l'environnement à l'aide de rayons laser et en mesurant les distances aux obstacles observés. Cette partie expose une modélisation mathématique précise du processus de mesure, en tenant compte de la géométrie des obstacles rencontrés.

Un capteur LiDAR 2D, situé en un point $\mathbf{x} = (x_r, y_r)$, envoie des rayons laser dans diverses directions $\theta \in [0°, 360°)$, avec une précision angulaire de 1°. Chaque direction est caractérisée par un modèle mathématique représentant la distance mesurée.

$$\rho(\mathbf{x},\theta) = \begin{cases} \min\limits_{\lambda\in[0,R]} d\big(\mathbf{x}, \mathbf{x}+\lambda[\cos\theta,\, \sin\theta]^\top\big), & \text{si obstacle détecté,} \\ R, & \text{sinon,} \end{cases} \tag{2.17}$$

où R est la portée maximale du capteur, et λ la distance parcourue par le faisceau dans la direction θ.

Ce modèle de perception est par la suite implémenté dans un cadre de simulation afin d'évaluer l'efficacité du contrôleur SVC à travers différents scénarios d'expérimentation.

2.2.5 Implémentation de l'approche SVC sous MATLAB

Cette section présente l'évaluation numérique du contrôleur *Safety Velocity Cones* (SVC) dans un environnement inconnu en 2D, avec perception locale assurée par un capteur LiDAR simulé. L'objectif est d'étudier l'impact du paramètre de lissage ε', de la portée du capteur, et de la géométrie des obstacles sur la qualité de navigation.

Simulation numérique des données LIDAR

La mesure finale $\rho(\mathbf{x},\theta)$ correspond à la distance jusqu'à la plus proche intersection valide, ou à R s'il n'y en a pas. La procédure est itérée pour chaque orientation, produisant ainsi un vecteur de mesures appelé `data_lid`.

- L'analyse est basée sur la résolution analytique des intersections entre le laser et l'obstacle.
- Les formes elliptiques et polygonales sont gérées différemment en utilisant des méthodes spécifiques : la méthode quadratique est appliquée aux ellipses, tandis que la méthode linéaire est utilisée pour les segments.
- Seules les intersections réelles, positives et situées dans l'intervalle R sont pris en compte.

Discontinuités et représentation en coordonnées polaires

Les discontinuités se produisent lorsque le discriminant de l'équation quadratique change de signe, ce qui entraîne la transition de deux solutions à aucune. Ces discontinuités indiquent la présence d'arêtes, d'obstacles ou d'espaces ouverts entre ces derniers.

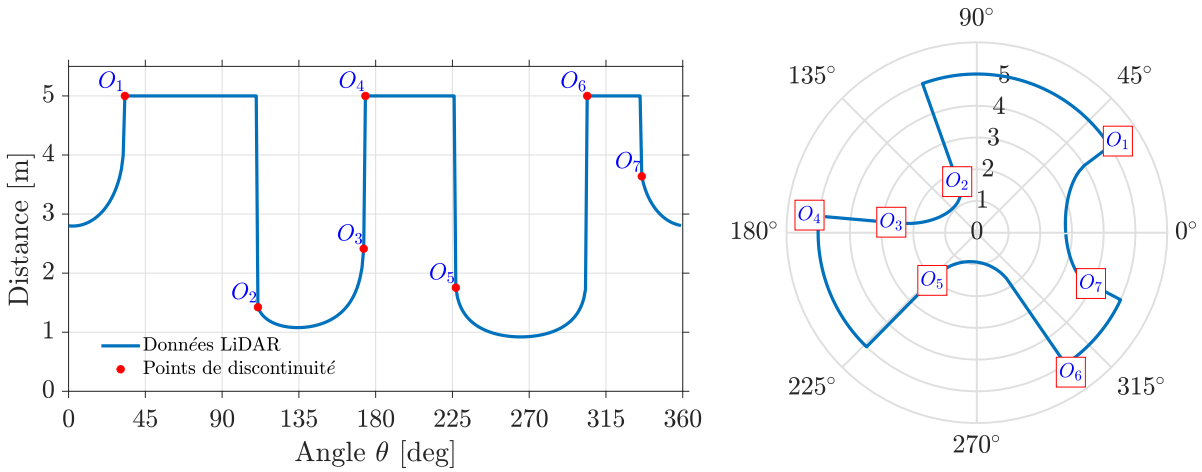

FIGURE 2.6 – Les données LiDAR sont représentées dans un système de coordonnées cartésiennes à gauche et dans un système de coordonnées polaires à droite. Les points de discontinuité signalent les transitions critiques.

Le robot évolue dans un environnement contenant plusieurs obstacles convexes, détectés à l'aide d'un capteur LiDAR 2D simulé numériquement, conformément à la modélisation présentée en section 2.2.4. Le contrôleur SVC est implémenté sous MATLAB selon la loi continue définie par l'équation (2.13), avec différentes valeurs du paramètre ε'. Deux cas sont considérés :

- **Portée infinie** du LiDAR : le robot perçoit l'environnement globalement.
- **Portée limitée** : perception locale simulant un capteur réel.

La figure 2.7 montre les trajectoires obtenues pour différentes valeurs de ε' avec une portée infinie. Une plus grande valeur de ε' induit une anticipation plus marquée et donc une trajectoire plus lissée.

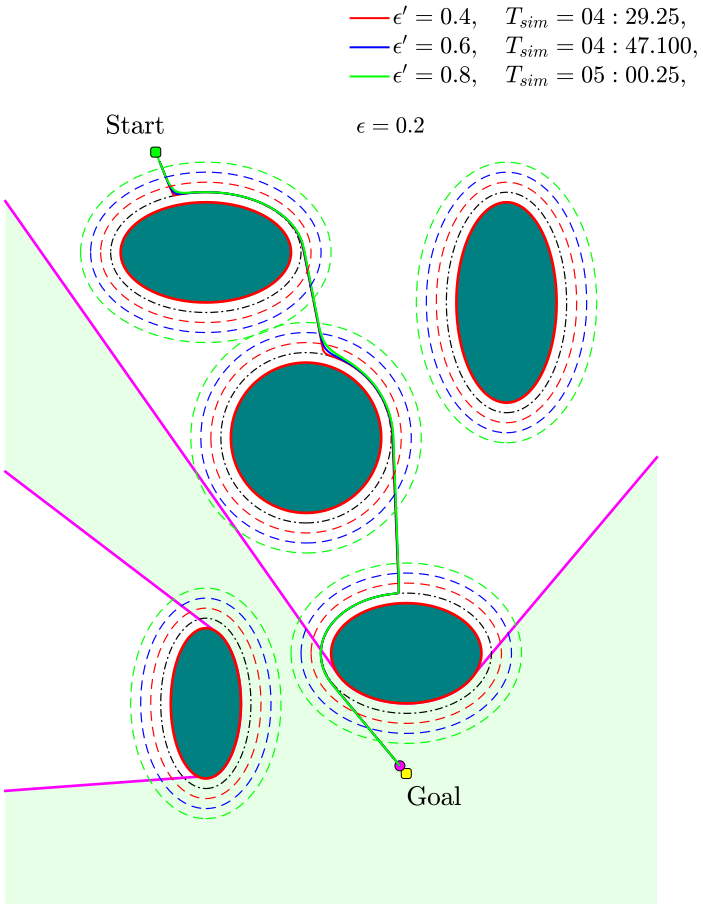

FIGURE 2.7 – Trajectoires de SVC avec différentes valeurs de ε', pour une portée infinie du capteur.

La figure 2.8 présente une simulation plus réaliste avec une portée limitée. Les effets ε' sont amplifiés : des valeurs trop faibles engendrent des comportements risqués et des manœuvres serrées à proximité des obstacles.

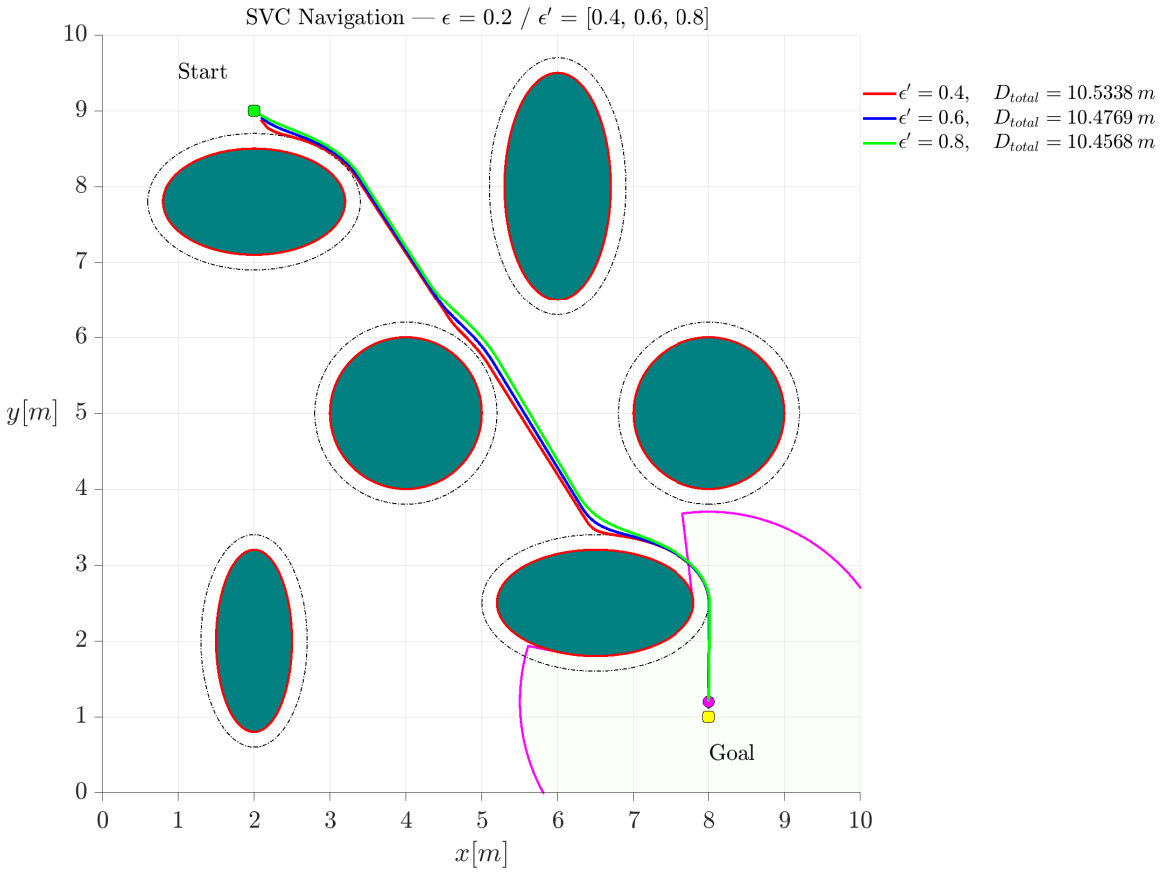

FIGURE 2.8 – Trajectoires de SVC avec différentes valeurs de ε' sous portée limitée.[1]

L'étude comparative des trajectoires générées pour différentes valeurs de ε' (figure 2.7) révèle que des valeurs plus élevées tendent à améliorer l'anticipation des obstacles, rendant les trajectoires plus lisses et sécurisées. Toutefois, un ε' trop grand peut allonger inutilement le chemin en éloignant le robot de son objectif. À l'inverse, une valeur trop faible réduit la marge de sécurité, exposant le robot à des trajectoires agressives à proximité des obstacles. Ainsi, le choix optimal de ε' dépend fortement de la densité et de la géométrie de l'environnement. Une piste prometteuse consiste à adapter dynamiquement ε' en fonction de la concentration locale d'obstacles, permettant un compromis intelligent entre réactivité, sécurité et efficacité de navigation.

1. Démo de SVC avec différentes valeurs de ε' : https://youtu.be/EsyBDyjwexs

Recommandations pratiques

- Privilégier un réglage intermédiaire de ε' (typiquement 0.6) pour équilibrer l'évitement et la convergence.
- Adapter dynamiquement ε' selon la densité locale d'obstacles et la portée effective du capteur.

Ces résultats numériques permettent de tirer plusieurs enseignements pratiques concernant le choix des paramètres et l'implémentation du contrôleur SVC dans un contexte robotique réel.

Les résultats issus de la simulation offrent des mesures précises pour l'ajustement du contrôleur SVC et son application sur un robot réel.

2.3 L'algorithme VFH (Histogramme de champ vectoriel)

L'algorithme VFH (Histogramme de champ vectoriel), introduit par Borenstein et Koren en 1991 [7], constitue une méthode de navigation locale réactive largement utilisée en robotique mobile. Son objectif est de diriger le robot vers une cible tout en assurant l'évitement des obstacles, sans nécessiter de cartographie préalable de l'environnement.

Le fonctionnement repose sur la conversion des mesures de distance, typiquement fournies par un capteur LiDAR 2D, en un histogramme polaire décrivant la densité d'obstacles autour du robot selon les différentes directions. Cette représentation permet d'identifier des zones dégagées, appelées « vallées », vers lesquelles le robot peut se diriger sans risque de collision.

À chaque instant, le VFH choisit la direction de déplacement optimale qui répond à ces 3 critères essentiels :

- **Sécurité** : éviter les directions associées à une forte densité d'obstacles.
- **Efficacité** : limiter l'écart angulaire avec la direction de la cible.
- **Continuité** : maintenir une trajectoire stable, sans changements brusques.

Ces choix sont effectués sur la base d'un histogramme local mis à jour en continu, centré sur la position instantanée du robot. Cela permet au VFH de réagir dynamiquement à l'apparition potentielle d'obstacles. Grâce à sa simplicité de calcul, sa réactivité,

sa robustesse face à l'incertitude et sa performance, l'approche VFH est bien adaptée à la navigation autonome dans des environnements inconnus.

2.3.1 Structure des données issues des capteurs

Le traitement des données dans VFH repose sur une architecture hiérarchique, permettant de convertir les mesures des capteurs en commandes de navigation efficaces et réactives. Cette structure assure un traitement en temps réel des mesures de distance, tout en maximisant la réactivité du système. On distingue les trois niveaux suivants [7] :

(**1**) **Niveau 1 – Grille cartésienne** : L'environnement est représenté par une grille d'histogramme bidimensionnelle $\boldsymbol{C}$, centrée sur le robot. Chaque cellule est mise à jour en continu à partir des lectures de capteurs (ex. : LiDAR), en accumulant une valeur de certitude reflétant la probabilité d'occupation.

(**2**) **Niveau 2 – Histogramme polaire** : La grille $\boldsymbol{C}$ est convertie en un histogramme polaire $\boldsymbol{H}$, divisé en N secteurs angulaires. Chaque secteur k fournit une estimation de la densité d'obstacles dans la direction correspondante, à partir des cellules actives $\boldsymbol{C}^*$.

(**3**) **Niveau 3 – Commande/direction du robot** : Ce niveau utilise l'analyse de l'histogramme polaire $\boldsymbol{H}$ pour générer des commandes de direction et de vitesse, ce qui permet de prendre la décision locale instantanément.

La figure 2.9 illustre le fonctionnement du capteur de distance. Le capteur LiDAR balaie l'espace avec 24 faisceaux répartis sur 360°, chacun mesurant la distance au premier obstacle détecté. La cellule correspondante dans la grille cartésienne $\boldsymbol{C}$ est alors incrémentée, indiquant une forte probabilité d'occupation.

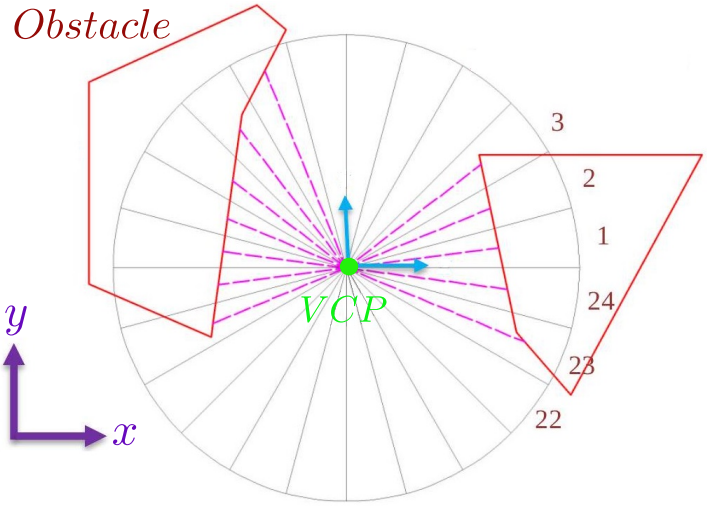

FIGURE 2.9 – Exemple d'acquisition de données par un capteur embarqué : chaque ligne rose indique une mesure de distance, projetée dans la grille d'histogramme.

La figure 2.10 illustre la mise à jour de la grille d'occupation. À gauche, seules les cellules visées par les capteurs actifs (ex. capteurs 3 et 4) sont incrémentées. À droite, ces mises à jour successives construisent progressivement une carte locale reflétant la probabilité d'occupation autour du robot.

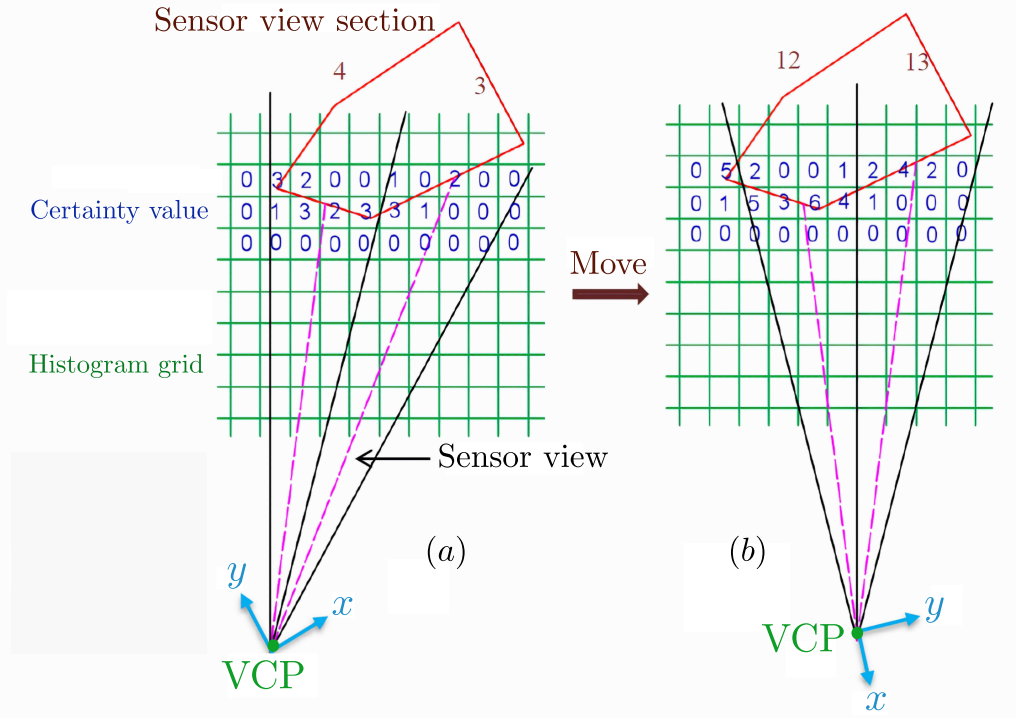

FIGURE 2.10 – Cartographie locale dans la grille d'histogramme. (a) Incrémentation individuelle des cellules à chaque mesure. (b) Accumulation progressive des zones occupées.

2.3.2 Première réduction des données : construction de l'histogramme polaire

Le VFH commence par convertir les mesures cartésiennes de la grille $\boldsymbol{C}$ issues des capteurs en un histogramme polaire $\boldsymbol{H}$, centré sur le robot. Cette représentation angulaire permet d'évaluer la densité d'obstacles dans chaque direction autour du robot.

Pour analyser localement son environnement, le robot utilise une **fenêtre active** carrée de taille $(w_s \times w_s)$, centrée sur sa position. Cette fenêtre se déplace avec le robot et sélectionne les cellules dites actives $\boldsymbol{C}^*$, situées à l'intérieur de cette zone.

Chaque cellule $(i, j) \in \boldsymbol{C}^*$ est interprétée comme un **vecteur obstacle**, défini par :

– **Direction angulaire** :

$$\beta_{(i,j)} = \tan^{-1}\left(\frac{y_j - y_{\text{vcp}}}{x_i - x_{\text{vcp}}}\right) \tag{2.18}$$

où (x_i, y_j) sont les coordonnées de la cellule, et $(x_{\text{vcp}}, y_{\text{vcp}})$ celles du centre du robot.

– **Magnitude du vecteur** :

$$m_{(i,j)} = \left(C^*_{(i,j)}\right)^2 \left(a - b \cdot d_{(i,j)}\right) \tag{2.19}$$

où :

— $C^*_{(i,j)}$ est la valeur de certitude d'occupation de la cellule active,

— $d_{(i,j)}$ est la distance entre la cellule et le centre du robot,

— a et b sont des coefficients empiriques choisis pour que :

$$a - b \cdot d_{\max} = 0, \quad \text{avec } d_{\max} = \frac{\sqrt{2}(w_s - 1)}{2}.$$

Ce processus illustré sur la figure 2.11 permet de transformer les données locales issues des capteurs en une représentation vectorielle utile pour la navigation.

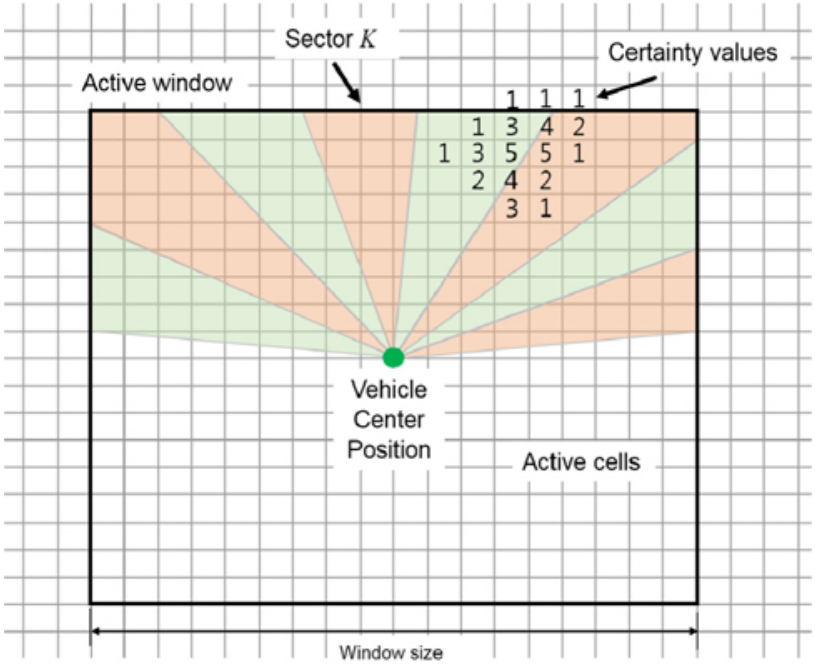

FIGURE 2.11 – Projection des cellules actives sur l'histogramme polaire dans l'approche VFH [7].

Construction de l'histogramme polaire

Une fois les vecteurs obstacles définis dans la fenêtre active, l'algorithme les projette dans un histogramme polaire unidimensionnel $\boldsymbol{H}$, centré sur le robot. Cet histogramme discrétise l'espace angulaire en N secteurs égaux de résolution α degrés, typiquement $\alpha = 5°$, soit $N = 72$ pour couvrir 360°.

Chaque cellule active $(i, j) \in \boldsymbol{C}^*$ contribue à un secteur k donné de l'histogramme selon :

$$k = \left\lfloor \frac{\beta_{(i,j)}}{\alpha} \right\rfloor \tag{2.20}$$

La **densité d'obstacle** dans le secteur k, notée h_k, représente la probabilité de rencontrer un obstacle dans la direction de ce secteur :

$$h_k = \sum_{(i,j)\in \boldsymbol{C}_k^*} m_{(i,j)} \tag{2.21}$$

où $\boldsymbol{C}_k^* \subset \boldsymbol{C}^*$ désigne l'ensemble des cellules actives associées au secteur k.

Ce mécanisme permet de construire un histogramme angulaire $\boldsymbol{H}$, reflétant la densité d'obstacles dans toutes les directions autour du robot, et constituant la base de la prise de décision.

Lissage de l'histogramme polaire : pour atténuer le bruit des capteurs et lisser les transitions directionnelles, un filtre linéaire symétrique est appliqué [7] :

$$h_k^* = \frac{1}{\sum_{n=-l}^{l} w_n} \sum_{n=-l}^{l} w_n h_{k+n}, \qquad \text{avec} \quad w_n = l - |n| + 1 \tag{2.22}$$

où :

- l est le rayon de la fenêtre de lissage [2] (ex. $l = 5$),
- w_n sont des poids décroissants symétriques autour de k,
- h_k^* est l'histogramme polaire lissé, utilisé dans les étapes de sélection de direction.

Comme présenté précédemment à la figure 2.11, la projection des cellules actives dans la fenêtre locale sur un histogramme polaire permet de synthétiser la densité d'obstacles

2. l est appelé aussi le facteur de lissage, correspondant à la moitié de la largeur de la fenêtre de convolution, soit une fenêtre totale de $2l + 1 = 11$ secteurs.

autour du robot. Après lissage (équation 2.22), cet histogramme sert de fondement à la sélection directionnelle décrite dans la section suivante.

2.3.3 Optimisation des données et sélection de la direction de déplacement

Une fois l'histogramme polaire lissé h_k^* obtenu, le VFH procède à l'analyse des directions sûres, correspondant aux zones de faible densité. L'objectif est d'identifier les vallées permettant au robot de se déplacer sans collision, comme illustré à la figure 2.12.

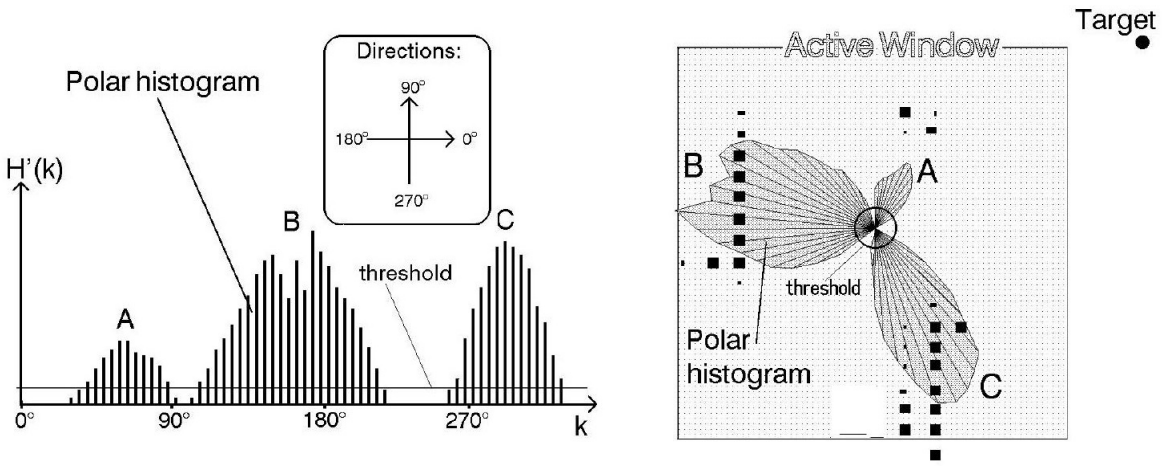

FIGURE 2.12 – Histogramme polaire lissé : les pics représentent les directions obstruées, tandis que les creux (sous le seuil η_1) indiquent les directions praticables.

Ces vallées sont détectées en comparant chaque valeur h_k^* à un seuil de décision η_1. Si $h_k^* < \eta_1$, le secteur angulaire k est considéré comme admissible et ajouté à l'ensemble des vallées candidates, défini par :

$$\mathcal{V} = \{k \in [0, N-1] \mid h_k^* < \eta_1\} \tag{2.23}$$

L'identification de ces vallées se fait par segmentation de l'histogramme lissé, ce qui permet de localiser les plages de directions adjacentes satisfaisant le critère du seuil. La figure 2.13 illustre ce processus : les creux sous le seuil représentent les directions libres (en vert), tandis que les pics indiquent les obstacles (en rouge). Ces vallées constituent les candidats potentiels pour orienter le robot.

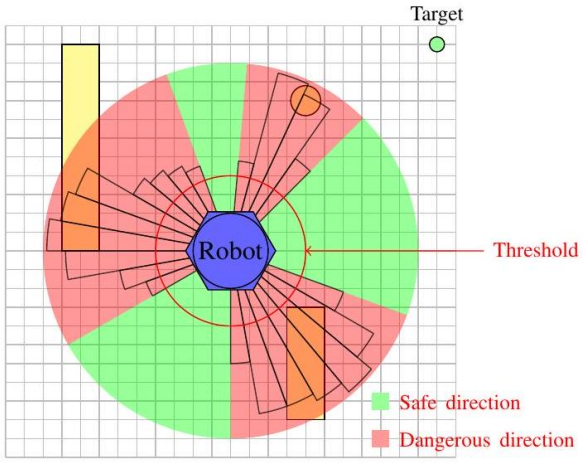

FIGURE 2.13 – Détection des vallées candidates à partir de l'histogramme lissé h_k^*. Les secteurs en vert représentent les directions sécuritaires ; ceux en rouge sont à éviter.

Une fois les vallées détectées, l'algorithme sélectionne celle dont l'axe est le plus proche de la direction cible. Ce choix équilibre trois critères : sécurité (éviter les zones denses), efficacité (réduction de l'écart angulaire avec la cible) et continuité (limitation des changements brusques de direction).

2.3.4 Stratégie de contrôle de direction

Une fois les vallées candidates identifiées, l'algorithme VFH doit choisir la direction vers laquelle orienter le robot. L'idée est de sélectionner la vallée dont l'axe médian est le plus proche de la direction cible k_T, afin d'assurer une progression rapide et sécurisée.

La direction optimale θ est calculée en prenant la moyenne entre les indices de bord de la vallée, notés k_n et k_f :

$$\theta = \frac{k_n + k_f}{2} \tag{2.24}$$

où :

- k_T est la direction vers la cible ;
- k_n est le bord d'entrée de la vallée le plus proche de k_T ;
- k_f est le bord de sortie de la vallée (égal à $k_n + S_{\max}$ pour une vallée large).

Ce choix assure que le robot se dirige au centre de l'ouverture détectée, minimisant ainsi les risques de collision.

En présence d'obstacles latéraux, l'orientation θ est ajustée dynamiquement selon trois situations typiques, illustrées en figures 2.14 et 2.15 :

(**a**) **Trop proche d'un obstacle** : θ est tournée vers l'extérieur pour éloigner le robot du danger (figure 2.14, gauche).

(**b**) **Trop éloigné** : θ est orientée vers l'intérieur pour recentrer la trajectoire dans la vallée (figure 2.14, droite).

(**c**) **Distance optimale** : si le robot est correctement centré (la distance latérale d_s est correcte), θ devient parallèle au bord, assurant une navigation stable (figure 2.15).

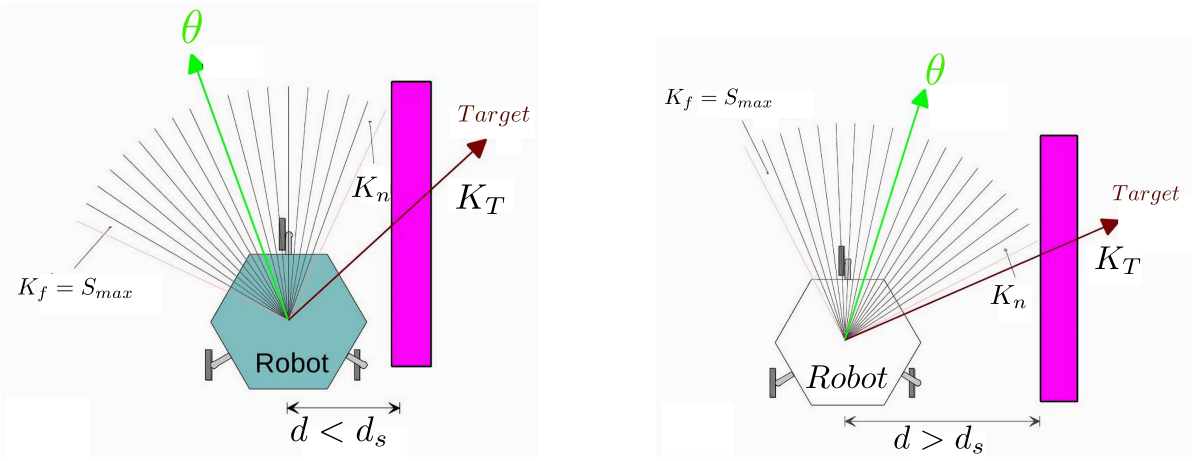

FIGURE 2.14 – Comportement latéral du robot : ajustement vers l'extérieur (gauche) ou recentrage (droite).

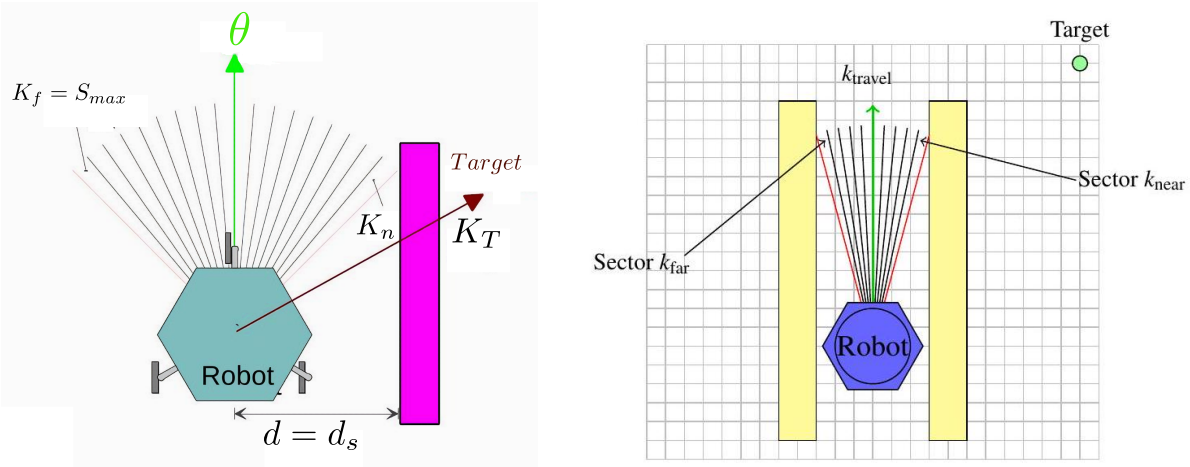

FIGURE 2.15 – Suivi latéral à distance constante (gauche) et navigation centrée dans une vallée étroite (droite).

Lorsque la direction k_T est bloquée par un obstacle, le robot doit sélectionner la vallée libre dont l'axe est le plus proche de la direction initiale souhaitée. La figure 2.16 illustre ce processus : le robot identifie une ouverture sûre, puis ajuste sa direction θ en se recentrant sur l'axe médian de cette vallée.

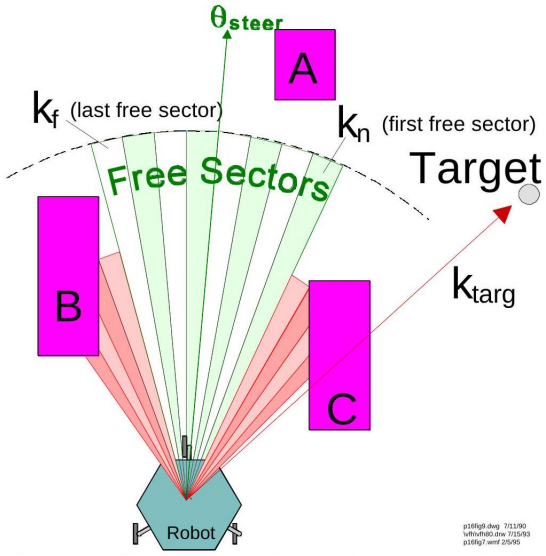

FIGURE 2.16 – Sélection d'une nouvelle direction lorsque la cible est bloquée.

Ainsi, le VFH assure une navigation réactive et fluide, capable de s'adapter dynamiquement aux obstacles tout en poursuivant efficacement la cible.

Le seuil de décision η_1

Le seuil η_1 est un paramètre important dans le VFH. Il permet de distinguer, dans l'histogramme polaire lissé h_k^*, les directions sûres (vallées) des zones à éviter. Concrètement, tous les secteurs angulaires k tels que $h_k^* < \eta_1$ sont considérés comme praticables et ajoutés à l'ensemble des directions admissibles. Les secteurs dépassant ce seuil sont exclus de la navigation.

Lafigure 2.12 illustre ce processus : le seuil η_1 est représenté par une ligne horizontale sur l'histogramme polaire, tandis qu'il est indiqué par un cercle rouge sur la figure 2.13, séparant les pics (zones occupées) des creux (directions libres). La valeur de η_1 influence directement la largeur et le nombre de vallées détectées.

Cependant, pour rendre la méthode VFH robuste face aux variations du seuil η_1 [7]. Deux cas principaux doivent être pris en compte :

– **Seuil trop élevé** : certaines directions sûres peuvent être ignorées, rendant la navigation hésitante ou bloquée dans des passages étroits.

– **Seuil trop faible** : des directions risquées peuvent être sélectionnées, augmentant le risque de collision.

Cependant, VFH reste relativement robuste aux variations du seuil η_1, car la direction finale est toujours choisie au centre d'une vallée libre, ce qui amortit les effets des fluctuations mineures du seuil.

Après la sélection de la direction, la vitesse du robot doit également être adaptée selon la densité d'obstacles pour garantir une navigation sûre.

Commande de vitesse

En complément du contrôle de direction, l'algorithme VFH ajuste dynamiquement la vitesse du robot en fonction de la densité d'obstacles dans sa direction de déplacement désirée. L'objectif est de ralentir dans les passages étroits ou encombrés, tout en maintenant une vitesse suffisante dans les zones dégagées.

La densité d'obstacles lissée dans la direction visée par la commande est notée h_c^*. Une vitesse cible v^* est alors définie selon la relation suivante [7] :

$$v^* = v_{\max}\left(1 - \frac{h_c^{**}}{h_m}\right), \qquad \text{où} \quad h_c^{**} = \min(h_c^*, h_m) \tag{2.25}$$

où :

– $v_{\max}$ est la vitesse maximale du robot ;

– h_m est un seuil de densité au-delà duquel la vitesse doit être fortement réduite ;

– h_c^{**} est une densité saturée pour éviter que v^* ne devienne négative.

De plus, pour assurer un déplacement fluide notamment lors des virages serrés, la vitesse est modulée en fonction du taux de braquage Ω :

$$v = v^*\left(1 - \frac{\Omega}{\Omega_{\max}}\right) + v_{\min} \tag{2.26}$$

où :

– Ω est le taux de rotation angulaire actuel ;

– $\Omega_{\max}$ est le taux de braquage maximal autorisé (par exemple, 120°/s) ;

– $v_{\min}$ est une vitesse minimale imposée pour éviter l'arrêt complet (ex : 1.2 cm/s).

Ainsi, même en cas de rotation maximale ($\Omega = \Omega_{\max}$), la vitesse du robot reste strictement positive grâce à l'offset $v_{\min}$.

2.3.5 Résultats de simulation et discussion

Cette section présente les résultats de simulation obtenus avec l'algorithme VFH implémenté sous MATLAB. Plusieurs simulations ont été réalisées pour évaluer les performances de la navigation en environnement inconnu. La figure 2.17 décrit l'organigramme du fonctionnement de l'algorithme VFH, de l'acquisition des données capteurs jusqu'à la commande de déplacement.

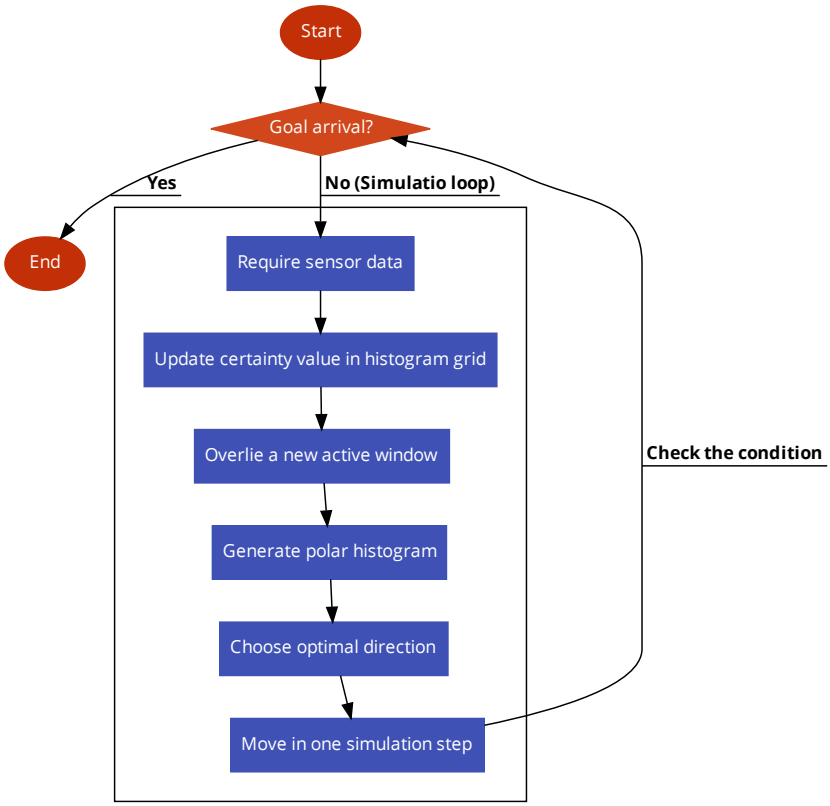

FIGURE 2.17 – Organigramme de l'algorithme VFH [7].

Les 6 étapes principales du processus de la simulation du VFH sont :

(1) Acquisition des distances par le capteur embarqué (LiDAR) ;

(2) Mise à jour de la grille locale de certitude ;

(3) Superposition de la fenêtre active centrée sur le robot ;

(4) Construction de l'histogramme polaire ;

(5) Détection des vallées et sélection de la direction optimale de déplacement ;

(6) Déplacement et mise à jour du robot.

Analyse de la navigation locale avec l'algorithme VFH

La figure 2.18 présente l'acquisition locale des données, la fenêtre active du robot, l'histogramme polaire lissé avec le seuil de décision, et la direction angulaire sélectionnée pour générer la trajectoire de navigation.

(a) En haut à gauche, la trajectoire noire du robot représenté par un cercle mauve, partant du point de départ S pour atteindre la cible G, tout en contournant les obstacles. La *fenêtre active*, représentée par un carré rouge, accompagne le déplacement du robot et permet une analyse dynamique de son environnement local.

(b) En haut à droite, la *fenêtre active* sous forme matricielle (15×15 cellules), où chaque valeur encode un degré de certitude d'occupation. Cette matrice fournit une vue instantanée de la proximité des obstacles autour du robot.

(c) En bas à droite, l'*histogramme polaire lissé* construit à partir des mesures locales. Chaque barre exprime la densité d'obstacles dans une direction angulaire. La *ligne rouge* indique la direction θ choisie par l'algorithme pour avancer en toute sécurité, en suivant l'axe d'une vallée libre détectée.

(d) En bas à gauche, présente la même information sous une forme graphique classique, liant la densité d'obstacles à l'angle. La *ligne verte* correspond au *seuil de décision* η_1, séparant les directions sûres (sous le seuil) des directions à éviter. Le *point rouge* localise l'angle sélectionné pour le déplacement du robot.

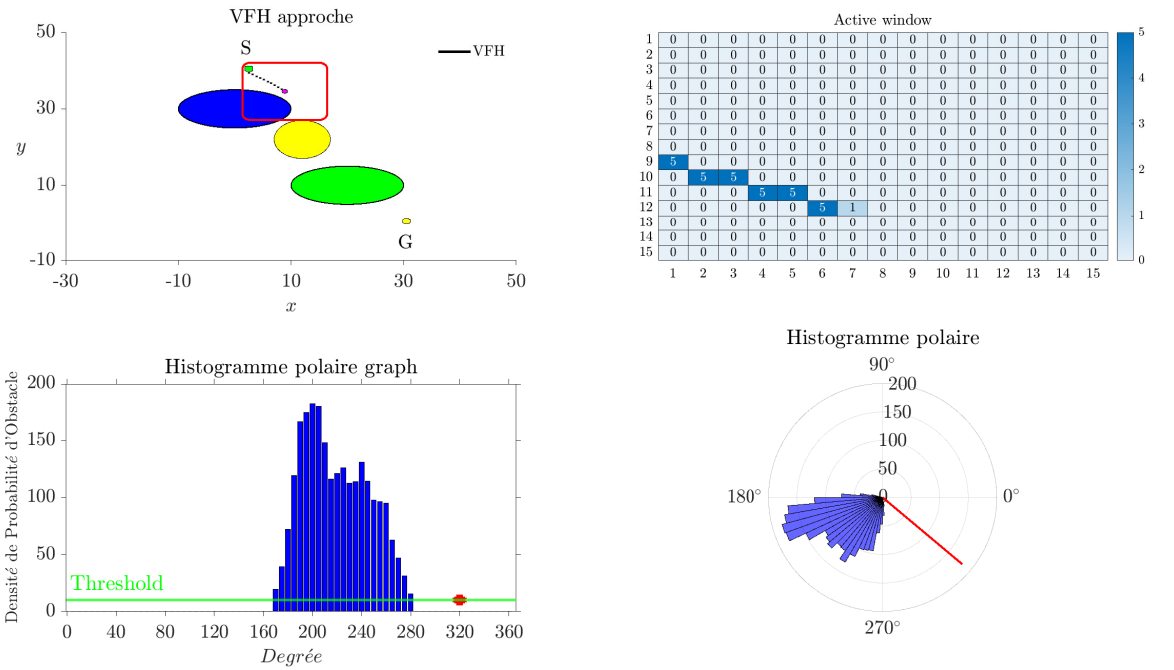

FIGURE 2.18 – Illustration du processus de navigation locale par le VFH.

La figure 2.19 détaille les différentes matrices extraites à partir de la fenêtre active pour une position instantanée donnée du robot.

1. En haut à gauche, la matrice **Active window** indique la carte locale d'occupation, avec des valeurs de 0 (cellule libre) à 5 (cellule fortement occupée).
2. En haut à droite, la matrice **Angle** fournit pour chaque cellule son angle polaire $\beta_{(i,j)}$, utilisé pour projeter les obstacles dans l'histogramme polaire.
3. En bas à gauche, la matrice **Distance** présente la distance euclidienne entre chaque cellule active et le centre du robot.
4. En bas à droite, la matrice **K-sector** attribue à chaque cellule un secteur angulaire discret, permettant de regrouper les obstacles selon leur direction.

Ces résultats illustrent de manière cohérente l'enchaînement du traitement effectué par l'algorithme VFH, depuis l'acquisition des mesures locales jusqu'à la sélection réactive de la direction de déplacement optimale.

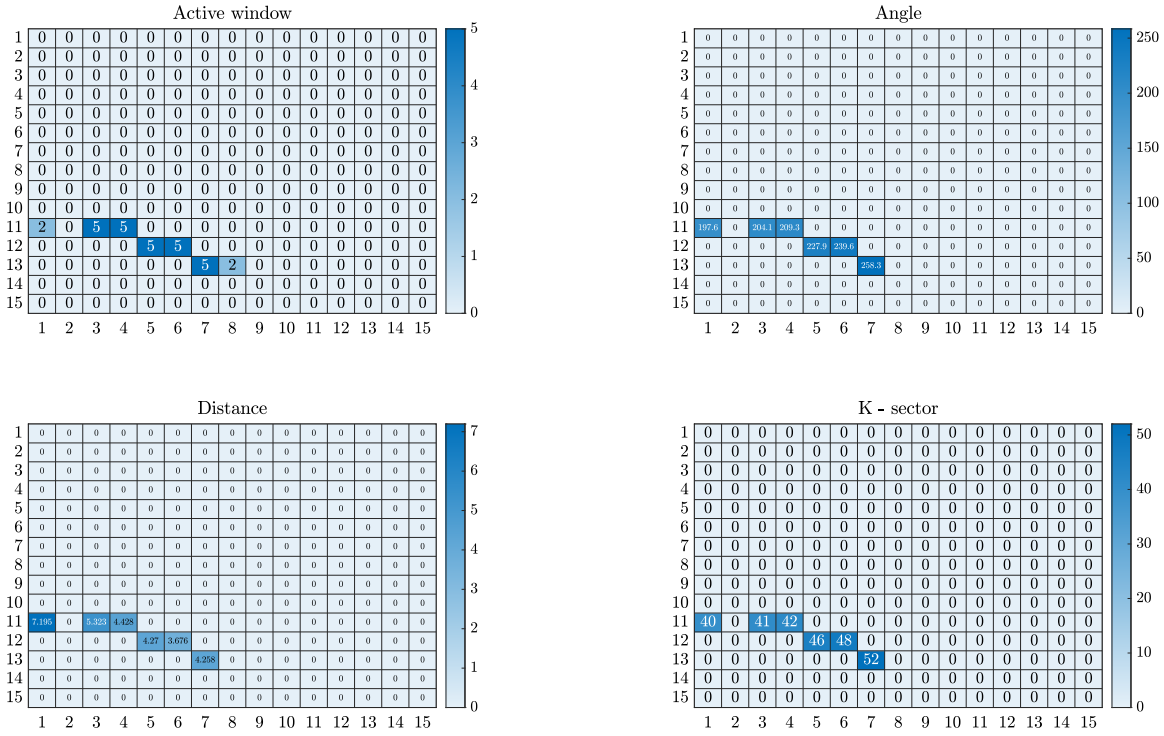

FIGURE 2.19 – Matrices issues de la fenêtre active : (a) carte locale d'occupation, (b) angles polaires, (c) distances euclidiennes aux obstacles, (d) secteurs angulaires associés.

Dans le but d'évaluer à nouveau les performances de VFH, une deuxième expérience a été réalisée en capturant l'état du robot à un instant donné, comme illustré sur la figure 2.20, puis en présentant la trajectoire complète de cette expérience dans la figure 2.21, accompagnée d'une animation.

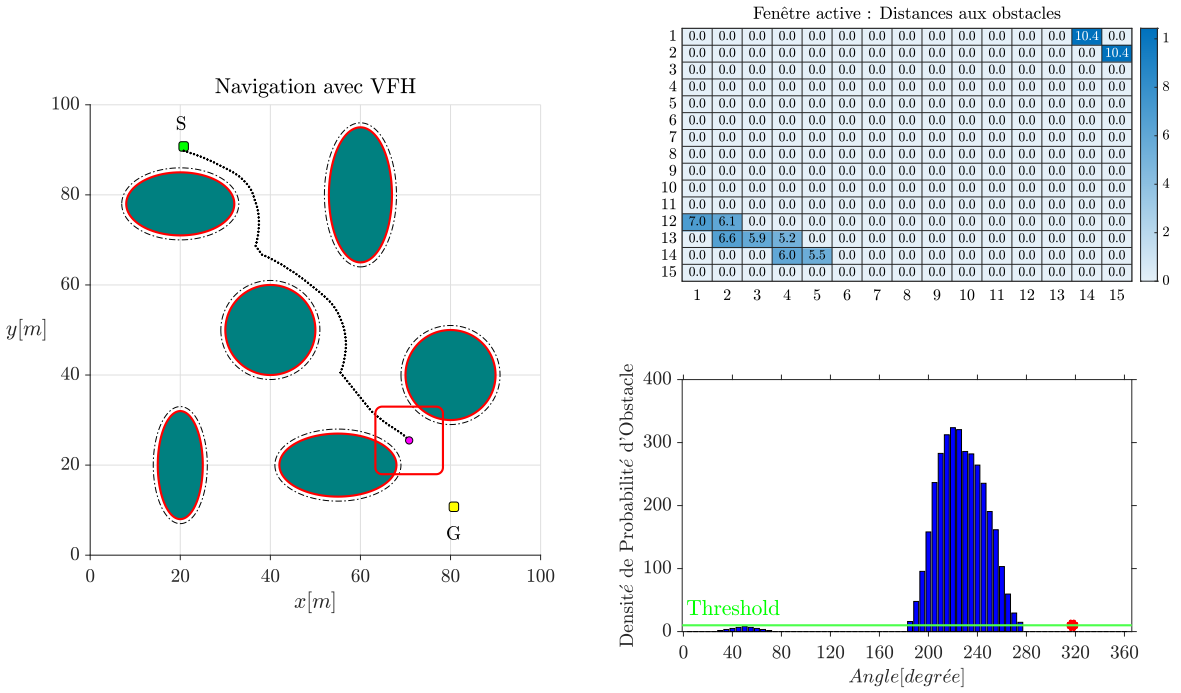

FIGURE 2.20 – Navigation avec l'algorithme VFH : trajectoire, fenêtre active et histogramme des densités d'obstacles. [3]

La figure 2.20 est composée de plusieurs parties :

- **Partie gauche** : navigation autonome avec VFH. Le robot est représenté par un cercle mauve, avec sa trajectoire en noir générée localement en naviguant du point de départ S vers le point d'arrivée G, tout en évitant les obstacles. La *fenêtre active*, représentée par un carré rouge, se déplace avec le robot et permet d'effectuer une analyse locale de l'environnement.
- **Partie supérieure droite** : visualisation des distances aux obstacles détectés dans la fenêtre active, représentées sous forme d'une matrice 15×15. Les cellules en bleu foncé signalent la présence d'obstacles proches, tandis que les cellules blanches traduisent des directions libres sans détection d'obstacle.
- **Partie inférieure droite** : représentation de l'histogramme polaire lissé h_k^* des densités d'obstacles sous forme de barres bleues. Le *seuil* η_1, indiqué par une ligne verte, permet de filtrer les directions présentant une faible densité d'obstacles.

3. Démo de VFH avec la fenêtre active et l'histogramme : https://youtu.be/ubqiEhVV-lA

Le *pic central*, situé dans l'intervalle $[190°; 250°]$, indique une forte densité d'obstacles, tandis qu'une autre zone de densité plus faible est observée dans l'intervalle $[38°; 70°]$. Enfin, le *marqueur rouge* identifie la direction sélectionnée par l'algorithme pour progresser vers la cible, correspondant à une vallée libre où la densité est minimale.

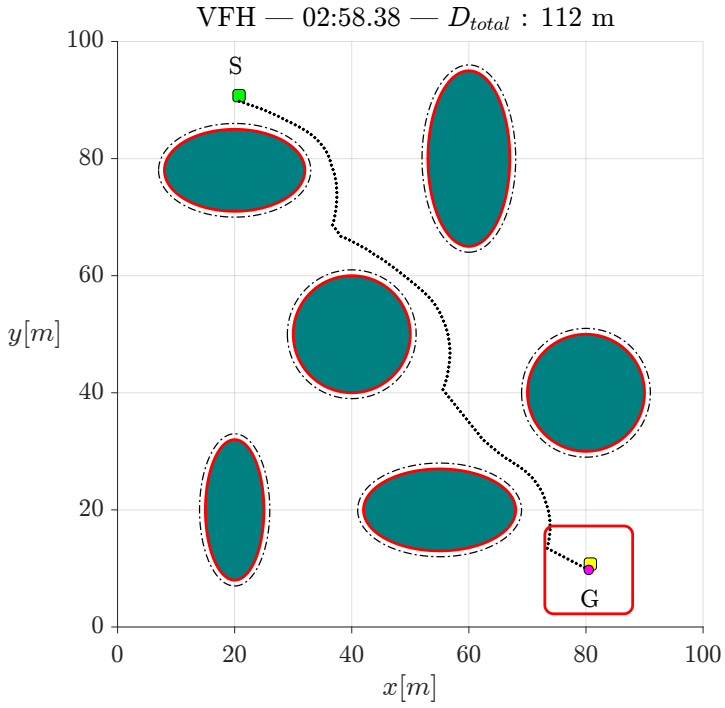

FIGURE 2.21 – Trajectoire finale de la navigation locale de l'algorithme VFH. [4]

Sur la figure 2.21, la trajectoire produite par l'algorithme VFH présente un chemin globalement fluide et régulier à travers un environnement encombré. Le robot ajuste progressivement sa direction face aux obstacles, bien que certains passages montrent des changements brusques sans pour autant provoquer d'arrêts soudains.

Dans cette section, une explication détaillée de l'algorithme de l'histogramme de champ vectoriel (VFH) a été présentée, en exposant sa base théorique ainsi que sa mise en œuvre pratique dans le contexte de la navigation locale. Le processus de génération de l'histogramme polaire, le mécanisme de sélection directionnelle et le contrôle de la vitesse ont été décrits en détail.

4. Démo de VFH : https://youtu.be/SC4eIBaNrKw

L'implémentation effectuée sous MATLAB en utilisant un capteur LiDAR simulé a offert une occasion d'examiner expérimentalement l'efficacité de l'algorithme dans un environnement inconnu. Les résultats obtenus par simulation confirment que VFH génère des trajectoires globalement homogènes et appropriées, garantissant un équilibre satisfaisant entre réactivité locale, sécurité et convergence vers la cible.

Cependant, certains ajustements abrupts ont été observés dans des environnements particulièrement contraints, soulignant la sensibilité de VFH aux variations locales de la densité d'obstacles. Ces résultats serviront de base pour envisager, dans la suite du mémoire, des améliorations permettant d'étendre l'approche à des scénarios 2D plus complexes et à des contraintes plus sévères dans le cadre de Gazebo. |

2.4 Algorithme Tangent Bug

L'algorithme T-bug, introduit en 1998 par Kamon et al. [23], constitue une méthode de navigation locale réactive pour les robots mobiles évoluant dans des environnements inconnus. Contrairement aux approches Bug classiques (Bug1, Bug2 [34]), il combine perception locale et raisonnement heuristique afin de guider le robot vers sa cible en l'absence de carte globale.

Reposant sur des données issues d'un capteur de type LiDAR omnidirectionnel, T-Bug permet au robot de réagir dynamiquement à la configuration des obstacles. L'algorithme s'appuie sur deux comportements principaux : un déplacement direct vers la cible lorsque celle-ci est visible, et un contournement local des obstacles dès qu'une obstruction est détectée.

Cette section présente les fondements de cette stratégie de navigation. Après avoir introduit le modèle cinématique du robot, nous décrivons la modélisation de la perception angulaire, les mécanismes de bascule entre modes de déplacement, ainsi que la logique de sélection des sous-objectifs intermédiaires. L'ensemble est illustré à l'aide de scénarios simulés représentatifs.

Modèle de déplacement

Le robot est modélisé par un point mobile dans $\mathbb{R}^2$, se déplaçant à vitesse constante v_{robot} sans tenir compte des contraintes dynamiques. Sa trajectoire est régie par :

$$\frac{dx(t)}{dt} = u(t), \tag{2.27}$$

et simulée par discrétisation explicite :

$$x_{t+\Delta t} = x_t + u_t \cdot \Delta t, \tag{2.28}$$

où $u(t)$ est orienté vers la cible q_{goal}. La direction est définie par :

$$\hat{d}_{\text{goal}} = \frac{q_{\text{goal}} - x}{\|q_{\text{goal}} - x\|}, \tag{2.29}$$

et la vitesse par :

$$u = v_{\text{robot}} \cdot \hat{d}_{\text{goal}}. \tag{2.30}$$

Ce modèle permet une mise en œuvre simple et efficace du comportement réactif dans T-Bug.

Perception et modélisation LiDAR

L'algorithme T-bug repose sur un capteur LiDAR omnidirectionnel à portée limitée, permettant au robot de balayer l'environnement à partir de sa position actuelle $x \in \mathbb{R}^2$. Pour chaque direction angulaire $\theta \in [0, 2\pi]$, le capteur mesure la distance à l'obstacle le plus proche.

En l'absence de contrainte de portée, la distance dans la direction θ est donnée par [12] :

$$\rho(x,\theta) = \min_{\lambda \in [0,\infty]} \left\{ d\left(x, x + \lambda \begin{bmatrix} \cos\theta & \sin\theta \end{bmatrix}^\top\right) \,\middle|\, x + \lambda \begin{bmatrix} \cos\theta & \sin\theta \end{bmatrix}^\top \in \bigcup_i \mathcal{W}O_i \right\}, \tag{2.31}$$

où $\mathcal{W}O_i$ représente l'ensemble des obstacles dans l'espace de travail.

Dans la réalité, cette fonction est saturée à une portée maximale R, définissant la fonction de distance limitée :

$$\rho_R(x,\theta) = \begin{cases} \rho(x,\theta), & \text{si } \rho(x,\theta) \leq R, \\ \infty, & \text{sinon.} \end{cases} \tag{2.32}$$

Chaque mesure $\rho_R(x,\theta)$ indique la distance au premier obstacle détecté dans la direction θ, ou l'infini s'il n'y en a pas. L'ensemble des points détectés est alors :

$$\mathcal{P}(x) = \left\{ x + \rho_R(x,\theta) \begin{bmatrix} \cos\theta & \sin\theta \end{bmatrix}^\top, \quad \theta \in [0, 2\pi] \right\}. \tag{2.33}$$

La structure angulaire de $\rho_R(x,\theta)$ présente des discontinuités aux transitions entre zones visibles et non visibles. Ces ruptures permettent d'identifier les extrémités des obstacles O_i, qui constituent les cibles locales potentielles dans la stratégie de navigation.

On distingue deux types de discontinuités :

- celles dues à l'entrée ou sortie d'un obstacle dans le champ de perception ;
- celles liées à la limite de portée du capteur ($\rho(x,\theta) > R$).

Les discontinuités angulaires détectées par le LiDAR révèlent les extrémités visibles O_i, utilisées pour définir dynamiquement des objectifs intermédiaires ; les segments continus entre ces ruptures correspondent aux portions observables des obstacles, guidant localement la navigation du robot (voir la figure 2.22).

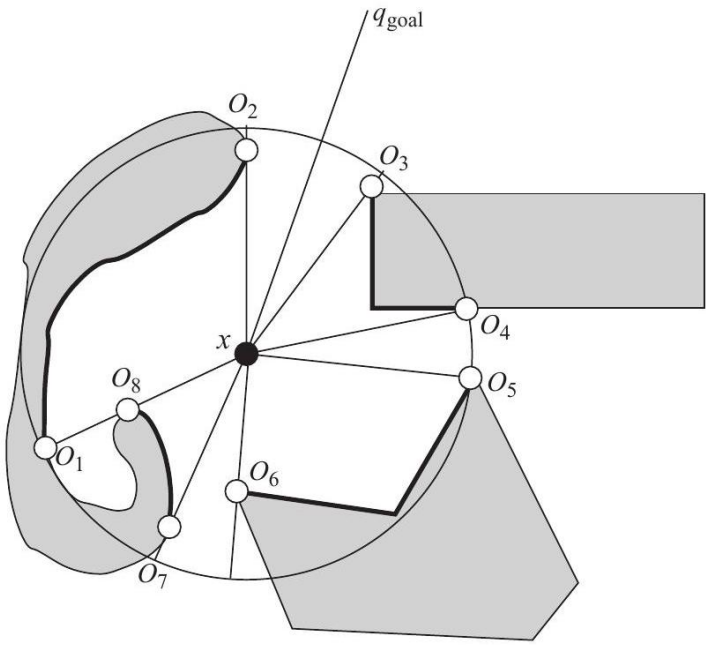

FIGURE 2.22 – Détection des extrémités O_i par le LiDAR ; les segments épais indiquent les zones visibles depuis la position x du robot.

2.4.1 Détection et sélection heuristique des objectifs

L'un des fondements de l'algorithme *Tangent Bug* réside dans la détection locale des extrémités d'obstacles, rendues visibles par les discontinuités angulaires de la fonction de distance saturée $\rho_R(x, \theta)$, calculée à partir des mesures du capteur LiDAR. Ces discontinuités marquent les transitions entre les zones perçues (mesures finies) et non perçues (valeurs infinies), et permettent d'identifier les points tangents O_i sur les frontières d'obstacles. À chaque itération, l'algorithme évalue ces points en minimisant une fonction heuristique exprimée par :

$$h(x, O_i) = d(x, O_i) + d(O_i, q_{\text{goal}}), \tag{2.34}$$

À chaque cycle, le robot sélectionne comme cible temporaire le point O_i^* qui minimise l'heuristique (2.34) :

$$O_i^* = \arg\min_{O_i} h(x, O_i), \tag{2.35}$$

Comme illustré à la figure 2.22, le robot extrait plusieurs extrémités visibles $O_1, O_2, O_3, \ldots$, sélectionne le point optimal O_i^* qui minimise l'heuristique (2.34), et l'utilise pour guider

temporairement sa progression jusqu'à réévaluation ou reprise du déplacement direct vers la cible q_{goal}.

2.4.2 Gestion des cas d'occlusion partielle

Cette approche repose sur une hypothèse fondamentale : en l'absence de carte globale, le robot peut estimer localement des options de navigation prometteuses, tant que l'environnement reste observable dans son champ de portée. Toutefois, un point O_i peut sembler optimal selon l'heuristique (2.34), mais mener à une impasse si la trajectoire $[O_i, q_{\text{goal}}]$ est en réalité obstruée par un obstacle non détecté depuis la position x. Ce cas est désigné sous le terme d'*occlusion partielle*.

La figure 2.23 illustre ce cas : le point O_2, initialement sélectionné, mène à une impasse. Une réévaluation permet alors d'identifier O_4 comme une alternative plus sûre.

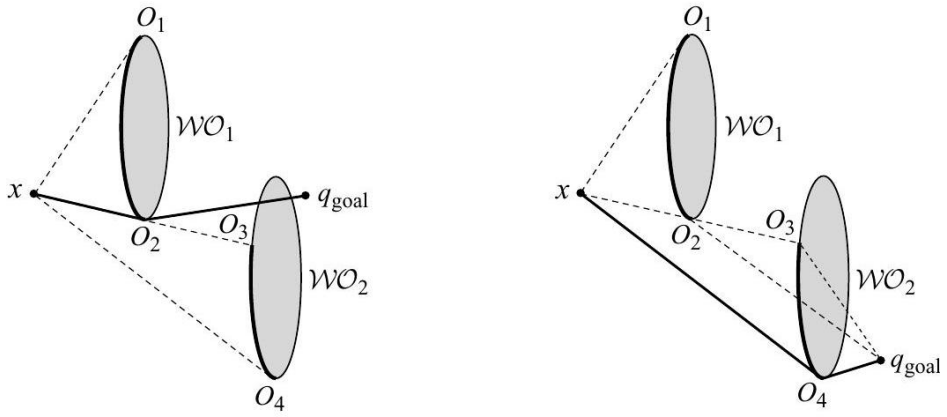

FIGURE 2.23 – À gauche : sélection initiale de O_2, bloquée par un obstacle non visible. À droite : réévaluation vers O_4, offrant un accès dégagé à la cible.

Dans ce cas, l'algorithme assigne un coût $h(x, O_i) = \infty$ aux trajectoires obstruées et réévalue les alternatives. Ce mécanisme local de correction adaptative renforce la robustesse de T-Bug face à l'incertitude perceptuelle, sans recourir à une carte globale.

2.4.3 Logique de navigation

L'algorithme T-Bug alterne entre deux modes de navigation en fonction de la visibilité de la cible depuis la position actuelle du robot. Cette stratégie permet une progression autonome, même dans un environnement partiellement observable ou inconnu.

Déplacement vers la cible (Motion-to-Goal)

Lorsque la cible q_{goal} est directement visible, le robot adopte le mode *Motion-to-Goal* (MTG), c'est-à-dire un déplacement direct vers celle-ci. La direction est donnée par le vecteur unitaire $\hat{d}_{\text{goal}}$, et la cinématique suit les équations (2.30) et (2.28).

Ce comportement est maintenu tant que, pour tout angle $\theta \in [0, 2\pi]$, la distance mesurée $d_{\text{obs}}(\theta)$ reste strictement supérieure à la distance directe vers la cible :

$$d_{\text{obs}}(\theta) > d_{\text{goal}}, \quad \forall\theta, \tag{2.36}$$

avec $d_{\text{goal}} = \|q_{\text{goal}} - x\|$. Cela indique que la cible est atteignable sans obstacle dans le champ de perception du capteur.

En revanche, si un obstacle bloque la ligne de visée, c'est-à-dire lorsque la distance mesurée dans la direction de la cible devient inférieure à celle-ci :

$$d(\theta_{\text{goal}}) < d_{\text{goal}}, \tag{2.37}$$

le robot bascule en mode *Suivi de bordure (BF)*, afin de contourner l'obstruction.

Suivi de bordure (Boundary Following)

Lorsque la cible n'est plus visible, le robot active le mode *Suivi de bordure* (BF). Il identifie le point de contact avec l'obstacle par lancer de rayon :

$$p_{\text{hit}} \in \text{raycast}(x, \hat{d}_{\text{goal}}), \tag{2.38}$$

et le robot commence à suivre la bordure de l'obstacle à partir de ce point tout en évaluant à chaque itération deux quantités essentielles : la distance d_{followed}, entre le point de contact p_{hit} et la cible, et d_{reach}, distance minimale atteignable depuis les points visibles sur la bordure. Le retour au mode MTG s'effectue dès que :

$$d_{\text{reach}} < d_{\text{followed}}. \tag{2.39}$$

Dès qu'un point plus favorable est détecté, le robot quitte le mode de suivi de bordure pour reprendre un déplacement direct. L'alternance fluide entre ces deux comportements

permet une navigation réactive et efficace, conduisant le robot vers sa cible sans nécessiter de carte préalable.

Critère de bascule entre les modes de navigation

Lorsque le robot suit la bordure d'un obstacle, il continue d'explorer les extrémités détectées à chaque itération. Il évalue en temps réel si l'une de ces nouvelles configurations permet un retour efficace vers l'objectif. Cette décision repose sur une comparaison entre la distance actuellement atteignable d_{followed} depuis une extrémité visible, et la distance enregistrée au moment où le contournement a commencé d_{reach}. On définit alors la condition de sortie du mode de suivi comme suit :

$$d_{\text{reach}} < d_{\text{followed}}, \tag{2.40}$$

indiquant que le robot peut désormais s'engager sur un chemin plus court vers la cible que celui qu'il avait initialement au moment de l'entrée en contournement.

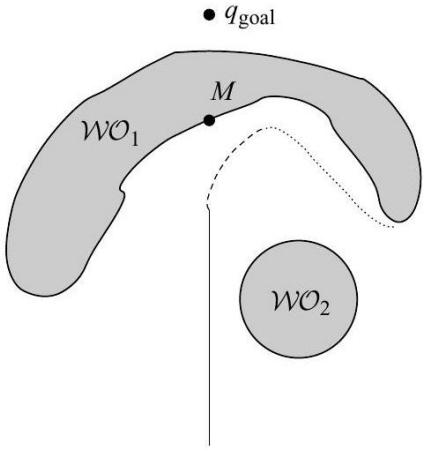

FIGURE 2.24 – Transition de navigation : passage du suivi de bordure au retour vers la cible, lorsque $d_{\text{reach}} < d_{\text{followed}}$.

Pour calculer d_{reach}, on considère l'ensemble des points visibles Λ sur la frontière courante $\partial\mathcal{WO}_f$, définis par :

$$\Lambda = \{y \in \partial\mathcal{WO}_f \,|\, \forall\lambda \in [0,1],\ \lambda x + (1-\lambda)y \in \mathcal{Q}_{\text{free}}\}\,, \tag{2.41}$$

où x est la position courante du robot. La distance atteignable minimale est alors donnée par :

$$d_{\text{reach}} = \min_{c \in \Lambda} d(q_{\text{goal}}, c). \tag{2.42}$$

Lorsque l'inégalité (2.40) est satisfaite, le robot désactive le mode *boundary-following* et relance le comportement de navigation directe vers la cible. Ce mécanisme empêche le robot de rester piégé dans des cycles de contournement et garantit un progrès continu vers q_{goal}.

2.4.4 Effet de la portée du capteur sur la stratégie de bascule

La portée effective du capteur R joue un rôle déterminant dans l'efficacité du critère de sortie du mode de suivi de bordure. Trois configurations typiques peuvent être distinguées.

Dans le cas d'une *portée nulle* (figure 2.25), le robot ne perçoit que le point de contact initial avec l'obstacle. Il prend ses décisions sur la base d'une information locale extrêmement restreinte, reproduisant ainsi un comportement proche de celui de Bug1.

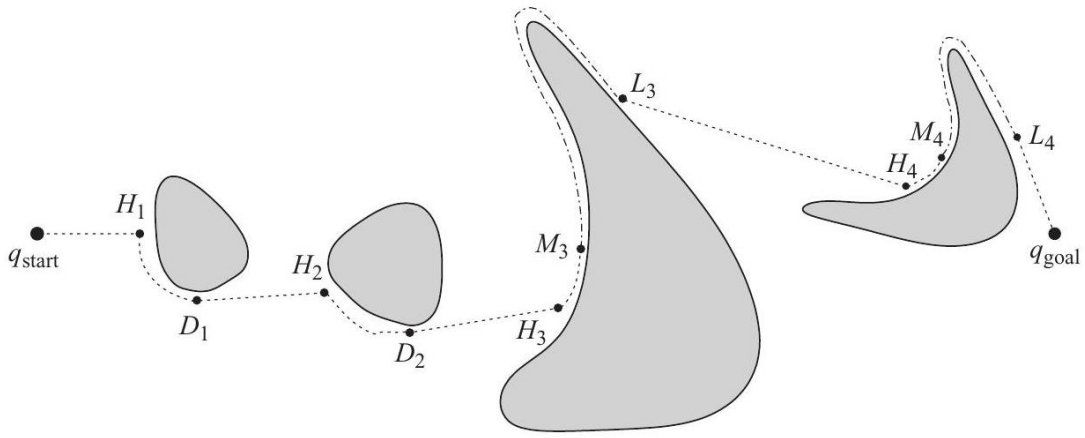

FIGURE 2.25 – Navigation avec portée nulle : succession rigide entre MTG et BF, sans vision anticipée.

Avec une *portée finie* (figure 2.26), le robot accède à une portion de la bordure de l'obstacle. Cette vision partielle lui permet de réévaluer régulièrement la distance atteignable d_{reach} et d'adapter sa stratégie de sortie.

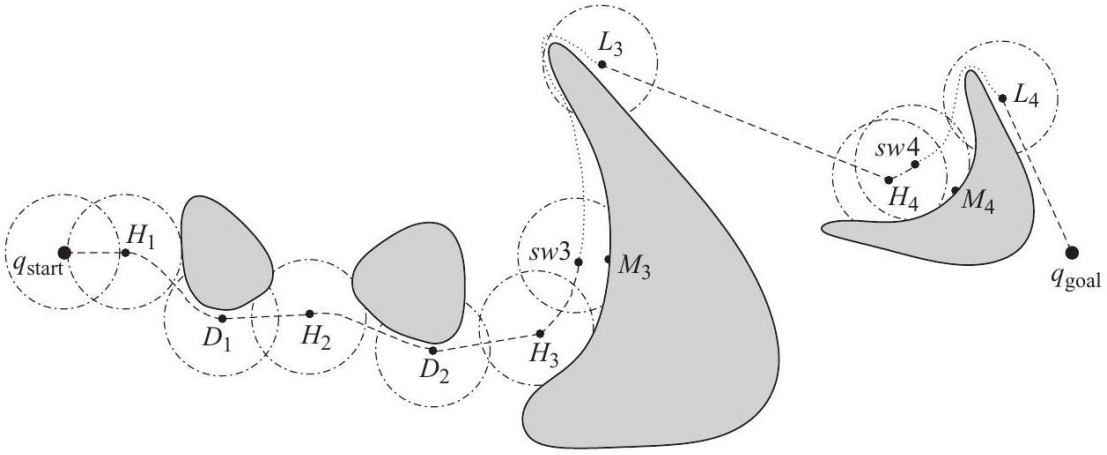

FIGURE 2.26 – Navigation avec portée finie : le robot alterne intelligemment entre MTG et BF, selon la visibilité des extrémités.

Enfin, dans le cas idéal d'une *portée infinie* (figure 2.27), l'ensemble des extrémités visibles O_i est accessible à tout instant. Le robot peut ainsi planifier un trajet direct sans recourir au contournement, sauf si la cible est véritablement inaccessible.

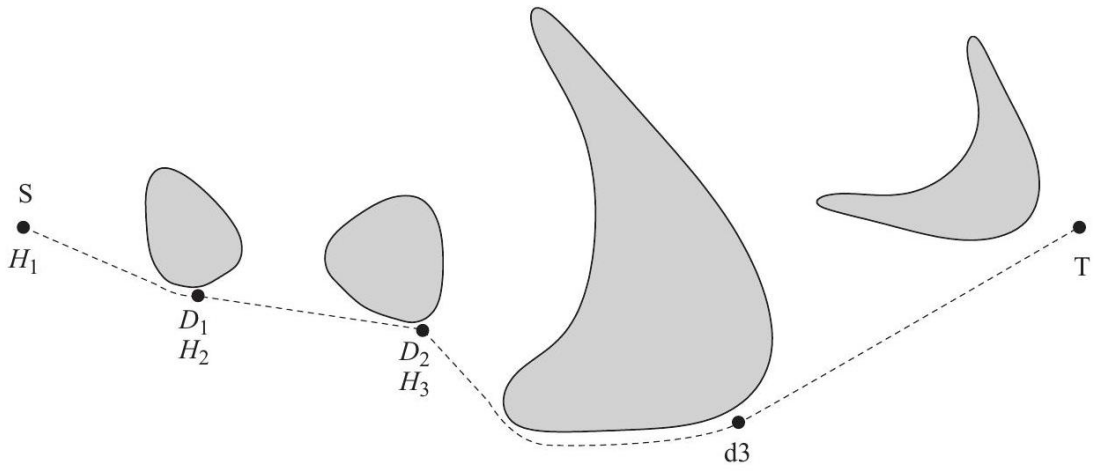

FIGURE 2.27 – Navigation avec portée infinie : toujours en mode MTG, car tous les obstacles sont visibles à l'avance.

Ce raisonnement confirme que plus la portée du capteur est grande, moins l'algorithme dépend du comportement de contournement. Une faible portée, en revanche, oblige le robot à composer avec une incertitude spatiale plus grande, mais rend son comportement plus localement réactif.

Organisation de l'algorithme T-bug

Le fonctionnement de l'algorithme T-bug repose sur une boucle principale de perception–décision–action, à travers laquelle le robot adapte en temps réel sa stratégie de déplacement en fonction des informations fournies par son capteur de portée. La figure 2.28 présente l'organigramme de cette logique de navigation.

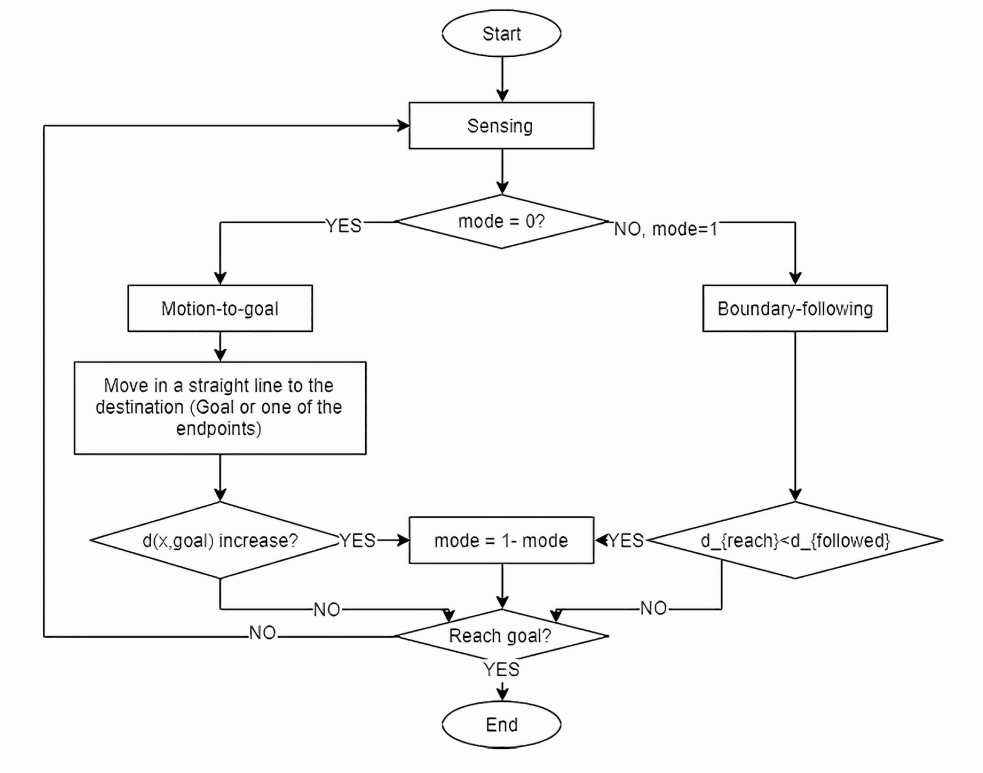

FIGURE 2.28 – Organigramme général de l'algorithme T-bug. Il illustre l'alternance entre les modes *move-to-goal* et *boundary-following*, selon la visibilité de la cible.

Résumé de la boucle de navigation

La boucle globale peut se formaliser ainsi :

$$\texttt{TangentBug:} \begin{cases} \text{Si cible visible} & \rightarrow \text{MTG}, \\ \text{Sinon} & \rightarrow \text{BF jusqu'à } d_{\text{reach}} < d_{\text{followed}}, \\ \text{Reprise MTG} & \rightarrow \text{Répéter}, \\ \text{Jusqu'à convergence.} & \end{cases}$$

Ce cycle réactif local est exécuté jusqu'à l'atteinte de la cible.

2.4.5 Implémentation de l'algorithme T-bug sous MATLAB

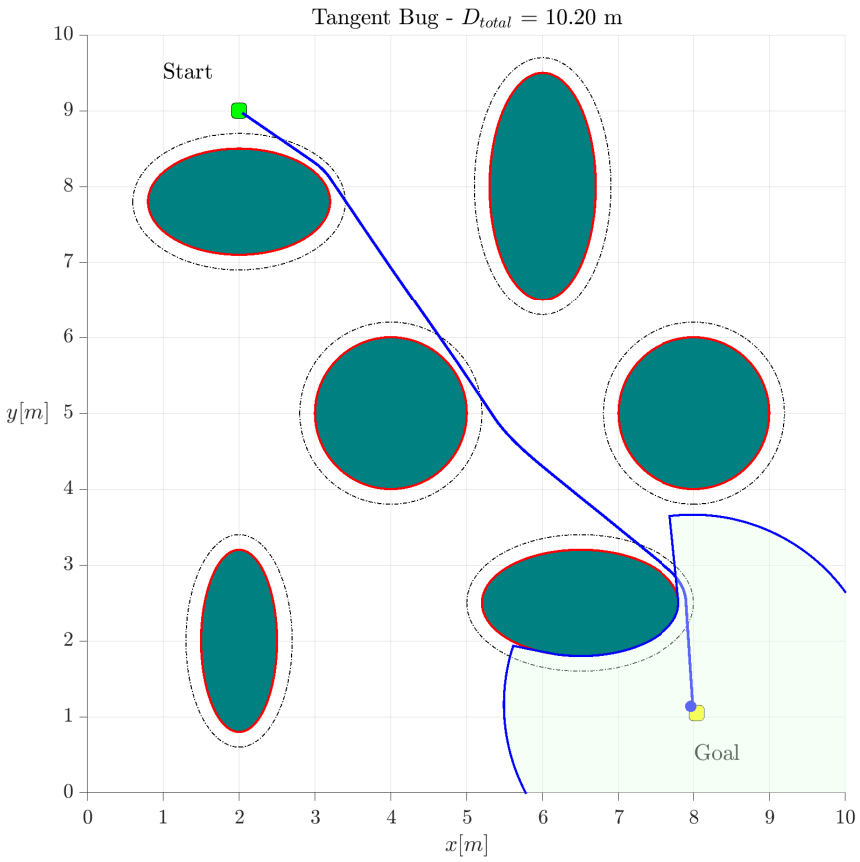

FIGURE 2.29 – Résultats de simulation de l'algorithme Tangent Bug. [5]

Interprétation des résultats

- **Capteur utilisé :** Le robot est équipé d'un capteur LiDAR 2D, qui lui permet de détecter les obstacles dans son environnement. Ce capteur fournit des données précises sur la position relative des obstacles par rapport au robot, ce qui est essentiel pour une navigation efficace.
- **Trajectoire suivie :** La trajectoire, représentée en bleu, montre le chemin emprunté par le robot pour atteindre l'objectif. Le robot part du point de départ

5. Démo de Tbug :https://youtu.be/bnlnaWw1uW4

(carré vert, "Start") et atteint l'objectif (cercle jaune, "Goal") en évitant les obstacles. L'algorithme Tangent Bug alterne entre deux modes de navigation : un mode de progression directe vers la cible lorsque le chemin est dégagé, et un mode de contournement des obstacles lorsque ceux-ci bloquent la trajectoire directe.

- **Contours détectés :** Les lignes rouges illustrent les contours des obstacles détectés par le capteur LiDAR. Ces contours permettent au robot de maintenir une distance de sécurité tout en naviguant autour des obstacles, assurant ainsi une navigation sans collision.
- **Distance parcourue :** La distance totale parcourue par le robot est de $D_{\text{total}} = 10.20\,\text{m}$. Cette valeur reflète l'efficacité de l'algorithme à minimiser la longueur du trajet tout en évitant les obstacles dans un environnement complexe.
- **Zones détectées :** La zone en pointillés noirs représente le périmètre global des obstacles détectés par le capteur. Cette information permet au robot de maintenir une cartographie locale et dynamique de son environnement, facilitant ainsi la prise de décision en temps réel.
- **Comportement global :** L'algorithme Tangent Bug démontre une capacité de navigation sûre et adaptative. En combinant une progression directe vers la cible et un suivi précis des contours des obstacles, le robot parvient à atteindre son objectif de manière efficace et fiable.

En somme, l'algorithme T-Bug constitue une approche fiable et réactive pour la navigation de robots mobiles en milieu inconnu. En combinant de manière adaptative la progression vers la cible et le contournement local des obstacles, il permet au robot d'avancer de façon autonome sans dépendre d'une connaissance préalable de l'environnement.

2.5 Comparaison quantitative des performances des méthodes SVC, VFH et T-Bug

Cette section présente une évaluation comparative des trois algorithmes, basée sur des critères quantitatifs définis en 2.5.1. L'analyse s'appuie à la fois sur les trajectoires générées (figure 2.30), l'évolution des distances de dégagement (figure 2.31), leur distribution statistique (figure 2.32), ainsi que sur les résultats synthétisés avec les paramètres utilisés dans ces expériences dans les tableaux 2.1 et 2.2.

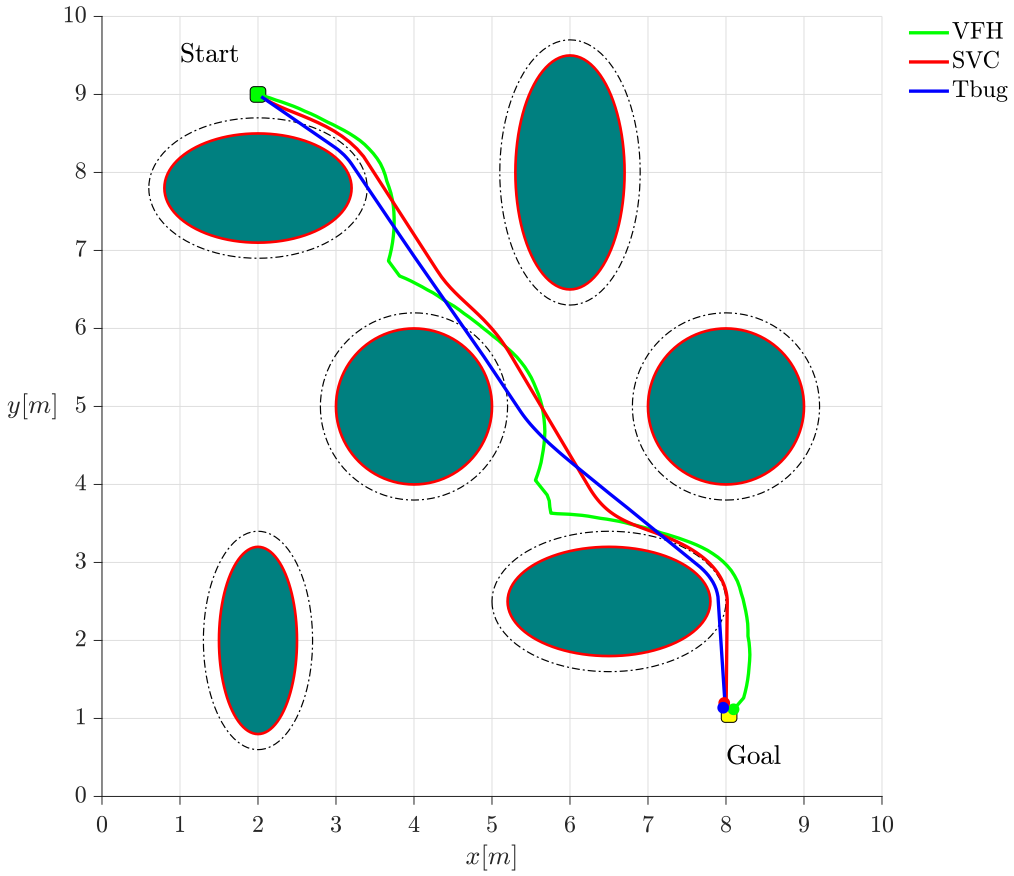

FIGURE 2.30 – Résultats de simulation des algorithmes VFH, SVC, et Tbug et leurs trajectoires pour atteindre l'objectif. [6]

6. Comparaison de SVC, VFH et Tbug : https://youtu.be/TaCTJ_5VmI8

2.5.1 Critères d'évaluation des algorithmes de navigation locale

Pour évaluer de l'efficacité des algorithmes **VFH**, **SVC** et **T-Bug** dans un environnement simulé, divers critères quantitatifs ont été élaborés sur la base des données expérimentales. Des essais ont été réalisés en utilisant un **LiDAR 2D** simulé, qui propose une couverture circulaire de 360° et une distance maximale de détection de $R = 2{,}5$ m.

Distance totale effectuée

Ce critère illustre l'efficacité géométrique de l'algorithme. Cela correspond à l'addition des distances entre les emplacements successifs du robot. La formule pour le calcul de la distance total est la suivante :

$$D_{\text{total}} = \sum_{i=2}^{n} \|\mathbf{p}_i - \mathbf{p}_{i-1}\|_2 \tag{2.43}$$

Distance de dégagement (Clearance)

La distance de dégagement indique la distance minimale entre le robot et l'obstacle le plus proche, calculée à partir des données obtenues par le LiDAR. Elle est fournie par :

$$C_i = \min_j \|\mathbf{p}_i - \mathbf{o}_j\|_2 \tag{2.44}$$

où $\mathbf{o}_j$ représente la localisation d'un obstacle identifié lors de l'itération i. La valeur moyenne de cette distance est la suivante :

$$\bar{C} = \frac{1}{n} \sum_{i=1}^{n} C_i \tag{2.45}$$

Distance de dégagement minimale et maximale

La distance de dégagement minimale et la distance de dégagement maximale sont définies comme suit :

$$C_{\min} = \min_i (C_i) = \min_i \left(\min_j \|\mathbf{p}_i - \mathbf{o}_j\|_2 \right) \tag{2.46}$$

$$C_{\max} = \max_i (C_i) = \max_i \left(\min_j \|\mathbf{p}_i - \mathbf{o}_j\|_2 \right) \tag{2.47}$$

où C_i désigne la distance de dégagement à l'itération i, $\mathbf{p}_i$ la position du robot, et $\mathbf{o}_j$ la position de l'obstacle j.

Temps moyen par itération

Ce critère reflète la charge de calcul associée à chaque algorithme. Il est défini comme la moyenne des temps de traitement par itération :

$$T_{\text{moy}} = \frac{1}{n} \sum_{i=1}^{n} t_i \tag{2.48}$$

où t_i représente le temps d'exécution de l'algorithme à l'itération i. Ce facteur est particulièrement important dans le cadre d'une application embarquée en temps réel.

Paramètres des algorithmes

Le tableau 2.1 collecte les paramètres appliqués pour les algorithmes **VFH** et **SVC** lors des tests effectués. À l'inverse de ces autres approches, T-Bug ne se base sur aucun critère heuristique. Il utilise plutôt une stratégie géométrique qui repose sur la distance à l'objectif, avec un champ de détection local commun R partagé par les trois méthodes.

Paramètre	**Valeur**	**Description**
Paramètres communs à tous les algorithmes		
$\boldsymbol{R}$ (LiDAR 2D)	2.5 m	Portée maximale de détection.
Paramètres de VFH		
$\boldsymbol{ws}$	15	Taille de la zone analysée.
Seuil direction libre	< 10	Critère d'acceptation directionnelle.
$\boldsymbol{a}$	10.6	Pondération de proximité.
$\boldsymbol{b}$	1	Facteur lié à la distance.

Suite à la page suivante...

Paramètre (suite)	**Valeur**	**Description**
$\boldsymbol{d_{\max}}$	≈ 9.9	Distance max dans la fenêtre.
Paramètres de SVC		
$\boldsymbol{\varepsilon}$	0.2	Marge de sécurité minimale.
$\boldsymbol{\varepsilon'}$	0.4	Seuil de lissage de la commande.
$\boldsymbol{k}$	1	Gain du contrôle nominal.

TABLE 2.1 – Paramètres utilisés dans les algorithmes VFH et SVC

Résultats quantitatifs

Le tableau 2.2 présente une comparaison des performances en se basant sur divers critères pour les trois algorithmes. Ces résultats proviennent de multiples simulations basées sur des trajectoires réelles simulées en MATLAB.

Critère	**VFH**	**SVC**	**Tbug**
Distance totale parcourue D_{total} [m]	11.60	10.32	10.20
Temps moyen par itération T_{moy} [s]	0.053	**0.025**	0.051
Mesures liées à la distance de dégagement (clearance)			
Moyenne de clearance $\bar{C}$ [m]	0.52	**0.56**	0.41
Minimum de clearance $C_{\min}$ [m]	0.27	**0.30**	0.10
Maximum de clearance $C_{\max}$ [m]	1.10	1.05	1.12

TABLE 2.2 – Comparaison des algorithmes selon plusieurs critères de performance. Toutes les mesures sont issues de simulations sur trajectoires réelles avec LiDAR 2D.

La figure 2.31 montre l'évolution temporelle de la distance de dégagement, tandis que sa distribution statistique est illustrée dans la figure 2.32. Ces représentations servent à comparer le comportement des algorithmes en termes de sécurité face aux défis.

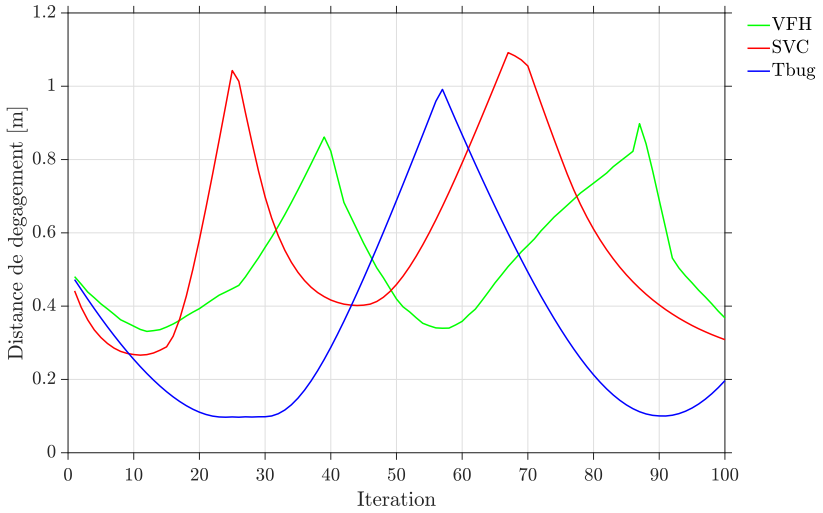

FIGURE 2.31 – Évolution de la distance de dégagement au cours du trajet

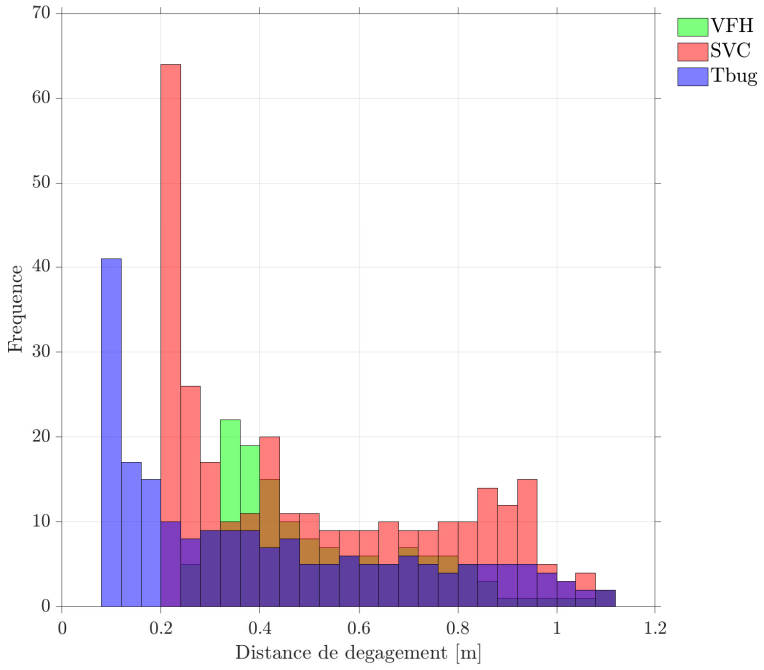

FIGURE 2.32 – Distribution statistique des distances de dégagement

Résumé comparatif des méthodes de navigation

Sur la base des critères définis en 2.5.1, des trajectoires observées (figure 2.30), de l'évolution des distances de dégagement (figure 2.31), de leur distribution statistique (figure 2.32), ainsi que des résultats numériques présentés dans le tableau 2.2, on peut conclure que :

1. **SVC :**
   - **Temps moyen par itération** très faible ($T_{\text{moy}} = 0.025$ s) — Très rapide, grâce à une formulation explicite, sans heuristiques ni traitement computationnel intensif.
   - Fonctionne directement en 2D/3D sans adaptation de structure.
   - Utilise uniquement la distance et la direction de l'obstacle le plus proche, limitant ainsi la charge mémoire et le traitement perceptif.
   - Assure une distance de dégagement sécurisée ($C_{\min} = 0.30$ m), avec une moyenne stable ($\bar{C} = 0.56$ m) contrôlée par ε'.
   - Compatible avec tout contrôleur nominal (PID, MPC, apprentissage par renforcement, etc).
2. **VFH :**
   - Rapide (se basant sur un histogramme angulaire fixe), avec un temps moyen par itération de $T_{\text{moy}} = 0.053$ s.
   - Limité à la navigation en 2D – pas d'élargissement naturel vers des milieux en 3D.
   - Sensible au bruit généré par les capteurs et aux ajustements de seuils (direction libre, pondérations).
   - Configuration complexe, capable de causer des oscillations ou des comportements instables.
   - Il n'y a aucune assurance théorique de convergence.
   - Conserve une distance de dégagement moyenne appropriée ($\bar{C} = 0.52$ m), avec une valeur minimale modeste ($C_{\min} = 0.27$ m) et un maximum de $C_{\max} = 1.10$ m, ce qui indique une conduite prudente.
   - Génère parfois des virages brusques, ce qui rend la trajectoire moins fluide.
3. **T-Bug :**

- Théoriquement complet : atteint toujours le but si celui-ci est accessible.
- Calcul plus exigeant ($T_{\text{moy}} = 0.051$ s) en raison des tests de visibilité directionnelle effectués à chaque itération.
- Difficile à généraliser à des environnements en 3D, car cela repose sur des opérations géométriques en 2D.
- Moins performant dans les environnements convexes ou très denses.
- Génère des trajectoires proches des obstacles, avec une distance de dégagement minimale critique ($C_{\min} = 0.10$ m) et une moyenne relativement faible ($\bar{C} = 0.41$ m), ce qui traduit un comportement potentiellement risqué.

2.6 Conclusion

Ce chapitre a mis en comparaison les algorithmes SVC, VFH et T-Bug selon des critères de performance quantitative via des simulations. Chaque approche offre des compromis entre efficacité, sûreté et complexité d'implémentation.

Le **SVC** se distingue par sa formulation géométrique précise, son efficacité d'exécution et sa puissance à conserver une distance de dégagement sécurisée. L'absence de réglages heuristiques et sa compatibilité particulière avec des systèmes en 3D en font une base robuste pour les applications embarquées, surtout lorsqu'il est couplé à un contrôleur PID.

VFH est rapide à l'exécution, cependant son implémentation se révèle complexe en raison du calcul intensif de l'histogramme polaire à chaque itération. Par ailleurs, sa grande dépendance à différents seuils le rend sensible au bruit du capteur et peut provoquer des oscillations ou des perturbations, sans aucune assurance théorique de convergence. Il génère parfois des virages brusques, ce qui rend la trajectoire moins fluide.

T-Bug assure théoriquement l'atteinte du but si accessible, cependant il génère fréquemment des itinéraires à proximité des obstacles, ce qui augmente les dangers en cas d'incertitudes. Son application à des contextes plus complexes reste limitée.

Ce chapitre a offert une première analyse comparative des algorithmes sous MATLAB, soulignant leurs compromis respectifs. Le prochain chapitre étend cette étude dans un cadre ROS/Gazebo plus réaliste, en réalisant une implémentation sur un robot différentiel simulé.

3

Implémentation d'un robot à entraînement différentiel sous Gazebo

3.1 Introduction

Ce chapitre présente l'implémentation d'un robot mobile à entraînement différentiel sous ROS/Gazebo, basé sur l'architecture du TurtleBot3. Il commence par la modélisation cinématique du robot, qui est ajustée à sa configuration différentielle et qui est en correspondance avec le simulateur. L'intégration de la configuration des capteurs (LiDAR, odométrie) est ensuite effectuée pour faciliter une communication authentique avec l'environnement simulé. Le chapitre continue avec la mise en place structurée de 4 algorithmes de navigation locale réactive (SVC, SVC-PID, VFH modifié et T-Bug), qui ont tous été élaborés en tant que nœud ROS autonome. Une partie est consacrée à l'explication du scénario expérimental, des paramètres de simulation et de la méthode expérimentale. La conclusion de l'étude comprend une évaluation comparative des performances réalisées, jugées selon trois critères essentiels : la réactivité, l'efficacité de la trajectoire et la sécurité.

3.2 ROS (Robot Operating System) et TurtleBot

Le *Robot Operating System* (ROS) est un outil de base polyvalent et open source qui jouit d'une large usage dans le domaine de la robotique mobile. Même s'il n'est pas considéré comme un système d'exploitation autonome, il occupe une place centrale dans le développement programmable des systèmes robotiques en coordonnant les divers composants matériels et logiciels. ROS propose un réseau de communication distribué

basé sur les concepts de *topics*, *services* et d'*actions*, facilitant la transmission asynchrone ou synchrone de données entre les différents nœuds du système [29].

ROS offre également une gamme d'outils pour simplifier le démarrage (`roslaunch`), la visualisation (`rviz`), l'enregistrement des données (`rosbag`) et le débogage des processus. En plus de ses fonctionnalités fondamentales, ROS comprend des ressources numériques dédiées à la gestion des capteurs, des actionneurs, de la navigation, de la planification de trajectoires et de la perception. En raison de son architecture modulaire et de sa communauté active, ROS s'est imposé comme une norme de référence dans le domaine de la recherche et de l'enseignement en robotique. Il sert également de fondement à des applications industrielles plus sophistiquées, notamment à travers ses déclinaisons telles que ROS2 et ROS-Industrial [36].

3.2.1 Composition de ROS

Le Robot Operating System (ROS) représente une installation logicielle modulaire élaborée dans le but de simplifier la conception de systèmes robotiques avancés. Il repose sur quatre éléments essentiels qui interagissent afin de fournir un cadre cohérent, flexible et extensible [41, 36] :

- La couche d'**infrastructure de communication**, également connue sous le nom de "plomberie", est responsable de la gestion des échanges entre les divers modules logiciels (noeuds) en utilisant des outils tels que les topics, les services et les actions. Elle facilite la communication entre les divers composants du système de manière répartie, transparente et asynchrone.
- **Outils de développement** : ROS met à disposition une variété d'outils qui simplifient la création, l'analyse et le débogage des applications robotiques. Comme l'outil le plus couramment utilisés `rviz` pour la visualisation en 3D, `rosbag` pour l'enregistrement et la lecture de données, ainsi que `rqt` pour l'inspection en temps réel.
- Les **fonctionnalités robotiques** regroupent les bibliothèques préétablies qui traitent des tâches robotiques essentielles telles que la navigation, la planification, la vision, la manipulation, la commande moteur, etc. Ces ensembles facilitent le développement rapide de comportements autonomes.
- L'un des points forts de ROS réside dans son écosystème communautaire riche et ouvert. Une vaste communauté d'utilisateurs, comprenant des chercheurs, des

ingénieurs, des industriels et des étudiants, assure la maintenance de milliers de packages, garantissant ainsi le développement continu du projet [15].

Ces éléments sont représentés dans la figure 3.1, laquelle résume leur fonction au sein de l'architecture globale de ROS.

FIGURE 3.1 – Les éléments fondamentaux de ROS comprennent la communication, les outils, les fonctionnalités robotiques et la communauté open-source [37].

En raison de son architecture modulaire et de l'engagement d'une communauté mondiale active, ROS s'est établi comme une norme en robotique, tant dans le domaine de la recherche que dans l'industrie. Les différentes versions de ROS, telles que ROS2 et ROS Industrial, sont conçues pour répondre à des exigences particulières comme le temps réel, la sécurité, la robustesse et la compatibilité avec des environnements industriels exigeants [45].

Comparaison des différentes versions de ROS

Depuis sa conception, le Robot Operating System (ROS) a connu une évolution constante, marquée par la sortie de plusieurs versions adaptées à divers contextes d'application. Les principales versions de Robot Operating System (ROS) sont `ROS-1`, `ROS-2` et ROS-Industrial (ROS-I), qui est une extension spécifiquement conçue pour l'automatisation industrielle [45]. Chacune de ces versions se caractérise par des architectures, des performances et des cas d'utilisation spécifiques.

Le tableau 3.1 se focalise sur la comparaison des deux variantes les plus stables et largement utilisées actuellement : `ROS-1 Noetic` (la dernière version officielle de `ROS-1`) et `ROS-2 Humble` (version LTS de `ROS-2`). Cette analyse souligne les distinctions entre eux en termes de compatibilité, de support en temps réel, de modularité et de niveau de développement dans des contextes de production [41, 36].

Critères	ROS-1 (Noetic)	ROS-2 (Humble)
Année de sortie	2007	2017
Support Ubuntu	Ubuntu 18.04, 20.04	Ubuntu 20.04, 22.04
Compatibilité Gazebo	Gazebo 9, 11	Gazebo 11
Meilleur TurtleBot	TurtleBot3 Burger	TurtleBot3 Waffle Pi, TurtleBot4
Support multi-agents	Pas idéalement	Oui
Capacité temps réel	Non	Oui
Support industriel	Non	Augmentant
Documentation	Mature	En augmentation
Support de maintenance	Proche EOL	Augmentant
Support des langages	C++11, Python2	C++11/14/17, Python3
Systèmes d'exploitation supportés	Linux, macOS	Linux, macOS, Windows, RTOS
Domaine recommandé	Loisir, Académique	Professionnel, Temps réel

TABLE 3.1 – Comparaison entre ROS-1 et ROS-2 selon différents critères [43]

La grande famille de TurtleBots

La série de plateformes robotiques mobiles TurtleBot est spécialement conçue pour l'apprentissage, la recherche et le prototypage. Depuis sa première introduction en 2010, cette gamme a subi plusieurs évolutions, chacune apportant des améliorations significatives en ce qui concerne le matériel, la compatibilité logicielle et les scénarios d'utilisation. L'évolution des divers modèles est illustrée dans la figure 3.2 [51].

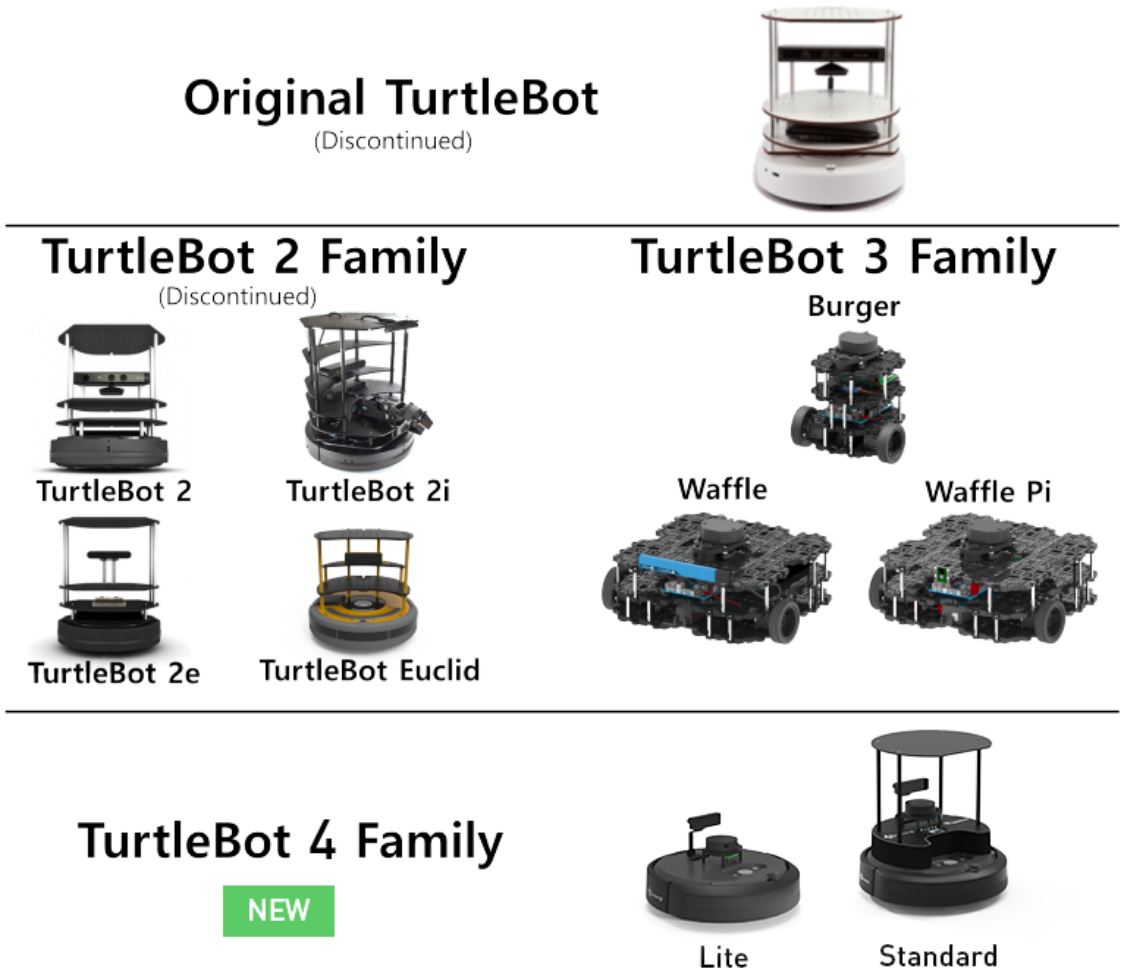

FIGURE 3.2 – L'évolution des plateformes TurtleBot s'étend du modèle initial aux différentes variantes du TurtleBot4 [43].

Les premières versions, à savoir le **TurtleBot 1** (2010) et le **TurtleBot 2** (2012), bien qu'ayant été innovantes à l'époque de leur sortie, ont depuis été arrêtées. Elles ont été conçues spécifiquement en s'établissant sur les plateformes *iRobot Create* et *Yujin Kobuki*, caractérisées par une structure imposante et une flexibilité restreinte. Leur tarif initial s'élevait à environ 1890 CAD pour le TurtleBot 1 et à 2025 CAD pour le TurtleBot 2 [51].

Lancée en 2017, la famille **TurtleBot3** est établie sur une architecture modulaire et compacte. Ce mémoire présente des expérimentations effectuées en simulation dans l'environnement `Gazebo`, grâce au robot **TurtleBot3 Burger**. Ce choix a été motivé par la taille, la facilité d'intégration et la conformité avec `ROS Noetic` de ce modèle. Son prix

abordable, d'environ 877,50 CAD, en fait une plateforme de choix pour la conception et l'apprentissage en robotique mobile [43].

La série **TurtleBot4**, introduite en 2022, incarne une nouvelle édition de plateformes spécialement conçues pour des utilisations robotiques de pointe. Elle est équipée de capteurs de haute précision tels que le LiDAR 3D, le RealSense et l'IMU, et elle est compatible nativement avec `ROS2`, ce qui la rend adaptée aux architectures distribuées et au traitement en temps réel. Proposé en versions *Lite* et *Standard*, ce produit est commercialisé à un prix d'environ **3000 CAD** [51].

Le tableau 3.2 expose de manière synthétique les distinctions techniques majeures entre les versions TurtleBot3 et TurtleBot4, mettant en lumière leur progression tant sur le plan matériel que logiciel.

Caractéristique	**TurtleBot3**	**TurtleBot4**
Base mobile	Burger, Waffle, Waffle Pi	Plateforme ROS2 de nouvelle génération
Batterie	Batterie lithium-ion 1800mAh	Batterie haute capacité
Ordinateur embarqué	Raspberry Pi 3/4, Intel Joule, OpenCR 1.0	Raspberry Pi 4, Jetson Nano, Jetson Xavier NX
Capteurs	LiDAR 2D, IMU 9 axes, encodeurs, caméra Pi/RealSense	LiDAR 3D, caméra de profondeur RealSense, IMU, encodeurs
Actionneurs	Servomoteurs Dynamixel	Servomoteurs Dynamixel de nouvelle génération
Systèmes pris en charge	ROS1 (`Noetic`), ROS2 (expérimental)	ROS2 (Foxy, `Humble`, Iron)
Année de sortie	2017	2023
Prix (CAD)	750–1200	3000–3200

Suite à la page suivante

Suite de la table 3.2

Caractéristique	TurtleBot3	TurtleBot4
Domaines d'application	Enseignement, recherche, prototypage académique	Applications industrielles, R&D, robotique collaborative
Développeurs	ROBOTIS, Open Robotics	ROBOTIS, Open Robotics

TABLE 3.2 – Comparaison des caractéristiques entre le TurtleBot3 et le TurtleBot4 [51]

Sélection des techniques pour les expérimentations simulées

Pour cette étude, les tests ont été réalisés en simulation grâce à `Gazebo 11.11.0`, sur une configuration logicielle bien établie et largement utilisée : `Ubuntu 20.04.6 LTS` avec `ROS Noetic` et le package `ros_comm` en version 1.17.0. Le robot employé ici est le `TurtleBot3 Burger`, dont le comportement a été simulé de manière virtuelle. Les algorithmes de navigation et de détection d'obstacles ont été intégrés en `Python 3.8.10`, dans un cadre compatible avec les exigences de développement dans ROS [41, 36, 45, 40].

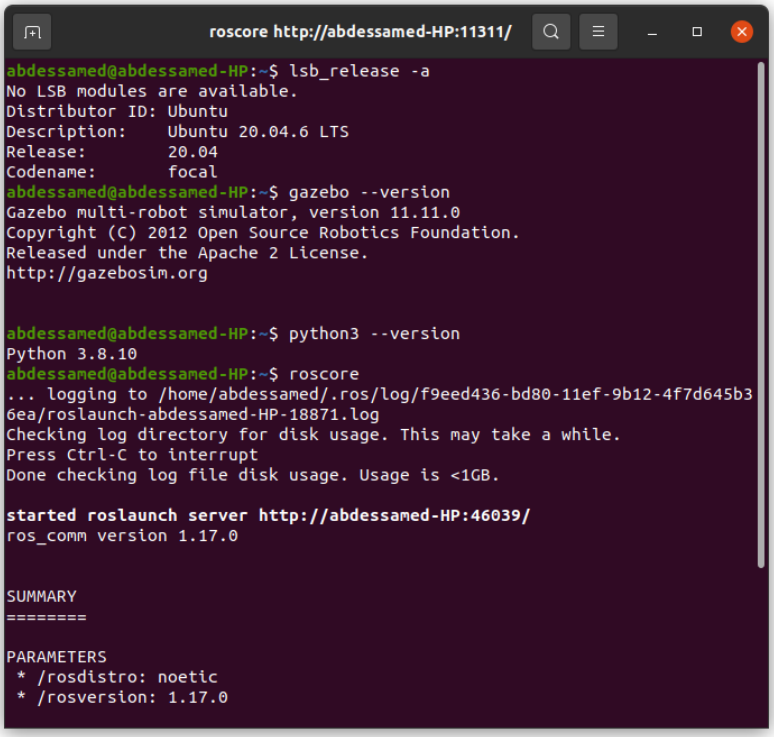

FIGURE 3.3 – Environnement logiciel utilisé pour les expérimentations : versions de `ROS`, `Ubuntu`, `Gazebo` et `Python`.

3.3 Modélisation du robot à entraînement différentiel

Cette section met en évidence la modélisation cinématique d'un robot mobile à entraînement différentiel, un modèle souvent utilisé dans le domaine de la robotique mobile. Le robot est décrit comme un système non holonome et rigide, ce qui signifie qu'il ne peut se déplacer que le long de l'axe longitudinal de son châssis, sans la capacité de changer de direction immédiatement. Cette contrainte découle de la configuration de l'entraînement différentiel, qui s'appuie sur deux roues motrices installées sur le même axe et gérées indépendamment [48].

L'objectif est d'établir un lien mathématique claire entre les vitesses angulaires des roues et l'état cinématique du robot, défini par sa vitesse linéaire et sa vitesse de rotation, afin de prédire le changement de sa position en fonction des ordres de mouvement. Ce modèle constitue un élément crucial pour la conception d'algorithmes de contrôle, de navigation autonome ou de localisation [20].

Le schéma 3.4 illustre les paramètres géométriques et cinématiques essentiels intégrés dans ce modèle. Le tableau 3.3 présente un récapitulatif organisé qui simplifie la compréhension et l'analyse des symboles utilisés.

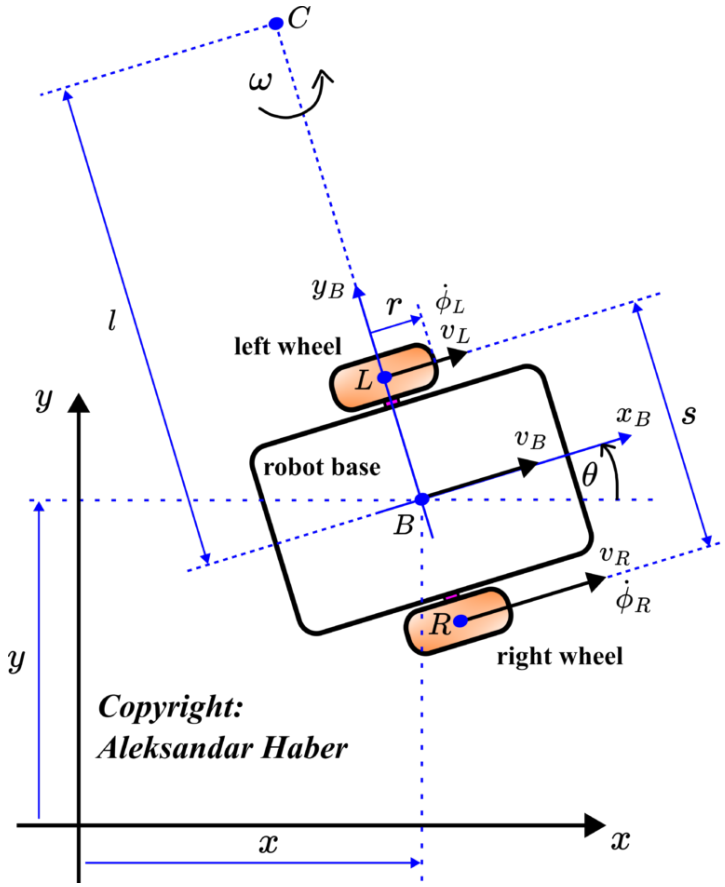

FIGURE 3.4 – Diagramme cinématique détaillé du robot différentiel [20].

Symbole	Description
x, y	Coordonnées du point B dans le repère inertiel.
θ	Angle d'orientation du robot par rapport à l'axe x inertiel.
x_B, y_B	Axes du repère local (attaché au robot, centré en B).
C	Centre instantané de rotation (CIR) du robot.
$\omega = \dot{\theta}$	Vitesse angulaire du robot autour de C.
L, R	Centres des roues gauche (L) et droite (R).

Suite à la page suivante

Suite de la table 3.3

Symbole	**Description**
B	Point médian entre L et R (centre géométrique du robot).
v_L, v_R	Vitesses linéaires des roues gauche et droite.
v_B	Vitesse linéaire du robot au point B.
$\dot{\phi}_L, \dot{\phi}_R$	Vitesses angulaires des roues gauche et droite.
l	Distance entre le point B et le centre instantané de rotation C.
r	Rayon des roues.
s	Empattement du robot : distance entre les roues gauche et droite.
$\dot{x}, \dot{y}$	Composantes de la vitesse du robot v_B dans le repère inertiel.

TABLE 3.3 – Résumé des notations et paramètres du modèle cinématique différentiel

Les vitesses angulaires des roues motrices $\dot{\phi}_L$ et $\dot{\phi}_R$ déterminent intégralement le mouvement du robot, étant reliées à leurs vitesses linéaires correspondantes par :

$$v_L = r\dot{\phi}_L, \quad v_R = r\dot{\phi}_R. \tag{3.1}$$

En prenant en compte le point central B, la vitesse angulaire du robot peut être formulée comme suit :

$$\omega = \frac{v_R - v_L}{s}, \tag{3.2}$$

et la vitesse linéaire du châssis au point B par :

$$v_B = \frac{v_R + v_L}{2}. \tag{3.3}$$

Une fois que les relations entre les vitesses des roues et celles du châssis ont été établies, le modèle cinématique direct du robot différentiel est obtenu. Les éléments constitutifs de la vitesse dans le référentiel inertiel sont ainsi définis comme étant la composante de la vitesse linéaire v_B projetée selon l'orientation θ du robot :

$$\dot{x} = v_B \cos(\theta), \tag{3.4}$$

$$\dot{y} = v_B \sin(\theta), \tag{3.5}$$

$$\dot{\theta} = \omega. \tag{3.6}$$

Ce système peut être exprimé sous forme matricielle :

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos(\theta) & 0 \\ \sin(\theta) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v_B \\ \omega \end{bmatrix}. \tag{3.7}$$

Ce modèle constitue la base de la commande cinématique pour le robot différentiel. Cet outil est conçu pour prévoir le mouvement en fonction des commandes du moteur, et possède une vaste utilisation dans les systèmes de navigation autonome, les simulateurs tels que Gazebo, ainsi que dans l'odométrie.

3.4 Architecture générale du nœud de navigation local

Les algorithmes de navigation examinés (VFH, SVC, Tangent Bug) s'appuient sur une structure ROS partagée, réalisée en tant que nœud modulaire. Cette méthode repose sur l'utilisation des messages standards de ROS pour établir une connexion entre les capteurs, comme le LiDAR et l'odométrie, et les actionneurs, tels que les commandes de vitesse. Le nœud principal se connecte à l'écosystème ROS en s'abonnant aux sujets de vision et de navigation, tout en émettant simultanément les consignes de déplacement et les informations visuelles.

3.4.1 Structure pratique du nœud de navigation

L'architecture logicielle est constituée de trois modules fonctionnels distincts : l'unité de **perception**, le module de **contrôle** et le module de **commande**. Chaque individu contribue de manière complémentaire au processus de prise de décision du robot :

– **Le module de perception** est responsable de l'acquisition, du filtrage et de l'interprétation des données issues du LiDAR et de l'odométrie. Il élabore une

cartographie spatiale de l'environnement, ce processus étant crucial pour la prise de décisions.

- **Le module de contrôle** intègre la logique de navigation propre à chaque algorithme (VFH, SVC, Tangent Bug). Il permet une transition dynamique entre différents modes de déplacement en fonction des conditions visibles de l'environnement (par exemple : ligne de visée dégagée ou présence d'obstacles).
- **Le module de contrôle et d'affichage** transforme les commandes du contrôleur en messages de type `Twist`, qui sont ensuite diffusés sur le topic `/cmd_vel`. Cette directive est périodiquement mise à jour dans la boucle principale (`run()`), assurant ainsi une réactivité aux changements de l'environnement.

La représentation graphique exposée dans la figure 3.5, générée à l'aide de la commande `rqt_graph`, met en évidence l'architecture de communication entre le nœud de navigation et les différents composants du système ROS. Elle souligne les liens entre le contrôleur VFH et les capteurs simulés, ainsi que les principes fondamentaux utilisés pour la commande et la surveillance du robot. Les algorithmes SVC et Tangent Bug partagent une structure similaire.

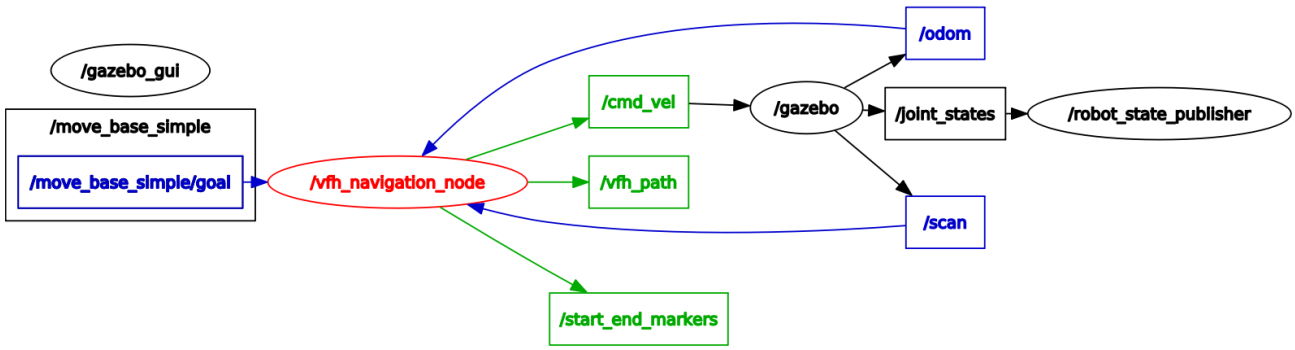

FIGURE 3.5 – Architecture ROS du système de navigation local.

Les flèches entrantes du schéma correspondent aux **abonnements** du nœud, à savoir :

- `/scan` : mesures brutes du télémètre (LiDAR 2D) ;
- `/odom` : estimation de la position et de l'orientation dans le repère odométrique ;
- `/move_base_simple/goal` : objectif défini par l'utilisateur via `RViz`.

Les flèches sortantes représentent les **publications**, notamment :

- `/cmd_vel` : consigne principale de vitesse spécifie les vitesses linéaire et angulaire.

– Parmi les caractéristiques complémentaires de `RViz` figurent la représentation visuelle de la trajectoire, des obstacles identifiés, ainsi que des points de départ et d'arrivée.

Chargement des paramètres ROS

Au démarrage, le nœud charge une série de paramètres à partir d'un fichier `YAML`, ce qui permet de personnaliser de manière dynamique les comportements internes en fonction de l'algorithme sélectionné. Diverses catégories de paramètres peuvent être distinguées :

– **Caractéristiques géométriques** : incluant la résolution angulaire, l'ouverture du cône frontal, les seuils de densité pour le groupement, etc. ;

– Les **paramètres de sécurité** font référence aux marges de sécurité ε et ε' qui sont employées dans le contrôleur (SVC) afin de superviser les zones d'alerte ;

– Les **paramètres de contrôle** incluent les gains du régulateur PID et la vitesse nominale utilisée en mode direct ;

– Les critères de convergence se réfèrent à la distance seuil qui est définie pour déterminer l'atteinte de l'objectif.

La fonction `rospy.get_param()` permet de récupérer dynamiquement ces réglages lors de l'exécution du fichier `.launch`, offrant ainsi la possibilité de modifier les comportements sans avoir à modifier le code source. Ensuite, chaque paramètre est enregistré dans une variable interne qui peut être accédée par les fonctions de traitement telles que le groupement, la navigation et la régulation.

Cette approche de configuration facilite l'adaptabilité de l'architecture aux pratiques. Elle permet en particulier :

– d'ajuster la résistance aux obstacles en changeant les seuils de distance ;

– de calibrer la précision d'atteinte des objectifs ;

– de modérer la réactivité du contrôleur en modifiant les gains PID ;

– d'impacter la logique de navigation (par exemple, le choix de direction dans VFH).

En somme, le fichier `YAML` joue le rôle d'une interface flexible et ajustable entre la configuration du système et la logique de navigation intégrée.

Une fois que les paramètres ont été chargés, le nœud initialise ses abonnements et publications en utilisant les méthodes `rospy.Subscriber` et `rospy.Publisher`, avant de passer à sa boucle principale d'exécution.

3.4.2 Traitement des données capteurs et modélisation de l'environnement

Le traitement des données des capteurs repose sur l'utilisation de fonctions de rappel (`callbacks`) qui garantissent la mise à jour en temps réel de l'état du système :

(i) `odom_callback()` : actualise la position du robot à partir de `/odom`.

(ii) `scan_callback()` : stocke la dernière mesure LiDAR issue de `/scan`.

(iii) `goal_callback()` : met à jour l'objectif de navigation.

Étape 1 : acquisition et préparation des données.

Le processus de traitement des données LiDAR s'articule autour d'une séquence d'étapes. Les données sont recueillies à partir du sujet `/scan` et sont exprimées en coordonnées polaires. En premier lieu, les données brutes subissent un processus de filtrage pour exclure les valeurs non valides, puis elles sont ensuite lissées à l'aide d'une moyenne mobile. Par la suite, chaque point valide est transformé du référentiel mobile du robot (`base_link`) vers le référentiel global (`odom`) en se basant sur les informations d'odométrie.

Étape 2 : classification des points par regroupement

Une fois les points LIDAR représentés dans le référentiel global, l'algorithme `DBSCAN` (Density-Based Spatial Clustering of Applications with Noise) est utilisé pour regrouper les mesures en structures [16]. Chaque ensemble est défini par un centre géométrique et un rayon d'enveloppe, ce qui permet de générer une représentation circulaire simplifiée de l'obstacle.

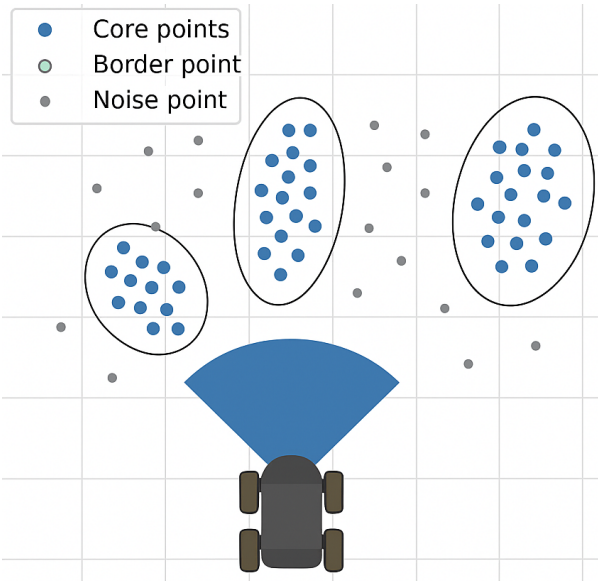

FIGURE 3.6 – Regroupement des données LiDAR en utilisant l'algorithme `DBSCAN`.

Chaque groupe de points déterminé de cette manière est caractérisé par un centre géométrique et un rayon d'enveloppe, créant ainsi un obstacle circulaire, comme le montre la figure 3.6. Cette illustration est largement claire pour la modélisation d'objets courants tels que des chaises, des coins de table ou des bords de mur, tout en restant facile à manipuler pour le contrôle.

Étape 3 : filtrage des points aberrants et exclusion des obstacles non pertinents.

La représentation circulaire permet de représenter l'environnement tout en assurant une marge de sécurité par rapport aux dimensions réelles des obstacles. Des filtres supplémentaires sont utilisés pour supprimer les artéfacts de perception. Plus précisément, les objets de dimensions très réduites qui se trouvent à proximité immédiate du robot et dont le diamètre est en dessous d'un seuil prédéfini sont rejetés, car ils sont habituellement considérés comme du bruit ou des réflexions parasites provenant du capteur LiDAR.

Étape 4 : Classification des obstacles principaux.

En cas de présence d'un obstacle large, tel qu'un mur ou une barrière, une méthode de subdivision est mise en œuvre : le groupe initial est divisé en plusieurs obstacles

ponctuels régulièrement répartis, chacun étant associé à une région de sécurité circulaire. Cette représentation plus détaillée permet de modéliser l'obstacle de façon plus précise, tout en répondant aux besoins de sécurité requise pour le calcul d'évitement.

Étape 5 : Établissement de la structure finale des données.

Les informations provenant de la perception structurée alimentent directement le processus décisionnel des divers algorithmes de navigation. Ces technologies sont utilisées pour évaluer les distances par rapport aux obstacles, pour ajuster de manière dynamique la commande motrice, en particulier à travers l'utilisation d'un contrôleur tel que le SVC, et pour assurer un comportement réactif et sécurisé. L'exactitude de cette carte géographique locale, élaborée à partir des données LiDAR, revêt une importance capitale afin de prévenir les collisions et garantir la fiabilité des déplacements autonomes dans des environnements partiellement explorés.

3.5 Implémentation des stratégies de navigation locales

Cette partie présente l'implémentation des algorithmes de navigation locale réactive sélectionnés pour ce projet : VFH modifié, T-Bug, SVC et SVC-PID. Chaque stratégie a été implantée en tant que nœud ROS distinct, branché aux capteurs simulés dans Gazebo et chargé de produire les instructions de mouvement. Les parties suivantes expliquent la logique derrière chaque méthode ainsi que son intégration dans l'architecture du robot simulé.

3.5.1 Implémentation modifiée de l'algorithme VFH

Dans cet implémentation Gazebo, nous avons utilisé une version modifie de l'algorithme `VFH`. Cette modification a pour but d'améliorer la réactivité et l'efficience de la navigation locale en milieu structuré, tout en optimisant l'efficacité du traitement numérique. Elle s'appuie sur une structure hybride où le système alterne de manière dynamique entre deux modes de fonctionnement : le mode direct, qui est activé lorsque la vue frontale est dégagée, et le mode d'évitement qui se base sur le calcul vectoriel traditionnel du VFH.

Cette approche a été inspirée des divers travaux précédents qui ont étudié des techniques de navigation conditionnelle ou adaptative. [53] a suggéré l'évolution `VFH+`, qui

perfectionne la prise de décision en introduisant des paramètres de direction privilégiée. Récemment, ils [35] ont mis en évidence l'importance d'éviter temporairement le calcul VFH lorsque l'itinéraire est clair, pour privilégier des trajectoires directes plus efficaces. Des recherches telles que celles de [54] et de [39] suggèrent également l'implémentation d'un mécanisme de validation préalable ou de contrôle commun, qui permet d'adapter le comportement du robot en fonction du niveau d'encombrement de l'environnement.

Mode de navigation directe

Dans l'architecture que nous avons implémentée sur ROS/Gazebo, le robot effectue d'abord une évaluation de son environnement proche grâce à un balayage angulaire. Plus spécifiquement, un essai est effectué dans un cône avant (habituellement fixé à 30°), centré sur l'orientation actuelle du robot. Si aucun obstacle n'est identifié dans cette zone, la voie vers l'objectif est estimée dégagée, et le signal de déplacement est alors produit directement, sans faire appel au module VFH.

Ce processus, présenté dans la figure 3.7, autorise le robot à se déplacer en milieu dégagé, tout en minimisant les calculs associés à l'élaboration de l'histogramme polaire. Le mode direct agit donc comme un contrôle décisionnel léger, renforçant la souplesse du mouvement tout en réduisant l'utilisation des capacités processeur. En fin de compte, cette approche mixte fait suite à l'évolution des VFH en cherchant à ajuster de manière dynamique les calculs de navigation en fonction des restrictions liées à l'environnement de travail.

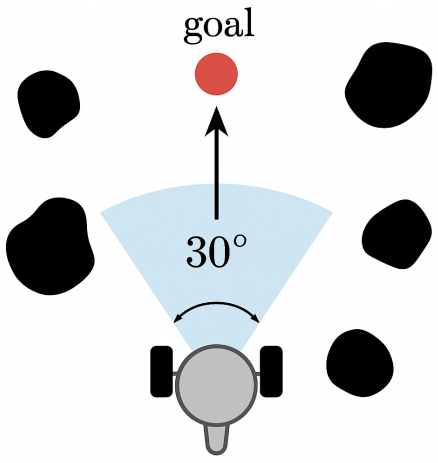

FIGURE 3.7 – Mode direct : cône frontal de 30 sans obstacle.

L'implémentation suggérée a démontré de bons résultats en simulation Gazebo/ROS, aussi bien en matière d'optimalité du chemin que de distance parcourue. Malgré la détection de certaines contraintes, en particulier dans des situations étroites ou complexes, les résultats atteints confirment l'efficacité de cette approche hybride. La section suivante comportera une évaluation comparative des performances, comprenant également les autres méthodes présentées dans cette étude.

3.5.2 Implémentation de l'algorithme T-Bug

Nous avons mis en œuvre l'algorithme Tangent Bug en respectant exactement la méthodologie exposée dans l'article de référence [22]. L'algorithme initial n'a pas subi de modifications structurelles. Les comportements de poursuite, de détection des points tangents et de suivi de contour ont été reproduits en respectant les définitions du modèle théorique d'origine. La section 2.4 a déjà fourni une explication approfondie de cet algorithme. Et les résultats de cette implémentation dans ROS/Gazebo seront présentés dans la section 3.5.5.

3.5.3 Implémentation du contrôleur SVC-PID

Le contrôleur `SVC` est une approche de navigation locale réactive qui vise à produire des commandes admissibles en conformité avec les limitations de sécurité associées à l'environnement [4]. Afin d'améliorer la stabilité dynamique et la robustesse du système, particulièrement en cas de perturbations, nous avons élaboré une structure hybride qui combine le `SVC` avec un contrôleur PID traditionnel.

Dans ce contexte, le régulateur PID détermine à l'avance un contrôle nominal basé sur l'écart de position par rapport à la cible. Puis, cette commande est envoyée au module SVC qui joue le rôle d'un filtre de sécurité : elle est intégrée dans l'espace des vitesses autorisées, en tenant compte de la distance minimale aux obstacles déterminée par le capteur `LiDAR`. La figure 3.8 présente le schéma général de cette architecture.

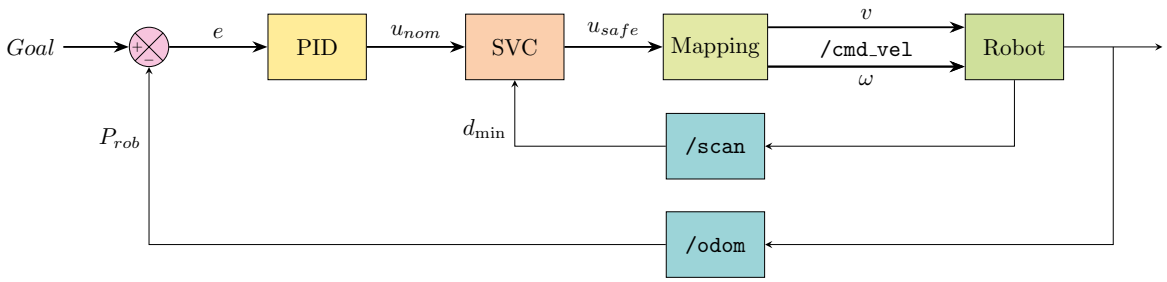

FIGURE 3.8 – Architecture de contrôle hybride SVC-PID : le régulateur PID génère une commande qui est ensuite sécurisée par le contrôleur SVC selon les contraintes perçues via le `LiDAR`.

Commande PID nominale

La commande nominale repose sur un régulateur PID vectoriel appliqué à l'erreur de position entre la position actuelle du robot $P = [x \;\; y]^T$ et la position cible $P_d = [x_d \;\; y_d]^T$. Elle est définie comme suit :

$$\mathbf{u}_{nom}(t) = k_P(P_d - P) + k_I \int_0^t (P_d - P)d\tau + k_D \frac{d}{dt}(P_d - P) \tag{3.8}$$

Ce vecteur $\mathbf{u}_{nom}(t)$ est ensuite sécurisé par le biais du filtre SVC pour produire le contrôlleur $\mathbf{u}_{safe}$. Ce dernier représente l'entrée en coordonnées cartésiennes, qui est ensuite transformée en commandes de vitesses cinématiques (v, ω) par le biais d'un processus de mapping.

En effet, le vecteur de commande $\mathbf{u}_{safe} = [u_x \;\; u_y]^T$ est transformé en direction cible θ_d et module de vitesse par les étapes suivantes :

(1) Calcul de l'orientation désirée :

$$\theta_d = \text{atan2}(u_y, u_x) \tag{3.9}$$

(2) Calcul de l'erreur d'orientation instantanée :

$$e_\theta = \theta_d - \theta \tag{3.10}$$

(3) La vitesse linéaire est modulée selon l'alignement angulaire :

$$v = \min\left(v_{\max}, \|\mathbf{u}\| \cdot \cos\left(\frac{e_\theta}{2}\right)\right) \tag{3.11}$$

(4) La vitesse angulaire est ajustée par :

$$\omega = k_p^\theta e_\theta + k_i^\theta \int e^\theta dt + k_d^\theta \frac{de_\theta}{dt} \tag{3.12}$$

3.5.4 Résultats expérimentaux et analyse comparative

La figure 3.9 illustre une vue simultanée de l'environnement Gazebo à droite et de la visualisation RViz à gauche. Le robot se déplace au milieu d'obstacles cylindriques, pendant que RViz visualise en temps réel la trajectoire effectuée (représentée par une courbe verte), les obstacles modélisés, ainsi que les repères de départ (en vert) et d'arrivée (en rouge).

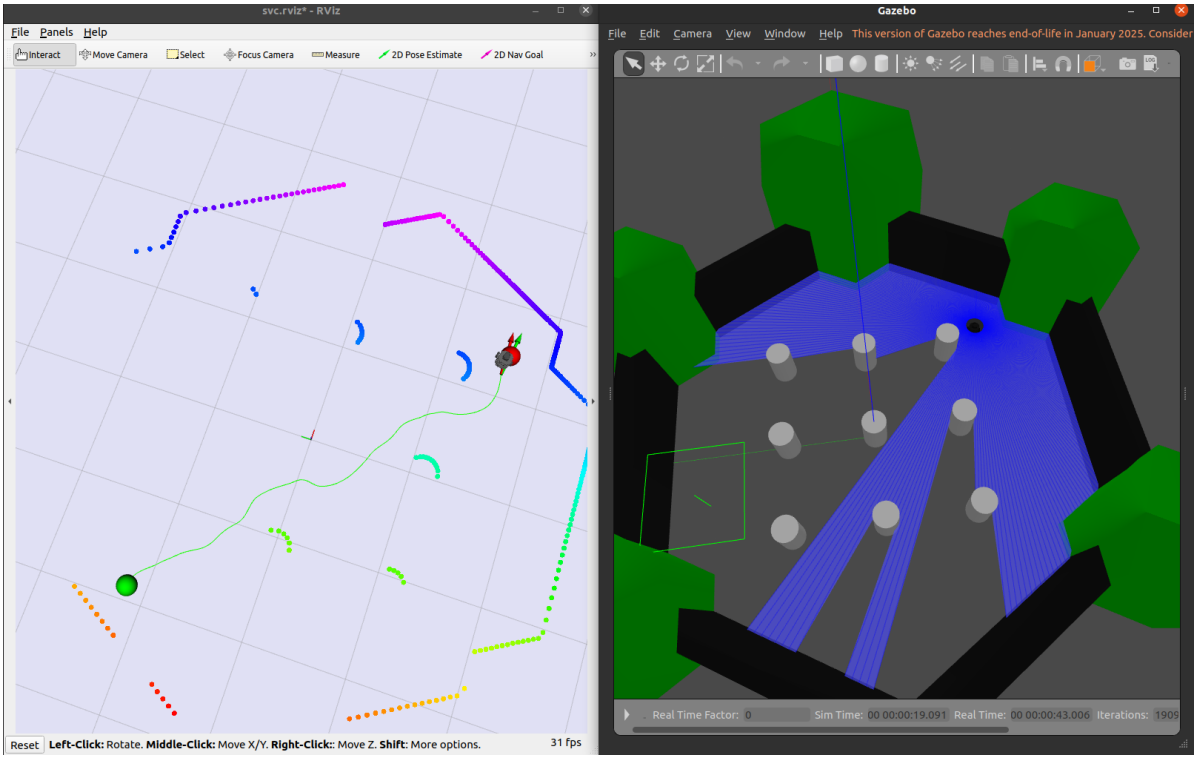

FIGURE 3.9 – Visualisation simultanée dans RViz (gauche) et Gazebo (droite) du comportement du contrôleur SVC-PID. [1]

1. Video : Real-time SVC+PID controller implementation — https://youtu.be/iNchHSrlQYE

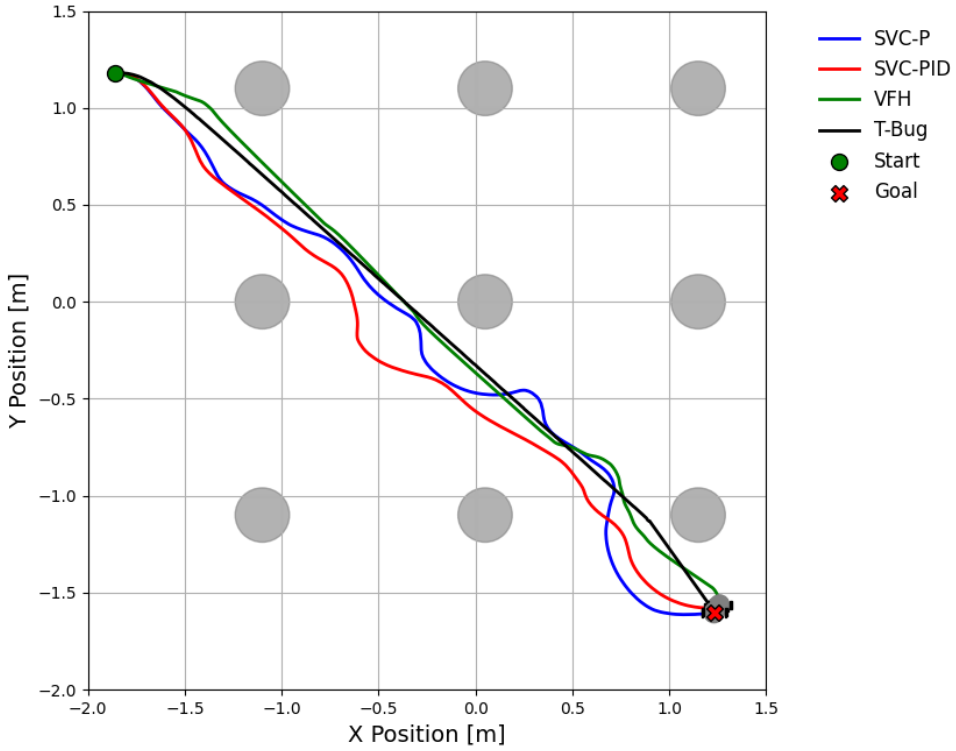

FIGURE 3.10 – Trajectoires du robot dans un environnement structuré avec obstacles : comparaison entre les algorithmes SVC, SVC couplé à un PID, VFH modifié et Tangent Bug.

Paramètre et valeur	**Description**
Paramètres communs à toutes les méthodes	
Portée LiDAR = 2.5 m	Distance maximale de détection.
LiDAR 2D (360°)	Pour la perception et le traitement des obstacles.
Paramètres de SVC / SVC-PID	
$\varepsilon = \mathbf{0.20}$ m	Marge de sécurité minimale autour des obstacles.
$\varepsilon' = \mathbf{0.25}$ m	Seuil supérieur pour la projection sécurisée du vecteur vitesse.

Suite à la page suivante...

Paramètre et valeur	**Description**
PID angulaire K_p = 2.5, K_i = 0.01, K_d = 0.5 ; **PID linéaire** K_p = 7.5, K_i = 0.01, K_d = 1.0	
Paramètres du VFH modifié	
Résolution angulaire = 20°	Taille d'un secteur de l'histogramme polaire.
Seuil d'occupation = 3.0	Valeur au-dessus de laquelle un secteur est considéré comme occupé.
Angle du cône frontal = 30°	Zone avant vérifiée pour passer en mode "**direct**".
$w_s = 20$ cellules	Taille de la fenêtre active carrée locale (2 m × 2 m).

TABLE 3.4 – Paramètres utilisés dans les implémentations ROS des algorithmes SVC, VFH et T-Bug

Comme discuté dans la partie comparative MATLAB, T-Bug se distingue des autres approches en ce qu'il ne repose pas sur des paramètres heuristiques explicites (comme des gains ou seuils d'activation). Il s'appuie plutôt sur une logique géométrique déterministe, fondée sur la visibilité directe, la mémorisation du point de contact avec l'obstacle et une détection de progrès vers le but. Ces mécanismes garantissent une navigation réactive sans nécessiter de réglages empiriques.

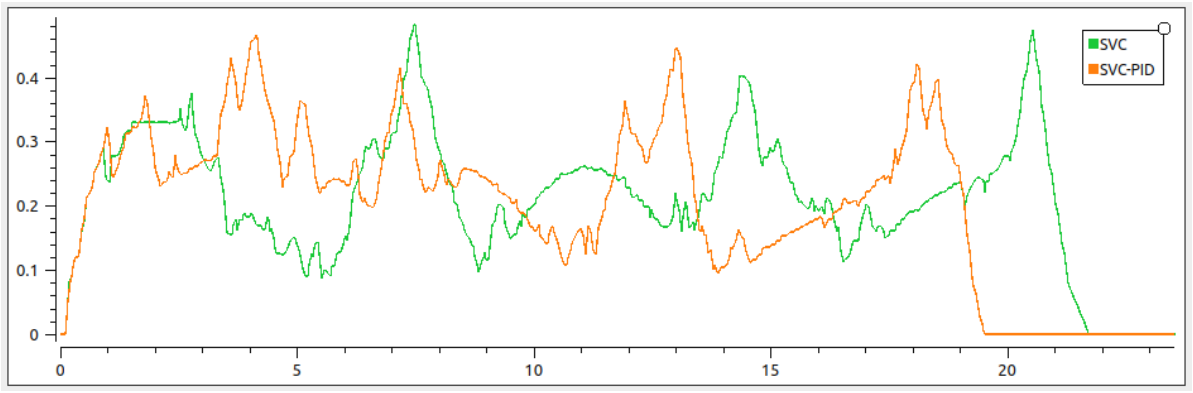

FIGURE 3.11 – Profil temporel des vitesses linéaires v (SVC et SVC-PID).

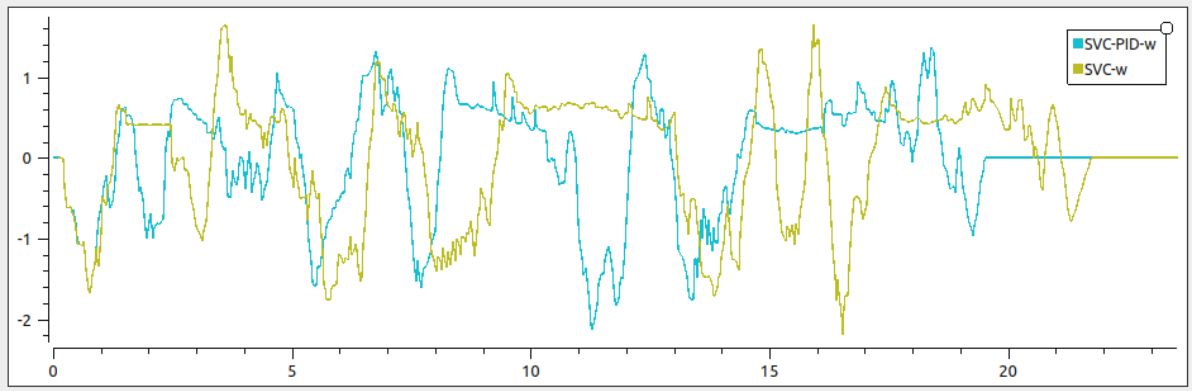

FIGURE 3.12 – Profil temporel des vitesses angulaires ω (SVC et SVC-PID).

3.5.5 Analyse comparative des résultats

Cette partie offre une étude quantitative des performances de quatre algorithmes de navigation locale : **SVC-PID**, **SVC-P**, **VFH** et **T-Bug**. On s'établit sur trois critères pour la comparaison : la longueur du parcours, le temps de traitement par itération et la distance de dégagement, comme définie dans la section 2.5.1.

Les évaluations ont été effectuées dans un environnement simulé identique, avec des obstacles fixes, en employant un détecteur LiDAR à 360. Des données présentées dans le tableau 3.5, provenant de plusieurs exécutions maîtrisées dans des conditions similaires, ont été prélevées à une fréquence de 20 Hz. L'écart de sécurité fait référence à la distance mesurée, à chaque cycle de traitement, entre le robot et l'obstacle le plus près identifié par le LiDAR.

Critère	**SVC-PID**	**SVC (P)**	**VFH**	**T-Bug**
Temps moyen / itération [s]	0.0053	**0.0042**	0.0136	0.0102
Distance totale D_{total} [m]	4.52	4.56	**4.29**	4.34
Distance de dégagement moyenne $\bar{C}$ [m]	0.326	**0.327**	0.279	0.265

TABLE 3.5 – Comparaison des algorithmes selon les critères de performance.

Interprétation détaillée des résultats expérimentaux

L'étude des trajectoires provenant de la simulation présentée dans la figure 3.10 et les critères quantitatifs indiqués dans le tableau 3.5 nous permet une comparaison approfondie des performances des quatre algorithmes examinés : SVC, SVC-PID, VFH modifié et T-bug.

SVC-PID :

Le contrôleur hybride SVC-PID révèle une conduite généralement équilibrée. Même si sa **distance totale parcourue est légèrement plus grande** que celle de VFH (4.52 m comparé à 4.29 m), il produit une **trajectoire plus fluide et mieux stabilisée** par rapport à un SVC seul. Cette constance est due à la régulation PID, qui atténue les oscillations dynamiques provoquées par les réactions brusques du SVC. La **durée moyenne par itération de 0.0053 s** demeure conforme aux exigences de temps réel. La **distance de dégagement moyenne** ($\bar{C} = 0.326$ m) témoigne d'une **navigation sûre**, garantissant une distance suffisante par rapport aux obstacles identifiés.

SVC :

La méthode SVC présente le temps de traitement le plus rapide (0.0042 s), démontrant une réactivité remarquable. Elle affiche aussi la distance de dégagement moyenne la plus basse ($\bar{C} = 0.327$ m), reflétant une approche prudente en termes de sécurité. Cependant, la trajectoire observée (courbe bleue) indique une instabilité croissante, surtout dans les zones étroites. On observe une distance totale légèrement plus longue (4.56 m) et des écarts plus marqués. Ce comportement met en évidence l'importance d'associer le SVC à un mécanisme de contrôle tel que le PID pour améliorer la stabilité.

VFH modifié :

L'algorithme VFH modifiée affiche la performance optimale en matière de distance parcourue (4.29 m), prouvant ainsi sa compétence pour produire des trajectoires optimisées dans un environnement structuré. Cette performance est entre autres due à la combinaison bénéfique avec le mode direct, qui autorise le robot à progresser en ligne droite dès qu'aucun obstacle n'est repéré dans un cône de devant. Cette combinaison de

mode direct et d'histogramme permet de produire une trajectoire plus tendue et directe, ce qui diminue la distance totale parcourue.

Toutefois, cette efficacité spatiale est liée au temps de calcul le plus élevé (0.0136 s), dû à la création et à l'examen d'histogrammes directionnels à chaque itération. De plus, la **distance de dégagement moyenne la moins élevée** ($\bar{C} = 0.279$ m) témoigne d'une navigation plus agressive, marquée par une proximité réduite avec les obstacles. Cette approche, bien qu'efficace dans un cadre organisé, peut provoquer des dangers supplémentaires en situation d'incertitude, de bruit ou de dynamisme, où l'exigence de marges de sécurité plus larges est souvent nécessaire.

T-Bug :

Grâce à l'algorithme T-Bug, le robot a pu atteindre l'objectif, attestant ainsi de sa caractéristique théorique de complétude. Il affiche une distance totale modérée (4,34 m), un temps moyen par itération raisonnable (0,0102 s) et la distance de dégagement la plus faible ($\bar{C} = 0{,}265$ m). Cette approche est efficace pour atteindre l'objectif, mais elle fait passer le robot très près des obstacles, notamment en mode de suivi des murs. Cela peut poser des défis dans un contexte de capteurs bruités ou d'environnements partiellement observables, où la sûreté du déplacement est primordiale.

3.6 Conclusion

Ce chapitre a présenté l'implémentation des algorithmes de navigation locale réactive sur un robot différentiel simulé dans l'environnement ROS/Gazebo. En s'appuyant sur le modèle cinématique du TurtleBot3 et l'intégration de capteurs (LiDAR 2D et odométrie), les méthodes SVC-PID, VFH modifié et T-Bug ont été évaluées dans des scénarios simulés reproduisant des configurations variées.

L'étude comparative a permis de mettre en évidence les comportements caractéristiques de chaque approche. L'algorithme VFH modifié a parfois produit des trajectoires relativement courtes, mais au prix d'une distance de dégagement réduite et d'une sensibilité accrue aux paramètres. Le contrôleur SVC combiné à un régulateur PID a offert un bon compromis entre robustesse, sécurité et régularité du mouvement, en amortissant les perturbations qui peuvent compromettre les garanties de sécurité du SVC pur (biais, glissements, retards). Bien que l'algorithme T-Bug montre une certaine efficacité dans un environnement structuré, il demeure sensible aux conditions d'incertitude : il produit

des trajets à proximité des obstacles et peut faire preuve d'instabilité lors de la phase finale, ce qui compromet parfois l'atteinte de l'objectif.

Ces résultats confirment que chaque méthode implique des compromis spécifiques entre distance parcourue, temps de traitement et sécurité de navigation. Parmi les variantes testées, l'approche SVC-PID apparaît comme la plus équilibrée, avec une exécution robuste, stable et relativement sûre. L'architecture mise en œuvre constitue une base pertinente pour des travaux futurs, notamment l'intégration de stratégies hybrides, l'adaptation contextuelle en temps réel, ou le transfert vers des plateformes physiques réelles.

4

Conclusion & Perspectives

Bilan des résultats obtenus

Ce travail de recherche a porté sur la navigation réactive de robots mobiles en environnement inconnu, à travers une analyse comparative approfondie de trois algorithmes d'évitement d'obstacles locaux : SVC, VFH et T-Bug. Ces approches ont été sélectionnées à l'issue d'une revue de littérature, distinguant les avantages et limites des méthodes de navigation globales par rapport aux approches réactives locales en temps réel.

Les expérimentations menées sous ROS/Gazebo ont mis en évidence le comportement spécifique de chaque algorithme dans un scénario commun simulé. En résumé :

- L'approche SVC : fournit des garanties théoriques rigoureuses. Quand elle est associée à un contrôleur PID, elle se révèle particulièrement résistante aux perturbations tout en produisant des trajectoires plus naturelles. En outre, cette approche, à la fois performante et simple en matière de calculs, permet de déterminer directement la commande à partir de l'angle de visée vers l'obstacle et de la distance.
- La version initiale de la méthode VFH (Histogramme de Champ Vectoriel) présente des problèmes d'oscillation dans les environnements complexes. Néanmoins, les modifications suggérées dans ce mémoire améliorent significativement la qualité des trajectoires produites. Toutefois, cette approche est limitée à 2D et ne s'étend pas naturellement en 3D comme le fait SVC. De plus, elle exige plus de ressources de calcul très complexes que SVC.
- L'algorithme T-Bug se caractérise par son optimalité de la longueur de trajectoire, en principalement due à sa stratégie de contournement plus directe. Cependant,

sans ajustements particuliers, cette technique a tendance à garder une distance de sécurité réduite face aux obstacles, ce qui peut mettre en risque la robustesse du déplacement dans des environnements incertains ou bruités.

L'analyse comparative a souligné les avantages particuliers et les contraintes de chaque technique. La méthode SVC s'est fait démontrer par sa stabilité et sa résistance aux perturbations, en particulier grâce à l'intégration d'un régulateur PID. La technique VFH, améliorée dans cette étude, a prouvé sa réactivité et sa puissance à produire des trajectoires lisses, même si cela entraîne une complexité de calculs. Finalement, l'algorithme T-Bug a démontré son efficacité à produire une trajectoire optimale avec une distance parcourue courte. Cependant, la distance de dégagement est trop basse, donc il n'offre pas une bonne sécurité de navigation.

Limites de l'étude

Bien que les résultats soient encourageants, cette étude comporte plusieurs limites. Initialement, les comparaisons se sont principalement réalisées en simulation sous MATLAB et ROS/Gazebo, ce qui peut entraîner des différences avec le rendement dans des conditions réelles. Des éléments comme le bruit du capteur, les incertitudes de calibration et les dynamiques réelles n'ont pas pu être pris en compte de façon complète.

Cependant, chaque algorithme comporte des limites. Le SVC, malgré sa performance et sa stabilité, pourrait produire des points d'équilibre stables indésirables si certaines conditions de courbure des obstacles ne sont pas observées. Le VFH nécessite une calibration manuelle précise de plusieurs paramètres essentiels, ce qui compromet sa robustesse dans des environnements dynamiques. Pour conclure, le T-Bug peut mener à des déviations excessives ou faire preuve d'inefficacité dans des situations géométriques complexes, sans oublier la difficulté de maintenir une marge de sécurité faible par rapport aux obstacles.

Perspectives futures

Ce travail ouvre le chemin à diverses perspectives de recherche. L'objectif immédiat est de tester de manière expérimentale les algorithmes étudiés sur une plateforme robotique concrète, comme le *TurtleBot3/4*. Cette phase permettra d'examiner leur so-

lidité face au bruit du capteur, aux perturbations dynamiques et aux incertitudes. Cela renforcerait les conclusions tirées de la simulation.

Il serait également envisageable d'apporter des améliorations spécifiques au niveau algorithme. Il serait possible d'optimiser l'approche SVC pour une exécution en temps réel plus légère, notamment par la simplification de certaines opérations géométriques. Une approche intéressante serait d'associer le SVC à un contrôleur nominal issu d'un planificateur global, comme l'RRT ou le PRM, pour profiter d'une orientation à long terme combinée à une réaction locale. En outre, le VFH serait plus robuste s'il incluait un ajustement dynamique de ses paramètres selon la condition locale de l'environnement. Une telle modification permettrait de réduire sa sensibilité aux ajustements manuels dans des environnements compliqués.

Finalement, l'intégration de méthodes d'apprentissage automatique, en particulier l'apprentissage par renforcement, représente un objectif ambitieux pour fournir les contrôleurs locaux d'une unité d'adaptation indépendante. Des méthodes similaires permettraient au robot de se former sur des stratégies optimales de navigation à travers l'expérience, même dans des environnements partiellement observables, dynamiques ou inconnus.

Ces divers points de vue permettent l'élaboration de systèmes de navigation hybrides, intelligents et robustes qui sont capables d'équilibrer la prise de décision locale, la planification globale et l'apprentissage en ligne.

Bibliographie

[1] Ames, A. D., Xu, X., Grizzle, J. W., and Tabuada, P. Control barrier function based quadratic programs for safety critical systems. *IEEE Transactions on Automatic Control 62*, 8 (2017), 3861–3876.

[2] Arslan, O., and Koditschek, D. Exact robot navigation using power diagrams. In *IEEE International Conference on Robotics and Automation (ICRA)* (2016), pp. 1–8.

[3] Bellman, R. On a routing problem. *Quarterly of Applied Mathematics 16*, 1 (1958), 87–90.

[4] Berkane, S. Navigation in unknown environments using safety velocity cones. *Proceedings of the American Control Conference* (2021), 2336–2341.

[5] Berkane, S., Bisoffi, A., and Dimarogonas, D. V. A hybrid controller for obstacle avoidance in an n-dimensional euclidean space. In *2019 18th European Control Conference (ECC)* (2019), pp. 764–769.

[6] Berkane, S., Bisoffi, A., and Dimarogonas, D. V. Obstacle avoidance via hybrid feedback. *IEEE Transactions on Automatic Control* (2021), 1–8.

[7] Borenstein, J., and Koren, Y. The vector field histogram-fast obstacle avoidance for mobile robots. *IEEE Transactions on Robotics and Automation 7*, 3 (1991), 278–288.

[8] Bouligand, G. Introduction à la géométrie infinitésimale directe.

[9] Cheniouni, I., Berkane, S., and Tayebi, A. Hybrid feedback control for global navigation with locally optimal obstacle avoidance in n-dimensional spaces. *IEEE Transactions on Robotics (submitted)* (2024).

[10] Choset, H., and Burdick, J. Sensor based planning. i. the generalized voronoi graph. 1649–1655 vol.2.

[11] Choset, H., and Burdick, J. Sensor-based exploration : The hierarchical generalized voronoi graph. *International Journal of Robotics Research 19*, 2 (2000), 96–125.

[12] Choset, H., Lynch, K. M., Hutchinson, S., Kantor, G. A., and Burgard, W. *Principles of Robot Motion : Theory, Algorithms, and Implementations*. MIT Press, 2005.

[13] Delfour, M. C., and Zolésio, J. P. Shapes and geometrics. *2011, vol. 53, no. 9* (2011).

[14] Dijkstra, E. W. A note on two problems in connexion with graphs. *Numerische Mathematik 1*, 1 (1959), 269–271.

[15] Documentation, R. Managing ros 2 environments. Online, 2024. Disponible sur : https://docs.ros.org/en/rolling/index.html.

[16] Ester, M., Kriegel, H.-P., Sander, J., and Xu, X. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD)* (1996), pp. 226–231.

[17] Floyd, R. W. Algorithm 97 : Shortest path. *Communications of the ACM 5*, 6 (1962), 345.

[18] Fox, D., Burgard, W., and Thrun, S. The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine 4*, 1 (1997), 23–33.

[19] Ge, S. S., and Cui, Y. J. Dynamic motion planning for mobile robots using potential field method. *Autonomous robots 13* (2002), 207–222.

[20] Haber, A. Clear and detailed explanation of kinematics equations and geometry of motion of differential wheeled robot (differential drive robot). aleksandarhaber.com/kinematics, 2023. Accessed : 2025-05-14.

[21] Hart, P. E., Nilsson, N. J., and Raphael, B. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics 4*, 2 (1968), 100–107.

[22] Kamon, I., Rimon, E., and Rivlin, E. Tangentbug : A range-sensor-based navigation algorithm. *The International Journal of Robotics Research 17*, 9 (1998), 934–953.

[23] Kamon, I., Rivlin, E., and Rimon, E. A new range-sensor based globally convergent navigation algorithm for mobile robots. In *Proceedings of IEEE International Conference on Robotics and Automation* (1996), vol. 1, IEEE, pp. 429–435.

[24] Karaman, S., and Frazzoli, E. Sampling-based algorithms for optimal motion planning. *The international journal of robotics research 30*, 7 (2011), 846–894.

[25] Kavraki, L. E., Švestka, P., Latombe, J.-C., and Overmars, M. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation 12*, 4 (1996), 566–580.

[26] Khatib, O. Real-time obstacle avoidance for manipulators and mobile robots. *The International Journal of Robotics Research 5*, 1 (1986), 90–98. Online Version.

[27] Koenig, S., and Likhachev, M. D* lite. *AAAI Conference on Artificial Intelligence* (2002), 476–483.

[28] Koren, Y., and Borenstein, J. Potential field methods and their inherent limitations for mobile robot navigation. In *Proceedings of the IEEE International Conference on Robotics and Automation* (1991), pp. 1398–1404.

[29] Kris, H. Robotic systems (draft), 2018.

[30] Labib, N. *Collision Avoidance and Simple Path Planning for Autonomous Robotic Exploration*. PhD thesis, 10 2014.

[31] LaValle, S. Rapidly-exploring random trees : A new tool for path planning. *Research Report 9811* (1998).

[32] LaValle, S. M. *Planning Algorithms*. Cambridge University Press, 2006.

[33] Lee, D.-T. *Proximity and reachability in the plane*. University of Illinois at Urbana-Champaign, 1978.

[34] Lumelsky, V. J., and Stepanov, A. A. Dynamic path planning for a mobile automaton with limited information on the environment. *IEEE Transactions on Automatic Control 31*, 11 (1986), 1058–1063.

[35] Luo, C., Krishnan, M., Paulik, M., and Wang, Q. Navigating with vfh : A strategy to avoid traps. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2010), IEEE, pp. 1433–1438.

[36] Macenski, S., Foote, T., Gerkey, B., Lalancette, C., and Woodall, W. Robot Operating System 2 : Design, Architecture, and Uses in the Wild. *Science Robotics 7*, 66 (2022), eabm6074.

[37] Militano, L., Arteaga, A., Toffetti, G., and Mitton, N. The cloud-to-edge-to-iot continuum as an enabler for search and rescue operations. *Future Internet 15*, 2 (2023), 55.

[38] Nagumo, M. Über die lage der integralkurven gewöhnlicher differentialgleichungen. *Proceedings of the Physico-Mathematical Society of Japan. 3rd Series 24* (1942), 551–559.

[39] Pappas, P., Chiou, M., Epsimos, G.-T., Nikolaou, G., and Stolkin, R. Vfh+ based shared control for remotely operated mobile robots. *arXiv preprint arXiv :2011.05228* (2020).

[40] PhoenixNAP. How to install python 3 on ubuntu, 2023.

[41] Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., and Ng, A. ROS : an Open-Source Robot Operating System. In *ICRA 2009 Workshop on Open Source Software* (2009).

[42] Rimon, E., and Koditschek, D. Exact robot navigation using artificial potential functions. *IEEE Transactions on Robotics and Automation 8*, 5 (1992), 501–518.

[43] ROBOTIS. Robotis e-manual. emanual.robotis.com, 2025. Accessed : 2025-05-14.

[44] Sawant, M., et al. Hybrid feedback for autonomous navigation in planar environments with convex obstacles. *IEEE Transactions on Automatic Control 68*, 12 (2023), 7342–7357.

[45] Shamshiri, R. R., Hameed, I. A., and Weltzien, M. K. a. *Robotic Harvesting of Fruiting Vegetables : A Simulation Approach in V-REP, ROS and MATLAB*. IntechOpen, 2018, ch. 1.

[46] Siegwart, R., Nourbakhsh, I. R., and Scaramuzza, D. *Introduction to autonomous mobile robots*. MIT press, 2011.

[47] Smaili, L., and Berkane, S. Real-time sensor-based feedback control for obstacle avoidance in unknown environments. In *2024 American Control Conference (ACC)* (2024), IEEE, pp. 2691–2696.

[48] Spong, M. W., and Vidyasagar, M. *Robot Dynamics and Control*, 2nd ed. John Wiley & Sons, Hoboken, NJ, 2008.

[49] Stentz, A. The focussed d* algorithm for real-time replanning. In *International Joint Conference on Artificial Intelligence (IJCAI)* (1995), pp. 1652–1659.

[50] Tan, X., and Dimarogonas, D. On the undesired equilibria induced by control barrier function based quadratic programs. *Automatica 159* (2024), 111359.

[51] TurtleBot. Turtlebot official website, 2024.

[52] Ubbriaco, G. Relazione di progetto : Robotica mobile, 2022. Université de Rome, projet académique.

[53] Ulrich, I., and Borenstein, J. Vfh+ : Reliable obstacle avoidance for fast mobile robots. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)* (1998), vol. 2, IEEE, pp. 1572–1577.

[54] Ulrich, I., and Borenstein, J. Vfh* : Local obstacle avoidance with look-ahead verification. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)* (2000), vol. 3, IEEE, pp. 2505–2511.

[55] Vasilopoulos, V., et al. Reactive semantic planning in unexplored semantic environments using deep perceptual feedback. *IEEE Robotics and Automation Letters 5*, 3 (2021), 4455–4462.

[56] Zhang, X., Liniger, A., and Borrelli, F. Optimization-based collision avoidance. *IEEE Transactions on Control Systems Technology 29*, 3 (2021), 972–983.