UNIVERSITÉ DE SHERBROOKE
Faculté de génie
Département de génie électrique et de génie informatique

PLANIFICATION DE TRAJETS POUR UN VÉHICULE À GUIDAGE AUTOMATIQUE HOLONOME À PARCOURS LIBRE

Cédric GODIN

Sherbrooke (Québec) Canada
Avril 2021

MEMBRES DU JURY

François MICHAUD

Directeur

François FERLAND

Évaluateur

François GRONDIN

Évaluateur

RÉSUMÉ

Les véhicules à guidage automatique (*Automated Guided Vehicule* – AGV) sont de plus en plus répandus dans le milieu industriel. Ces véhicules permettent le transport automatisé des biens dans une usine, ce qui libère de la main d'œuvre qui peut réaliser des tâches ayant une plus grande valeur ajoutée. Récemment, certains manufacturiers ont débuté la mise en marché d'AGV holonomes. Ces derniers peuvent se déplacer latéralement, libérant l'orientation du véhicule pendant ses mouvements. Ce degré de liberté supplémentaire n'est toutefois pas géré par les planificateurs de trajectoires existants. En absence d'obstacles pour limiter la rotation de l'AGV, ce dernier a tendance à pivoter de manière erratique lorsqu'il se déplace. Cela entraîne un inconfort chez les personnes à proximité et peut endommager une charge fragile placée sur le véhicule. Aussi, les logiciels existants ne permettent pas à un utilisateur de communiquer au véhicule les règles de circulation spécifiques à un milieu de travail.

Les travaux présentés dans ce mémoire se découpent en trois volets. D'abord, un planificateur global, qui trouve un trajet composé d'une suite de points de cheminement (combinaison d'une position et d'une orientation) entre un point de départ et un point d'arrivée, est décrit. Ce planificateur est livré sous la forme d'un paquet pouvant être utilisé directement avec les logiciels de navigation existant dans l'environnement du *Robotic Operating System* (ROS). L'algorithme utilise des intervalles d'orientation au lieu de valeurs discrètes afin de réduire le nombre de nœuds dans le graphe A*. Ensuite, un planificateur local existant dans ROS est modifié pour les besoins des AGV holonomes. Il permet de calculer les commandes de vitesses qui doivent être adoptées par l'AGV afin de suivre le trajet calculé par le planificateur global. Il est livré comme une version discrète d'un paquet ROS et remplace l'original. Le bon fonctionnement des planificateurs est validé en simulation et sur un véritable AGV. Enfin, une preuve de concept en simulation est réalisée pour la gestion des règles de circulation. Une interface permet de définir des comportements par zones, qui sont ensuite encodées dans des grilles d'occupation et sauvegardées sur le robot. Un logiciel de navigation adapté se réfère ensuite à ces grilles pour ajouter la bonne orientation au trajet et pour circuler aux bons endroits.

Ce projet de recherche a montré qu'il est possible d'exploiter l'holonomie d'un AGV dans un logiciel de navigation intégré à la pile de navigation ROS originale. Toutefois, certains enjeux demeurent au niveau du planificateur local qui tente d'établir un compromis entre le suivi exact du trajet et la progression vers le but. Comme travaux futurs, il serait intéressant d'ajouter la gestion de l'interaction avec les systèmes de sécurité présents sur les véhicules industriels.

Mots-clés : Automatisation, Localisation et cartographie simultanée, Odométrie, Contrôle, Navigation, AGV

TABLE DES MATIÈRES

CHAPITRE 1 INTRODUCTION 1
CHAPITRE 2 DÉPLACEMENT DES ROBOTS HOLONOMES 7
2.1 Base robotique holonome 8
2.2 Cartographie et localisation 10
2.3 Planification globale 11
2.4 Planification locale 13
2.5 Interaction humaine 15
CHAPITRE 3 PLANIFICATEUR GLOBAL 17
3.1 Discrétisation de l'orientation en intervalles libres 19
3.1.1 Nœuds 19
3.1.2 Graphe de nœuds 20
3.1.3 Heuristique et coût de déplacement 25
3.1.4 Implémentation 27
3.2 De l'algorithme A* à un planificateur global complet 29
3.2.1 Implémentation 31
3.3 Réduction du temps de recherche du coût de l'empreinte 33
3.3.1 Méthode des quatre coins 34
3.3.2 Méthode des deux cercles 36
3.3.3 Implémentation 38
CHAPITRE 4 PLANIFICATEUR LOCAL 40
4.1 Planificateur DWA 40
4.2 Ajout d'une nouvelle fonction de coût 42
4.3 Implémentation 42
CHAPITRE 5 GESTION DES RÈGLES DE CIRCULATION 44

5.1 Encodage dans les couches des grilles d'occupation 44
5.2 Interface graphique 46
5.3 Intégration avec la pile de navigation ROS-NH 47
CHAPITRE 6 TESTS ET RÉSULTATS 48
6.1 Tests avec l'AGV VII 48
6.2 Tests en simulation 57
6.2.1 Circulation dans un espace ouvert 59
6.2.2 Circulation d'un mur à l'autre 62
6.2.3 Circulation entre deux cavités 65
6.2.4 Stationnement dans un espace restreint 68
6.2.5 Sommaire des scénarios simulés 72
6.3 Démonstration de la gestion des règles de circulation 73
CHAPITRE 7 CONCLUSION 76

LISTE DES FIGURES

Figure 1.1 L'AGV VII d'Inogec 2
Figure 1.2 Orientation d'un AGV en mouvement (les rectangles noirs représentent des obstacles) 3
Figure 1.3 Pile de navigation ROS-H 5
Figure 3.1 Intervalles d'orientation 20
Figure 3.2 Processus pour déterminer l'orientation à adopter 31
Figure 3.3 Méthode des quatre coins 35
Figure 3.4 Collision entre les coins éloignés du centre 36
Figure 3.5 Méthode des deux cercles 37
Figure 4.1 Fonctions de coût de DWA : a) but; b) trajet; c) obstacles 41
Figure 5.1 Interface des paramètres de circulation avec une zone d'orientation spécifique 46
Figure 5.2 Zone préférée et interdite dans l'interface 47
Figure 6.1 Carte produite par le SLAM lors de l'ajustement des poids 49
Figure 6.2 Écart entre le trajet et le but 51
Figure 6.3 Environnement de test chez Inogec 52
Figure 6.4 Carte produite par le SLAM 53
Figure 6.5 Trajet lors de la série A 54
Figure 6.6 Vitesses lors de la série A 54
Figure 6.7 Trajet lors de la série B 55
Figure 6.8 Vitesses lors de la série B 56
Figure 6.9 Environnement de simulation 58
Figure 6.10 Carte produite par le système SLAM 58
Figure 6.11 Trajet dans un espace ouvert 60
Figure 6.12 Déplacement dans un espace ouvert sans poids sur l'orientation 60
Figure 6.13 Déplacement dans un espace ouvert avec poids sur l'orientation 61
Figure 6.14 Vitesses dans un espace ouvert sans poids sur l'orientation 61
Figure 6.15 Vitesse dans un espace ouvert avec un poids sur l'orientation 62
Figure 6.16 Trajet mur à mur 63

Figure 6.17 Déplacement entre deux murs sans poids sur l'orientation.....63
Figure 6.18 Déplacement entre deux murs avec un poids sur l'orientation 64
Figure 6.19 Vitesses entre deux murs sans poids sur l'orientation.....64
Figure 6.20 Vitesses entre deux murs avec un poids sur l'orientation 65
Figure 6.21 Trajet entre deux cavités.....66
Figure 6.22 Collision avec la péninsule 66
Figure 6.23 Déplacement entre deux cavités avec un poids sur l'orientation 67
Figure 6.24 Vitesses entre deux cavités avec un poids sur l'orientation 67
Figure 6.25 Trajet dans un espace restreint.....68
Figure 6.26 Insertion série 9 avec **Yaw** = 0 69
Figure 6.27 Insertion série 9 avec **Yaw** = 6 69
Figure 6.28 Insertion série 4 avec **Yaw** = 6 69
Figure 6.29 Blocage lors de la sortie avec la série 4 70
Figure 6.30 Collision lors de la sortie avec la série 9 sans poids sur l'orientation (**Yaw** = 0) 70
Figure 6.31 Déplacement hors du stationnement.....71
Figure 6.32 Vitesses lors de la sortie du stationnement 71
Figure 6.33 Démonstration de l'orientation spécifiée : a) orientation originale, tangente au déplacement; b) orientation spécifiée par l'utilisateur 74
Figure 6.34 Démonstration des zones de circulation préférées : a) trajet original, qui coupe le coin; b) zone de circulation préférée spécifiée par l'utilisateur 74
Figure 6.35 Démonstration des zones de circulation interdites : a) Trajet original, plus court; b) trajet plus long, qui contourne la zone interdite par l'utilisateur 75

LISTE DES TABLEAUX

Tableau 1 Profilage du planificateur 34
Tableau 2 Profilage du graphe 34
Tableau 3 Profilage du graphe avec la méthode des quatre coins 36
Tableau 4 Comportements de l'AGV VII avec diverses combinaisons de poids 50
Tableau 5 Poids utilisés lors des tests avec l'AGV VII 52
Tableau 6 Résumé des tests en conditions réelles 56
Tableau 7 Poids utilisés lors des tests en simulation 59
Tableau 8 Vitesses et rotation lors des scénarios simulés 72

CHAPITRE 1

INTRODUCTION

Un véhicule à guidage automatique (*Automated Guided Vehicule* – AGV) est un engin relativement simple et de plus en plus répandu [1] qui permet d'acheminer des biens d'un endroit à un autre dans un environnement industriel. Puisque les tâches de manutention ne constituent pas une valeur ajoutée pour la production, il est avantageux de libérer les ressources humaines associées à ces tâches à l'aide d'AGV. Selon Fedorko et coll. [2], sur un AGV traditionnel, le système de guidage est dépendant de l'infrastructure présente dans l'environnement. Ainsi, il est nécessaire d'installer un câble guide, un ruban magnétique ou tout autre marqueur afin de tracer les routes que les AGV suivront. Cette installation initiale entraîne des coûts importants et réduit grandement la possibilité d'altérer les tracés pour s'adapter aux changements dans la production. Ainsi, le coût d'installation élevé d'un système d'AGV traditionnel empêche plusieurs entreprises de s'en procurer [1].

Parallèlement, les récentes avancées dans le domaine de la navigation autonome de robots mobiles ont permis l'apparition d'AGV dits à parcours libre. Ce nouveau type d'AGV n'est plus dépendant d'une infrastructure présente dans l'environnement pour se diriger [3]. Le véhicule génère plutôt un trajet approprié à la réception d'une destination puis se dirige à l'aide de ses capteurs. Cela réduit le coût d'installation puisqu'il n'est plus nécessaire d'aménager un réseau de guides. Aussi, il est facile de modifier l'aménagement de la ligne de production, puisque les routes d'AGV ne sont plus fixes.

Récemment, des AGV holonomes sont apparus sur le marché, comme le *KUKA Mobile Platform 1500*[1]. Ces nouveaux AGV peuvent se déplacer de manière omnidirectionnelle,

[1] https://www.kuka.com/fr-ca/produits-et-prestations/mobilité/plateformes-mobiles/kmp-1500

c'est-à-dire latéralement sans changer leur orientation. Il existe différentes manières pour concevoir des plateformes omnidirectionnelles : avec des roues orientables (ce qui peut rendre la plateforme non holonome) [4] et un planificateur de déplacement permet de tenir compte des contraintes de la plateforme [5]; ou avec des roues omnidirectionnelles [6] (ce qui rend la plateforme holonome) et le planificateur peut exploiter le degré de liberté supplémentaire pour optimiser le déplacement, par exemple en limitant la consommation d'énergie [7]. La grande agilité conférée par cette nouvelle capacité rend ce type de véhicule particulièrement attrayant pour l'ajout à une chaine de production existante. Inogec inc. s'est lancée dans ce marché avec la conception de l'AGV VII[2] (véhicule industriel intelligent) montré à la figure 1.1. Pour lui donner cette intelligence, le véhicule doit pouvoir planifier ses trajectoires de manière à optimiser ses déplacements en exploitant son omnidirectionalité. L'objet de la recherche présentée dans ce mémoire est de développer un système de navigation pour l'AGV VII d'Inogec.

Figure 1.1 L'AGV VII d'Inogec

[2] https://www.inogec.com/agv

La navigation autonome des robots mobiles holonome est un sujet bien maîtrisé. Des logiciels efficaces existent dans l'environnement ROS[3]. Toutefois, l'utilisation directe de ces logiciels sur les AGV holonomes ne conduit pas à un comportement acceptable du véhicule. En effet, puisque le véhicule n'est plus contraint de s'orienter dans la direction de son déplacement, comme pour une plateforme différentielle ou d'Ackermann, son orientation varie énormément lors de ses déplacements. Comme le montre la figure 1.2, à proximité d'un obstacle l'AGV s'oriente pour éviter les collisions, mais dans un espace ouvert il tend à osciller d'un côté à l'autre de manière aléatoire. Ce comportement erratique peut être acceptable pour un robot mobile de petite taille, mais il s'avère dangereux pour un AGV massif qui porte une lourde charge.

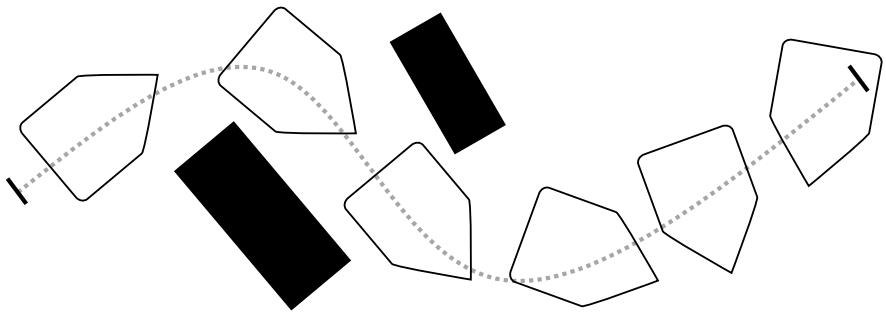

Figure 1.2 Orientation d'un AGV en mouvement (les rectangles noirs représentent des obstacles)

Le comportement aléatoire et un peu saccadé est également une source de stress pour les gens présents dans l'environnement. Afin d'augmenter la confiance et l'efficacité des travailleurs à proximité, l'AGV doit se déplacer d'un mouvement continu le plus rectiligne possible. Les logiciels de navigation existants dans ROS ne permettent pas non plus de tenir compte des consignes de circulation d'un environnement de production, comme des corridors de circulation à privilégier, une orientation à adopter ou encore une limite de vitesse différente dans certaines zones de son environnement d'opération.

[3] http://wiki.ros.org/navigation

Ainsi, la recherche proposée vise à déterminer si la pile de navigation ROS, dénommée ROS-NH (ROS non-holonome), peut être optimisée pour commander l'orientation d'un AGV holonome à parcours libre tout en se conformant aux règles établies dans une entreprise. Pour répondre à cette question, un planificateur de trajectoire optimisé pour les AGV holonomes est développé et implémenté dans l'environnement ROS. Les étapes suivantes doivent être franchies pour atteindre cet objectif :

1. Un algorithme et son implémentation doivent être développés pour trouver le trajet devant être suivi par l'AGV entre son point de départ et son but. Ce trajet doit être composé d'une série de couples (x, y, θ), où x et y décrivent la position horizontale du véhicule dans l'environnement et θ son orientation, assurant un mouvement fluide et une distance sécuritaire des obstacles;
2. La pile de navigation ROS-NH devra être adaptée pour que l'AGV suive le trajet déterminé et respecte les règles de circulation. Le logiciel développé doit se greffer à la structure existante et ne pas affecter le bon fonctionnement de la pile sur un robot non holonome. Le temps de calcul ajouté doit également être suffisamment faible pour que la fréquence de mise à jour de la vitesse commandée soit maintenue, puisqu'un ralentissement réduirait les performances de l'AGV;
3. Le planificateur développé doit prouver son efficacité en simulation et sur l'AGV réel. Une réduction des oscillations et le respect des règles de circulation doivent être démontrés. Le logiciel doit également permettre le calcul du trajet dans un temps suffisamment court pour permettre l'opération de l'AGV dans un environnement de production;
4. Les règles de circulation s'appliquant sur le lieu d'opération de l'AGV doivent être traduites dans une représentation pouvant être comprise par le logiciel de navigation. De là, l'orientation à adopter pendant le trajet de l'AGV doit être déterminée. Cette orientation doit permettre un déplacement efficace et prévisible par les travailleurs qui se trouvent à proximité du véhicule.

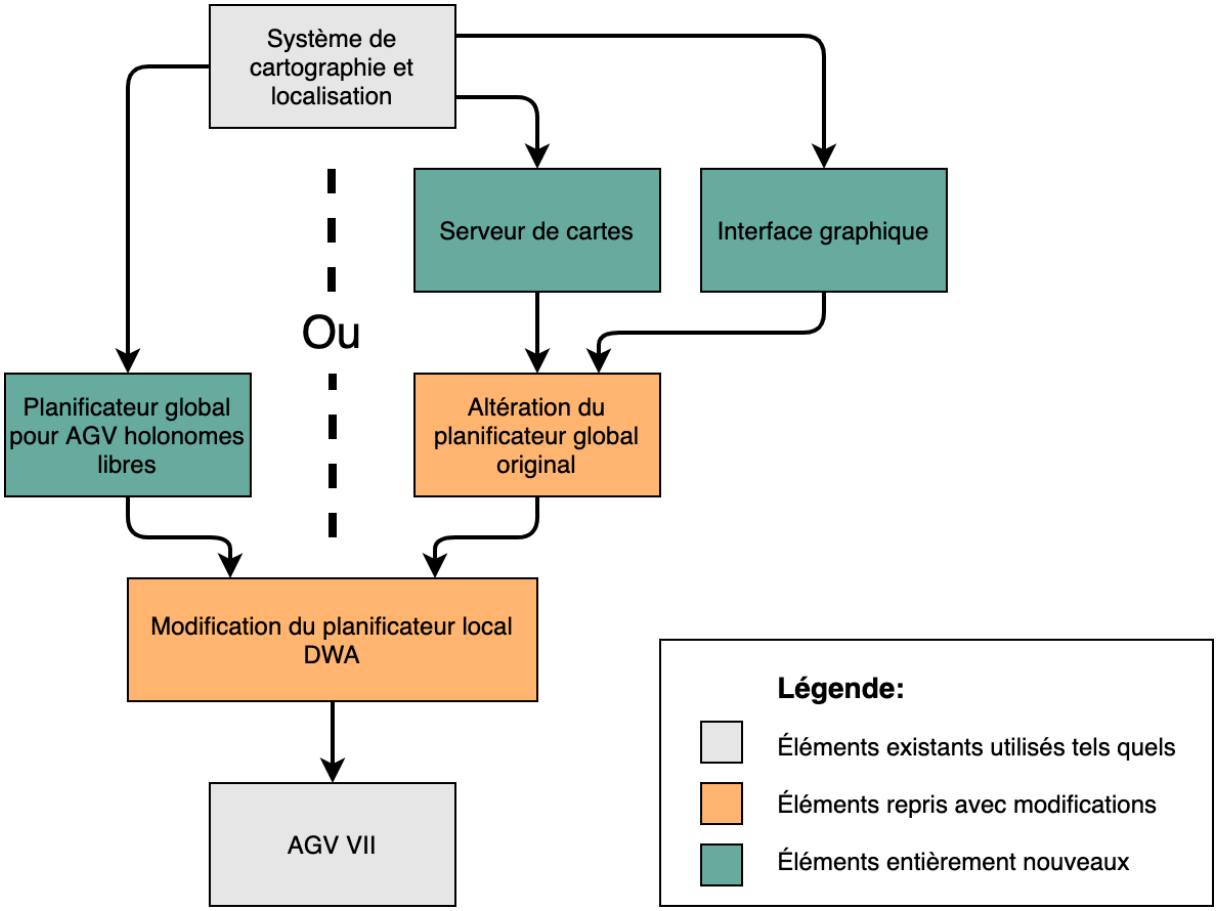

Figure 1.3 Pile de navigation ROS-H

La figure 1.3 présente la solution conçue, soit une pile de navigation dérivée de la pile de navigation ROS-NH, adaptée aux AGV holonomes à parcours libre, nommée ROS-H (ROS holonome). La portion à gauche de la ligne pointillée représente le planificateur global de trajets pour AGV holonomes. La portion à droite de la ligne pointillée représente la preuve de concept de la gestion des règles de circulation. Un seul de ces logiciels peut être démarré à la fois au lancement du véhicule. Il y a trois contributions originales :

- Un planificateur global de trajets pour AGV holonomes libres qui considère la position et l'orientation du véhicule est conçu pour être utilisé avec la pile de navigation ROS-NH. Il est livré dans un paquet ROS et se présente comme un plugin qui peut être spécifié lors de la configuration de la pile de navigation ROS-NH;

- Un planificateur local de trajectoire DWA de la pile de navigation ROS-NH est modifié pour considérer les orientations contenues dans le trajet, car le planificateur d'origine considère seulement les positions. Ces modifications sont disponibles comme un embranchement du dépôt Git de la pile de navigation ROS-NH et remplace le paquet DWA original;
- Une preuve de concept de l'utilisation des grilles d'occupation ROS pour l'encodage des règles d'opération sur le lieu d'opération est livrée comme un paquet ROS. La preuve de concept comprend un serveur de cartes qui s'exécute sur le robot, une interface graphique qui s'exécute sur l'ordinateur de l'opérateur et une altération du planificateur global de trajets de la pile de navigation ROS-NH afin de respecter les consignes fournies par le serveur de cartes.

Ce mémoire est organisé de la manière suivante. Le chapitre 2 présente une revue de la littérature sur le déplacement des robots mobiles holonomes. Le chapitre 3 couvre le développement du planificateur global, du développement des algorithmes à l'implémentation du logiciel. Adoptant la même structure que le chapitre 3, le chapitre 4 porte plutôt sur le planificateur local. Le chapitre 5 présente la preuve de concept de la gestion des règles de circulation. Les tests ainsi que leurs résultats sont présentés au chapitre 6. Enfin, le chapitre 7 conclut le document.

CHAPITRE 2

DÉPLACEMENT DES ROBOTS HOLONOMES

La robotique mobile est un domaine de recherche et d'ingénierie apparue vers la fin des années 60 [8]. Pour la première fois, les robots n'étaient plus ancrés au sol, mais étaient plutôt libres de se déplacer dans leur environnement d'opération. En 1969, Nils J. Nilson de l'Institut de recherche de Stanford définit les grands défis qui devront être relevés pour la réalisation de robots mobiles efficaces [9] : la perception de l'environnement, la cartographie, la planification des mouvements et l'architecture de contrôle qui rend ces capacités possibles. Encore aujourd'hui, ces défis occupent les chercheurs en robotique.

Ce chapitre aborde, dans l'ordre, les quatre piliers du déplacement d'un robot holonome dans son environnement [8] : 1) la base robotique et le développement de sa cinématique directe et inverse; 2) la modélisation de l'environnement et la localisation du robot dans ledit environnement afin de planifier un trajet; 3) la planification du trajet global qui permet de déterminer, à partir de la carte, le trajet qui permet à un robot situé à un emplacement initial de se rendre à une destination autre; 4) la planification de la trajectoire locale indiquant une vitesse à suivre au robot. Il se conclut avec l'intégration des enjeux humains à la navigation des robots mobiles.

Une architecture logicielle commune doit toutefois être capable d'interconnecter ces quatre éléments. Dans le monde de la robotique, l'environnement de développement ROS (*Robot Operating System*) [10] est souvent utilisé pour jouer ce rôle. Il s'agit d'un intergiciel distribué très polyvalent basé sur le concept de nœuds reliés entre eux par des messages où transitent des données. Dans ROS, la pile de navigation ROS-NH[4] permet la navigation

[4] http://wiki.ros.org/navigation

efficace d'un robot mobile. Elle est basée sur les travaux de Marder-Eppstein et coll. [11]. Ces travaux présentent un système de navigation implémenté dans ROS qui prouve son efficacité dans un environnement de bureau encombré. La pile de navigation ROS-NH se décompose en trois étages:

1. Le module de cartographie et de localisation simultanée (*Simultaneous Localization And* Mapping - SLAM) crée une carte de l'environnement du robot à partir des mesures des capteurs et détermine la position du robot dans la carte;
2. Le planificateur global trouve un trajet formé d'une série de points de cheminement entre un point de départ et un point d'arrivée dans la carte;
3. Le planificateur local calcule les vitesses que le robot doit adopter afin de parcourir le trajet calculé par le planificateur global tout en évitant les obstacles rencontrés.

La pile de navigation ROS-NH est bien adaptée à la recherche puisqu'il est facile d'interchanger les modules ou de réutiliser des modules existants étant donné que des interfaces connues pour chaque étages soient imposées. Bien que conçue pour les environnements institutionnels et le milieu de la recherche, la pile de navigation ROS-NH a su démontrer une bonne performance en milieu industriel. Par exemple, Unhelkar et coll. [12] l'utilisent avec succès pour la navigation d'un robot holonome sur une chaîne de montage. Les éléments de ROS pertinents à ce projet sont abordés à même les sections de ce chapitre.

2.1 Base robotique holonome

Une base mobile capable de déplacements holonomes est requise pour développer un AGV holonome à parcours libre. Il est également nécessaire de développer la cinématique directe et inverse associée à la base mobile afin de la contrôler correctement. La cinématique inverse permet de commander les moteurs des roues afin que le robot se déplace à une vitesse commandée. La cinématique directe permet quant à elle de calculer l'odométrie, soit une estimation de la position du robot par rapport à son point de départ, à partir des mesures de

rotation des roues [13]. Il s'agit toutefois de questions bien étudiées qui ne font pas l'objet de la recherche proposée, le point de départ étant une base robotique opérationnelle acceptant des commandes de vitesse et fournissant une odométrie de bonne qualité.

Il existe plusieurs bases robotiques holonomes avec leur cinématique directe et inverse. Doroftei et coll. [14] ont conçu un robot mobile holonome destiné à un usage éducationnel. Le robot est compact et utilise des roues omnidirectionnelles de type Mecanum montées sur une suspension passive. La cinématique directe est une simple addition des forces exercées par les roues selon leur vitesse de rotation et les dimensions du châssis exprimées sous la forme d'une multiplication matricielle. La cinématique inverse est simplement l'inversion de la multiplication matricielle de la cinématique directe. Cette approche a toutefois ses limites puisqu'elle ne tient pas compte du glissement important relié aux roues Mecanum. L'odométrie est donc fournie par une caméra pointée vers le sol qui calcule le déplacement et la vitesse du robot en déterminant la transformation entre les images captées consécutives. Liu et coll. [15] ajoutent la dynamique du robot et des actionneurs au modèle d'addition des formes afin de développer un contrôleur qui tient compte de la réponse des moteurs utilisés ainsi que de l'inertie du robot pour suivre une trajectoire de manière plus précise.

Des travaux ont également été menés sur des robots plus volumineux. Xie et coll. [16] présentent une plateforme mobile dotée de roues Mecanum pour la recherche à grande échelle. L'objectif est de pouvoir étudier les performances de ce type de roues pour des applications industrielles. En effet, le glissement et les vibrations peuvent devenir problématiques lorsque la taille du robot augmente. Xie et coll. utilisent toutefois le même type de contrôleur que celui employé pour les robots plus petits.

Gmerek et coll. [17] optent plutôt pour des roues montées sur pivot afin d'obtenir un robot pouvant opérer sur une surface qui n'est pas nécessairement propre, comme un chantier de construction. Les équations de cinématiques sont donc différentes puisque l'algorithme doit

fournir une vitesse et une orientation pour chaque roue. Le glissement plus faible permet toutefois d'obtenir une meilleure odométrie à partir des encodeurs de roue.

Il est à noter que tous ces robots utilisent ROS.

2.2 Cartographie et localisation

La modélisation de l'environnement forme une carte de l'environnement à partir des capteurs du robot lorsqu'il se déplace. La localisation détermine la position du robot dans la carte. Puisque la position doit être connue pour former la carte à partir des mesures, mais que la carte est requise pour connaitre la position, ces deux aspects sont combinés pour former le SLAM [18]. Les deux approches largement répandues pour le SLAM sont la combinaison probabiliste de lectures de LiDAR [19] et la reconstruction basée sur une série d'images d'une caméra [20]. Toutefois, les approches visuelles ne produisent pas la grille d'occupation nécessaire au fonctionnement de la pile de navigation ROS-NH [21] et ne sont donc pas considérées ici.

GMapping [22] est le logiciel de SLAM par défaut de ROS. Il est basé sur un filtre particulaire qui utilise l'odométrie pour calculer les échantillons prédits. L'approche est efficace si la variance de l'odométrie indiquée correspond à la réalité et si le nombre d'échantillons est suffisant. GMapping n'est toutefois pas en mesure de fournir la position du robot dans la carte. AMCL [23], basé sur l'approche de Monte-Carlo, est donc souvent utilisé en complément pour la portion localisation. AMCL permet également de fusionner la position obtenue par la localisation dans la carte avec l'odométrie fournie par les autres capteurs d'un robot [24].

Hector SLAM [25] est une alternative qui permet la génération de cartes en 3D avec un très haut taux de mise à jour. L'approche exploite le fait que les LiDARs modernes ont un taux de rafraîchissement élevé et un bruit de mesure faible pour aligner l'extrémité des rayons avec la portion de carte existante. La prémisse est que le LiDAR est moins bruité que

l'odométrie, cette dernière n'est donc pas nécessaire et peut être omise. Hector SLAM ne détecte toutefois pas les fermetures de boucles, ce qui rend l'approche moins appropriée pour la cartographie de grands espaces avec de multiples trajets communicants comme dans un entrepôt.

Google Cartographer [26] est un logiciel de SLAM conçu à l'origine pour la mesure de bâtiments à l'aide d'un LiDAR monté sur un sac à dos. Il a depuis été utilisé sur des robots mobiles. Google Cartographer utilise une approche semblable à Hector SLAM où les rayons sont alignés pour former une carte. Toutefois, Google Cartographer crée des sous-cartes qui sont ensuite reliées par des contraintes lors des détections de boucles. Ainsi, le logiciel demeure rapide tout en permettant l'optimisation de la carte. L'odométrie peut également être utilisée si elle est disponible sur la plateforme utilisée.

RTAB-Map [21], intégré dans ROS, fournit la grille d'occupation et les transformations entre les repères nécessaires au fonctionne de la pile de navigation ROS-NH. Pendant la cartographie, RTAB-Map crée un nouveau nœud à intervalle régulier qui contient la lecture du LiDAR et/ou l'image de la caméra à cet endroit selon la configuration choisie. Les nœuds sont reliés par des liens basés sur l'odométrie reçue. Lorsque des caractéristiques communes à deux nœuds sont identifiées, un lien de proximité est créé et la carte est optimisée. RTAB-Map se démarque des autres logiciels de SLAM par son utilisation judicieuse de la mémoire, ce qui permet de cartographier une très grande zone sans que la recherche des correspondances ralentisse où que l'utilisation de la mémoire devienne trop élevée. En effet, les nœuds sont échangés entre une mémoire de travail et un stockage à long terme afin de ne garder que les nœuds utiles dans l'immédiat en mémoire.

2.3 Planification globale

Le planificateur global opère dans le référentiel de la carte. Puisque la position du robot dans ce référentiel peut changer abruptement lorsque la localisation du module SLAM converge, le plan global doit être mis à jour périodiquement et ne doit pas être utilisé directement pour

calculer les commandes de vitesse à envoyer au robot. Généralement, les contraintes cinématiques du robot ne sont pas prises en compte à cette étape afin de simplifier le problème d'optimisation. Les mesures des capteurs, qui ont généralement une portée limitée, ne sont pas considérées non plus.

Le planificateur global de la pile de navigation ROS-NH est basé sur un algorithme de recherche de type A*. Dans [27] et [28], les auteurs présentent des robots mobiles efficaces qui utilisent cet algorithme. Puisque le planificateur ne génère qu'une série de coordonnées (x, y), le robot est approximé par un point et les obstacles de la carte sont gonflés pour correspondre au rayon du cercle dans lequel le périmètre du robot peut être inscrit. Il ne fournit donc pas l'orientation à adopter le long du trajet.

Les planificateurs basés sur A* sont toutefois lents puisqu'ils doivent traverser toutes les cases de la grille d'occupation. Les diagrammes de Voronoi [29] permettent d'accélérer la recherche. En effet, en suivant les contours des régions d'un diagramme de Voronoi, la trajectoire emprunte naturellement le trajet le plus éloigné des obstacles. Par exemple, le planificateur présenté par Sprunk et coll. [30] utilisent un diagramme de Voronoi pour produire un trajet sécuritaire par défaut, puisqu'il est le trajet le plus éloigné des obstacles, puis l'optimise en le réduisant au besoin. En cas d'échec de la planification dans le diagramme de Voronoi, un planificateur basé sur A* et opérant directement dans la grille d'occupation prend le relai.

Dakulovic et coll. [31] s'intéressent à la planification globale de trajectoires. Ils ajoutent l'orientation du robot dans le graphe de recherche du trajet et considèrent une empreinte au sol asymétrique plutôt qu'un point pour détecter la collision avec un obstacle. À l'aide de l'algorithme de recherche D* [32], il parvient à générer des trajets comportant l'orientation du robot qui traverse des passages plus étroits que la plus grande dimension du robot.

2.4 Planification locale

Le planificateur local reçoit une portion réduite du trajet global et produit une trajectoire, soit une suite de vitesses à suivre qui permet au robot de se rapprocher au maximum du trajet global. Le robot s'en écarte toutefois puisque le planificateur local tient compte des contraintes cinématiques du robot ainsi que des mesures des capteurs. La trajectoire locale peut donc s'écarter du trajet prévu initialement pour éviter des obstacles non cartographiés ou bien adoucir des virages trop serrés.

Le planificateur par défaut de la pile de navigation ROS-NH implémente l'approche par fenêtre dynamique (*Dynamic Window Approach* - DWA). C'est un algorithme simple qui trouve la vitesse instantanée optimale à commander au robot [33]. DWA est configuré avec la plage de vitesses en translation et en rotation admissible pour la base robotique à contrôler. Ces plages sont ensuite échantillonnées pour obtenir des combinaisons de vitesses possibles. À chaque exécution, le déplacement du robot dans les prochaines secondes est simulé pour chaque combinaison de vitesses. Les simulations sont ensuite comparées à l'aide de fonctions de coût. Les coûts utilisés par le planificateur DWA classique sont [34] :

- L'angle entre l'orientation du robot à la fin de la trajectoire simulée et l'orientation directe vers le but soustrait de 180°;
- La distance avec l'obstacle le plus près le long de la trajectoire simulée. Si aucun obstacle n'intercepte la trajectoire, cette fonction est fixée à une grande valeur;
- La vitesse choisie utilisée directement.

Il est pertinent de noter qu'il est facile de modifier les fonctions de coût ou d'en ajouter de nouvelles pour changer le comportement du robot. La combinaison de vitesses permettant d'obtenir la trajectoire avec le coût le plus élevé est sélectionnée et envoyée à la base robotique. Ainsi, le robot tend à se diriger vers sa cible, à la plus grande vitesse possible tout en s'éloignant des obstacles.

L'approche par bande élastique, en anglais *Elastic Band* (EB), [35] est une alternative intéressante aux algorithmes d'échantillonnage comme DWA. Le principe est de considérer la portion du trajet global près du robot comme un élastique. Une force de contraction interne tente de réduire la longueur du trajet au minimum. Cette force est équilibrée par la force de répulsion des obstacles. Le trajet final est donc une trajectoire lisse qui évite les collisions, les virages serrés et les détours inutiles. La bande élastique permet notamment d'éviter des obstacles absents de la carte qui sont détectés par les capteurs du robot en déformant le trajet reçu du planificateur global.

La bande élastique n'a toutefois pas de dimension temporelle : la vitesse du robot le long de la trajectoire n'est pas déterminée. Ainsi, la forme de la trajectoire conduit au trajet le plus court, qui n'est toutefois pas le plus rapide. C'est pourquoi Rösmann et coll. [36] ajoutent le temps à la bande élastique pour créer la bande élastique temporisée, en anglais *Timed Elastic Band* (TEB). En évaluant la vitesse du robot le long de la bande élastique, cet ajout permet de déformer la trajectoire pour réduire les accélérations du robot, réduisant ainsi le temps de trajet. Le résultat est une trajectoire qui s'exécute le plus rapidement possible tout en respectant les limites cinématiques du robot et de la charge utile transportée.

Les bandes élastiques ont toutefois une limite importante lorsqu'elles sont confrontées à des obstacles dynamiques. Repoussées par un obstacle qui traverse la trajectoire, elles vont demeurer du même côté de l'obstacle, ce qui peut allonger considérablement le trajet, voir mener à un échec de la planification. C'est pourquoi Rösmann et coll. [37] ajoutent plusieurs élastiques parallèles. Ceux-ci sont répartis de chaque côté des obstacles rencontrés. Le planificateur choisit ensuite l'élastique qui représente le trajet le plus rapide. Ainsi, le planificateur peut contourner correctement un obstacle mobile qui s'approche de la trajectoire initialement planifiée.

Sprunk et coll. [38] développent un générateur de trajectoires optimisé pour les robots holonomes non circulaires qui évoluent dans un milieu industriel. Le générateur réduit d'abord

le trajet fourni par le planificateur global à une série de points de cheminement avec l'algorithme de Douglas-Peucker [39], puis il crée une représentation compacte du trajet complet sous forme de splines en utilisant la technique présentée dans [40]. Les paramètres des splines sont fixés de manière à assurer la continuité du trajet, ce qui réduit le nombre de degrés de liberté.

Afin d'assurer une bonne performance avec un robot holonome, il n'est pas suffisant de simplement interpoler l'orientation entre les points de cheminements obtenus avec Douglas-Peucker. En effet, cela entraînerait une rotation constante du véhicule le long du trajet, alors que le planificateur global assume une orientation fixe le long des segments : des collisions pourraient survenir. Des points de contrôles en rotation sont donc ajoutés de part et d'autre de chaque point de cheminement. Ces points sont initialement placés très près des points de cheminement, forçant le robot à essentiellement pivoter en place, ce qui correspond au comportement prévu par le planificateur global.

Les paramètres libres des *splines* ainsi que le profil de vitesse le long du trajet sont ensuite optimisés itérativement afin de permettre le déplacement le plus rapide possible, tout en demeurant dans l'enveloppe de vitesse acceptable pour le véhicule telle que définie dans les paramètres du robot. Sprunk et coll. [30] mettent en œuvre leur méthode dans sur les plateformes holonomes KUKA omniRob et KUKA Moiros. Des essais en simulation, en laboratoire, puis une démonstration pendant une conférence représentant plusieurs kilomètres de navigation efficaces, démontrent l'efficacité de l'approche.

2.5 Interaction humaine

Les travaux présentés jusqu'à maintenant traitent seulement de l'AGV et les personnes qui l'entourent sont considérées comme de simples obstacles [41]. Toutefois, les AGV opèrent dans un espace partagé avec des humains et sont au service de ceux-ci. Il est donc essentiel de s'intéresser aux opérateurs et aux observateurs qui interagissent avec l'AGV. De plus,

certains milieux de travail comportent des consignent particulières pour le déplacement des humains et des véhicules qu'un AGV doit être en mesure de comprendre et de respecter.

Des chercheurs se sont penchés sur la performance et le confort d'une personne qui doit travailler avec un robot collaboratif [42]. Différents types de déplacements de l'effecteur ont été comparés quant à l'effet qu'ils ont sur un collaborateur humain. Les mouvements prévisibles et uniformes se sont révélés les plus agréables pour l'humain. Une revue de différentes études sur la lisibilité des trajectoires détermine trois facteurs contributifs principaux [43]: conserver un tracé rectiligne puisqu'un opérateur humain est davantage porté à ralentir pour éviter une collision plutôt qu'à éviter un piéton; toujours suivre la même trajectoire pour l'atteinte d'un but similaire; ajouter un geste complémentaire comme tourner le regard vers l'objectif.

Il existe une distinction entre la prévisibilité du mouvement d'un robot, qui est une mesure de la correspondance du mouvement aux attentes d'un observateur humain, et la lisibilité, qui exprime la capacité d'un observateur à prédire la finalité d'un mouvement [44]. Ces deux critères sont parfois contradictoires et un choix doit être fait, comme l'emploi d'une pile de navigation sociale qui s'adapte au contexte [45]: un module d'intelligence artificielle détermine l'interaction en cours, comme suivre une personne ou éviter une collision dans un couloir, et le planificateur local ajuste ses paramètres selon le contexte détecté.

Une autre solution est d'utiliser une interface permettant à un opérateur d'annoter une carte afin d'influencer le fonctionnement d'un planificateur de trajets [46]. L'objectif était d'orienter la mission de drones de reconnaissance opérant dans un environnement urbain dense. La carte est présentée sur un écran tactile et l'opérateur peut indiquer des rues à emprunter et à éviter, ainsi que des zones ayant un coût plus ou moins faible pour forcer le planificateur de trajet à produire un tracé qui répond aux besoins de la mission en cours, par exemple, éviter une zone à risque ou circuler dans une rue particulièrement intéressante.

CHAPITRE 3

PLANIFICATEUR GLOBAL

Pour bien saisir le processus de développement présenté, il importe de rappeler la finalité du projet. Le logiciel de navigation produit est destiné à un véhicule industriel beaucoup plus gros et lourd que les robots mobiles typiquement utilisés avec la pile de navigation ROS-NH. Néanmoins, comme présenté au chapitre 2, malgré les lacunes de la pile au niveau du comportement du véhicule, elle présente de nombreux avantages pour le développement. L'utilisation des interfaces et du mode d'opération ROS est donc un enjeu tout au long du projet. Il est aussi souhaitable de réutiliser un maximum de logiciel existant dans ROS.

Inogec a également posé des balises au développement tirées de leur expérience préalable avec la mise en œuvre de la pile de navigation ROS-NH sur leur AGV. Des commentaires de clients potentiels complètent ces balises. Ainsi :

- Les oscillations et les mouvements brusques doivent être absolument évités pour protéger la charge, le véhicule et les gens à proximité;
- Le véhicule est holonome et souvent symétrique, toutefois, la charge peut requérir une orientation définie à l'arrivée;
- Il est préférable de circuler vers l'avant ou l'arrière plutôt que latéralement afin de minimiser l'usure des roues et la dépense énergétique;
- Contrairement à un robot mobile compact, il est préférable de demeurer le plus près possible du trajet précalculé et d'attendre si un obstacle imprévu est détecté plutôt que de tenter une manœuvre de contournement.

Comme exposé au chapitre 2, utiliser l'orientation comme degré de liberté supplémentaire pour se faufiler dans des espaces restreints est très avantageux pour le déploiement d'un système d'AGV. Toutefois, les logiciels actuellement disponibles dans l'environnement ROS

utilisent des approximations qui ne permettent pas de l'utiliser : le robot est approximé par un cercle au moment de planifier le trajet. Il faut alors choisir une des approches suivantes : utiliser un cercle correspondant au rayon circonscrit et planifier des trajets conservateurs toujours traversables qui ne peuvent pas emprunter des passages où l'orientation est importante; utiliser un cercle correspondant au rayon inscrit et planifier des trajets optimistes, parfois impossibles, qui assument que le véhicule présentera toujours son côté étroit. Il revient ensuite au planificateur local, qui n'a pas la vue d'ensemble, de décider de l'orientation. Le résultat est un véhicule qui reste souvent coincé, qui n'est pas très agile ou qui pivote erratiquement en l'absence de contraintes cinématiques ou environnementales.

Rappelons également que l'algorithme A*, à la base du planificateur de trajets de la pile de navigation ROS-NH, dépend d'une représentation du monde sous la forme d'un graphe. Cela implique de discrétiser l'environnement. Dans ROS, la discrétisation en (x, y) découle naturellement de la grille d'occupation générée par le système SLAM. Il pourrait donc sembler naturel de simplement ajouter une troisième dimension pour former des couples (x, y, θ). Toutefois, comme le soulignent Gammel et coll. [47], cela augmente le nombre de nœuds dans le graphe au point ou l'algorithme devient trop lourd. Gammel et coll. proposent d'utiliser un algorithme qui découle de RRT [48] afin de ne pas avoir à discrétiser l'environnement. Toutefois, cette approche s'éloigne de l'intention de demeurer proche de la pile de navigation puisqu'il s'agit d'un algorithme complètement différent. Le comportement pour des robots non holonomes s'en trouverait modifié.

Une solution alternative est présentée par Dakulovic et coll. [31]. Ils proposent de discrétiser l'orientation en intervalles libres, où un intervalle représente une plage d'orientations admissibles, plutôt que de créer un nœud par pas de discrétisation d'angle. C'est l'approche qui est retenue et qui est présentée à la section 3.1.1. La section 3.2 présente ensuite l'implémentation de l'algorithme A* réalisée qui utilise le graphe et ses intervalles

d'orientation. Enfin, la section 3.3 présente des améliorations apportées à l'implémentation originale après que des problèmes de temps de calcul aient été rencontrés lors des tests.

3.1 Discrétisation de l'orientation en intervalles libres

D'abord, établissons les requis d'un graphe utilisé par A*. Ce graphe doit discrétiser l'environnement d'opération de l'AGV sous la forme de nœuds et d'arcs qui peuvent être parcourus par l'algorithme. En plus, le graphe doit fournir certaines informations à l'algorithme afin de permettre son exécution, dont :

- Une liste des nœuds pouvant être rejoints à partir d'un nœud donné;
- Le coût d'un déplacement entre un nœud et un nœud adjacent;
- Le coût pour traverser un nœud;
- Une heuristique entre deux nœuds, qu'ils soient adjacents ou non, sans calculer le trajet entre ces nœuds;
- L'équivalence entre deux nœuds.

3.1.1 Nœuds

Il est pertinent de présenter les nœuds utilisés par l'implémentation originale du planificateur global de la pile de navigation ROS-NH. De la grille d'occupation SLAM, chaque nœud représente une case de la grille, et donc une position (x, y). Le coût c_{noeud} pour traverser le nœud est la valeur de la case de la grille d'occupation qui lui est associée. Ainsi, le coût pour traverser un nœud augmente au fur et à mesure que la distance avec les obstacles diminue.

Afin de tenir compte de l'orientation, un changement de paradigme est appliqué. Plutôt que d'avoir un nœud par case de la grille d'occupation, il peut maintenant y en avoir plusieurs. Chacun de ces nœuds possède la même position (x, y), mais contient également un intervalle d'orientations admissibles. Cet intervalle est unique et contigu. Un robot dont le centre se trouve à la position du nœud peut adopter n'importe quelle orientation, pourvu qu'elle se

trouve dans l'intervalle. Si plusieurs intervalles existent pour une même position, plusieurs nœuds y sont associés. Cette situation est représentée à la figure 3.1. Le triangle vert montre l'intervalle d'orientation lorsque l'avant de l'AGV pointe vers la droite et le triangle orangé l'intervalle avec l'avant vers la gauche. Il y aurait donc deux nœuds associés à la position (x, y) centrale de l'AGV. Étant donné la symétrie d'un AGV rectangulaire, il en résulte généralement un ou deux nœuds par case de la grille : toutes les orientations sont admissibles et le véhicule peut être orienté côté étroit devant ou derrière. Cela rend la recherche dans le graphe avec l'algorithme A* tout à fait réaliste.

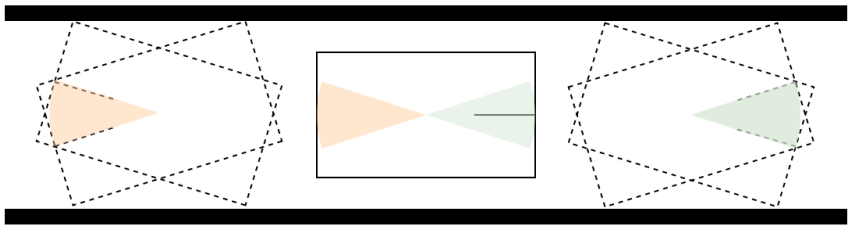

Figure 3.1 Intervalles d'orientation

Il faut également être en mesure de déterminer si deux nœuds sont équivalents. Cela est essentiel afin de reconnaître que le robot est arrivé à destination lorsque la destination se retrouve parmi les voisins du nœud courant. En termes de position, il est intuitif et approprié de considérer que deux nœuds sont équivalents si leurs positions sont les mêmes. En termes d'orientation, la notion d'intervalles rend toutefois ce genre de comparaison impossible. Il faut plutôt qu'une superposition existe entre les intervalles, c'est-à-dire qu'il doit exister une orientation que le robot peut adopter qui fait en sorte qu'il ne soit en collision dans aucun des deux nœuds.

3.1.2 Graphe de nœuds

Un graphe à nœud unique ne serait pas très utile, sauf si le but à atteindre se trouve exactement au point de départ. Il est donc essentiel d'être en mesure de trouver les nœuds voisins à un nœud donné afin de voyager de nœud en nœud jusqu'à atteindre le but. Toutefois,

avant d'aborder la recherche des voisins, il est pertinent de définir trois types de position, puisque la recherche des nœuds voisins débute par la recherche des positions voisines :

- Une position libre est une position où toutes les orientations sont libres. Elle se reconnait à son coût nul dans la grille d'occupation. Un seul nœud y est associé dont l'intervalle d'orientation est une plage de 360 degrés;
- Une position inscrite est une position où aucune orientation n'est libre. Elle s'identifie par son coût d'obstacle inscrit ou occupé dans la grille d'occupation. L'obstacle le plus proche est à une distance inférieure au rayon du cercle inscrit dans l'empreinte au sol du robot. Aucun nœud n'y est associé puisqu'un robot dont le centre se trouve à cette position est toujours en collision;
- Une position circonscrite est une position où certaines orientations peuvent être libres. Son coût est non-nul mais inférieur au coût d'obstacle inscrit dans la grille d'occupation. L'obstacle le plus proche peut être à une distance inférieure au rayon du cercle qui circonscrit l'empreinte au sol du robot. Si c'est le cas, il existe des orientations où le robot est en collision. Toutefois, le coût peut représenter une inflation pour s'éloigner des obstacles, alors il est possible que toutes les orientations soient libres. Plusieurs nœuds peuvent y être associés.

L'algorithme 1 permet de trouver les nœuds associés à une position. La première étape de recherche est de récupérer le coût de la position dans la grille d'occupation. Si le coût est nul, la recherche est terminée puisque qu'un seul nœud existe à cette position. De même, si le coût est inscrit ou occupé, aucun nœud n'existe. Dans les autres cas, l'algorithme vérifie la collision de l'empreinte avec un obstacle dans toutes les orientations possibles. Un intervalle est ouvert lorsque l'empreinte est dans une orientation où elle n'est pas en collision. L'intervalle est refermé à la prochaine orientation où l'empreinte est en collision et le nœud associé à cet intervalle est ajouté à la liste.

Algorithme 1 Recherche des nœuds à une position donnée

```
paramètres
    Position dont les nœuds sont requis
    position : Position (x,y)

    Grille d'occupation fournie par ROS
    grille : Grille d'occupation

    Le rayon du cercle qui circonscrit l'empreinte au sol du robot
    rayon circonscrit : Nombre positif

sortie
    La liste des nœuds pour la position donnée
    nœuds : Liste<Nœuds>

début nœuds pour position
    coût : Nombre := grille -> coût(position)
    nœuds := nouvelle Liste

    Position libre : toutes les orientations (en degré) sont admissibles
    si coût == COÛT LIBRE faire
        nœud : Nœud := nouveau Nœud
        nœud -> début intervalle := 0
        nœud -> longueur intervalle := 359
        nœuds -> ajouter(nœud)

    Position incrite : il faut vérifier les collisions à différentes orientations
    sinon si coût < COÛT INSCRIT faire
        début intervalle : Nombre := 0
        était sécuritaire : Booléen := faux

        Un incrément doit déplacer n'importe quelle partie du robot d'une case de la grille
        au maximum
        rayon en cases : Nombre := rayon circonscrit / grille-> résolution
        nombre incréments : Nombre := ⌈2π * rayon en cases⌉
        incrément : Nombre := 359 / nombre incréments
```

```
        orientation : Nombre := 0
        tant que orientation < 360 faire
            en collision : Booléen := grille -> en collision(position, orientation)

            Début d'un nouvel intervalle
            si pas en collision && n'était pas sécuritaire faire
                début intervalle := orientation
                était sécuritaire := vrai

            Fin d'un intervalle
            sinon si en collision && était sécuritaire faire
                nœud : Nœud := nouveau Nœud
                nœud -> début intervalle := début intervalle
                nœud -> longueur intervalle := orientation - début intervalle
                nœuds -> ajouter(nœud)
                était sécuritaire : = faux
            fin si

            orientation := orientation + incrément
        fin tant que

        Le dernier intervalle est encore ouvert
        si était sécuritaire faire
            nœud : Nœud := nouveau Nœud
            nœud -> début intervalle := début intervalle
            nœud -> longueur intervalle := 359 - début intervalle
            nœuds -> ajouter(nœud)
        fin si
    fin si

    retourner nœuds
fin
```

Il est à noter que le coût de la position dans grille et la vérification de la collision utilise les interfaces standards de ROS et n'ont pas eu à être développée dans le cadre du projet.

Il est maintenant possible de formellement définir les critères pour que deux nœuds soient voisins:

- Leurs positions sont associées à deux cases adjacentes dans la grille d'occupation;

- Une superposition existe entre leurs intervalles d'orientation.

De ces critères découle l'algorithme 2 qui permet de trouver les nœuds voisins d'un nœud donné. La recherche débute avec la liste des huit déplacements autour de la position du nœud donné. Pour chaque déplacement, si le déplacement mène à une position contenue dans la grille d'occupation, les nœuds correspondant sont obtenus avec l'algorithme 1. Les nœuds voisins sont alors les nœuds, parmi ceux obtenus, dont l'intervalle d'orientation se superpose à celui du nœud de départ.

Algorithme 2 Recherche des nœuds à une position donnée

```
paramètres
    Nœud duquel les voisins sont à identifer
    noeud : Nœud

    Grille d'occupation fournie par ROS
    grille : Grille d'occupation

sortie
    La liste des voisins d'un nœud donné
    voisins : Liste<Nœuds>

début voisins d'un nœud
    voisins := nouvelle Liste
    position : Position := nœud -> position

    Les huit positions adjacentes à la position de départ sont vérifiées
    déplacements : Liste<Position> := {
            (1;0), (0;1), (-1;0), (0;-1), (1;1), (1;-1), (-1;1), (-1;-1), (-1;1)
    }
    pour chaque déplacement dans déplacements faire
        position décalée : Position := nouvelle Position
        position décalée -> x := position -> x + déplacement ->x
        position décalée -> y := position -> y + déplacement ->y

        Validation que le décalage ne sort pas de la grille d'occupation
        si grille -> contient(position décalée) faire
            potentiels : Liste<Nœud> := nœuds pour position(position décalée)
```

```
        pour chaque potentiel dans potentiels faire

            Calcul de la superposition des orientations
            superposition : Nombre
            si position -> début intervalle <= potentiel -> début intervalle faire
                superposition := position -> longueur intervalle - ...
                        (potentiel -> début intervalle - position -> début intervalle)
            sinon faire
                superposition := potentiel -> longueur intervalle - ...
                        (position -> début intervalle - potentiel -> début intervalle)
            fin si

            Le nœud potentiel est un voisin s'il y a superposition
            si superposition > 0 faire
                voisins -> ajouter(potentiel)
            fin si
        fin pour
    fin si
fin pour

retourner voisins
fin
```

3.1.3 Heuristique et coût de déplacement

Disposant maintenant d'un moyen de déterminer la liste des nœuds voisins à un nœud donné, il reste à choisir le prochain nœud à explorer. En effet, l'intérêt d'A* est de pouvoir prédire quel voisin fait partie du trajet optimal et ainsi sauver du temps de calcul en n'explorant pas les nœuds qui éloignent le robot de son but. Pour ce faire, une heuristique est calculée. Cette heuristique, dont la valeur diminue alors que les nœuds s'approchent du but, prend un nœud en entrée et fournit un estimé optimiste du coût pour voyager du nœud donné jusqu'au but. Un choix classique lors de la recherche d'un trajet dans une grille d'occupation est la distance octale. Il s'agit de faire la somme du déplacement sur les deux axes entre les deux positions, puis de soustraire les déplacements diagonaux possibles. Formellement, avec deux positions $p_1 = (x_1, y_1)$ et $p_2 = (x_2, y_2)$, les écarts entre les coordonnées sont :

$$\Delta x = |x_1 - x_2| \text{ et } \Delta y = |y_1 - y_2| \quad (1)$$

La distance octale est donnée par :

$$h(p_1, p_2) = D * (\Delta x + \Delta y) + (D_2 - 2 * D) * \min(\Delta x, \Delta y) \quad (2)$$

où $D = 1$ et $D_2 = \sqrt{2}$.

Toutefois, dans un espace ouvert, cette heuristique peut être équivalente pour de nombreux trajets comportant le même nombre de cases. Ainsi, de nombreux nœuds sont explorés inutilement. Il faut donc ajouter un terme pour briser l'égalité. L'approche retenue est de gonfler artificiellement l'heuristique. Ainsi :

$$h'(p_1, p_2) = h(p_1, p_2) * (1 + p) \quad (3)$$

et p est choisi toujours positif plus petit que le coût minimal pour voyager entre deux nœuds divisés par la longueur maximale d'un trajet valide, avec Dim_x et Dim_y les dimensions de la grille d'occupation :

$$p = \frac{1}{2*(Dim_x + Dim_y)} \quad (4)$$

En plus de l'heuristique, il faut également être en mesure de calculer le coût réel pour voyager entre deux nœuds, soit le coût de l'arc entre ces nœuds. Ainsi, il est possible de calculer le coût total d'un trajet en additionnant le coût de tous les arcs et de tous les nœuds traversés. L'implémentation originale utilise seulement la distance euclidienne d entre les positions, soit, avec les positions p_1 et p_2:

$$d(p_1, p_2) = \sqrt{(x_1 - x_2) + (y_1 - y_2)} \quad (5)$$

Toutefois, il est intéressant ici de considérer également l'orientation. Le niveau de superposition entre les intervalles, exprimé en degrés, est donc également pris en compte. La prémisse étant que plus les intervalles sont superposés, moins le véhicule a à pivoter et plus il demeure éloigné des obstacles. Avec $\theta_{sup}(n_1, n_2)$ qui représente la superposition des

intervalles en degrés, la fonction de coût total d'un arc c_{arc} entre les nœuds n_1 et n_2 aux positions p_1 et p_2 devient :

$$c_{arc}(n_1, n_2) = d(p_1, p_2) + \left(1 - \frac{\theta_{sup}}{360°}\right) \quad (6)$$

Afin de simplifier l'implémentation à venir, le coût pour traverser le nœud de destination c_{noeud},définit à la section 3.1.1, est également ajouté au coût. Ainsi, pour une exploration de A*, soit de visiter le nœud n_2 à la position p_2 à partir du nœud n_1 à la position p_1, le coût c est donné par :

$$c(n_1, n_2) = c_{arc}(n_1, n_2) + c_{noeud}(n_2) \quad (7)$$

3.1.4 Implémentation

Le graphe est implémenté en C++ pour s'intégrer à la pile de navigation ROS. Trois classes sont utilisées afin de représenter les entités du graphe : une position, un nœud et un graphe. L'utilité des capacités de hachage est expliquée à la section 3.3.3. La position représente un emplacement (x, y) dans la grille d'occupation. Elle contient :

- Sa position en x dans le référentiel de la grille d'occupation;
- Sa position en y dans le référentiel de la grille d'occupation.

La position peut :

- Se représenter comme un *hash* : un identifiant numérique unique calculé à partir d'une position et qui permet de la reconnaître dans une liste en comparant les *hash* plutôt que les positions complètes;
- Fournir son coût dans la grille d'occupation fournie en paramètre;
- Fournir le coût pour se déplacer jusqu'à une autre position;
- Fournir sa position dans le référentiel du monde à partir de la grille d'occupation fournie en paramètre;

- Indiquer si elle représente un emplacement à l'intérieur des dimensions de la grille d'occupation passées en paramètres;
- Calculer le déplacement en x et en y requis pour se déplacer jusqu'à une autre position.

Deux opérateurs sont aussi implémentés sur un objet de position : l'addition et l'égalité. L'addition retourne une nouvelle position dont les coordonnées sont les sommes des coordonnées des positions fournies en entrée. L'égalité est vraie si les coordonnées en x et en y de deux positions sont égales.

Le nœud représente un nœud tel qu'il peut être visité par A*. Il contient :

- Un objet de type position, tel que défini au début de la section 3.1.4, qui indique sa position dans la grille;
- Le début de son intervalle d'orientations admissibles en degrés;
- La longueur de son intervalle d'orientations admissibles en degrés;
- Le coût pour le visiter.

Tout comme la position, le nœud a plusieurs fonctionnalités :

- Indiquer s'il est possible d'atteindre un autre nœud passé en paramètre;
- Indiquer le coût pour se déplacer jusqu'à un autre nœud passé en paramètre;
- Fournir la valeur de la fonction heuristique jusqu'au but passé en paramètre pour le nœud;
- Retourner sa position décalée d'une position fournie en paramètre;
- Fournir sa position;
- Fournir le début de son intervalle d'orientation en degrés;
- Fournir la longueur de son intervalle d'orientation en degrés;
- Indiquer si une orientation en degré passée en paramètre est comprise dans son intervalle d'orientations admissibles;

- Se représenter comme un *hash*.

Le nœud implémente également l'opérateur d'égalité. Il retourne vrai si les positions des nœuds reçus sont égales et s'il est possible de voyager d'un nœud à l'autre, autrement dit, si les intervalles d'orientations admissibles se superposent.

Enfin, le graphe est l'interface utilisée par A* lors de la recherche du trajet. Il contient :

- La dimension en x de la grille d'occupation dans le référentiel de la grille;
- La dimension en y de la grille d'occupation dans le référentiel de la grille.

Sa seule fonction est de fournir les nœuds voisins d'un nœud passé en paramètre. C'est dans le graphe que sont implémentés les algorithmes à l'aide des capacités des nœuds et des positions.

3.2 De l'algorithme A* à un planificateur global complet

L'algorithme A* est un algorithme de recherche classique [49]. Il est ici utilisé tel quel, les particularités propres aux AGV libres holonomes se trouvant dans les nœuds présentés à la section 3.1.1. Une particularité est que la sortie de A* est une suite de nœuds. Toutefois, un planificateur global conforme aux spécifications de la pile de navigation ROS doit produire une série de couples (x, y, θ). Il faut donc, à partir de la série d'intervalles d'orientations admissibles, produire une série de valeurs de θ.

Trois critères principaux guident le choix de l'orientation le long du trajet. D'abord, il faut éviter les collisions. Ensuite, il est préférable de circuler avec le côté étroit devant, autant pour l'intuition des humains à proximité, qui piloteraient naturellement l'engin dans ce sens, mais également puisque c'est la direction qui minimise l'usure des roues et l'utilisation de la batterie étant donné la conception mécanique de la plateforme. Enfin, il faut minimiser autant que possible les variations d'orientations qui rendent les humains à proximité ainsi que les propriétaires de la charge utile un peu inquiets. La figure 3.2 présente le processus décisionnel qui associe une orientation à chaque nœud. À chaque nœud du trajet trouvé par

A*, s'il est possible de regarder en avant, c'est-à-dire s'il reste une distance suffisante avant la fin du trajet pour calculer une tangente au déplacement en (x, y) du robot, l'orientation tangente au mouvement et son inverse est calculée. Ces deux orientations tangentes sont alors comparées à l'orientation actuelle afin de déterminer laquelle requiert le moins de rotation. Si cette orientation est comprise dans l'intervalle d'orientation du

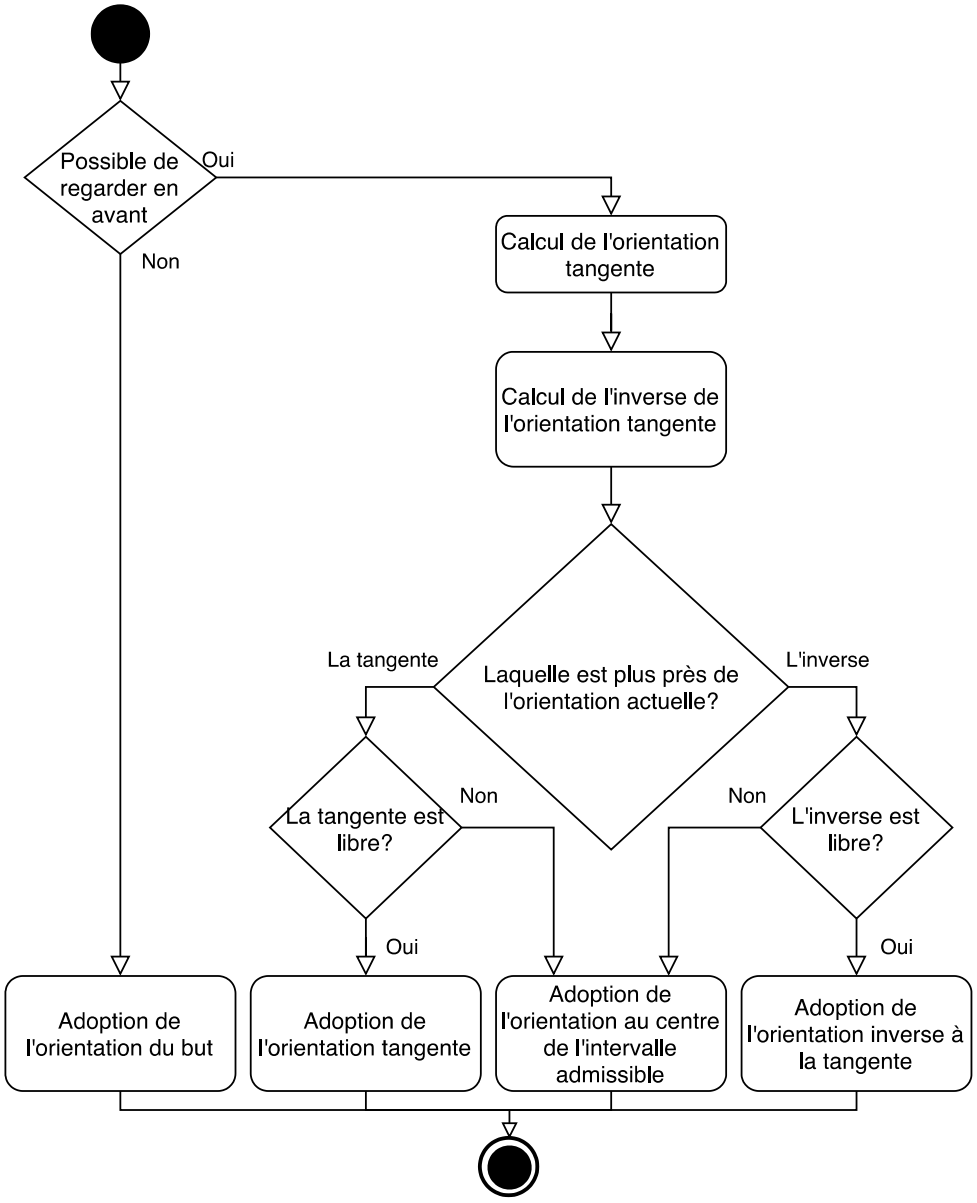

Figure 3.2 Processus pour déterminer l'orientation à adopter

nœud, elle est adoptée. Sinon, c'est l'orientation au centre de l'intervalle qui est retenue. Les nœuds en fin de trajet reçoivent l'orientation du but.

3.2.1 Implémentation

Afin d'être utilisable avec la pile de navigation ROS-NH, le planificateur global doit être une classe C++ héritant de *nav_core::BaseGlobalPlanner*. Cette classe doit implémenter une méthode publique d'initialisation ainsi qu'une méthode publique de planification pour

produire une série de couples (x, y, θ) entre un couple de départ et un couple d'arrivée donnée. La méthode d'initialisation réalise des tâches administratives de ROS et récupère la grille d'occupation qui est utilisée par le planificateur. La méthode de planification recherche le trajet avec A*, puis reconstruit l'orientation à l'aide du processus présenté à la figure 3.2. Cette approche permet à la pile de navigation de charger dynamiquement le bon planificateur global selon le fichier de configuration du robot en créant simplement une instance de la classe correspondante.

À la base de l'implémentation d'A* se trouve la frontière. C'est une structure de données qui doit retourner le prochain nœud à explorer, c'est à dire le nœud avec le plus faible coût pour le rejoindre. La *priority_queue* de la librairie standard C++ est retenue pour jouer ce rôle. Son utilisation requiert trois éléments :

- Un item;
- Une fonction qui permet de comparer deux items;
- Une structure de données pour stocker les items.

Une fois ces trois éléments implémentés, la *priority_queue* retourne toujours l'item pour lequel la fonction de comparaison est toujours fausse en premier. Ici, un item se définit comme une paire de la librairie standard entre un nœud du graphe et un nombre qui est le coût pour l'atteindre. La fonction de comparaison compare simplement le coût des deux items. Enfin, un vecteur de la librairie standard est utilisé pour stocker les items.

A* requiert également des structures de données pour stocker des associations entre des nœuds (quel nœud a mené à quel nœud) et des associations entre des nœuds et des nombres (quel est le coût total pour rejoindre un nœud). Des *map* de la librairie standard sont utilisées à cette fin. Toutefois, les éléments d'une *map* doivent pouvoir être représentés sous forme de *hash*, c'est pourquoi les nœuds implémentent une fonction de hachage. L'ensemble du code du planificateur global se trouve dans le paquet ROS *ino_planner*. C'est la classe *ino_planner::InoPlanner* qui contient la frontière et les associations. Elle expose également

les méthodes requises qui sont appelées par la pile de navigation ROS-NH lors de la planification d'un trajet.

3.3 Réduction du temps de recherche du coût de l'empreinte

Une fois la première itération du planificateur global implémentée, un problème de performance s'est présenté : le temps de recherche du trajet était de quelques minutes plutôt que de quelques fractions de secondes avec la pile de navigation ROS-NH. Ce temps est inacceptable pour l'utilisation prévue sur un AGV, puisque le temps de planification est du temps où le véhicule n'est pas productif. Un travail de profilage du code a d'abord été fait pour identifier la portion du planificateur à accélérer. La librairie standard *chrono* a été utilisée pour ajouter des points de mesure du temps écoulé dans le code. Les trois portions de code ciblées par ce profilage étaient : l'insertion et la lecture dans la *priority_queue*, les appels au graphe pour l'obtention des nœuds voisins et le calcul du coût de déplacement entre les nœuds. Le tableau 1 présente les résultats du profilage du planificateur complet lors de la recherche du trajet entre deux points à l'aller (recherche 1) et au retour (recherche 2). Le temps passé dans quatre portions de l'exécution du planificateur global a été mesuré, soit :

- L'obtention du prochain nœud dans la *priority_queue;*
- La recherche des voisins du nœud courant;
- Le calcul du coût du déplacement entre deux nœuds;
- Le reste de l'exécution.

À la lumière de ces résultats, il est apparu évident que la recherche des voisins dans le graphe était la portion à accélérer puisque c'était celle dont le temps d'exécution était le plus long.

Tableau 1 Profilage du planificateur

Portion	Temps (us)	
	Recherche 1	Recherche 2
priority_queue	17577	15168
Recherche des voisins	44775229	62823145
Coût de déplacement	4736	2998
Autre	1760855	1893421

Une autre ronde de profilage a été menée sur le graphe. Les portions ciblées étaient : la vérification d'un lien entre deux nœuds, la transformation des coordonnées entre le référentiel de la grille d'occupation et le référentiel du monde, le calcul du coût de l'empreinte et le calcul du coût entre deux nœuds. Le tableau 2 présente les résultats du profilage du graphe. Ces données montrent clairement que c'est le calcul du coût de l'empreinte qui cause un ralentissement du logiciel. Il faut donc modifier la fonction existante dans la pile de navigation pour obtenir le coût exact de l'empreinte du robot, lequel permet de détecter les collisions avec les obstacles. Puisque cette méthode est appelée pour chaque orientation de chaque position circonscrite explorée, il est logique qu'elle ait un grand impact sur la performance de l'ensemble.

Tableau 2 Profilage du graphe

Portion	Temps (us)	
	Recherche 1	Recherche 2
Lien	0	0
Transformation	49	1820
Empreinte	5841542	49154046
Coût	311	2938

3.3.1 Méthode des quatre coins

La méthode employée par la pile de navigation ROS-NH est de projeter chaque segment du périmètre de l'empreinte dans la grille d'occupation et de vérifier qu'aucune case occupée n'est touchée. Ensuite, le coût de l'empreinte correspond à la somme du coût des cases le

long de son périmètre. Cela est illustré à la figure 3.3 où les cases colorées de la grille sont toutes les cases dont le coût est évalué par la méthode originale. Avec un robot de 3 m de long par 2 m de large et une résolution de grille de 10 cm, cela nécessite 50 vérifications pour évaluer une empreinte. La solution proposée, appelée méthode des quatre coins, assume un robot rectangulaire. Ensuite, elle valide que le centre n'est pas dans une case inscrite ou occupée et qu'aucun des coins ne se trouve dans une case occupée. Enfin, en l'absence de collision, le coût de l'empreinte correspond au coût du centre. Cela réduit le nombre de vérifications par un facteur 10. Cela est illustré à la figure 3.3 où seules les cases vertes doivent être évaluées. La théorie est qu'avec la forme rectangulaire de l'empreinte, la vérification des quatre coins est suffisante pour vérifier l'absence de collision, de la même manière que l'algorithme original de la pile de navigation utilise seulement le centre en assumant un robot circulaire.

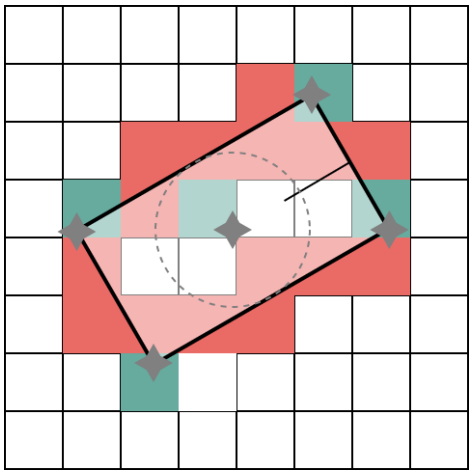

Figure 3.3 Méthode des quatre coins

Le tableau 3 montre les résultats d'une ronde de profilage du graphe avec la nouvelle méthode. Immédiatement, un gain de performance important est observable. Toutefois, la méthode s'avère imparfaite. Dans un cas comme celui présenté à la figure 3.4, l'AGV entre en collision avec un obstacle, mais la méthode des quatre coins ne détecte pas la collision. En effet, un obstacle peut s'approcher du véhicule entre les coins les plus éloignés du centre. Cet obstacle n'est alors pas détecté par les coins, ni par le centre qui vérifie seulement le

rayon inscrit, donc seulement les obstacles qui s'approchent entre les coins les plus rapprochés. Une meilleure stratégie est nécessaire.

Tableau 3 Profilage du graphe avec la méthode des quatre coins

Portion	Temps (us)	
	Ronde 1	Ronde 2
Lien	0	0
Transformation	48	93
Empreinte	15139	17642
Coût	284	689

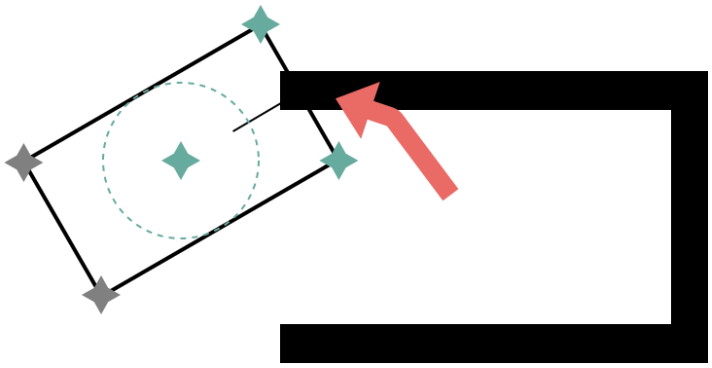

Figure 3.4 Collision entre les coins éloignés du centre

3.3.2 Méthode des deux cercles

Pour bien comprendre la méthode des deux cercles, il est pertinent de discuter davantage de la simplification de l'empreinte employée par la pile de navigation ROS-NH. En effet, ses développeurs ont également eu à employer une stratégie pour accélérer la recherche des collisions entre une empreinte et une grille d'occupation. Il est important de noter que la pile de navigation originale est bâtie autour d'un robot rond ou circulaire. L'approche est la suivante :

- Calculer le rayon $r_{inscrit}$ d'un cercle pouvant être inscrit dans l'empreinte. La figure 3.5 montre le cercle inscrit rougeâtre au centre des empreintes rectangulaires. Cette opération est effectuée à chaque reconfiguration des dimensions de l'empreinte;

- Dans la grille d'occupation, attribuer un poids inscrit élevé connu aux cases à moins de $r_{inscrit}$ d'une case occupée. La figure 3.5 montre cette zone en rougeâtre et les obstacles en noir. Cette opération est effectuée à chaque mise à jour de la grille d'occupation, mais diverses optimisations sont implémentées dans le paquet ROS qui fournit la grille d'occupation afin de rendre le processus rapide;
- Vérifier la présence d'une collision en vérifiant si le coût de la case occupée par le centre du robot est inférieur au poids inscrit. Si ce n'est pas le cas, l'empreinte est en collision, sinon, la position est admissible.

Toutefois, cette approche perd son sens lorsque l'empreinte est rectangulaire comme l'illustre la figure 3.5. Si le côté large du rectangle fait face à l'obstacle, comme dans la configuration (c), le raisonnement fonctionne. La collision débute lorsque le centre pénètre dans la zone rougeâtre. Toutefois, comme dans la configuration (b), lorsque c'est le côté étroit qui est face à l'obstacle, la collision n'est pas détectée.

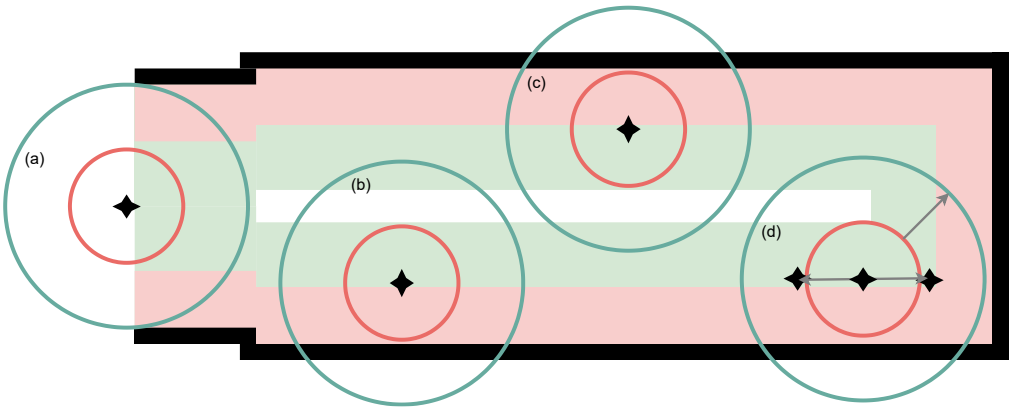

Figure 3.5 Méthode des deux cercles

Il est toutefois possible de tracer le cercle de rayon $r_{circonscrit}$, en vert sur la figure 3.5, qui circonscrit l'empreinte et attribuer un poids circonscrit aux cases de la grille. Il en résulte la zone verte de la figure 3.5. Contrairement à la zone rougeâtre, le centre du robot peut s'y trouver. Dans le cas contraire, le robot pourrait refuser d'emprunter des passages étroits qui sont tout de même praticables comme dans la configuration (a). Lorsque le centre du robot se trouve dans la zone circonscrite, cela indique plutôt que davantage de vérifications

sont nécessaires. Pour faire ces vérifications, deux points sont placés aux coordonnées suivantes :

$$\{x, y\} = \{\pm(r_{circonscrit} - r_{inscrit}), 0\} \quad (8)$$

Ces points sont montrés dans la configuration (d) de la figure 3.5. Ils sont décalés du centre vers le côté étroit de l'empreinte d'une distance équivalente à la différence entre les deux rayons. Le même raisonnement par rapport à la zone rougeâtre est maintenant applicable. Comme montré dans la configuration (d), si les deux points sont hors de la zone inscrite, l'empreinte n'est pas en collision. Cette approche est très efficace, puisque seulement trois vérifications sont requises. Elle a également démontré sa robustesse en permettant au planificateur de produire des trajets adéquats en quelques secondes, ce qui est approprié pour l'utilisation prévue sur un AGV.

3.3.3 Implémentation

La pile de navigation ROS-NH utilise une classe *world_model* pour fournir le coût d'une empreinte à une position donnée. Cette classe expose une interface qui fournit le coût d'une empreinte donnée à la position spécifiée. L'implémentation par défaut est *costmap_model* qui implémente la stratégie de projection du périmètre de l'empreinte dans la grille. Ce modèle du monde est instancié lors de l'initialisation d'un planificateur. Afin d'utiliser la méthode des deux cercles, une nouvelle classe *simple_costmap_model* est implémentée. Elle expose la même interface, mais cette fois avec la méthode des deux cercles au lieu de la projection.

Le planificateur global implémenté par la classe *ino_planner::InoPlanner* instancie une instance du *simple_costmap_model* lors de l'initialisation. Toutefois, contrairement au *costmap_model*, l'instanciation ne se fait pas avec la définition complète de l'empreinte. Le modèle simplifié reçoit plutôt seulement les deux points décalés. Lors de son utilisation par le planificateur, la position du centre dans la grille est passée en paramètre lors de l'appel pour l'obtention

du coût. En décalant le centre, le modèle dispose de toute l'information nécessaire pour vérifier la présence d'une collision. L'ensemble du code est disponible sur GitHub[5].

[5] https://github.com/GodCed/ino_planner

CHAPITRE 4

PLANIFICATEUR LOCAL

Dans la pile de navigation ROS-NH, le planificateur local détermine la vitesse à laquelle la base robotique doit se déplacer pour atteindre le but à partir du trajet reçu du planificateur global. Contrairement à ce dernier, le planificateur local opère seulement dans un horizon restreint autour du robot, plutôt que dans l'ensemble de l'environnement. Le planificateur local n'utilise pas la grille d'occupation produite par le système SLAM. Il utilise plutôt directement les mesures des capteurs pour produire une grille d'occupation des alentours du robot et reçoit la portion du trajet global qui traverse cette grille. Le point où le trajet rencontre la frontière de la grille d'occupation est appelé le but local et c'est ce dernier que le planificateur local tente d'atteindre. Le comportement du robot entre sa position actuelle et le but local est à la discrétion du concepteur du planificateur local, le robot n'étant pas tenu de demeurer sur le trajet précalculé. Un planificateur local polyvalent est fourni avec la pile de navigation ROS-NH, DWA, est utilisé dans le présent projet et son fonctionnement est décrit dans la section 4.1.

4.1 Planificateur DWA

Comme son nom l'indique, le planificateur DWA est basé sur l'idée d'une fenêtre dynamique dans laquelle le robot doit opérer. Cette fenêtre est centrée sur la vitesse actuelle du robot et est limitée par ses spécifications ainsi que par le temps de simulation donné. La fenêtre donne la plage de vitesse pouvant être accomplie par le robot dans le temps donné, les vitesses étant limitées par l'accélération et la vitesse actuelle ou bien par la vitesse maximale du robot. Tout comme le planificateur global, DWA est un planificateur par échantillonnage. Toutefois, plutôt que l'environnement, c'est la fenêtre de vitesse qui est échantillonnée. Un cycle de planification débute donc avec une liste des couples de vitesses (x, y, θ) possibles.

Ensuite, des trajectoires sont formées. Chaque trajectoire correspond à un couple de vitesses et contient les positions futures du robot s'il adopte le couple de vitesses pendant le temps de simulation. La figure 4.1 présente les fonctions de coût qui sont alors appliquées à chaque trajectoire afin d'en sélectionner une. Un poids est attribué à chaque fonction de coût par le concepteur et le coût de la trajectoire est donné par la somme pondérée du coût associé à chaque fonction de coût. Naturellement, la trajectoire retenue et commandée au robot est celle dont le coût est le plus faible.

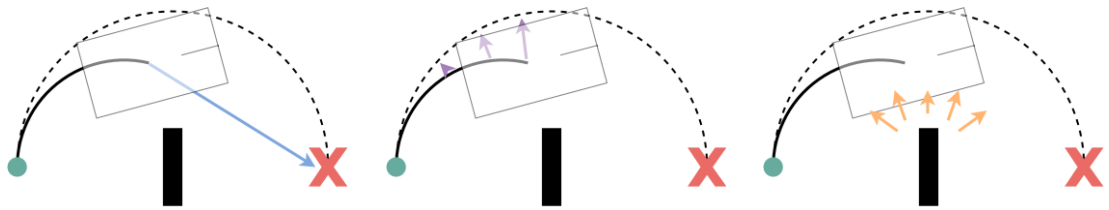

Figure 4.1 Fonctions de coût de DWA : a) but; b) trajet; c) obstacles

La première fonction, a), pénalise la distance du but local. Elle a pour objectif d'inciter le robot à se déplacer le plus rapidement possible. Elle est calculée en mesurant la distance entre le but local et le dernier point de la trajectoire. Plus la distance est élevée, plus le coût est élevé. La deuxième fonction, b), pénalise la distance du trajet fourni par le planificateur global. La plus courte distance entre chaque point de la trajectoire et le trajet est calculée. Le coût est donné par la somme de ces distances. L'objectif est d'inciter le robot à demeurer sur le trajet plutôt qu'à se diriger en ligne droite vers le but local. Enfin, une troisième fonction de coût, c), pénalise la proximité avec les obstacles. Elle utilise une grille d'occupation bâtie à partir des capteurs pour assigner un coût à chaque point de la trajectoire selon le coût de case de la grille qui contient le point à évaluer. Les cases plus près des obstacles ont un coût plus élevé. Cette fonction doit garder le robot à l'écart des obstacles. Cette fonction peut également marquer une trajectoire comme invalide en lui assignant un coût fatal si un des points de la trajectoire entre en collision avec un obstacle.

4.2 Ajout d'une nouvelle fonction de coût

Il est important de noter que la fonction de coût qui évalue la distance entre la trajectoire et le trajet ne considère que la distance linéaire entre les coordonnées (x, y). C'est logique puisque le planificateur global original de la pile de navigation ROS-NH ne fournit pas d'information quant à l'orientation à adopter le long du trajet. Ainsi, seules les capacités cinématiques et la proximité avec les obstacles limitent la vitesse en rotation dans DWA. Cela rejoint la problématique présentée au chapitre 1, c'est-à-dire que l'orientation d'un AGV holonome n'est pas contrainte dans un espace ouvert lorsque celui-ci est piloté par la pile de navigation ROS-NH.

En conservant l'esprit de DWA, la solution retenue est d'ajouter une fonction de coût supplémentaire. Cette fonction a deux rôles : favoriser une rotation du robot vers l'orientation dictée par le planificateur global et l'empêcher de se déplacer de manière linéaire si l'orientation est incorrecte afin d'éviter les blocages dans des espaces restreints qui ne sont pas visibles dans l'horizon du planificateur local. Ainsi, la fonction de coût compare d'abord l'orientation commandée par le trajet du planificateur global à l'emplacement actuel du robot et l'orientation prévue à la fin de la trajectoire. Un coût est alors attribué à l'écart entre ces deux orientations. La fonction de coût attribue également un coût à la vitesse linéaire du robot. Ce coût est inversement proportionnel à l'écart entre l'orientation commandée et finale. Ainsi, le robot va ralentir et prendre le temps de pivoter vers la bonne orientation avant de se déplacer.

4.3 Implémentation

Limiter les modifications du planificateur DWA à l'ajout d'une fonction de coût est avantageux puisque cela permet de réutiliser l'ensemble du code du paquet ROS original. Toutefois, l'implémentation de DWA ne permet pas le chargement dynamique de fonctions de coût supplémentaires. Ainsi, une copie parallèle du paquet *dwa_local_planner* a dû être créée afin d'apporter les modifications requises au code. L'environnement ROS charge alors

le planificateur du paquet local, plutôt qu'à partir du paquet fourni avec la pile de navigation. Les modifications apportées au planificateur DWA original sont les suivantes :

- Ajout d'une nouvelle classe *YawCostFunction*, héritant de *TrajectoryCostFunction* qui implémente la nouvelle fonction de coût;
- Ajout du poids associé à la *YawCostFunction* à la configuration de *DWALocalPlanner*;
- Création d'une instance de la *YawCostFunction* et ajout de l'instance à la liste des fonctions de coût à appliquer lors de l'initialisation du planificateur;
- Transfert des paramètres du robot, du but, de la position actuelle et de la vitesse linéaire actuelle vers la *YawCostFunction* lors de la préparation du planificateur à l'évaluation des trajectoires.

L'ensemble du code est disponible sur GitHub[6].

[6] https://github.com/GodCed/dwa_local_planner

CHAPITRE 5

GESTION DES RÈGLES DE CIRCULATION

Bien qu'il soit possible de peaufiner un algorithme de planification de trajets indéfiniment pour optimiser de multiples critères mesurables comme le temps de trajet, la distance parcourue, l'utilisation de l'énergie et bien d'autres, certains cas d'utilisation requièrent des trajets volontairement non optimaux. Les règles de sécurité en place dans un environnement industriel peuvent commander que la circulation soit interdite dans certaines zones et priorisée dans d'autres. Un outil de manutention installé sur le véhicule peut nécessiter des déplacements avec une orientation fixe entre des rangées de palettes pour en lire les étiquettes. Bref, un utilisateur d'AGV doit pouvoir ajouter des critères arbitraires à la planification du trajet afin que le véhicule s'y conforme pour que tous les déplacements puissent se faire de façon autonome.

Ce chapitre présente une preuve de concept de la réalisation d'un tel système de navigation qui encode les consignes dans les grilles d'occupation existantes de ROS, fournit une interface graphique conviviale pour l'utilisateur et respecte les consignes lors des déplacements autonomes d'un AGV. Les consignes retenues pour la preuve de concept sont :

- Les zones où il est préférable de circuler;
- Les zones où il est interdit de circuler;
- Les zones où l'orientation à adopter est spécifique.

5.1 Encodage dans les couches des grilles d'occupation

La pile de navigation ROS-NH utilise abondamment les grilles d'occupation pour stocker et transporter les informations requises pour le déplacement des robots. Le paquet *grid_map* offre plusieurs outils pour les stocker, les transporter et les utiliser tout en assurant une

excellente performance. Divers outils existent également pour l'affichage. Ainsi, les paramètres de circulation sont stockés dans des grilles d'occupation. Les grilles ont toutefois une limitation importante : chaque case contient un seul nombre. Les différents paramètres doivent donc être encodés. Puisque *grid_map* supporte plusieurs couches, une couche est créée pour chaque type de paramètre.

Les paramètres de circulation sont représentés comme les obstacles. Toutes les cases débutent à un coût faible mais non nul. Les cases où il est préférable de circuler se voient attribuer un coût nul. Le planificateur global les préfère donc aux autres. Le faible coût des autres cases peut être ajusté pour prioriser soit les zones soit la longueur des trajets. Pour les zones interdites, le coût des cases est fixé au coût d'une case occupée tel que défini dans les spécifications de la grille d'occupation ROS. Ainsi, elles apparaissent comme des obstacles et sont systématiquement évitées par le planificateur global. Pour l'orientation, une transformation est nécessaire. En effet, chaque case doit contenir deux valeurs : le mode et l'angle en degrés. La transformation suivante permet de les combiner :

$$valeur = 1000 * mode + angle \qquad (9)$$

Au moment de la planification du trajet, la valeur de l'orientation est récupérée pour chacune des cases traversées. Selon le mode, la bonne orientation est alors ajoutée au point de cheminement.

Les grilles elles-mêmes sont contenues dans un nœud ROS s'exécutant sur le robot qui agit comme un serveur de cartes. Ce nœud assure la persistance des paramètres en sauvegardant les grilles sur le disque dur entre les utilisations. Ce nœud expose également les cartes aux nœuds de la pile de navigation et à l'interface de l'utilisateur. Des services ROS sont également rendus disponibles pour permettre l'ajout et la suppression des zones.

5.2 Interface graphique

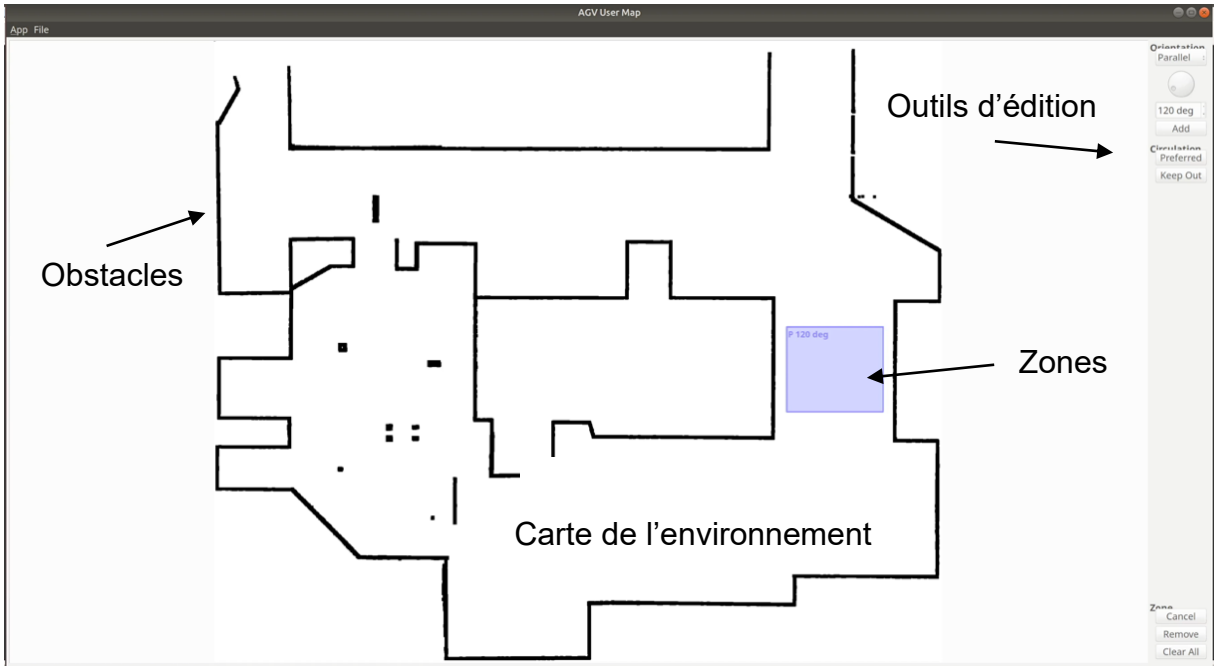

Figure 5.1 Interface des paramètres de circulation avec une zone d'orientation spécifique

Les grilles d'occupation ne peuvent malheureusement pas être remplies directement par l'utilisateur. Une interface graphique a donc été développée pour spécifier les paramètres. Cette interface s'exécute sur l'ordinateur de l'utilisateur et interagit avec le robot à travers le serveur de cartes.

La figure 5.1 montre une capture d'écran de l'interface où une zone d'orientation spécifique a été ajoutée. Le grand panneau central montre la carte de l'environnement du robot récupérée du système SLAM. Les obstacles sont présentés en noir et les règles de circulation par des rectangles de couleur. Les boutons du panneau de droite permettent d'ajouter les différentes zones de circulation : orientation spécifique (bleue), zone préférée (verte) et zone interdite (rouge). La molette permet d'indiquer l'angle pour l'ajout des zones d'orientation spécifiques. Pour ajouter une zone, l'utilisateur commence par placer la molette à la bonne valeur au besoin. Ensuite, il clique sur le bouton correspondant au type de zone à ajouter. Enfin, il clique dans la zone de la carte, puis fait glisser le curseur en maintenant le clic

jusqu'au coin opposé de la zone rectangulaire voulue. La figure 5.2 montre les deux autres types de zones dans l'interface.

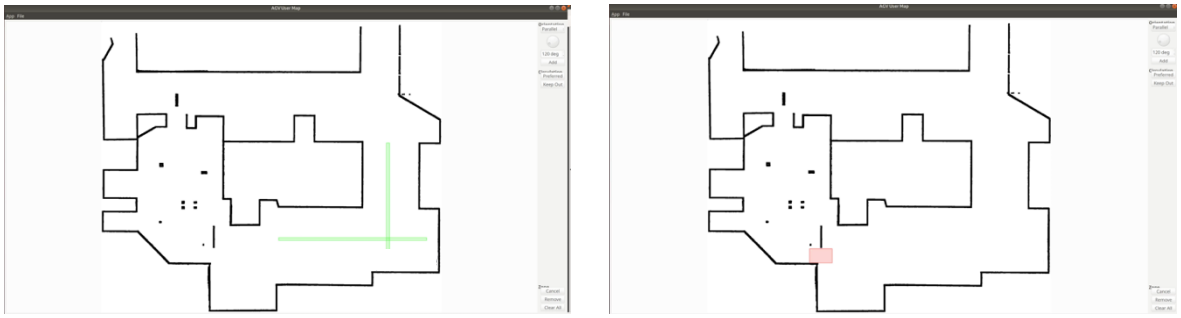

Figure 5.2 Zone préférée et interdite dans l'interface

5.3 Intégration avec la pile de navigation ROS-NH

Afin de démontrer la viabilité du concept, le simulateur Gazebo de l'AGV a été adapté pour utiliser le serveur de carte et l'interface utilisateur. Puisque les travaux sur la gestion des paramètres de circulation ont été réalisés avant le développement du planificateur *ino_planner* présenté au chapitre 3, celui-ci n'était pas disponible. Un planificateur global a donc été produit pour utiliser l'orientation de la carte des paramètres de circulation. Ce dernier recherche le trajet entre le point de départ et le but avec l'algorithme A* et le graphe original de la pile de navigation ROS-NH. Il ajoute ensuite l'orientation indiquée dans la couche d'orientation de la carte des paramètres de circulation à chaque point de cheminement. Le planificateur local présenté au chapitre 4 est repris intégralement puisque ce dernier assure le respect de l'orientation spécifiée dans le trajet reçu. Le code permettant la gestion des paramètres utilisateur est disponible sur GitHub[7].

[7] https://github.com/GodCed/user_map

CHAPITRE 6

TESTS ET RÉSULTATS

Le véhicule étant encombrant et l'atelier assez exigu, la plupart des tests ont été réalisés dans des environnements simulés sur Gazebo. Un représente un grand entrepôt afin de valider le logiciel sur de longs trajets dans espaces représentatifs de l'environnement d'opération de l'AGV. Le second environnement est une copie virtuelle de l'atelier d'Inogec, afin de reproduire le comportement observé lors des tests avec le vrai véhicule. Lors des essais, l'outil *rosbag* a servi pour enregistrer toutes les informations circulant sur le réseau ROS. Les paramètres qui se sont avérés pertinents pour le traitement des résultats sont :

- La carte produite par le système SLAM;
- Les grilles d'occupation de la pile de navigation ROS-NH;
- La position du véhicule;
- La vitesse du véhicule.

L'outil *rostopic* a servi à extraire les données numériques qui devaient être traitées dans MATLAB pour la production de graphiques ou de tableaux comparatifs.

6.1 Tests avec l'AGV VII

Une fois l'implémentation de la pile de navigation ROS-H complétée et avant de passer aux tests, un travail d'optimisation des poids associés aux différentes fonctions de coût du planificateur local a été réalisé sur l'AGV VII dans les locaux du partenaire industriel Inogec. La figure 6.1 présente la carte de l'environnement opérationnel de l'AGV VII produite par le SLAM lors de ces essais. L'espace gris clair représente l'espace libre et l'espace noir l'espace occupé. Les mesures des LiDARs concordent avec les obstacles et sont en rouge. Le gradient de couleur translucide indique la valeur des cases de la grille d'occupation. Les

cases turquoise ont un coût inscrit puis la valeur diminue tandis que la couleur tend progressivement vers le bleu foncé. Les cases libres ne sont pas colorées.

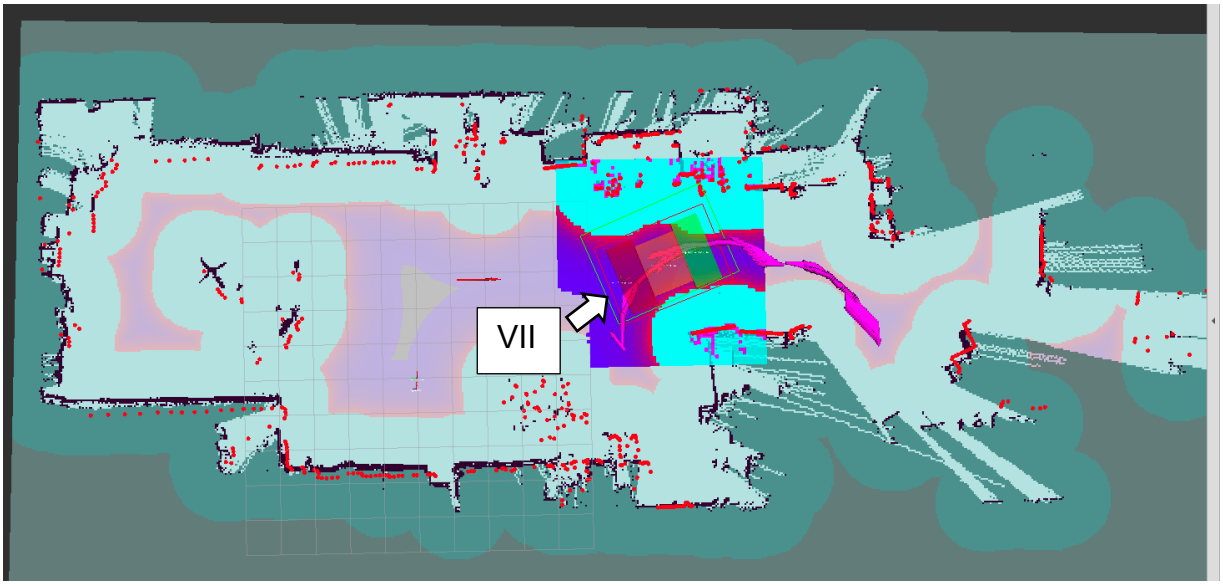

Figure 6.1 Carte produite par le SLAM lors de l'ajustement des poids

Différentes combinaisons des poids associés aux fonctions de coût de DWA ont été utilisées afin de s'approcher au maximum du comportement recherché pour le véhicule. Le tableau 4 présente les combinaisons de cette série de tests et les comportements observés. Les poids modifiés sont :

- **Goal :** le poids associé à la fonction de coût qui attire le robot vers le but;
- **Plan :** le poids associé à la fonction de coût qui garde le robot sur le trajet fourni par le planificateur global;
- **Obs :** le poids associé à la fonction de coût qui éloigne le robot des obstacles;
- **Yaw :** le poids associé à la fonction de coût ajouté à DWA pour respecter l'orientation contenue dans le trajet prévu par le planificateur global.

Pour chaque combinaison, les impressions de l'opérateur sur le comportement du véhicule sont notées. Le comportement obtenu est toujours un compromis entre le suivi du trajet et le déplacement fluide du véhicule. Cette limitation vient de la contradiction entre la pénalité

sur la distance avec le but et la pénalité sur l'écart entre la trajectoire et le trajet planifié par le planificateur global.

Tableau 4 Comportements de l'AGV VII avec diverses combinaisons de poids

Poids				
Goal	Plan	Obs	Yaw	Commentaires
24	6	0.05	32	Beaucoup d'oscillations aux points où il y a un grand changement d'orientation. Hésitation à proximité des obstacles.
12	24	0.01	6	Plus naturel, mais éloigné du trajet.
14	16	0.001	6	Mouvement souple, mais éloigné du trajet. L'AGV VII se bloque contre un obstacle que le planificateur global avait contourné.
28	30	0.001	6	Semble prometteur, l'AGV VII négocie avec succès des passages restreints.
24	16	0.005	32	Se déplace de manière fluide, mais coupe des portions du trajet.
6	10	0.005	32	S'écarte du trajet et se bloque.
32	24	0.005	32	Fluide une fois en mouvement, mais beaucoup d'hésitations et d'oscillations lorsque la vitesse courante est faible.

La figure 6.2 présente un cas typique de la contradiction entre la fonction de coût du trajet et la fonction de coût du but. Le trajet planifié par le planificateur global est en pointillé. Le point de départ est le rond vert et le but est la croix rouge. L'évitement de l'obstacle, le rectangle noir, par le planificateur global est bien visible. La trajectoire évaluée par le planificateur local est en noire. La flèche bleue représente l'objectif de la fonction de coût qui minimise la distance avec le but. Les flèches mauves représentent l'objectif de la fonction de coût qui minimise l'écart avec le trajet. Il s'avère évident que les flèches bleues et mauves sont contradictoires : l'AGV ne peut pas demeurer sur le trajet et avancer vers son but simultanément.

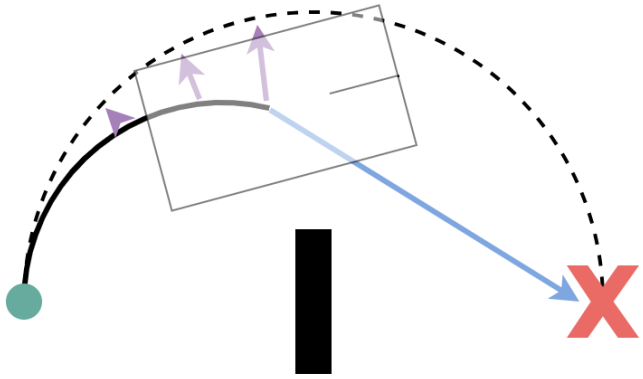

Figure 6.2 Écart entre le trajet et le but

Lors des essais, plusieurs combinaisons de poids ont été testées afin d'arriver à un compromis acceptable. Toutefois, ces tentatives ne furent pas concluantes :

- Avec un poids faible pour le suivi du trajet et un poids fort pour l'atteinte du but, le robot dévie du trajet et se bloque dans un obstacle;
- Avec un poids faible pour l'atteinte du but et un poids fort pour le suivi du trajet, le robot ne bouge pas parce qu'il est aussi payant de demeurer sur place que d'avancer;
- Avec des poids équivalents, le robot oscille sur place entre le trajet et le but alors que les deux fonctions de coût s'échangent la priorité.

Une nouvelle fonction de coût pour remplacer celle basée sur la distance du but s'avère nécessaire pour forcer l'AGV à avancer sans pour autant s'éloigner de son trajet. Elle n'a toutefois pas été réalisée dans le cadre de ce mémoire. Une approche envisageable dans le futur est de se baser sur la progression le long du trajet, par exemple en évaluant le nombre de points de cheminement atteints.

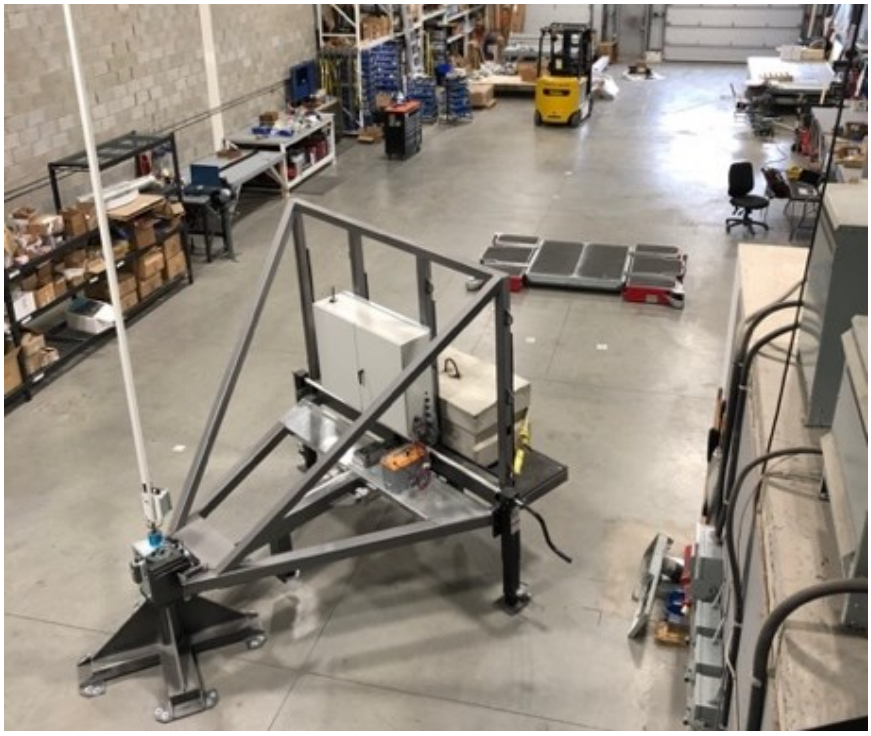

Figure 6.3 Environnement de test chez Inogec

Une fois les poids des diverses fonctions de coût fixés, la performance de la pile de navigation ROS-H a été évaluée en conditions réelles chez Inogec afin de déterminer si elle était avantageuse par rapport à la pile de navigation originale. Étant donné la disposition de l'espace, seul le comportement dans un espace ouvert a pu être évalué. La figure 6.3 montre l'environnement de test et la figure 6.4 la carte produite par le SLAM. Deux séries de déplacements entre les extrémités de l'espace libre au centre de l'atelier était commandée à l'AGV VII. Le tableau 5 indique les poids associés aux différentes fonctions de coût lors des tests.

Tableau 5 Poids utilisés lors des tests avec l'AGV VII

Série	Poids			
	Goal	Plan	Obs	Yaw
A	30	38	0.005	0
B	30	38	0.005	6

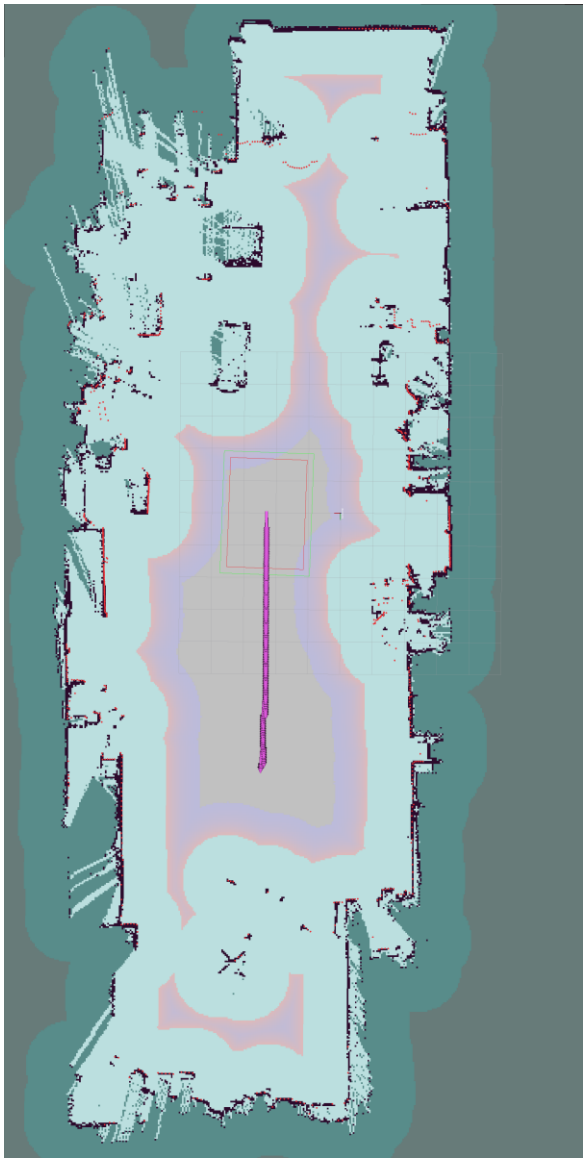

Figure 6.4 Carte produite par le SLAM

Lors de la première série de déplacements, nommée A, le poids associé au respect de l'orientation calculée était nul pour que l'AGV VII se comporte comme s'il était commandé par la pile de navigation originale (**Yaw** = 0). La figure 6.5 montre le trajet du véhicule lors du test. Bien que le véhicule ait atteint ses buts, il a effectué de nombreuses rotations inutiles en chemin. La figure 6.6 illustre bien ce comportement puisque les vitesses en y et en θ sont aussi élevées que la vitesse en x alors que l'AGV VII se déplace le long d'une ligne droit où il pourrait circuler avec son côté étroit devant

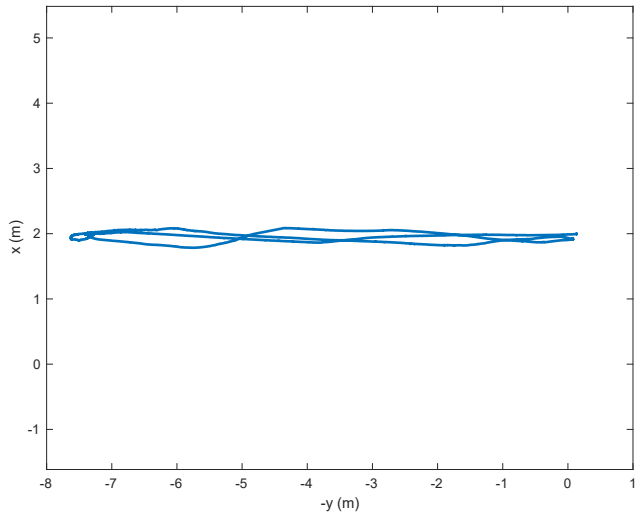

Figure 6.5 Trajet lors de la série A

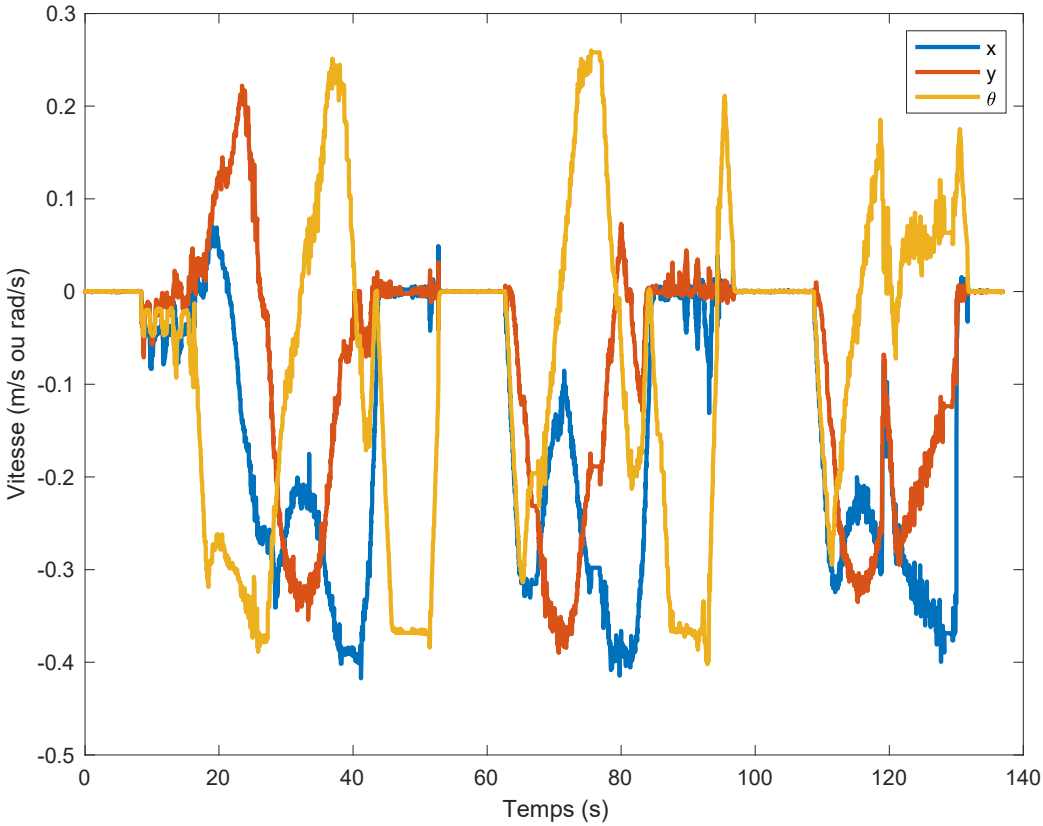

Figure 6.6 Vitesses lors de la série A

La figure 6.7 montre le trajet du véhicule lors du test de la deuxième série de déplacement, nommée B, avec **Yaw** = 0 afin que les contributions présentées aient un effet. Cette fois, l'essentiel de la vitesse est en x, ce qui démontre un comportement acceptable. Les vitesses restantes en y et en θ correspondent aux rotations pour que l'AGV VII demeure tangent au déplacement lors des virages.

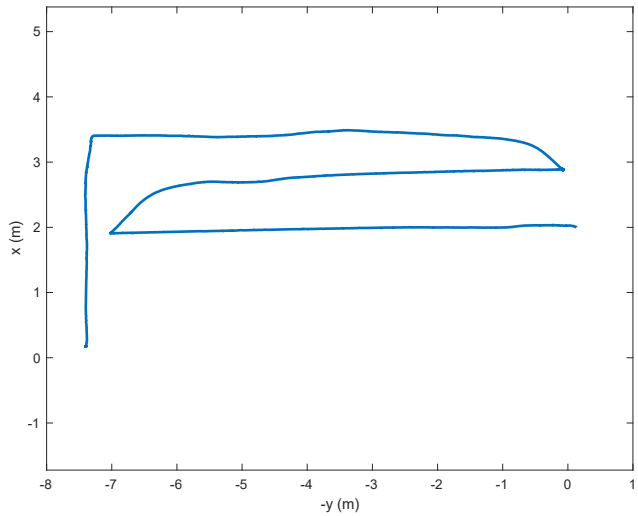

Figure 6.7 Trajet lors de la série B

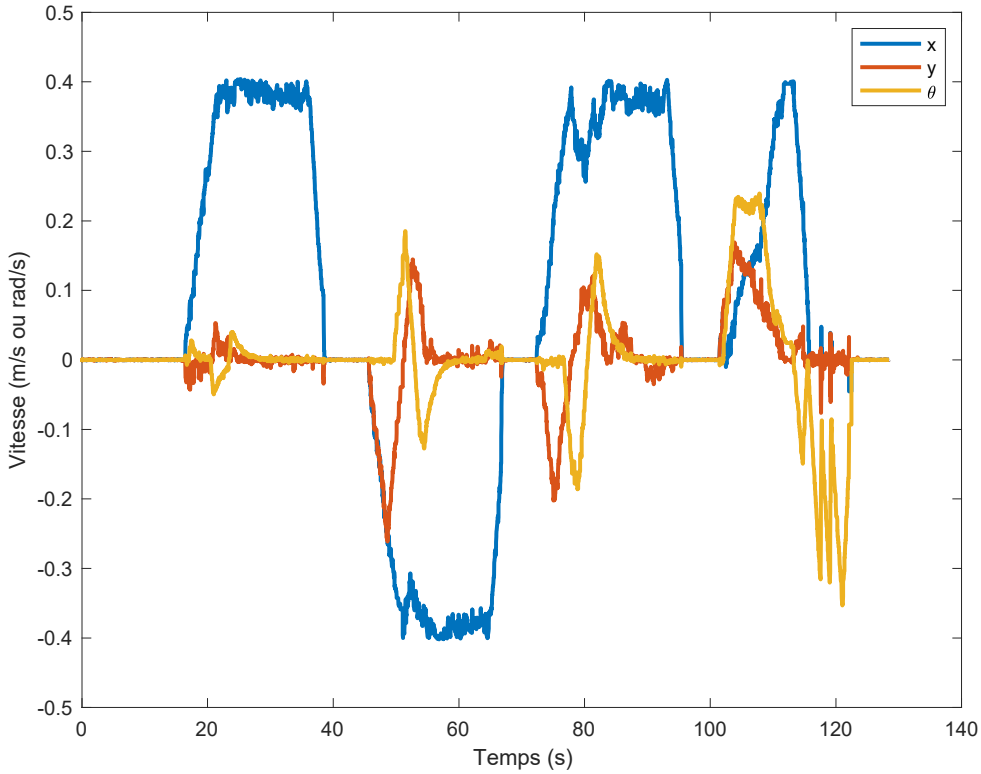

Figure 6.8 Vitesses lors de la série B

Ces tests suggèrent que le concept d'ajouter une fonction de coût pour commander l'orientation du robot à partir de l'orientation spécifiée par le planificateur global est viable en conditions réelles et conduit à un meilleur comportement du véhicule. Le tableau 6 présente une synthèse des vitesses moyennes lors des deux séries de déplacements ainsi que la distance angulaire parcourue. Il montre l'augmentation de la vitesse en x et la diminution des vitesses en y et θ observées dans les graphiques. La réduction de la distance angulaire parcourue témoigne également de l'efficacité de l'approche présentée.

Tableau 6 Résumé des tests en conditions réelles

Série	$\overline{v_x}$ (m/s)	$\overline{v_y}$ (m/s)	$\overline{v_\theta}$ (rad/s)	θ (rad)
A	0.1311	0.0901	0.1237	17.02
B	0.1862	0.0248	0.0367	4.748

Un problème a également fait surface lors des tests avec le véhicule réel. Pour des raisons de sécurité, l'AGV VII est doté de capteurs LiDARs qui limitent la vitesse de déplacement du véhicule ou l'immobilise selon la distance de l'obstacle le plus près. La vitesse de déplacement du véhicule détermine la distance minimale à respecter par rapport à un obstacle. C'est un comportement qui n'était pas reproduit dans le simulateur et qui cause parfois des déplacements erratiques en conditions réelles. En effet, les grilles d'occupation sont fixes peu importe la vitesse du robot. Aussi, les limites cinématiques sur lesquelles reposent DWA sont données et ne dépendent pas de la proximité des obstacles.

Il arrive donc que la vitesse commandée par le planificateur local ne soit pas admissible du point de vue des LiDARs qui bloquent alors le déplacement. La vitesse est alors plus faible, les LiDARs quittent le mode de ralenti et l'AGV VII accélère brutalement à la vitesse commandée par DWA qui déclenche immédiatement un nouveau freinage. Ces variations brutales de vitesse ne sont pas acceptables pour un AGV industriel. La solution temporaire utilisée pour la suite des travaux est donc de choisir une limite cinématique acceptable par les LiDARs à la distance minimale des obstacles autorisée par la pile de navigation. Cela a l'inconvénient de ne pas utiliser la pleine vitesse du véhicule, mais les déplacements sont beaucoup plus fluides.

6.2 Tests en simulation

La performance de la pile de navigation ROS-H a ensuite été validée dans plusieurs scénarios avec l'environnement de simulation Gazebo. La figure 6.9 montre l'environnement chargé dans le simulateur. Il est grossièrement inspiré d'un environnement industriel avec une aire de travail et des zones d'entreposage. C'est un espace vaste, mais avec certains endroits exigus pour valider le comportement du véhicule dans des espaces restreints. Comme lors du test en conditions réelles, la figure 6.10 montre la carte produite par le SLAM.

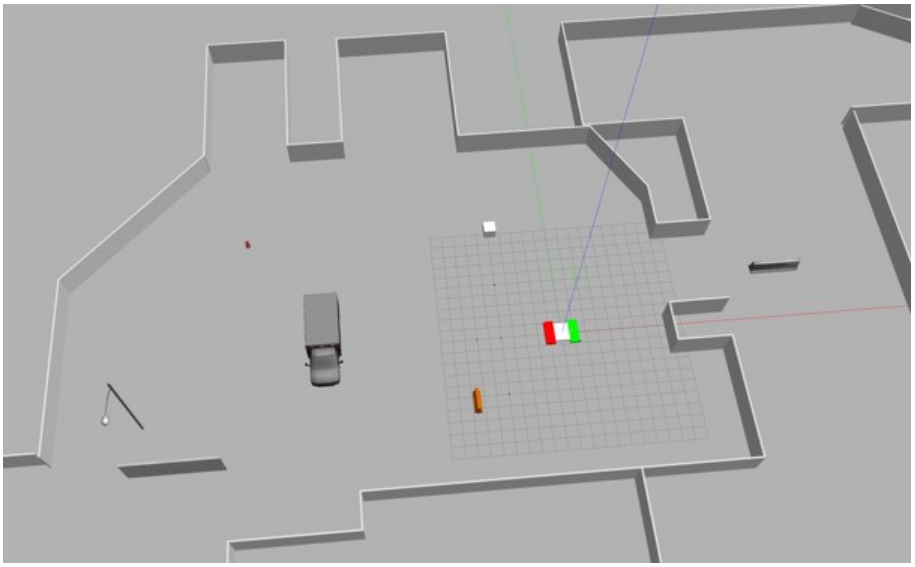

Figure 6.9 Environnement de simulation

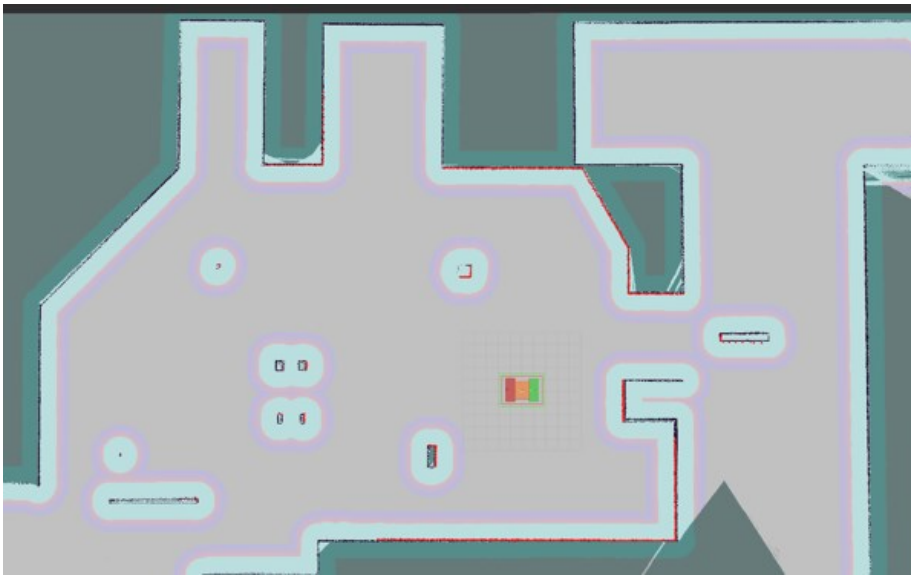

Figure 6.10 Carte produite par le système SLAM

Étant donné les limitations du planificateur local identifiées à la section 6.1, deux ensembles de poids ont été utilisés lors des simulations selon le scénario en cours tels que donnés par le tableau 7. La série de poids 4 priorise l'atteinte du but et la série 9 le suivi du trajet.

Tableau 7 Poids utilisés lors des tests en simulation

Série	Poids		
	Goal	Plan	Obs
4	30	28	0.005
9	28	32	0.005

Quatre scénarios sont étudiés pour valider le bon comportement de l'AGV lorsqu'il effectue différentes tâches :

1. La circulation dans un espace ouvert représente le déplacement d'une charge entre deux postes d'assemblage dans une aire de travail dégagée. L'environnement n'impose pas de contraintes, mais l'AGV ne doit pas pivoter inutilement et circuler avec son côté étroit devant;
2. La circulation d'un mur à l'autre représente le déplacement de l'AGV entre un espace de stationnement et un point de recharge. Ici, le véhicule doit s'éloigner et s'approcher du mur avec la bonne orientation pour éviter les collisions;
3. La circulation entre deux cavités représente le déplacement d'une charge entre deux zones d'entreposage. L'AGV doit coordonner sa vitesse en x et en θ afin de conserver le côté étroit devant lors des virages.
4. Le stationnement dans un espace exiguë représente l'échange d'une charge avec un autre équipement de manutention. Le trajet combine des espaces ouverts, des murs et des virages afin d'atteindre les limites du logiciel de navigation.

6.2.1 Circulation dans un espace ouvert

Ce scénario reproduit le test en conditions réelles réalisé chez Inogec. L'AGV doit circuler de l'espace vert vers l'espace rougeâtre puis rejoindre l'espace orangé en repassant par l'espace vert. La figure 6.11 montre les buts commandés. Aucune rotation n'est nécessaire, mais il est préférable que l'AGV pivote pour orienter son côté étroit devant lorsqu'il circule entre les espaces vert et orangé. La série de poids 4 est utilisée pour ce scénario puisque le

véhicule n'a pas à suivre rigoureusement le trajet du planificateur global pour éviter de se bloquer contre un obstacle.

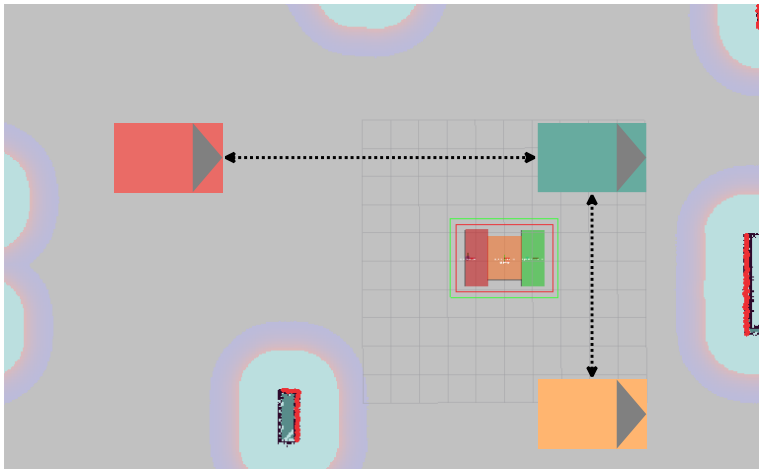

Figure 6.11 Trajet dans un espace ouvert

La figure 6.12 et la figure 6.13 montrent le déplacement de l'AGV lors des essais sans (**Yaw** = 0) et avec (**Yaw** = 6) poids associé à l'orientation. Le trajet parcouru est très similaire dans les deux cas.

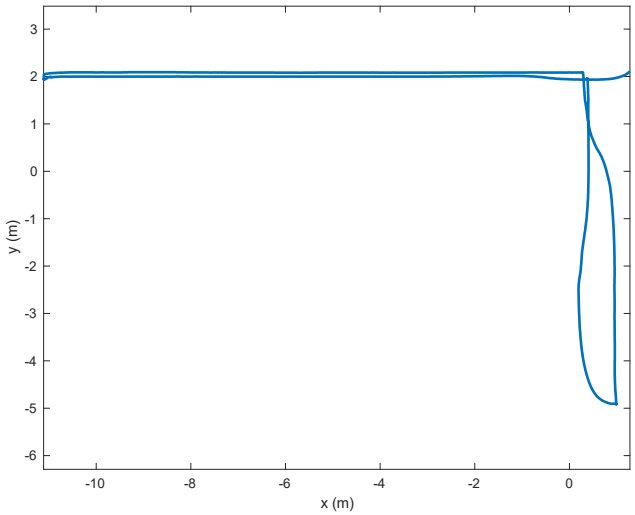

Figure 6.12 Déplacement dans un espace ouvert sans poids sur l'orientation

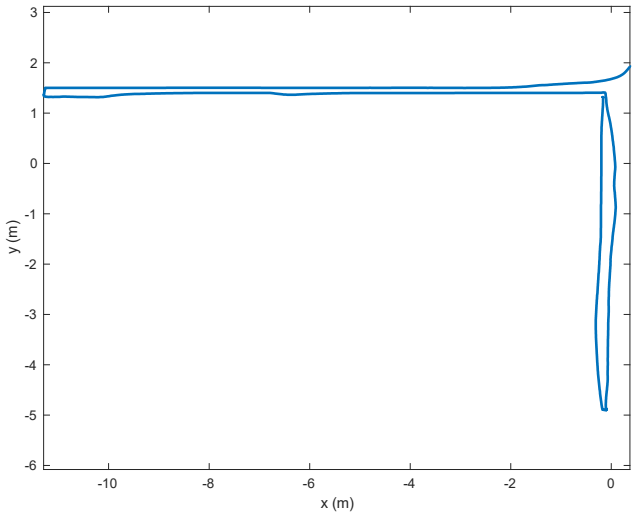

Figure 6.13 Déplacement dans un espace ouvert avec poids sur l'orientation

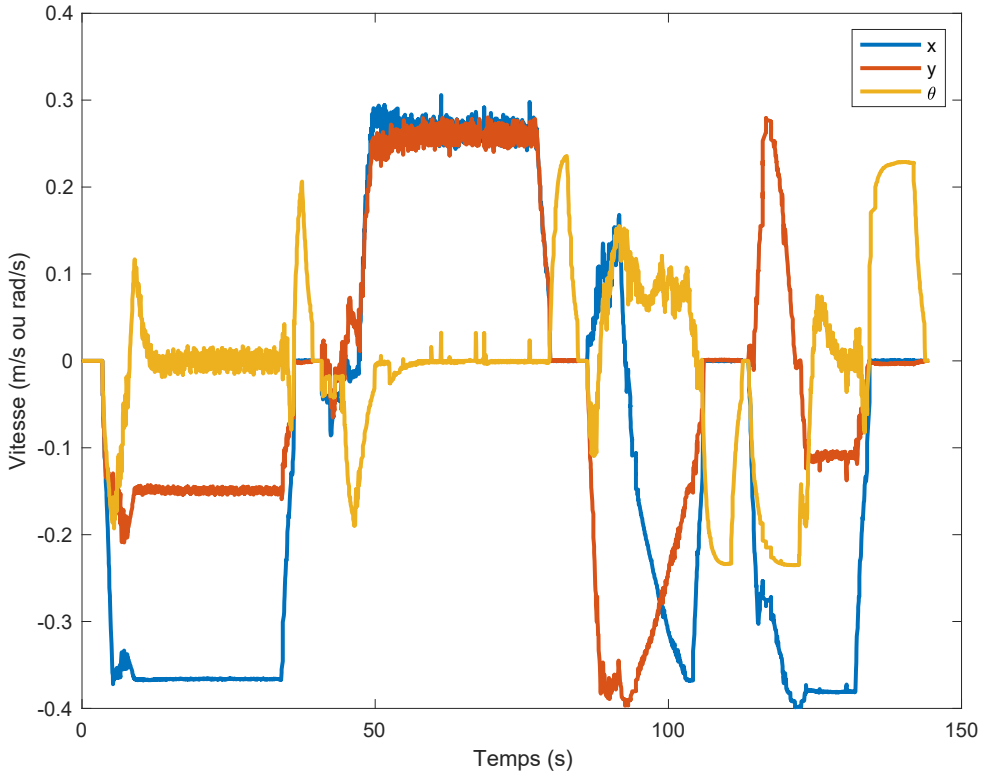

Figure 6.14 Vitesses dans un espace ouvert sans poids sur l'orientation

La figure 6.14 montre les vitesses lors de l'essai sans poids associé à l'orientation (**Yaw** = 0). La courbe de vitesse en y orange montre bien que l'AGV ne place pas systématiquement

son côté étroit devant puisqu'il se déplace également en y plutôt que seulement en x. À l'inverse, lors de l'essai avec un poids associé à l'orientation (**Yaw** = 6), la figure 6.15 montre bien la réduction de la vitesse en y lorsque l'orientation tangente au mouvement est respectée.

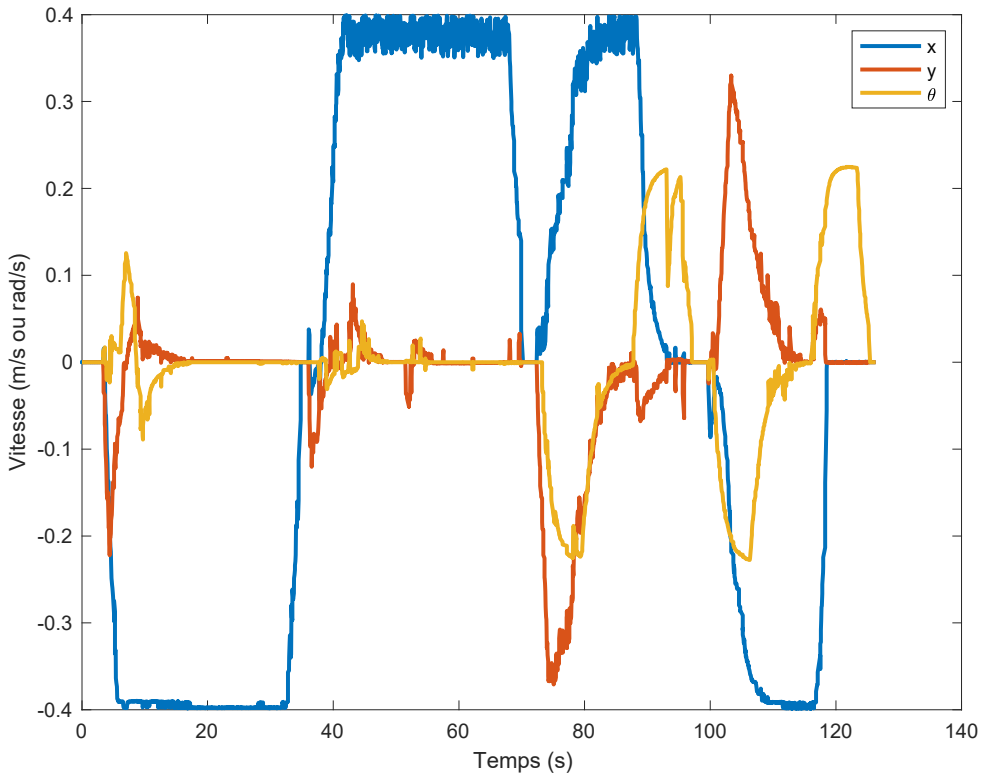

Figure 6.15 Vitesse dans un espace ouvert avec un poids sur l'orientation

6.2.2 Circulation d'un mur à l'autre

Lors de ce scénario, l'AGV doit démarrer en position de stationnement le long d'un mur puis rejoindre un espace de stationnement le long d'un autre mur. Une rotation de 90° est requise entre les deux espaces. La figure 6.16 présente le scénario. Il n'y a pas d'obstacles le long du trajet, alors la série de poids 4 est utilisée.

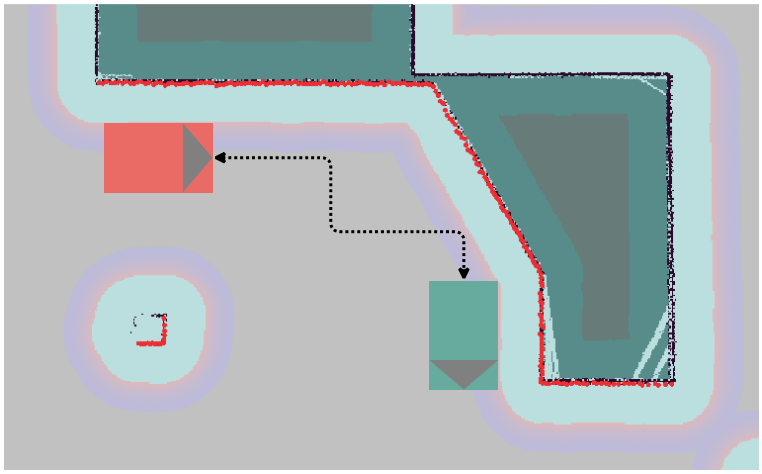

Figure 6.16 Trajet mur à mur

La figure 6.17 et la figure 6.18 montrent le déplacement de l'AGV lors des essais sans (**Yaw** = 0) et avec (**Yaw** = 6) poids associé à l'orientation. Le trajet parcourut est très similaire dans les deux cas.

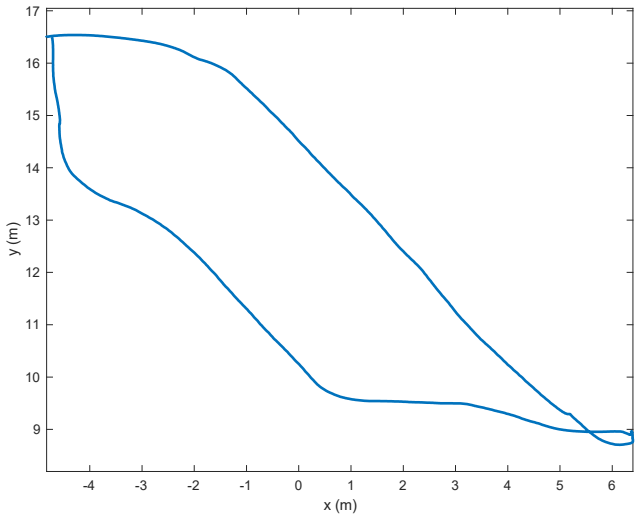

Figure 6.17 Déplacement entre deux murs sans poids sur l'orientation

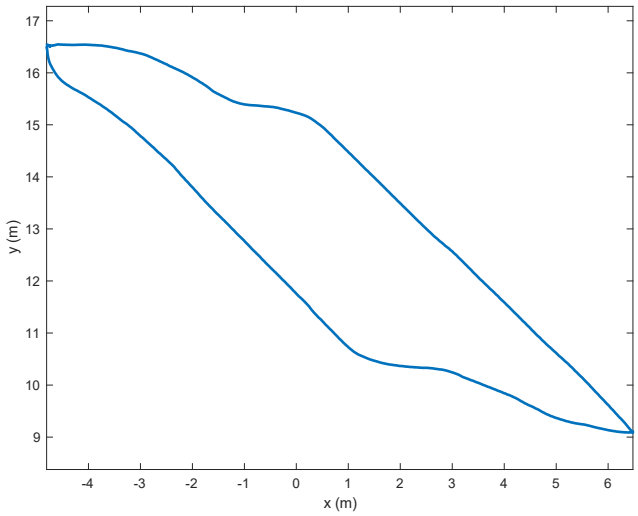

Figure 6.18 Déplacement entre deux murs avec un poids sur l'orientation

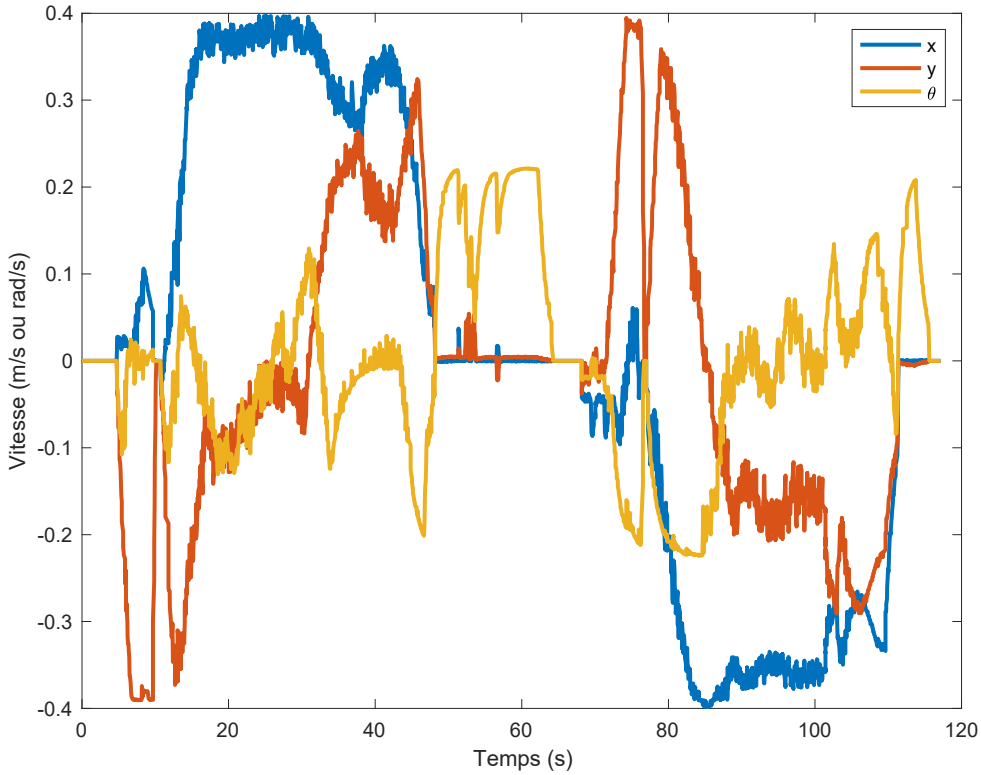

Figure 6.19 Vitesses entre deux murs sans poids sur l'orientation

La figure 6.19 montre les vitesses lors de l'essai sans poids sur l'orientation (**Yaw** = 0). Bien que la vitesse en x domine, la vitesse en y demeure présente. La vitesse en θ suggère

également plusieurs rotations réparties tout au long du trajet. La figure 6.20 montre les vitesses lors de l'essai avec un poids associé à l'orientation (**Yaw** = 6). La baisse de la vitesse en y est bien visible et les rotations se limitent aux rotations requises pour garder l'AGV tangent à son déplacement. En effet, la vitesse en θ concorde avec la vitesse en y.

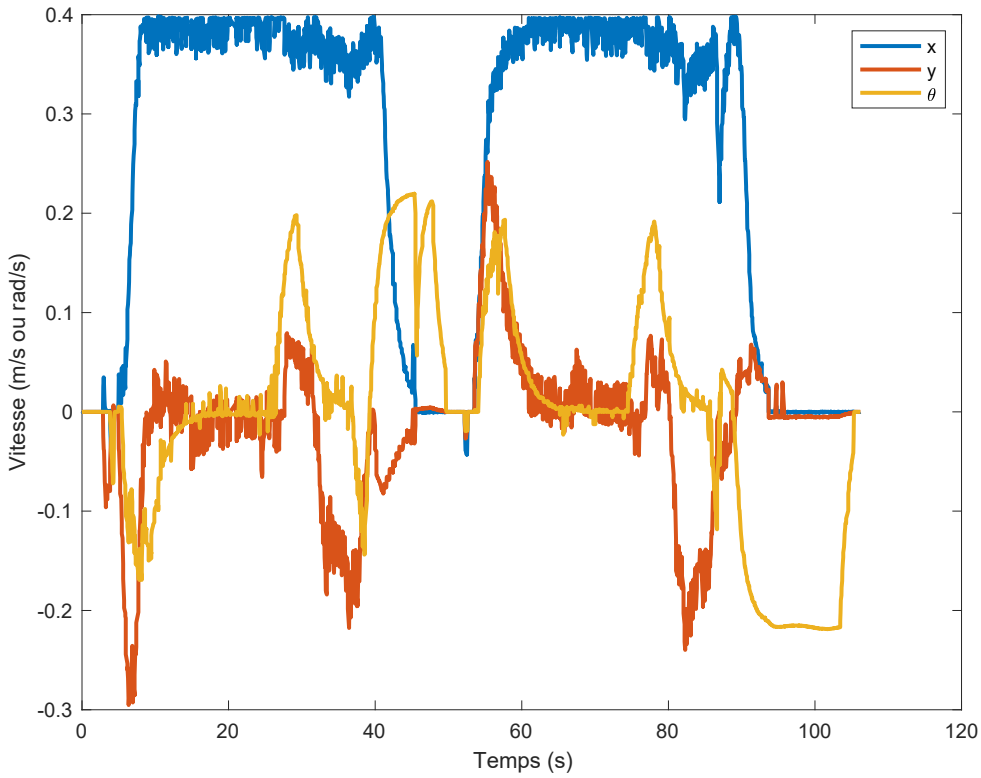

Figure 6.20 Vitesses entre deux murs avec un poids sur l'orientation

6.2.3 Circulation entre deux cavités

Ce scénario, montré à la figure 6.21, est très semblable au précédent. Toutefois, l'AGV doit contourner une péninsule et effectuer un virage de 180°. Puisque l'espace est assez dégagé, le scénario débute avec la série de poids 4. Toutefois, lors du premier essai, l'AGV se bloque contre le coin de la péninsule puisque le planificateur local tente de court-circuiter le trajet calculé par le planificateur global. La figure 6.22 montre l'AGV coincé près du mur et le trajet calculé en mauve qui contournait l'obstacle. La série de poids 9 est donc retenue pour la suite du scénario. Les poids sont modifiés en obtenant la version correspondante du fichier

de paramètres ROS à partir de git avant le démarrage de la pile de navigation au début de chaque test.

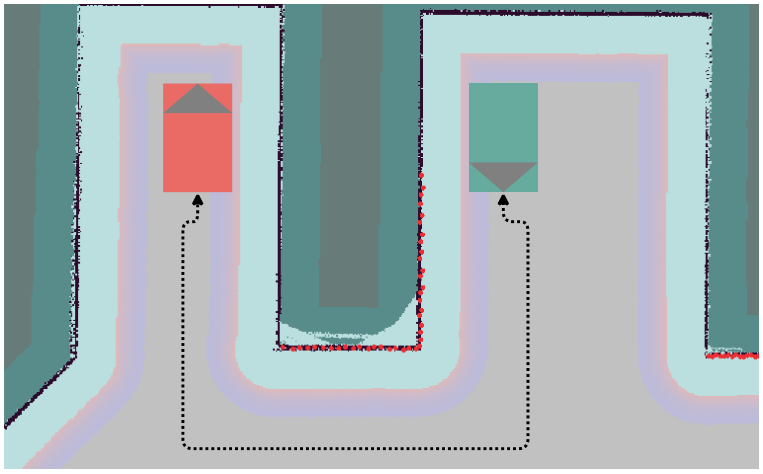

Figure 6.21 Trajet entre deux cavités

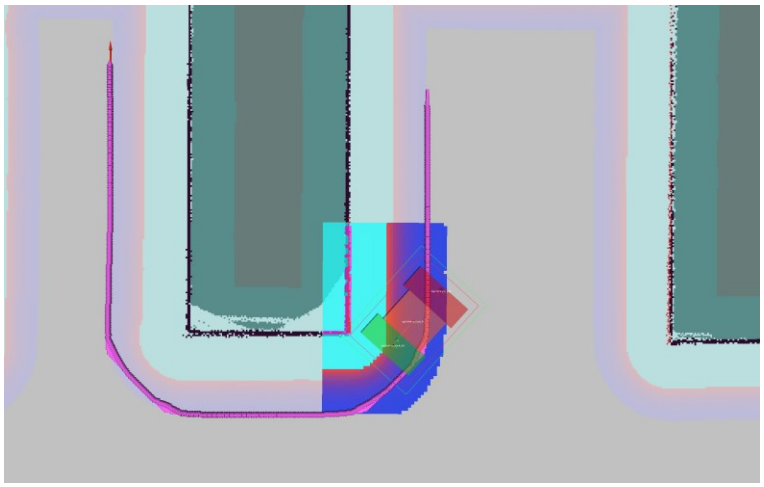

Figure 6.22 Collision avec la péninsule

Au point de départ le long du mur, la série 9 avec **Yaw** = 0 n'incite pas l'AGV à bouger et il reste sur place. Les résultats sont donc obtenus seulement avec la série 9 et **Yaw** = 6. La figure 6.23 montre le trajet parcouru lors de l'essai et la figure 6.24 les vitesses. Cet essai montre que le comportement optimal de l'AGV peut être atteint avec la pile de navigation proposée. En effet, le véhicule maintient sa vitesse maximale en x pour toute la durée du trajet. La vitesse en θ se limite aux virages et la vitesse en y est très faible.

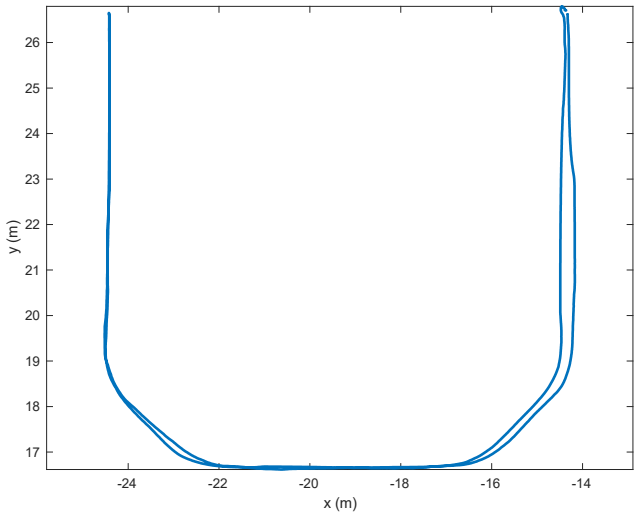

Figure 6.23 Déplacement entre deux cavités avec un poids sur l'orientation

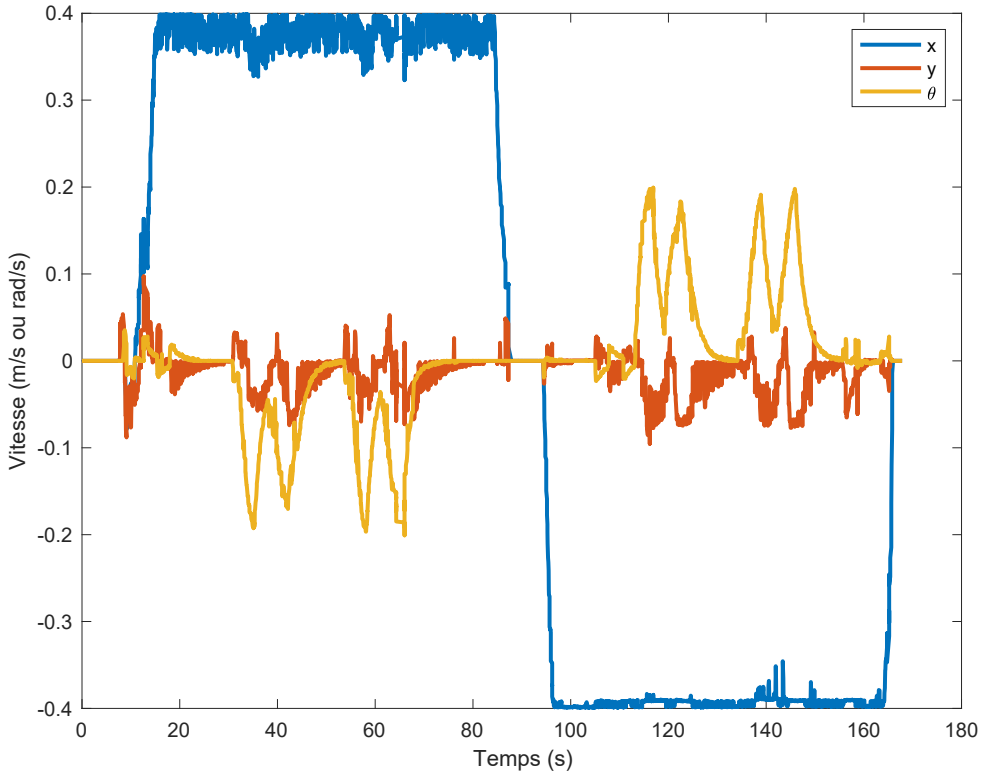

Figure 6.24 Vitesses entre deux cavités avec un poids sur l'orientation

6.2.4 Stationnement dans un espace restreint

Ce scénario est divisé en deux tests. La figure 6.25 montre le déplacement que doit effectuer l'AGV. Lors du premier test, le véhicule démarre à l'emplacement vert et doit se stationner sur l'emplacement rougeâtre entouré de murs. Le deuxième test consiste à quitter l'emplacement rougeâtre, puis à rejoindre l'emplacement vert. Les deux cas requièrent une rotation de 180°. Étant donné sa complexité, ce scénario est tenté avec la série 9 puis la série 4 de poids.

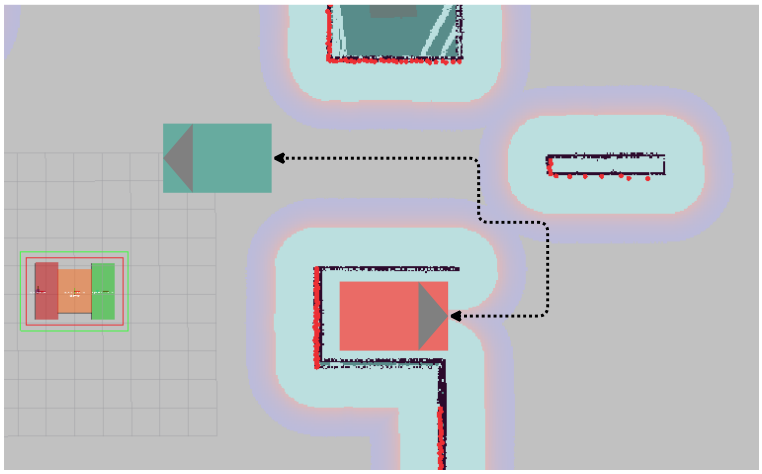

Figure 6.25 Trajet dans un espace restreint

Lors de l'insertion dans l'espace de stationnement avec la série de poids 9 et **Yaw** = 0, l'AGV se bloque contre le coin du mur puisqu'il n'adopte pas la bonne orientation. La figure 6.26 montre la collision. Avec **Yaw** = 6, le véhicule s'immobilise dans le virage puisque les fonctions de coûts sont contradictoires entre le virage, le trajet et le but local. La figure 6.27 montre l'AGV immobilisé. Avec la série 4 et **Yaw** = 0, l'AGV demeure au point de départ. Avec **Yaw** = 6, le planificateur local permet à l'AGV de dévier du trajet et il entre en collision avec le mur. La figure 6.28 montre l'AGV contre le mur et le trajet du planificateur global, en mauve, qui contourne l'obstacle.

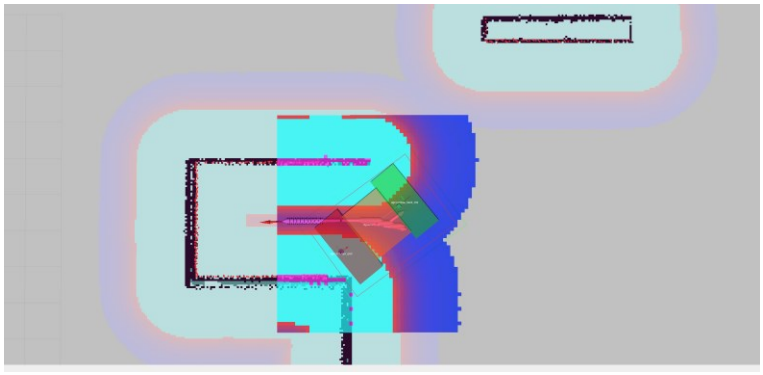

Figure 6.26 Insertion série 9 avec **Yaw** = 0

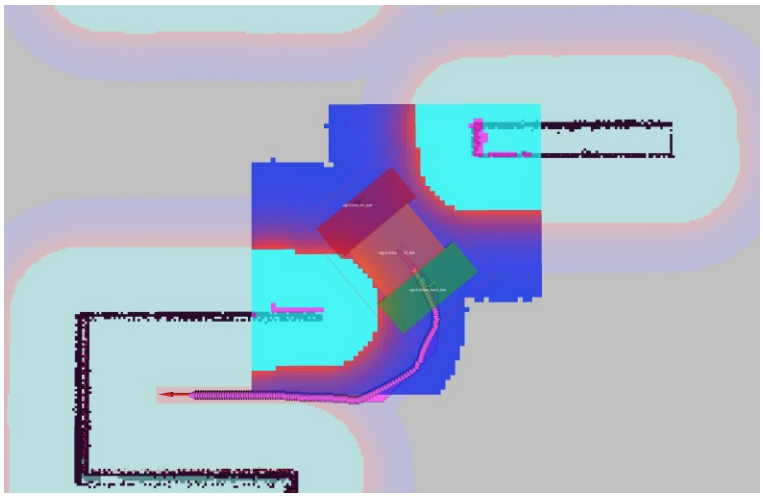

Figure 6.27 Insertion série 9 avec **Yaw** = 6

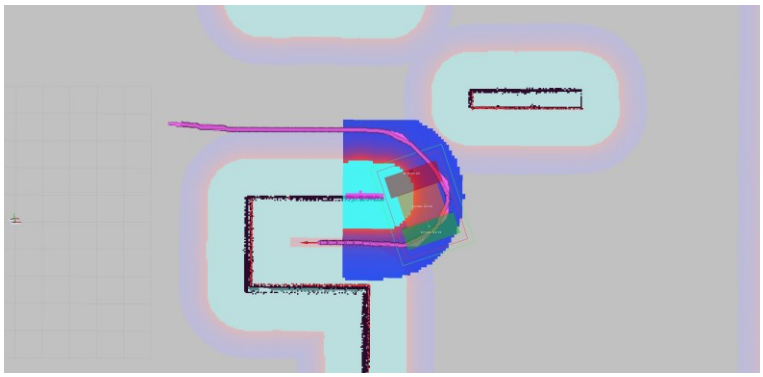

Figure 6.28 Insertion série 4 avec **Yaw** = 6

Lors de la sortie de l'espace de stationnement, la série de poids 4 bloque l'AGV contre le mur en déviant du trajet planifié peu importe le poids associé à l'orientation. La figure 6.29 montre l'AGV contre le mur et le trajet prévu en mauve qui contourne l'obstacle.

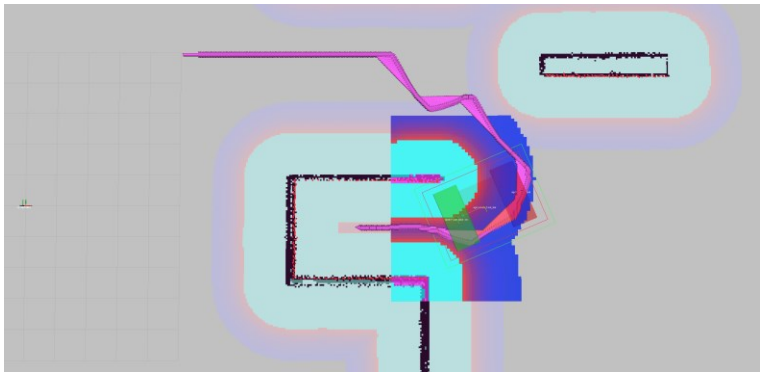

Figure 6.29 Blocage lors de la sortie avec la série 4

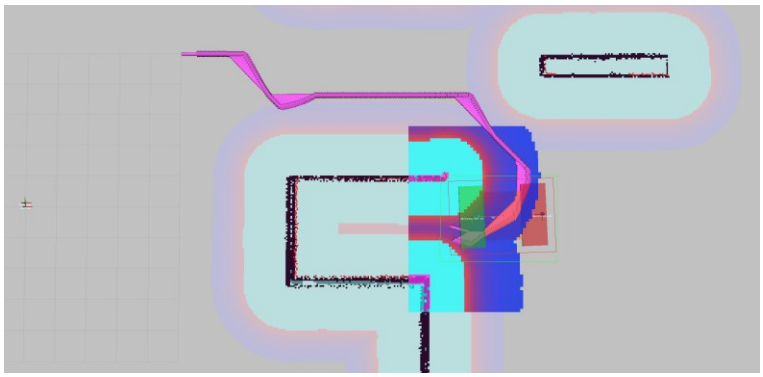

Figure 6.30 Collision lors de la sortie avec la série 9 sans poids sur l'orientation (**Yaw** = 0)

Avec la série 9 et **Yaw** = 0, l'AGV ne dévie pas du trajet prévu mais se bloque tout de même contre l'obstacle puisqu'il n'adopte pas la bonne orientation. La figure 6.30 montre la collision. L'ajout d'un poids sur l'orientation permet au véhicule de quitter l'espace de stationnement avec succès. La figure 6.31 montre le trajet parcouru. La figure 6.32 montre les vitesses pendant le scénario. C'est un autre cas où la pile de navigation ROS-H améliore le comportement. En plus d'éviter la collision, la vitesse en x est maintenue au maximum de la capacité du véhicule, la vitesse en y est faible et la vitesse en θ est utilisée seulement pour compléter les virages.

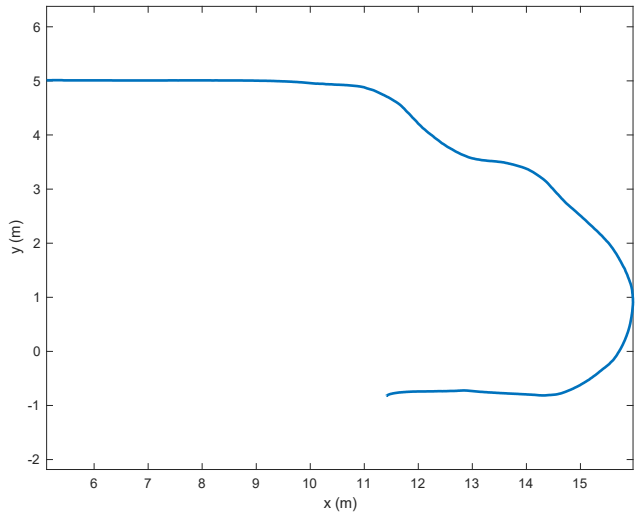

Figure 6.31 Déplacement hors du stationnement

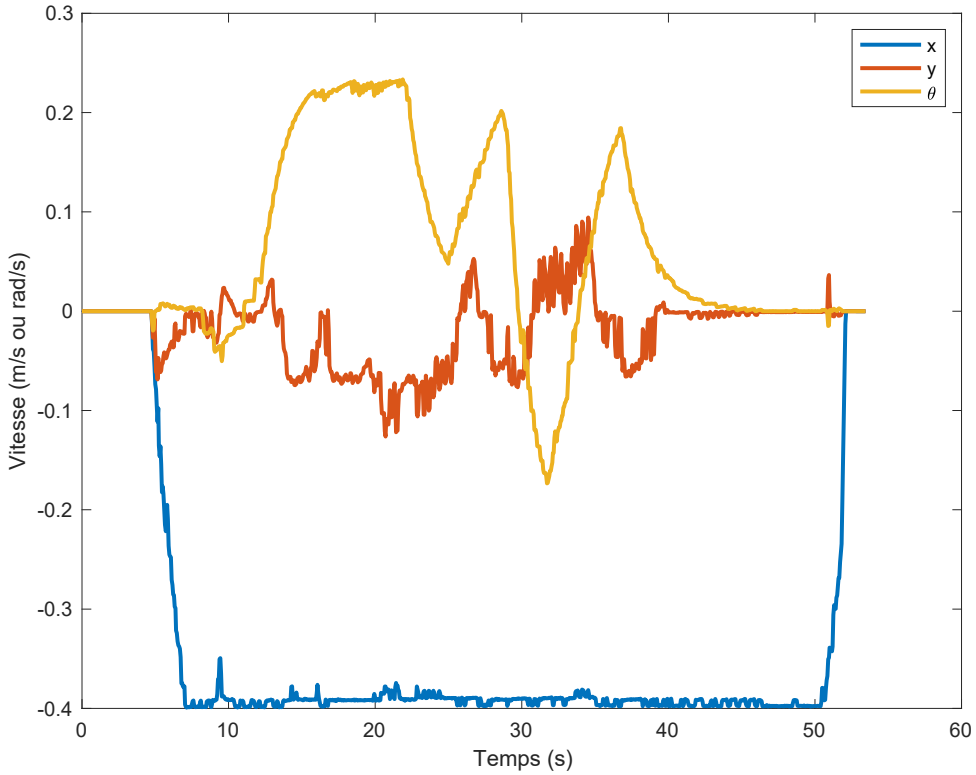

Figure 6.32 Vitesses lors de la sortie du stationnement

6.2.5 Sommaire des scénarios simulés

Le tableau 8 présente les vitesses moyennes de déplacement et la rotation totale effectuée par l'AGV lors des différents scénarios simulés. La mention Gel indique que le véhicule est demeuré sur place et la mention Collision indique un blocage contre un obstacle.

Tableau 8 Vitesses et rotation lors des scénarios simulés

Scénario	Série de poids	Yaw	$\overline{v_x}$ (m/s)	$\overline{v_y}$ (m/s)	$\overline{v_\theta}$ (rad/s)	θ (rad)
Espace ouvert	4	0	0.2087	0.1427	0.0673	9.7071
		6	0.2655	0.0408	0.0523	6.6043
Mur à mur	4	0	0.1952	0.1267	0.0795	9.3231
		6	0.2536	0.0457	0.0776	8.2727
Cavité à cavité	4	6	Collision			
		0	Collision			
	9	0	Gel			
		6	0.1311	0.0901	0.1237	17.02
Insertion dans le stationnement	4	0	Gel			
		6	Collision			
	9	0	Collision			
		6	Collision			
Sortie du stationnement	4	0	Collision			
		6	Collision			
	9	0	Collision			
		6	0.3361	0.0272	0.0761	4.0801

Les lignes avec une case verte sont les combinaisons d'un poids et d'un scénario où les tests suggèrent que l'ajout de la gestion de l'orientation (colonne orientation) présente un avantage. Cet avantage est démontré par une réduction de la vitesse moyenne en y, une réduction de la distance angulaire parcourue et une augmentation de la vitesse moyenne en x. Il est à noter qu'il n'y a aucun cas où l'ajout de la gestion de l'orientation a dégradé la performance. Les scénarios suggèrent que le pile de navigation ROS-H améliore le comportement d'un AGV holonome libre si les poids du planificateur local sont adaptés à la situation. Malheureusement, les poids ne sont pas partagés entre les espaces ouverts et les espaces restreints. L'insertion dans un espace de stationnement exiguë positionné après un

virage semble également impossible étant donné le compromis dans l'ajustement du planificateur local expliqué à la section 6.1 : la priorité sur le trajet (Plan > Goal) cause le blocage de l'AGV lorsque le trajet et le but du planificateur local sont contradictoires, par exemple pour contourner l'extrémité d'un mur; la priorité sur le but (Plan < Goal) cause la collision de l'AGV avec un obstacle lorsque l'espace entre la position actuelle de l'AGV et le but du planificateur local est occupé puisque le véhicule coupe le trajet calculé qui contourne l'obstacle.

6.3 Démonstration de la gestion des règles de circulation

Les figures 6.33 à 6.35 montrent les tests effectués dans Gazebo avec le modèle simplifié de l'AGV et une carte représentant un vaste espace ouvert. Ils suggèrent la viabilité de la preuve de concept de la méthode de gestion des règles de circulation présentée au chapitre 5.

La figure 6.33, où l'orientation dans le couloir a été fixée à une valeur arbitraire, montre que l'AGV suit la consigne de l'utilisateur. En a), les flèches mauves, qui représentent les points du trajet produit par le planificateur global, pointent vers la direction du déplacement de haut en bas. En b), les flèches pointent dans la direction de l'orientation indiquée par l'utilisateur pour la portion du trajet qui traverse la zone d'orientation spécifiée. L'AGV respecte cette orientation en s'écartant de l'orientation tangente pour s'aligner avec les flèches.

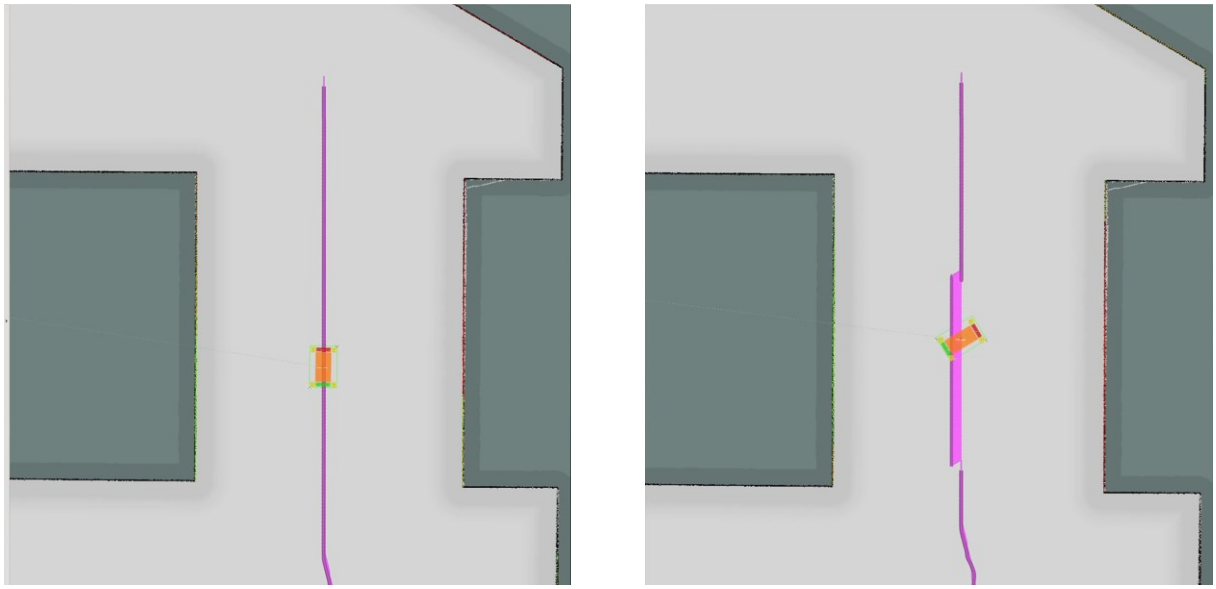

Figure 6.33 Démonstration de l'orientation spécifiée : a) orientation originale, tangente au déplacement; b) orientation spécifiée par l'utilisateurSur la figure 6.34, des zones de circulation préférées ont été ajoutées pour que l'AGV demeure au centre de l'espace libre plutôt que de couper le coin en s'approchant du mur. La figure 6.34 b) montre cette déviation tandis que l'AGV s'approche du coin pour réduire la longueur du trajet original montré en a).

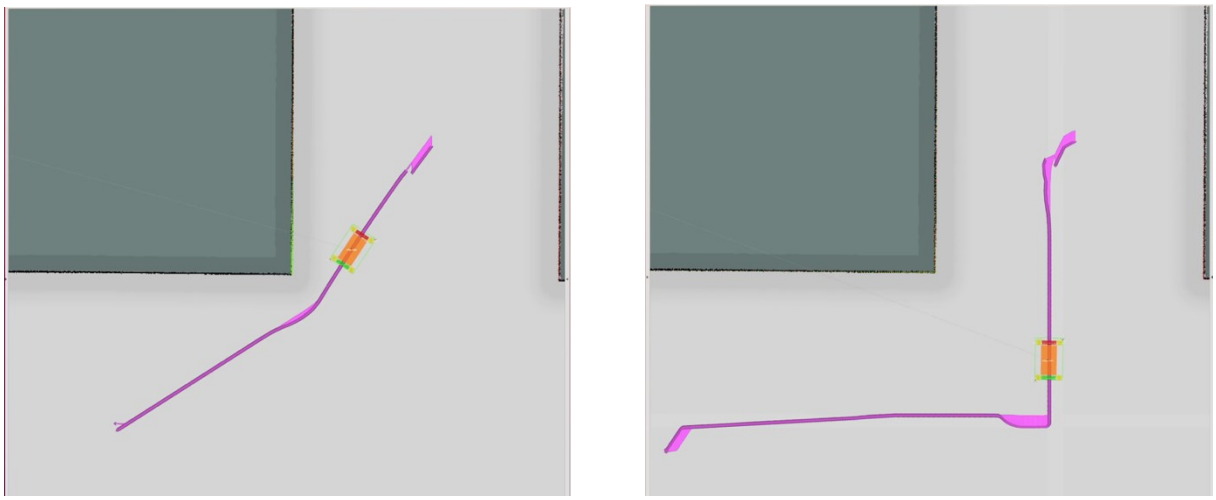

Figure 6.34 Démonstration des zones de circulation préférées : a) trajet original, qui coupe le coin; b) zone de circulation préférée spécifiée par l'utilisateur

Sur la figure 6.35, une zone de circulation interdite a été ajoutée pour empêcher la circulation dans la zone en bas de l'obstacle. La figure de gauche montre le trajet par défaut, plus court, qui passe en bas du mur, tandis que celle de droite montre que l'AGV respecte la consigne et prend un détour par le haut.

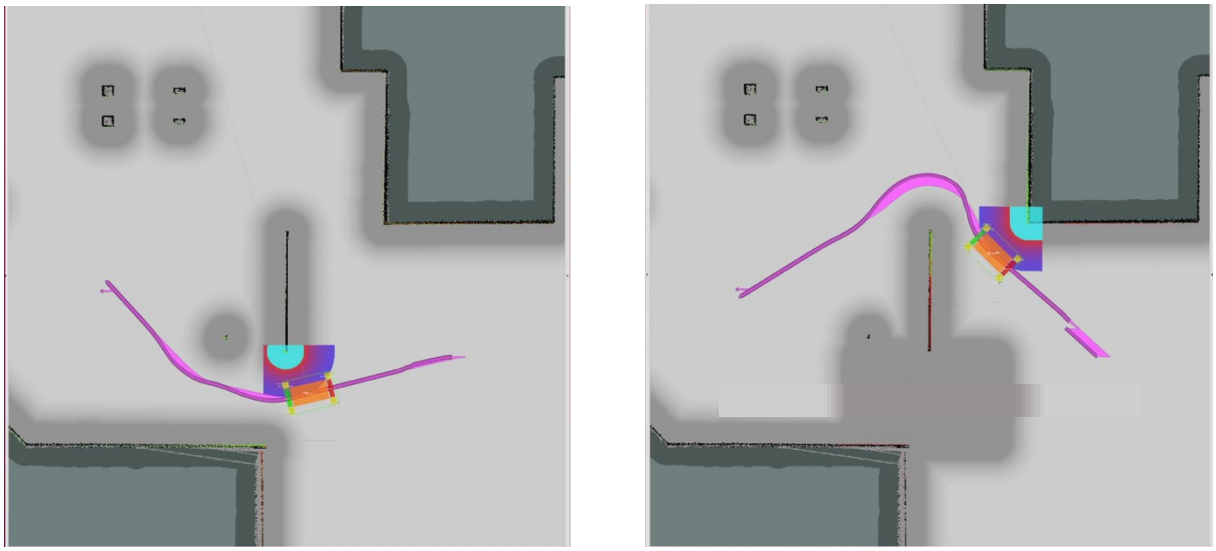

Figure 6.35 Démonstration des zones de circulation interdites : a) Trajet original, plus court; b) trajet plus long, qui contourne la zone interdite par l'utilisateur

Ces tests suggèrent que l'encodage des règles de circulation dans des grilles d'occupation est viable. En effet, trois types de zones ont été encodées et utilisées avec succès par un AGV en simulation.

CHAPITRE 7

CONCLUSION

Les travaux de recherche présentés suggèrent qu'il est possible d'adapter la pile de navigation ROS-NH pour commander l'orientation d'un AGV holonome à parcours libre de manière optimale. La démonstration d'une méthode pour assurer le respect des règles de circulation propre à un milieu d'opération est également réalisée. Le graphe utilisé par l'algorithme A* est augmenté d'intervalles d'orientation afin de tenir compte de ce paramètre lors de la recherche d'un trajet. Ce graphe augmenté est utilisé par le planificateur global développé qui produit un trajet optimal composé de couples (x, y, θ) entre le point de départ et la destination. Un important travail d'optimisation du graphe a été réalisé afin de maintenir le temps de planification dans l'ordre de quelques secondes afin que le logiciel soit utilisable dans un environnement de production industrielle. Le planificateur local existant de la pile de navigation ROS-NH est bonifié d'une nouvelle fonction de coût pour l'évaluation des trajectoires candidates. Cette fonction de coût permet de considérer l'orientation dictée par le planificateur global lors du calcul des vitesses à transmettre à la base mobile. Cette considération assure le respect de l'orientation par l'AGV lors de ses déplacements. Toutefois, un conflit inhérent à la conception du planificateur local original demeure, car l'atteinte du but et le suivi exact du tracé du trajet est un compromis : les deux objectifs sont contradictoires.

Enfin, une preuve de concept permettant le respect des règles de circulation établies est développée. Cette preuve de concept comprend un encodage des règles dans des grilles d'occupation, un serveur de carte pour le stockage et la modification des grilles, un planificateur global qui exploite l'information contenue dans les grilles et une interface graphique pour l'entrée des règles par l'utilisateur.

Tous les logiciels développés sont livrés sous la forme de paquets ROS utilisant les interfaces standards. Ils se divisent en trois paquets :

- *ino_planner*, le planificateur global qui implémente l'interface *nav_core::BaseGlobalPlanner* pour être chargé par *plugin_lib* lors de la configuration de la pile de navigation ROS-NH;
- *dwa_local_planner*, le planificateur local qui est une version modifiée du paquet original qui modifie directement la pile de navigation RPS;
- *user_map*, le paquet qui fournit les outils pour la gestion des règles de circulation par l'utilisateur.

Il serait pertinent que des travaux futurs se penchent sur le conflit entre le but et le trajet dans le planificateur local. La fonction de coût qui cherche à minimiser la distance avec le but pourrait vraisemblablement être remplacée par une fonction de coût qui cherche à maximiser la distance parcourue le long du trajet. Le planificateur global devrait également être modifié afin de fonctionner avec les grilles de règles de circulation, puisque ces dernières existent présentement dans une structure parallèle. Enfin, lors des tests avec le vrai véhicule, il arrivait fréquemment que la pile de navigation commande des vitesses à l'AGV VII que les capteurs LiDARs de sécurité ne jugeaient pas admissibles. Les diverses zones de sécurités entourant les véhicules autonomes en milieux industriels devraient être prises en compte lors des prochains développements.

LISTE DES RÉFÉRENCES

[1] S. G. Kumbhar, R. B. Thombare, et A. B. Salunkhe, « Automated guided vehicles for small manufacturing enterprises: A review », *SAE International Journal of Materials and Manufacturing*, vol. 11, n° 3, p. 253-258, sept. 2018.

[2] G. Fedorko, S. Honus, et R. Salai, « Comparison of the traditional and autonomous AGV systems », dans *Proceedings of the MATEC Web of Conferences*, 2017, vol. 134, p. 00013.

[3] A. Kelly, B. Nagy, D. Stager, et R. Unnikrishnan, « An infrastructure-free automated guided vehicle based on computer vision », *IEEE Robotics & Automation Magazine*, vol. 14, p. 24-34, 2007.

[4] F. Ferland, L. Clavien, J. Frémy, D. Létourneau, F. Michaud, et M. Lauria, « Teleoperation of AZIMUT-3, an omnidirectional non-holonomic platform with steerable wheels », dans *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, oct. 2010, p. 2515-2516.

[5] S. Chamberland, É. Beaudry, L. Clavien, F. Kabanza, F. Michaud, et M. Lauria, « Motion planning for an omnidirectional robot with steering constraints », dans *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, oct. 2010, p. 4305-4310.

[6] A. S. Kundu, O. Mazumder, R. Chattaraj, et S. Bhaumik, « Door negotiation of a omni robot platform using depth map based navigation in dynamic environment », dans *Proceedings of the International Conference on Contemporary Computing*, août 2014, p. 176-181.

[7] F. Künemund, D. Heß, et C. Röhrig, « Energy efficient kinodynamic motion planning for holonomic AGVs in industrial applications using state lattices », dans *Proceedings of the International Symposium on Robotics*, juin 2016, p. 8.

[8] B. Siciliano et O. Khatib, *Springer Handbook of Robotics*. Cham, Switzerland: Springer, 2016.

[9] N. J. Nilsson, « A mobile automaton: An application of artificial intelligence techniques », Defense Technical Information Center, Fort Belvoir, VA, janv. 1969.

[10] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, et A. Ng, « ROS an open-source Robot Operating System », dans *Proceedings of the ICRA workshop on open source software*, mai 2009, p. 6.

[11] E. Marder-Eppstein, E. Berger, T. Foote, B. Gerkey, et K. Konolige, « The Office Marathon: Robust navigation in an indoor office environment », dans *Proceedings of the IEEE International Conference on Robotics and Automation*, mai 2010, p. 300-307.

[12] V. V. Unhelkar, S. Dorr, A. Bubeck, P. A. Lasota, J. Perez, H. C. Siu, J. C. Boerkoel, Q. Tyroller, J. Bix, S. Bartscher, et J. A. Shah, « Mobile robots for moving-floor assembly lines: design, evaluation, and deployment », *IEEE Robotics & Automation Magazine*, vol. 25, n° 2, p. 72-81, juin 2018.

[13] B. Siciliano, L. Sciavicco, L. Villani, et G. Oriolo, *Robotics*. London: Springer London, 2009.

[14] I. Doroftei, V. Grosu, et V. Spinu, « Omnidirectional mobile robot - Design and implementation », dans *Bioinspiration and Robotics Walking and Climbing Robots*, M. K, Éd. I-Tech Education and Publishing, 2007.

[15] Yong Liu, Xiaofei Wu, J. Jim Zhu, et Jae Lew, « Omni-directional mobile robot controller design by trajectory linearization », dans *Proceedings of the American Control Conference*, 2003, vol. 4, p. 3423-3428.

[16] A. J. Gmerek, A. Plastropoulos, P. Collins, M. Kimball, A. Wheatley, J. Liu, P. T. Karfakis, K. Shah, J. Carroll, et G. S. Virk, « A novel holonomic mobile manipulator robot for construction sites », dans *Proceedings of the International Conference on Climbing and Walking Robots and Support Technologies for Mobile Machines*, 2018, p. 10.

[17] L. Xie, C. Scheifele, W. Xu, et K. A. Stol, « Heavy-duty omni-directional Mecanum-wheeled robot for autonomous navigation: System development and simulation realization », dans *Proceedings of the IEEE International Conference on Mechatronics*, mars 2015, p. 256-261.

[18] C. Stachniss, J. J. Leonard, et S. Thrun, « Simultaneous localization and mapping », dans *Springer Handbook of Robotics*, B. Siciliano et O. Khatib, Éd. Cham: Springer International Publishing, 2016, p. 1153-1176.

[19] S. Thrun, « Robotic mapping: A survey », dans *Exploring Artificial Intelligence in the New Millenium*, p. 31.

[20] J. Fuentes-Pacheco, J. Ruiz-Ascencio, et J. M. Rendón-Mancha, « Visual simultaneous localization and mapping: A survey », *Artificial Intelligence Review*, vol. 43, n° 1, p. 55-81, janv. 2015.

[21] M. Labbé et F. Michaud, « RTAB-Map as an open-source LiDAR and visual simultaneous localization and mapping library for large-scale and long-term online operation », *Journal of Field Robotics*, vol. 36, n° 2, p. 416-446, mars 2019.

[22] G. Grisetti, C. Stachniss, et W. Burgard, « Improved techniques for grid mapping with rao-blackwellized particle filters », *IEEE Transactions on Robotics*, vol. 23, n° 1, p. 34-46, févr. 2007.

[23] D. Fox, W. Burgard, F. Dellaert, et S. Thrun, « Monte Carlo localization: Efficient position estimation for mobile robots », dans *Proceedings of the National Conference on Artificial Intelligence and Eleventh Conference on Innovative Applications of Artificial Intelligence*, juill. 1999, p. 7.

[24] L. P. N. Matias, T. C. Santos, D. F. Wolf, et J. R. Souza, « Path planning and autonomous navigation using AMCL and AD* », dans *Proceedings of the Latin American Robotics Symposium and Brazilian Symposium on Robotics*, oct. 2015, p. 320-324.

[25] S. Kohlbrecher, O. von Stryk, J. Meyer, et U. Klingauf, « A flexible and scalable SLAM system with full 3D motion estimation », dans *Proceedings of the IEEE International Symposium on Safety, Security, and Rescue Robotics*, nov. 2011, p. 155-160.

[26] W. Hess, D. Kohler, H. Rapp, et D. Andor, « Real-time loop closure in 2D LIDAR SLAM », dans *Proceedings of the IEEE International Conference on Robotics and Automation*, mai 2016, p. 1271-1278.

[27] W. Y. Loong, L. Z. Long, et L. C. Hun, « A star path following mobile robot », dans *Proceedings of the International Conference on Mechatronics*, mai 2011, p. 1-7.

[28] S. Kuswadi, J. W. Santoso, M. Nasyir Tamara, et M. Nuh, « Application SLAM and path planning using A-Star algorithm for mobile robot in indoor disaster area », dans *Proceedings of the International Electronics Symposium on Engineering Technology and Applications*, oct. 2018, p. 270-274.

[29] B. Lau, C. Sprunk, et W. Burgard, « Efficient grid-based spatial representations for robot navigation in dynamic environments », *Robotics and Autonomous Systems*, vol. 61, n° 10, p. 1116-1130, oct. 2013.

[30] C. Sprunk, B. Lau, P. Pfaff, et W. Burgard, « An accurate and efficient navigation system for omnidirectional robots in industrial environments », *Autonomous Robots*, vol. 41, n° 2, p. 473-493, févr. 2017.

[31] M. Dakulovic, C. Sprunk, L. Spinello, I. Petrovic, et W. Burgard, « Efficient navigation for anyshape holonomic mobile robots in dynamic environments », dans *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, nov. 2013, p. 2644-2649.

[32] A. Stentz, « Optimal and efficient path planning for unknown and dynamic environments », Carnegie Mellon University, Pittsburgh, PA, Technique, 1993.

[33] D. Fox, W. Burgard, et S. Thrun, « The dynamic window approach to collision avoidance », *IEEE Robotics & Automation Magazine*, p. 11, mars 1997.

[34] P. Saranrittichai, N. Niparnan, et A. Sudsang, « Robust local obstacle avoidance for mobile robot based on dynamic window approach », dans *Proceedings of the International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology*, mai 2013, p. 1-4.

[35] S. Quinlan et O. Khatib, « Elastic bands: Connecting path planning and control », dans *Proceedings of the IEEE International Conference on Robotics and Automation*, 1993, p. 802-807.

[36] C. Rösmann, W. Feiten, T. Wösch, F. Hoffmann, et T. Bertram, « Trajectory modification considering dynamic constraints of autonomous robots », dans *ROBOTIK: German Conference on Robotics*, mai 2012, p. 6.

[37] C. Rosmann, F. Hoffmann, et T. Bertram, « Planning of multiple robot trajectories in distinctive topologies », dans *Proceedings of the European Conference on Mobile Robots*, sept. 2015, p. 1-6.

[38] C. Sprunk, B. Lau, P. Pfaffz, et W. Burgard, « Online generation of kinodynamic trajectories for non-circular omnidirectional robots », dans *Proceedings of the IEEE International Conference on Robotics and Automation*, mai 2011, p. 72-77.

[39] D. H. Douglas et T. K. Peucker, « Algorithms for the reduction of the number of points required to represent a digitized line or its caricature », dans *Classics in Cartography*, M. Dodge, Éd. Chichester, UK: John Wiley & Sons, Ltd, 2011, p. 15-28.

[40] F. Künemund, C. Kirsch, D. Heß, et C. Röhrig, « Fast and accurate trajectory generation for non-circular omni-directional robots in industrial applications », dans *Proceedings of the ROBOTIK: German Conference on Robotics*, mai 2012, p. 6.

[41] C. Sprunk, J. Röwekämper, G. Parent, L. Spinello, G. D. Tipaldi, W. Burgard, et M. Jalobeanu, « An experimental protocol for benchmarking robotic indoor navigation », dans *Experimental Robotics*, vol. 109, M. A. Hsieh, O. Khatib, et V. Kumar, Éd. Cham: Springer International Publishing, 2016, p. 487-504.

[42] D. Bortot, M. Born, et K. Bengler, « Directly or on detours? How should industrial robots approximate humans? », dans *Proceedings of the ACM/IEEE International Conference on Human-Robot Interaction*, mars 2013, p. 89-90.

[43] C. Lichtenthäler et A. Kirsch, « Towards legible robot navigation - How to increase the intend expressiveness of robot navigation behavior », dans *Proceedings of the International Conference on Social Robotics*, 2013, p. 7.

[44] A. D. Dragan, K. C. T. Lee, et S. S. Srinivasa, « Legibility and predictability of robot motion », dans *Proceedings of the ACM/IEEE International Conference on Human-Robot Interaction*, mars 2013, p. 301-308.

[45] S. B. Banisetty et D. Feil-Seifer, « Towards a unified planner for socially-aware navigation », *arXiv e-prints*, p. 8, oct. 2018.

[46] S. S. Mehta, C. Ton, M. J. McCourt, Z. Kan, E. A. Doucette, et W. Curtis, « Human-assisted RRT for path planning in urban environments », dans *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, oct. 2015, p. 941-946.

[47] J. D. Gammell, S. S. Srinivasa, et T. D. Barfoot, « Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic », dans *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, sept. 2014, p. 2997-3004.

[48] S. M. LaValle et J. J. Kuffner, « Randomized kinodynamic planning », dans *Proceedings of the IEEE International Conference on Robotics and Automation*, 1999, vol. 1, p. 473-479.

[49] P. Hart, N. Nilsson, et B. Raphael, « A formal basis for the heuristic determination of minimum cost paths », *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, n° 2, p. 100-107, 1968.