Navigation Autonome d'un Robot Mobile en Environnement Dynamique et Incertain

Frédéric Large

THÈSE

préparée à

L'Institut National de Recherche
en Informatique et en Automatique

en vue d'obtenir le titre de

Docteur de l'Université de Savoie
(arrêté ministériel du 30 mars 1992)

Spécialité
INFORMATIQUE

par

Frédéric LARGE

Navigation Autonome D'un Robot Mobile en Environnement Dynamique et Incertain

Directeur de Thèse : Christian Laugier

Soutenue le 5 novembre 2003 devant le jury composé de

Président :	**M. Flavio Oquendo**
Rapporteurs :	**M. Rachid Alami**
	M. Roland Siegwart
Examinateurs :	**M. Christian Laugier**
	M. Sepanta Sekhavat

Remerciements

Je remercie tout d'abord les membres du jury pour leur grande disponibilité malgré des emplois du temps très chargés : Flavio Oquendo, professeur à l'*Ecole Supérieure d'Ingénieurs d'Annecy* (ESIA), qui a accepté de présider mon jury, ainsi que Rachid Alami, directeur de recherche au *Laboratoire d'Analyse et d'Architecture des Systèmes* (LAAS), et Roland Siegwart, professeur à l'*Ecole Polytechnique Fédérale de Lausanne* (EPFL), qui ont accepté tous deux d'être mes rapporteurs. Je remercie également Michel Parent, responsable du programme *La Route Automatisée* (LaRA) à l'INRIA, pour m'avoir fait confiance et octroyé une bourse de thèse.

Je tiens à remercier tout particulièrement Zvi Shiller, professeur au *College of Judea and Samaria* en Israël, pour ses conseils éclairés sur les $\mathcal{V}$*-Obstacles*. Ses visites au sein de l'équipe *Sharp/e-Motion* et les longues discussions qu'elles ont permis m'ont été d'une aide capitale.

Il est difficile et serait trop long de remercier individuellement toutes les personnes que j'ai eu le privilège de cotoyer durant ces années de thèse. Je tiens cependant à leur exprimer mes plus sincères remerciements à tous car une thèse est avant tout une expérience humaine. J'ai notamment une pensée émue pour les membres du projet *Sharp/e-Motion* au sein duquel les relations de travail se sont rapidement transformées en liens d'amitié parfois très forts. Je conserve le souvenir de personnes d'une grande richesse, tant sur le plan professionnel que personnel et auprès desquelles j'ai beaucoup appris.

Je souhaite remercier en premier lieu Christian Laugier, professeur et directeur de recherche à l'INRIA, pour m'avoir accueilli au sein de cette équipe exceptionnelle, ainsi que pour sa patience et son soutien. Il a été et reste bien plus qu'un directeur de thèse. Je remercie également Sepanta Sekhavat, chargé de recherche, pour sa gentillesse, pour nos discussions parfois agitées mais toujours très enrichissantes, et son aide précieuse tout au long de la thèse jusqu'aux dernières minutes. Je voudrais aussi exprimer mes remerciements à mon complice Jorge Hermosillo-Valadez, docteur depuis peu, avec qui j'ai partagé le bureau, mais surtout des moments inoubliables, de cogitation intense sur des problèmes sérieux, et de franche camaraderie sur des sujets bien plus légers... Enfin, je voudrais remercier Thierry Fraichard, chargé de recherche, pour ses conseils et son immense patience. J'ai pour toutes ces personnes de cœur une profonde admiration.

Je n'oublie pas non-plus le trio Mariama-Véro-Soraya à l'énergie débordante, parfois épuisante, mais toujours stimulante, ainsi que toute l'équipe des moyens robotiques pour leur extrême gentillesse et leur disponibilité.

Je voudrais également exprimer mon immense gratitude envers mes parents et ma famille qui, sans nécessairement partager mon enthousiasme pour la recherche, m'ont depuis toujours fait confiance et ont répondu présents dans les bons moments comme dans les moins bons. Marise, Jean-Pierre, Corinne, Henriette, René, Monique et Daniel sont quelques prénoms qui me sont particulièrement chers et à qui je dois le fait d'en être arrivé là. Mon seul regret est d'avoir pris trop de temps pour que tous ne puissent assister au dénouement.

Et puisqu'il est de coutume de garder la meilleure pour la fin, je tiens à remercier tout particulièrement Priscilla, devenue ma femme pour mon plus grand bonheur au cours de ma (je devrais dire « notre ») thèse. J'ai pu éprouver et apprécier son immense patience en particulier durant la phase de rédaction de ce mémoire. Son soutien inconditionnel et permanent m'ont été d'une aide capitale et ô combien précieuse pour mener à bien ma mission et tenir la promesse faite à mon père. Un grand merci à toi ma princesse.

Table des matières

Introduction

Robots Mobiles Autonomes (RMA)

Depuis toujours l'Homme a su se doter d'**outils** performants pour prolonger sa main et réaliser des actions qui n'auraient pas été possibles sans. Cette évolution est motivée par la nécessité de satisfaire des besoins impératifs, mais également par l'envie de disposer d'un outil assez performant pour pouvoir le **seconder** voire le **remplacer** totalement dans toutes les tâches ingrates ou dangereuses.

Le terme « **robot** » correspond à cette idée d'esclave mécanique. D'après le Larousse, il s'agit d'un « appareil automatique pouvant se substituer à l'Homme pour exécuter diverses actions » . Créé par Karel Capek en 1927, le mot vient de « robuta » signifiant « travaux forcés » en tchèque. Dans sa pièce de théâtre « Rossum's Universal Robots », il désigne des êtres exploités par l'Homme (figure 1), contre lequel ils finissent par se révolter. La peur de la rébellion des robots restera d'ailleurs un sujet privilégié des auteurs de science fiction.

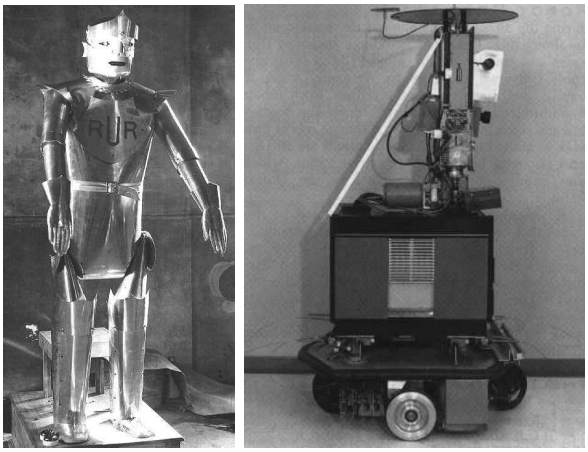

FIG. 1 – *Premiers robots* Une représentation d'un robot de R.U.R., le premier robot littéraire officiel (à gauche), et le premier robot autonome reconnu comme tel, Shakey (à droite).

Les robots de la littérature et du cinéma, à l'apparence plus ou moins humaine, sont capables d'**observer**, de **penser**, de **se déplacer** et d'**agir** de façon parfaitement auto-

nome. Ces copies en quelque sorte de l'homme relèvent cependant encore du domaine de l'imaginaire. Les robots actuels ont certes envahi notre quotidien, mais sous des formes bien moins évoluées. Leur étude, la robotique, à débuté avec l'ère de l'industrialisation.

Au début du XXème siècle, le développement de la robotique répond essentiellement à un besoin de production dans les usines. Les robots ne se déplacent pas ou seulement dans un espace connu et parfaitement contrôlé. Ils sont dépourvus de toute intelligence, et leur rôle se limite à l'accomplissement d'actions entièrement prédéfinies. Cette simplicité des robots, réduits à l'état d'automates, est rendue possible grâce à des environnements étudiés spécifiquement pour chaque robot et chaque application (figure 2). En contre-partie, tout imprévu implique un blocage du robot. En marge de ce mouvement apparaissent, dans les laboratoires de recherche, les premiers rares robots mobiles capables de réagir de manière simple et pré-conditionnée à un environnement moins structuré.

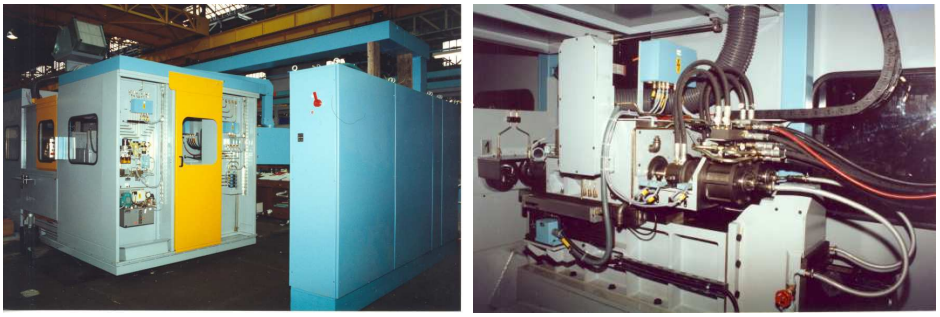

FIG. 2 – *Machine d'usinage* Les machines-outils modernes remplacent avantageusement l'Homme dans les tâches difficiles et laborieuses. Tout y est pensé, étudié et calculé dans le moindre détail pour qu'aucun imprévu n'ait sa place, et que la tâche du robot soit réduite à celle d'un automate. *(concepteur J-P. Large - photos Rouchaud SA)*

Avec l'arrivée de l'électronique, ils deviennent capables de suivre des lignes au sol ou de réagir à la lumière, à la chaleur ou à la présence d'un obstacle situé sur leur route. Ils restent toutefois plus proches des automates mécaniques du moyen-âge que des robots pensants de Capek.

C'est l'informatique qui va finalement permettre d'apporter aux robots « l'intelligence » qui leur manquait, c'est-à-dire la capacité de « **raisonner** » sur leur environnement. Le premier robot mobile capable de percevoir puis de modéliser son environnement pour décider seul de son prochain déplacement, est créé en 1967. Il est baptisé *Shakey* (figure 1). L'absence d'intervention humaine dans le processus de décision du robot lui confère la dénomination de "**robot autonome**".

Les premiers robots mobiles autonomes (RMA) ont en fait une autonomie très restreinte. Les moyens de perception et de calcul de *Shakey* par exemple, sont encore limités et planifier un mouvement lui prend des heures. De plus, tout changement dans l'environnement l'oblige à s'arrêter pour planifier un nouveau mouvement. Se déplacer en présence d'obstacles mobiles lui est par exemple impossible.

Les premières approches proposées pour permettre aux RMA de se « débrouiller » seuls, consistent à les « aider » en adaptant leur environnement comme pour les robots industriels. Les technologies et les techniques évoluant, les robots « voient » plus de choses plus rapidement, « pensent » davantage et plus vite, et l'exploration d'espaces moins travaillés et moins figés devient envisageable. Une réelle autonomie des robots offrirait de nombreuses perspectives et dans cet espoir, la robotique mobile autonome devient une discipline scientifique à part entière.

Les robots de service (pour aider dans les hôpitaux, les aéroports, les bureaux, les usines ou à domicile), les robots de maintenance pour les milieux dangereux ou difficiles d'accès pour l'Homme (fonds sous-marins, terrains minés, espace, centrales nucléaires), les robots de divertissement (comme les robots-chiens de Sony apparus récemment en provenance du Japon) ou encore les moyens de transports intelligents (véhicules autonomes ou systèmes d'assistance) sont quelques exemples d'applications potentielles des RMA.

Les résultats obtenus dans ces domaines au cours des dix dernières années témoignent des progrès considérables réalisés. Pourtant, depuis *Shakey*, peu de RMA sont réellement « sortis » des laboratoires pour se confronter au monde réel. La raison principale en est le nombre important de problèmes technologiques et scientifiques encore non-résolus pour des environnements *incertains* (informations incomplètes, bruitées, imprécises et/ou imprédictibles), *dynamiques* (présence d'obstacles mobiles, changeants, déformables et/ou déplaçables) et *ouverts* (virtuellement illimités). De telles caractéristiques définissent les milieux dans lequel l'Homme évolue et auxquels les RMA sont destinés.

Problème abordé dans ce document

Le problème abordé dans ce document est celui de la « *navigation autonome en environnement incertain et dynamique* ». L'objectif est de développer des modèles et des techniques informatiques permettant à un robot terrestre de se déplacer de manière autonome, c'est-à-dire sans intervention humaine, dans un environnement mal maîtrisé et en présence d'obstacles mobiles. Nous avons étudié deux thèmes importants :

Modélisation d'un environnement dynamique et incertain L'absence de connaissance a priori complète sur l'environnement impose l'utilisation de la perception. Locale et imparfaite, elle ne permet pas de lever toutes les incertitudes sur l'environnement (voir notion d'incertitude à la page 11).

En présence d'obstacles mobiles, cela implique de faire des hypothèses, notamment sur les trajectoires des obstacles (bornées dans le temps) et les mouvements autorisés du robot. Ces hypothèses pouvant s'avérer inexactes, l'actualisation des connaissances et la prise de décision doivent se faire de manière itérative et dans un temps court et borné dépendant de la tâche (nous parlerons de contrainte *temps-réel*). Pour assurer la sécurité du robot, ce temps de réponse devra être d'autant

plus court que les accélérations possibles des obstacles seront importantes, c'est-à-dire que l'environnement sera plus dynamique (voir page 11 pour notre définition de la dynamique).

Modèles et algorithmes pour la navigation autonome Les approches classiques de planification de trajectoire ne fonctionnent pas en environnement à la fois dynamique et incertain : les calculs sont trop nombreux et complexes pour être exécutés en temps-réel. Seules les méthodes purement réactives d'évitement d'obstacles sont applicables. Ces dernières sont cependant mal adaptées à la prise en compte de contraintes de tâche (e.g. atteindre une position donnée). Le chapitre 1 présente les plus représentatives.

Développer de nouvelles méthodes itératives est donc nécessaire pour permettre le calcul d'un déplacement sûr (sans collision) et la prise en compte de ces contraintes. D'autre part, il est important que ce déplacement soit exécuté par le robot conformément aux prévisions, ce qui nécessite de bien connaître les paramètres de contrôle du robot. Ce mémoire présente nos contributions principales sur ces thèmes.

Approche étudiée et contributions

Au vue des observations précédentes, nous avons scindé le problème de la navigation en environnement dynamique en trois tâches consécutives. Le caractère incertain de l'environnement implique que toutes doivent respecter des contraintes de temps-réel afin d'assurer la survie du robot. Ces tâches et les contributions que nous y avons apportées sont les suivantes :

modélisation d'un environnement dynamique Elle consiste à obtenir une représentation des obstacles statiques ou mobiles, utilisable pour la recherche de déplacements sûrs. Nous avons choisi de raisonner dans des espaces de dimension réduite pour maîtriser le nombre et la complexité des calculs : Il s'agit de l'espace $\mathcal{V}$ des vitesses linéaires instantanées du robot et de l'espace $\mathcal{V} \times \mathcal{T}$ des trajectoires rectilignes constantes engendrées par ses vitesses.

Notre première contribution est le ***concept de* $\mathcal{V}$-Obstacle Non-Linéaire** (abrégé en *NLVO*) défini dans ces espaces, et qui permet de calculer l'ensemble des déplacements sûrs du robot sur un intervalle de temps donné, ainsi que les temps avant impact des déplacements en collision sur ce même intervalle.

calcul d'un déplacement sûr vers un but Nous avons considéré une approche itérative qui prenne en compte des contraintes de tâche dans le processus de décision (i.e. en combinant les concepts précédents avec des méthodes de planification), tout en garantissant la sécurité du robot.

Nos contributions prennent la forme de deux méthodes dérivées des *NLVO* : Une ***méthode locale d'évitement des obstacles, et une méthode hybride***[1] ***de planification itérative de trajectoires sûres globales*** (i.e. jusqu'au but).

génération de la commande Elle a pour objectif de traduire un déplacement désiré en ordres compréhensibles par le robot pour en permettre l'exécution au plus proche. Nous avons envisagé une approche classique basée sur l'utilisation d'un modèle simplifié du robot et d'une loi de commande pour le suivi de trajectoire. Nous nous sommes alors intéressés au moyen d'améliorer le modèle en lui permettant de s'adapter par lui-même et dans une certaine mesure, aux caractéristiques du robot et à celles du terrain.

Nous avons utilisé un ***réseau de neurones artificiels pour apprendre les paramètres de commande*** du robot en ligne, et réduire ainsi les écarts entre le modèle et la réalité. L'originalité de notre approche réside dans la mise en place d'un système logiciel de surveillance, qui évite les problèmes liés habituellement à ce type d'approche en cas d'apprentissage insuffisant ou erroné.

Tous ces travaux ont été validés en simulation. Certains ont été portés sur la plateforme expérimentale Cycab (annexe C) utilisée par le projet *Sharp/e-Motion*[2] de l'INRIA[3] Rhône-Alpes, au sein duquel s'est déroulé cette thèse et dont la problématique principale porte sur l'autonomie de mouvements pour robots mobiles, qu'ils soient réels (voitures automatisées, robots de service) ou virtuels (avatars, mondes virtuels).

Organisation du rapport

Ce document comporte quatre chapitres encadrés par une introduction générale et une conclusion générale.

- Le chapitre 1 introduit les concepts de base de la robotique mobile autonome et la problématique étudiée dans ce rapport. Il présente les principaux travaux déjà réalisés sur ces thèmes. Enfin, il introduit notre approche et nos contributions qui seront développées dans les chapitres suivants.
- Les chapitres 2 et 3 présentent respectivement le concept de $\mathcal{V}$*-Obstacle Non-Linéaire* (*NLVO*) et deux fonctions dérivées de ce dernier. Il s'agit d'une méthode locale d'évitement d'obstacles mobiles (page 51) et d'une méthode de planification itérative de trajectoires sûres vers un but (page 68). Des résultats obtenus en simulation sont présentés pour chacune d'elles.

[1] Le terme hybride dénote ici le fait que l'approche combine une méthode locale d'évitement réactif des obstacles (pour assurer la survie du robot) avec une méthode globale de planification de trajectoire au but (pour prendre en compte la mission du robot).

[2] Serveur du projet *Sharp/e-Motion* : *http ://www.inrialpes.fr/sharp/*

[3] Institut National de Recherche en Informatique et en Automatique : *http ://www.inria.fr*

– Le chapitre 4 reprend les concepts présentés aux chapitres précédents dans l'optique d'une utilisation en situation réelle. Il s'agit notamment de modèles et algorithmes permettant l'exécution en temps-réel de ces méthodes et leur généralisation à des robots et des obstacles quelconques.

Ce document comporte des annexes où sont introduits des concepts ou des termes utilisés dans ce document. L'annexe A introduit les méthodes principales de modélisation de l'espace libre, auxquelles le chapitre 1 fait parfois référence. L'annexe D introduit les réseaux de neurones artificiels et en particulier ceux à fonctions de base radiale que nous avons utilisés au chapitre 4. Notre plate-forme robotique expérimentale, le Cycab et sa cinématique particulière sont présentées à l'annexe C. Enfin l'annexe B est la démonstration d'un calcul utilisé au chapitre 2.

Chapitre 1

Navigation autonome : Problématique et Approches

Ce chapitre présente dans un premier temps les différents concepts de base introduits dans l'introduction (section 1.1) ainsi que la problématique traitée dans ce document (section 1.2).

Dans un deuxième temps, les principales méthodes existantes de génération de mouvement en environnement dynamique sont présentées (section 1.3)

Enfin, à la section 1.4, nous présentons notre propre approche et nos contributions principales qui seront développées à travers les chapitres 2 à 4.

1.1 Notions de robotique mobile autonome

Concevoir un système capable de se déplacer seul en environnement incertain, dynamique et ouvert, pose de nombreux problèmes qui font intervenir des spécialistes de domaines très divers. L'intégration de ces différents travaux est elle-même un problème à part entière. Pour plus de détails, le lecteur pourra se référer à [Brady, 89] où quelques thèmes fondamentaux de la robotique mobile sont introduits.

1.1.1 Aspects matériels d'un robot mobile autonome

D'un point de vue technologique, la configuration de base d'un robot mobile autonome (RMA) comprend trois types d'organes :

- les ***organes de perception***
- les ***unités de traitement***
- les ***organes de locomotion***

D'autres organes tels que des bras manipulateurs peuvent lui être ajoutés pour une application particulière.

1.1.1.1 Organes de perception

Ils comprennent les capteurs proprioceptifs (renseignant le robot sur lui-même : encodeurs, centrale inertielle) et les capteurs extéroceptifs (renseignant le robot sur son environnement : caméras, télémètres laser, radars hyperfréquences, proximètres à ultrasons). Certains capteurs modernes sont capables de pré-traiter leurs données pour fournir des informations de plus haut niveau (segmentation d'une image vidéo, suivi automatique de cible, filtrage du bruit).

1.1.1.2 Organes de locomotion

Aussi appelés *effecteurs*, ils ont pour but de permettre au robot d'évoluer dans un monde prévu à l'origine pour l'homme. Les plus courants sont les systèmes à roues (de type voiture, différentiels, synchro-drive ou à roues à galets), mais il existe aussi des robots à chenilles, à « pattes » (du robot sauteur sur un « pied » au robot araignée à huit pattes) ou se déplaçant par reptation (comme un serpent).

Le type de locomotion définit deux types de contraintes :

- les ***contraintes cinématiques***, qui portent sur la géométrie des déplacements possibles du robot (eg. bornes sur la courbure des trajectoires),
- les ***contraintes dynamiques***, liées aux effets du mouvement (accélérations bornées, vitesses bornées, présence de forces d'inertie ou de friction)

Selon sa cinématique, un robot sera dit :

- ***holonome***, s'il peut se déplacer instantanément dans toutes les directions (tel que le Nomad, figure 1.1),
- ***non-holonome***, si ses seuls déplacements autorisés sont des courbes dont la courbure est bornée (tel qu'une voiture ou le Cycab, figure 1.1).

Ces définitions informelles sont suffisantes pour notre étude, toutefois leur interprétation mathématique peut être trouvée dans [Latombe, 90].

1.1.1.3 Unités de traitements

Elles regroupent tout ce que l'électronique moderne compte de calculateurs, ordinateurs, et autres systèmes capables de manipuler et stocker des données numériques.

1.1.2 Aspects Fonctionnels d'un robot mobile autonome

Les traitements de base réalisés par un RMA, c'est-à-dire par ses unités de traitements, suivent un découpage fonctionnel similaire à celui de ses équipements. Ainsi, nous trouvons :

FIG. 1.1 – *Types de robots étudiés* Le robot Nomad 200 (à gauche) est un robot expérimental holonome de base cylindrique, de type robot de service. Le robot Cycab de l'INRIA (à droite) est un véhicule expérimental non-holonome, de base rectangulaire, de type système de transport automatisé. Ils sont représentatifs de la plupart des robots à roues existants, et des types de robots que nous serons amenés à considérer dans ce document.

- la ***perception***
- le ***raisonnement***
- l'***action*** (à savoir ici l'exécution d'un déplacement désiré)

1.1.2.1 Perception

La perception (schématisée par la figure 1.2) regroupe l'ensemble des traitements visant à minimiser le volume des informations reçues par le robot afin de n'en conserver que les plus pertinentes pour sa mission (localisation dans l'environnement, carte des obstacles). Il s'agit essentiellement de filtrage, de mise en forme et de fusion des données issues des capteurs et de la connaissance a priori du robot (cartes de l'environnement, données constructeur).

1.1.2.2 Raisonnement

Il constitue l'intelligence du robot. À l'instar de l'Homme, le raisonnement du robot lui permet de décider d'une action appropriée à une situation donnée, compte-tenu d'une mission à réaliser. Plusieurs tâches élémentaires s'exécutent en parallèle pour synthétiser un comportement. La façon dont elles inter-agissent est définie par l'architecture décisionnelle du robot. Celle-ci comprend notamment un contrôleur d'exécution, dont le but est de lancer l'exécution d'un groupe de tâches donné à intervalles de temps réguliers. Chaque cycle (ou itération) du contrôleur correspond à une décision du robot, et génère un ordre (appelé consigne) envoyé à la partie action. Nous parlerons de *traitement itératif*.

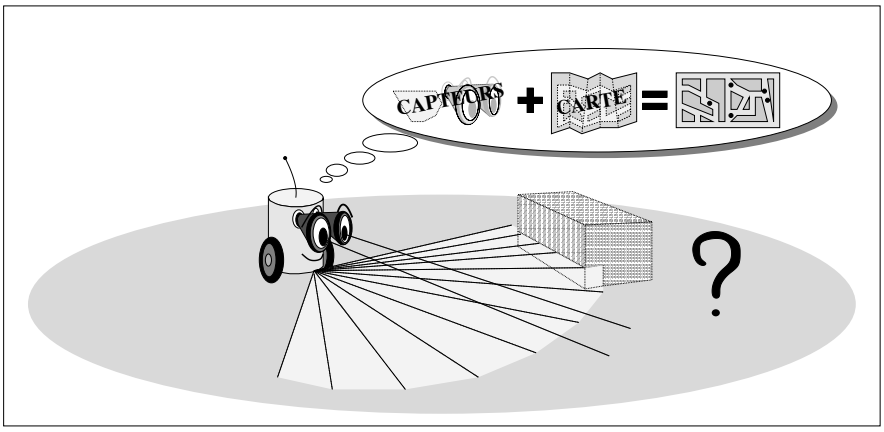

FIG. 1.2 – *Problèmes liés à la perception de l'environnement* La "perception" désigne généralement les traitements permettant au robot d'extraire des informations pertinentes sur son environnement, à partir des capteurs seulement. Nous désignons également, sous ce terme, le traitement d'informations ne provenant pas de capteurs telle que la connaissance a-priori sur l'environnement, lorsque celle-ci est disponible (plan d'un bâtiment par exemple).

1.1.2.3 Action

Cette fonction est également appelée « ***contrôle*** » du robot. L'action essentielle du RMA est de se déplacer. Par conséquent, la consigne est un déplacement désiré et le contrôle regroupe l'ensemble des opérations permettant au robot d'effectuer ce déplacement. Il se fait en deux étapes (figure 1.3) :

1. La *génération de la commande* (ensemble de paramètres de contrôle des effecteurs) supposée impliquer le déplacement escompté.
2. L'*application de cette commande*, c'est-à-dire sa conversion en signaux directement utilisables par les effecteurs : On parle alors de *contrôle bas niveau.*

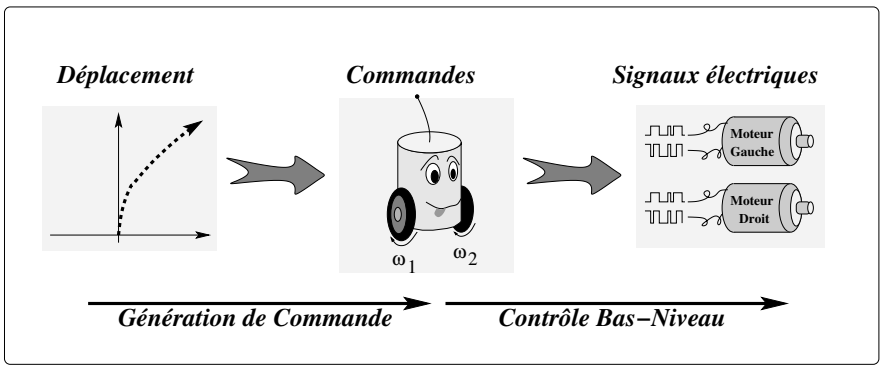

FIG. 1.3 – *Contrôle du robot* La génération de commande traduit le déplacement désiré (la consigne) en une commande, que le contrôle bas niveau traduit à son tour en signaux directement utilisables par les effecteurs.

1.1.3 Aspects étudiés dans ce mémoire

Donner au robot la possibilité d'obtenir les informations dont il a besoin pour raisonner et le doter de capacités de locomotion adaptées à son environnement sont essentiels à son autonomie. Cependant, ceci implique des systèmes complexes dont la réalisation mais aussi la maîtrise posent d'importants problèmes non seulement technologiques mais aussi scientifiques. Les défis principaux de la robotique mobile autonome concernent donc la perception et la locomotion.

Actuellement, beaucoup de capteurs ou d'effecteurs sont sous-utilisés en raison du manque d'outils et de techniques suffisamment efficaces. C'est le cas de la vision et des systèmes de locomotion bipèdes. Leurs équivalents chez l'Homme lui permettent de voir son environnement et de s'y déplacer librement. En revanche, chez un robot, les calculs très complexes mis en œuvre lui permettent à peine, dans le meilleur des cas, d'identifier quelques éléments d'une scène et de s'y déplacer de manière très lente et hésitante. De plus, les différents capteurs et effecteurs étant des systèmes électroniques et mécaniques, ils sont sujet aux pannes, aux bruits et aux dérèglements.

Nous ne nous intéressons pas ici aux problèmes de perception ni de locomotion, mais à leurs conséquences sur les traitements, et plus particulièrement sur la ***génération des mouvements en environnement dynamique et incertain***. Pour un tour d'horizon des différents types de capteurs ainsi que les traitements de perception associés, le lecteur pourra consulter l'ouvrage « Where Am I » ([Borenstein et al., 96]), consacré au problème de la localisation d'un robot mobile.

1.2 Environnement dynamique et incertain

1.2.1 Notion d'environnement dynamique

Nous dirons qu'un environnement est dynamique lorsque l'ensemble des positions occupées par les obstacles est susceptible de changer au cours du temps.

C'est le cas des obstacles qui se déplacent, de ceux qui changent de forme (e.g. un piéton tenant un chien en laisse est perçu par le robot comme un seul objet à géométrie variable) ou de ceux qui apparaître/disparaître (e.g. une porte coulissante semble « disparaître » dans le mur quand elle s'ouvre).

1.2.2 Notion d'incertitude

La notion d'incertitude est souvent sujette à discussions (pour quelques définitions proposées en IA, voir [Hacking, 75] [Saffiotti, 87] [Krause et Clark, 93] ou [Smets, 95]). Dans la suite de ce mémoire, nous dirons d'une manière générale qu'une information est *incertaine*, si elle est bruitée (mauvaises conditions de mesures), incomplète (obstruction d'un capteur ou portée limitée, absence d'informations sur l'évolution d'un objet ou d'un phénomène) ou imprécise (les glissements des roues par rapport au sol sont observables mais rarement mesurables avec précision).

1.2.3 Implications sur les traitements

L'absence de connaissance a priori sur l'environnement contraint le robot à faire des hypothèses qui peuvent s'avérer fausses par la suite. Une réactualisation permanente de sa connaissance ainsi qu'une capacité à réagir rapidement aux changements sont nécessaires dans ce contexte.

En environnement statique, les hypothèses portent uniquement sur la position du robot et la présence de nouveaux obstacles à proximité du robot. Celles-ci sont levées au fur-et-à-mesure de la progression du robot.

En environnement dynamique, les hypothèses portent également sur les trajectoires des obstacles mobiles. Celles-ci doivent être prises en compte par le robot lors du choix d'un déplacement pour anticiper les collisions et assurer sa sécurité. En raison du nombre important de possibilités dans ce cas et de l'absence de connaissance a priori complète, les erreurs sont nombreuses et la validité d'une solution est de courte durée.

En résumé, un environnement à la fois dynamique et incertain implique ***davantage de calculs*** (liés à l'anticipation des collisions) et demande une ***plus forte réactivité du robot*** et donc des temps de réponse plus courts (en raison du caractère éphémère des solutions).

1.3 Méthodes de génération de mouvement existantes

Le problème de la génération de mouvement au sens large consiste à calculer un déplacement sûr du robot puis à l'exécuter. Tandis que l'Homme apprend à lever rapidement les incertitudes grâce à ses expériences passées, cela représente un véritable challenge pour un RMA, dont le raisonnement purement logique et mathématique demande du temps. En environnement dynamique et incertain, les méthodes actuelles de planification d'un déplacement sûr ne sont pas assez rapides pour assurer la sécurité du robot. Seules les méthodes réactives le permettent, mais elles ne prennent pas en compte (ou mal) les contraintes de tâche comme atteindre un but.

Par conséquent, un des grands problèmes ouverts de la robotique mobile autonome est celui de la navigation autonome pour ces environnements particuliers. Dans ce rapport, nous avons choisi d'étudier deux thèmes fondamentaux :

- ***le calcul d'un déplacement sûr*** du robot, compte-tenu de sa connaissance actuelle sur lui-même et sur son environnement,
- ***le calcul de la commande*** du robot permettant de réaliser ce déplacement, compte-tenu de sa connaissance actuelle sur lui-même et sur ses interactions avec son environnement.

1.3.1 Calcul d'un déplacement sûr

La littérature oppose généralement deux grandes approches pour calculer un déplacement sûr, indépendamment du type d'environnement considéré :

- les méthodes d'***évitement d'obstacles***,
- les méthodes de ***planification de trajectoire***[1].

Des approches récentes tendent à les combiner dans des méthodes hybrides appelées ***méthodes itératives*** ou à réduire la complexité de la modélisation de l'espace libre par des ***approches statistiques***.

1.3.1.1 Notion d'espace de recherche

Les traitements réalisés en navigation reposent essentiellement sur des outils de géométrie. Typiquement, la recherche d'un déplacement libre fait appel à des techniques classiques de test de collision (calculs d'intersections, calculs de distances).

Le choix des espaces mathématiques dans lesquels sont réalisés les calculs est fonction de l'environnement et des informations à représenter (i.e. à prendre en compte). Leur dimension peut par conséquent être très élevée ce qui a une influence directe sur la quantité et la complexité des traitements. Sous contrainte de temps-réel, des compromis doivent être faits pour réduire cette complexité. Cela se traduit souvent par la recherche avant tout d'un espace de dimension plus réduite pour pouvoir maîtriser la complexité des calculs, tout en conservant les informations essentielles.

L'espace de modélisation le plus intuitif pour un robot est son espace de travail. Noté $\mathcal{W}$, il représente l'ensemble des positions que ce dernier peut atteindre. Par conséquent, sa dimension est faible et reste constante quelque soit la complexité du robot (2 pour les robots mobiles considérés dans ce rapport et 3 pour des robots qui se déplacent dans l'espace, comme un drône ou un bras de robot).

À chaque type d'environnement et à chaque traitement correspond un espace. $\mathcal{W}$ est généralement utilisé pour formuler le problème. En revanche, il ne permet pas de le traiter car la représentation du robot et des obstacles dans $\mathcal{W}$ n'est pas appropriée pour les calculs. Nous donnons ici les principaux espaces utilisés en robotique à titre indicatif, avec leur utilisation respective la plus courante :

- Espace des configurations [Goldstein, 80] ($\mathcal{C}$) : La configuration d'un robot est l'ensemble des n paramètres qui permettent de définir de manière unique et sans équivoque, la posture (position+orientation) de chaque partie d'un robot dans son environnement. $\mathcal{C}$ permet de représenter par un point donné une configuration donnée. Sa dimension est donc n. Permettant de ramener un problème de planification complexe à un problème purement géométrique, $\mathcal{C}$ est très employé en robotique (voir [Lozano-Perez, 83]), en particulier pour la planification de chemin géométrique en environnement statique avec prise en compte de la cinématique du robot.

[1] Le terme trajectoire désigne en robotique un déplacement dont la vitesse est imposée. Lorsque ce n'est pas le cas et que seule la géométrie du déplacement importe comme en environnement statique, le terme de chemin est utilisé.

- Espace des états [Canny et al., 88] ($\mathcal{S}$) : L'état du robot peut être vu comme l'union des paramètres de configuration du robot, et de leurs dérivées (premières et éventuellement secondes) par rapport au temps. Il permet de prendre en compte des contraintes dynamiques et par conséquent est utilisé pour la planification de trajectoire en environnement statique avec prise en compte de la cinématique et de la dynamique du robot.
- Espace des configurations-temps [Erdmann et Lozano-Perez, 86] ($\mathcal{CT}$) : il s'agit de $\mathcal{C}$ augmenté d'une dimension de temps pour pouvoir traiter les environnements dynamiques, d'où son utilisation pour la planification de trajectoire en environnement dynamique avec prise en compte de la cinématique du robot.
- Espace des états-temps [Fraichard, 93] ($\mathcal{ST}$) : Il s'agit de l'espace des états augmenté d'une dimension de temps pour les mêmes raisons que pour $\mathcal{CT}$. Il est utilisé en planification de trajectoire en environnement dynamique avec prise en compte de la cinématique et de la dynamique du robot.
- Espace des commandes immédiates ($\mathcal{U}$) : Cet espace ne permet pas de représenter une trajectoire, mais uniquement le prochain déplacement élémentaire du robot. Il est de faible dimension et permet une représentation simple des contraintes sur le robot. Son utilisation typique est l'évitement d'obstacles en environnement statique ou dynamique avec prise en compte de la cinématique et de la dynamique du robot pour le prochain déplacement.
- Espace des trajectoires de fuite [Novalès, 94] (Γ) : Il s'agit d'une extension de $\mathcal{U}$. Γ contient un sous-ensemble des trajectoires immédiatement réalisables par le robot sur un intervalle de temps borné et généralement court. Le test d'une trajectoire au lieu de se limiter au prochain mouvement permet de mieux traiter les environnements dynamiques tout en conservant une dimension raisonnable. Cela permet surtout une meilleure prise en compte des contraintes cinématiques et de la dynamiques du robot. Γ est utilisé par exemple pour l'évitement réactif d'obstacles.

1.3.1.2 Méthodes d'évitement d'obstacles

Les méthodes d'évitement d'obstacles sont des méthodes locales dont l'objectif est d'assurer la sécurité du robot sur un temps très court. Elles peuvent être classées en deux catégories :

- celles qui calculent directement une commande à partir des informations disponibles sur l'environnement. Il s'agit des approches de l'intelligence artificielle inspirées du vivant (réseaux de neurones artificiels [Gauthier, 99], approche bayésienne [Lebeltel, 99], logique floue [Garnier, 95]), ou des approches basées sur des phénomènes physiques naturels (comme les champs de potentiels [Khatib, 80] [Khatib, 86]). Ces méthodes confèrent un comportement de base très réactif au robot, bien adapté aux environnements dynamiques. Cependant la prise en compte

de contraintes de tâche (atteindre un but) est difficile voire impossible avec ces méthodes et nous ne nous y intéresserons pas davantage.

– celles qui calculent un ensemble de solutions potentielles compte-tenu des informations sur l'environnement, puis sélectionnent une solution particulière afin de satisfaire des contraintes de tâche. Les solutions peuvent prendre la forme d'une direction privilégiée du robot ou d'une consigne en vitesse. Nous allons nous intéresser ici à ces méthodes.

Leur principal avantage est leur grande réactivité. Elle est due au fait qu'en ne considérant que des déplacements locaux, le nombre de calculs est réduit. Le temps ainsi « économisé » permet de réduire les temps de réponse du robot ou de prendre en compte des paramètres tels que la cinématique et la dynamique du robot, pour améliorer la qualité (ie. l'exécutabilité) des mouvements proposés.

En contre-partie, leur caractère local leur confère un inconvénient majeur : les « minima locaux » de la fonction de sélection. En effet, un coût minimal peut être attribué à un déplacement répondant localement aux critères de choix désirés, mais qui, dans le futur, conduira le robot à une situation de blocage ou à une collision (figure 1.4).

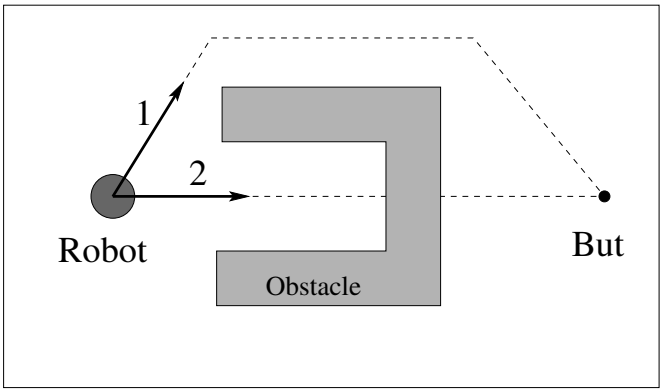

FIG. 1.4 – *Minimum local* Les déplacements 1 et 2 sont tous deux libres pour le robot. Localement, le déplacement 2 étant dans la direction du but, il est meilleur, pourtant, il conduit à un blocage.

Une solution couramment rencontrée au cours des dernières années, consiste à étendre ces méthodes avec des techniques de graphe ou à les combiner avec un planificateur global utilisant une fonction de navigation classique NF1 ([Latombe, 90]) de type « propagation de vague ».

Nous ne citons ici que les approches les plus utilisées. Nous avons conservé leur dénomination anglosaxone lorsque la traduction française ne nous semblait pas naturelle :

Vector Field Histogram (*VFH*) et extensions Dans leur version d' origine, les *VFH* ([Borenstein et Koren, 91]) consistent à représenter dans un premier temps

l'environnement par une grille d'occupation. Chaque cellule contient une grandeur réelle correspondant en quelque sorte à la probabilité de trouver un obstacle à cet emplacement. La grille est ensuite traduite en un histogramme dont chaque « barre » représente une direction du robot et dont la hauteur est proportionnelle à la probabilité de percuter un obstacle en suivant cette direction. Par seuillage, il est ainsi possible d'identifier les directions libres du robot, appelées passages. Une fonction de coût prenant en compte la direction du but, la direction précédemment suivie et l'orientation courante des roues permet de sélectionner un des passages et la direction correspondante. La vitesse est ensuite calculée en fonction de la distance du robot aux obstacles.

Une version plus récente (VFH^+) ([Ulrich et Borenstein, 98]) prend en compte la largeur du robot et ses contraintes cinématiques (les trajectoires ne sont plus assimilées à des droites mais à des arcs de cercles).

Un des inconvénients majeur de l'approche est la difficulté à régler le seuil déterminant les « passages » et la difficulté à naviguer en milieu contraint (couloirs étroits, passage de portes). De plus, la dynamique du robot n'est pas réellement prise en compte et le traitement purement local empêche de garantir la prise en compte des contraintes de tâches. Ce dernier point a toutefois été traité dans la dernière version de cette approche sous la dénomination VFH^* ([Ulrich et Borenstein, 00]), où plusieurs déplacements sont testés à l'avance. Ceci a pour effet de limiter les situations de blocage et d'améliorer la convergence vers le but.

Curvature Velocity (CV) et Lane Curvature Velocity (LCV) L'idée consiste à déterminer non plus une orientation mais une commande. Elle permet de prendre en compte aussi bien des contraintes cinématiques et dynamiques sur le robot : La méthode des « Curvature Velocities » ([Simmons, 96]) représente les obstacles dans l'espace des vitesses de translation et de rotation (v, ω) du robot. Une fonction de coût permet de prendre en compte des critères d'alignement avec le but comme pour les VFH, mais permet en plus de considérer la longueur des trajectoires libres avant que le robot n'entre en collision avec un obstacle.

La méthode de base a été étendue sous la dénomination de « Lane Curvature Velocity » ([Ko et Simmons, 98]). Les trajectoires du robot considérées sont quelconques et les solutions permettent de passer plus loin des obstacles grâce à la notion de voie (« Lane »). Bien que la méthode soit purement locale et présente les inconvénients classiques de ce type d'approche, elle permet de mieux passer les passages étroits et prend mieux en compte les contraintes cinématiques et dynamiques du robot, que les précédentes.

Fenêtres dynamiques (DW) La méthode des fenêtres dynamiques ([Fox et al., 97]) reprend en tout point le principe des CV y compris dans le choix de la fonction de coût mais dans un espace des vitesses (v, ω) du robot discrétisé. Les contraintes dynamiques du robot sont prises en compte en limitant les vitesses du robot sélectionnables. Elles définissent une fenêtre dite dynamique dans l'espace des vitesses,

autour de la vitesse courante du robot. Une vitesse est considérée potentiellement solution, si elle est admissible compte tenu de ces contraintes, et si elle permet au robot de s'arrêter avant de percuter un obstacle. Les fenêtres dynamiques existent en deux versions, pour robots holonomes ([Brock et Khatib, 99]) ou non-holonomes (dans ce cas les trajectoires du robot sont assimilées à des arcs de cercle).

L'inconvénient de cette approche réside dans le choix de la discrétisation de l'espace des vitesses. Ainsi une implémentation temps-réel nécessite de réduire la résolution ou la taille de la fenêtre dynamique ([Arras et al., 02]). Limitée à un traitement local et donc sujette aux minima locaux dans sa version de base, cette approche a été étendue vers une version globale ([Brock et Khatib, 99]) à l'aide d'une fonction NF1. Dans les expérimentations, celle-ci est calculée uniquement à partir de l'environnement perçu par le robot. Cela pose des problèmes de mise-à-jour de la carte en environnement dynamique, comme cela est noté dans [Minguez et al., 02]. Dans ce cas, l'approche est applicable en environnement statique ou peu dynamique, plutôt qu'en environnement réellement dynamique.

Diagramme de proximité (ND) Les ND ([Minguez et Montano, 00]) sont basés sur l'extraction d'informations de haut niveau sur l'environnement, à savoir l'identification d'une situation particulière parmi 5 prédéfinies. Cela permet de générer la commande la plus appropriée. La méthode comprend 4 étapes :

1. la construction de deux diagrammes polaires représentant la distribution des obstacles autour du robot et leur distance, respectivement au robot et à son point de référence.
2. la recherche de régions de passage à partir du second diagramme, et la sélection de l'une d'elles
3. l'analyse du premier diagramme pour définir le niveau de sécurité du robot
4. enfin, la combinaison de ces informations pour identifier une situation prédéfinie parmi 5, et générer la commande appropriée à cette situation.

La première chose à remarquer est la phase d'analyse préalable, rencontrée nulle part ailleurs dans les autres approches. Cela permet au robot d'éviter les pièges classiques sur lesquelles les autres méthodes locales butent, tels que le traitement des obstacles en forme de U (figure 1.4), ou la navigation en milieu très contraint sans osciller. Nous pensons que cela fait de cette approche la plus efficace pour affronter des environnements totalement inconnus. Cependant, la survie du robot en milieu dynamique n'est due qu'à la réactivité de la méthode, comme pour les approches précédentes, et n'est pas due à la prise en compte explicite de cette dynamique. De plus, la complexité des traitements implique des temps de réponse plus longs, qui peuvent à notre avis lui faire perdre son avantage en présence d'obstacles mobiles, par rapport à une approche moins « pensante » mais plus rapide telle que les DW. Toute comme celles-ci, en revanche, elle a été récemment étendue à une version globale ([Minguez et al., 02]) dont elle a pris le qualificatif.

Le principe est le même à savoir l'utilisation combinée de l'approche réactive avec une fonction de navigation de type NF1. L'actualisation de la carte locale tire cependant davantage parti des ressources capteurs et permet un comportement plus naturel en milieu dynamique que les GWA (voir [Minguez et al., 02] pour un comparatif).

1.3.1.3 Méthodes de planification de trajectoire

Les méthodes de planification de trajectoire, par opposition aux précédentes, calculent un déplacement complet (global) jusqu'au but. Elles supposent une connaissance suffisante de l'environnement. Leur principe consiste à obtenir une représentation de l'espace libre et de sa connectivité pour y rechercher une trajectoire reliant le robot à son but. Différentes contraintes (cinématique, dynamique, incertitudes) peuvent être prises en compte lors de la recherche de la trajectoire dans l'espace libre.

Ces méthodes permettent d'éviter les problèmes de minima locaux et de proposer des solutions optimales[2], mais leur temps de réponse est généralement supérieur aux contraintes imposées pour la navigation : La modélisation de l'espace libre dans le cas général nécessite en effet des calculs complexes. De plus, l'étendue de cet espace implique un nombre plus important de solutions potentielles à envisager. Les études menées sur le sujet sont moins nombreuses que pour les environnements statiques connus (introduites en annexe A), et d'une manière générale, ne sont pas utilisables en temps-réel. Nous citons cependant les plus courantes pour information.

Planification dans $\mathcal{CT}$ et $\mathcal{ST}$ L'approche générale consiste à ajouter une dimension de temps à l'espace $\mathcal{C}$ des configurations, généralement utilisé pour la planification en environnement statique. Lorsque la dynamique du robot doit être prise en compte, l'espace étendu est $\mathcal{S}$ ([Fraichard, 99]). Les méthodes applicables dans $\mathcal{C}$ ont été étendues pour les nouveaux espaces, notamment la méthode du graphe de visibilité ([Erdmann et Lozano-Perez, 86] [Fujimura et Samet, 90] [Reif et Sharir, 85]) ou celle des décompositions cellulaires ([Fujimura et Samet, 89] [Shih et al., 90]).

Ces approches s'avèrent très coûteuses en calculs et non-envisageables pour une utilisation temps-réel autre que pour des cas très simplifiés ([Fujimura et Samet, 89] [O'Dunlaing, 87]).

Décomposition Chemin-vitesse L'idée consiste à calculer, à l'aide des méthodes citées en annexe A, le chemin géométrique permettant d'éviter tous les obstacles statiques connus. Un profil de vitesse est ensuite calculé le long de ce chemin de manière à éviter les obstacles en mouvement. Il est réalisé dans un espace de dimension 2, dont l'abscisse est l'abscisse curviligne du chemin et l'ordonnée est

[2] L'optimalité d'une trajectoire peut concerner son temps de parcours par le robot ou sa longueur. Il s'agit dans les deux cas d'une trajectoire qui minimise ce critère.

le temps. Il s'agit d'une courbe monotone selon l'axe du temps. Les contraintes dynamiques imposent des bornes de sa pente ([Kant et Zucker, 86]). Cette approche a permis le calcul de trajectoires optimales en temps pour des robots dont les vitesses et les accélérations sont bornées ([Canny et al., 90] [O'Dunlaing, 87]), ainsi que pour des robots soumis à des contraintes dynamiques plus complètes ([Bobrow et al., 85] [Shiller et Dubowsky, 85] [Shiller et Lu, 90]). Dans ce cas cependant, les contraintes d'optimalité peuvent être relâchées légèrement. L'idée consiste à calculer une version approchée seulement de la trajectoire optimale, par le biais de grilles définies dans l'espace de travail $\mathcal{W}$ ([Shiller et Dubowsky, 88]), dans l'espace des configurations $\mathcal{C}$ ([Sahar et Hollerbach, 85]) ou dans l'espace des états $\mathcal{S}$ ([Canny et al., 88] [Donald et Xavier, 90] [Jacobs et al., 89]).

Le temps de calcul des méthodes à deux étapes est considérablement réduit par rapport aux autres approches et une utilisation en ligne est possible. Néanmoins, ces approches ne sont pas complètes et écartent un grand nombre de solutions potentielles lors de la sélection d'un chemin géométrique particulier.

1.3.1.4 Méthodes hybrides ou itératives

Les méthodes itératives utilisent des techniques de planification classiques basées sur l'exploration d'un arbre de recherche. La différence avec les approches classiques porte sur le fait que la modélisation de l'espace libre et son exploration ne sont pas deux tâches séparées et consécutives mais une seule répétée de manière itérative. Cette procédure permet de réduire considérablement la taille de l'espace exploré, et donc les temps de calcul, en ne modélisant que ce qui nécessite de l'être au fur-et-à mesure des besoins. De plus, elle peut être interrompue à tout instant pour obtenir la trajectoire de coût minimal parmi celles déjà explorées.

En théorie, ces méthodes sont moins sensibles aux minima locaux car elles anticipent les trajectoires du robot sur des temps plus longs (jusqu'au but dans le cas idéal où elles ne sont pas interrompues). Elles permettent donc une meilleure prise en compte des contraintes de tâche, tout en assurant la réactivité du robot en environnement changeant.

En pratique, les méthodes itératives actuelles utilisent des méthodes locales d'évitement d'obstacles pour étendre l'arbre, or celles-ci sont mal adaptées aux environnements dynamiques :

- Pour chaque déplacement du robot, les unes demandent des calculs coûteux (Voir ([Laumond, 98]) pour un aperçu des principales approches analytiques locales). Pour compenser, la plupart utilisent une approche probabiliste pour développer l'arbre et couvrir l'espace de recherche plus rapidement et de manière plus homogène ([LaValle, 98] [Ahuactzin, 94] et [Ahuactzin et Gupta, 99] par exemple).
- Les autres ([Brock et Khatib, 99] [Minguez et al., 02] [Ulrich et Borenstein, 00]) considèrent les obstacles statiques, y compris s'ils ne le sont pas réellement. Les déplacements calculés à chaque itération doivent par conséquent être suffisamment

courts pour que cette hypothèse ne mette pas le robot en danger. Or, cela a une influence directe sur la durée de validité d'une solution. De plus, l'absence d'anticipation de ces méthodes conduit à des trajectoires moins naturelles.

Une approche se distingue de toutes les autres : La bande élastique (EB). Son principe consiste à maintenir en permanence un passage (appelé bande) sans collision entre le robot et son but. Ce passage est représenté par un chapelet de disques chevauchant appelés des bulles. Chaque bulle représente l'espace libre local le long du passage. Son diamètre peut par conséquent s'accroître ou se réduire pour couvrir au mieux cet espace libre, en fonction des changements dans l'environnement. Leur nombre peut également varier pour éviter toute interruption de la bande. Celle-ci est soumise à deux familles de forces : des forces attractives entre bulles consécutives afin de maintenir la bande « tendue ». Et des forces répulsives avec les obstacles pour maintenir la bande éloignée de ces derniers. L'équilibre de ces forces tend vers un chemin sûr jusqu'au but. La mise à jour des forces assure l'adaptation de ce chemin en temps-réel en fonction des obstacles.

L'approche a initialement été prévue pour planifier des chemins géométriques en temps-réel ([Quinlan et Khatib, 93] [Quinlan, 94a]). Elle a été étendue ensuite dans [Khatib et al., 97] à des robots non-holonomes. Le principe reste le même et seule la métrique considérée change (métrique non-holonome de Reed et Shepp). Pour éviter que la bande ne soit totalement déformée voire coincée (figure 1.5) en présence d'obstacle mobiles, une extension a été proposée sous le terme de bandelettes élastiques [Brock et Khatib, 00].

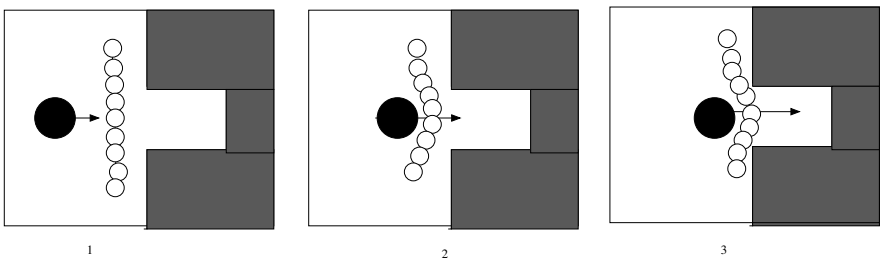

FIG. 1.5 – *Bandes élastiques.* Un obstacle mobile peut venir « coincer » la bande.

L'idée consiste à relâcher temporairement la contrainte entre deux bulles adjacentes pour « laisser passer » un obstacle mobile (figure 1.6). De plus, pour limiter les cas où une replanification complète de la bande est néanmoins nécessaire, un ensemble de bandes de secours est conservé dans [Brock et Khatib, 00] comme solutions alternatives rapidement disponibles.

Ceci rend la méthode des bandelettes élastiques utilisable en environnement dynamique. Néanmoins, aucune anticipation des collisions n'est faite et il nous semble que cela fait défaut à la méthode pour garantir totalement la sécurité du robot en présence d'obstacles mobiles proches.

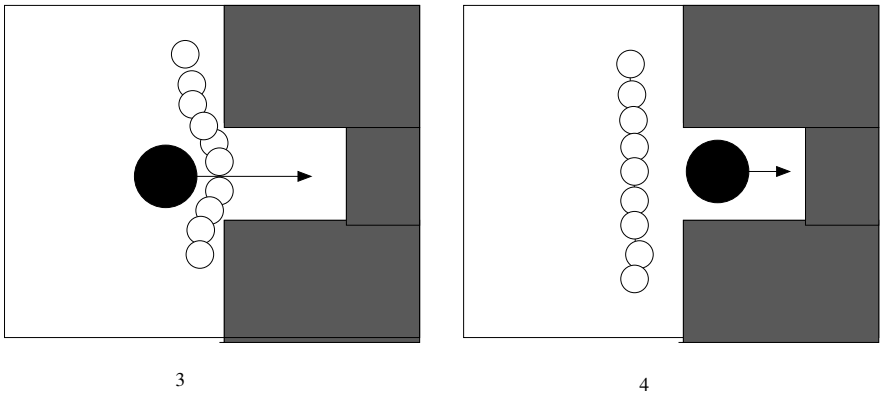

FIG. 1.6 – *Bandelettes élastiques.* Les bandelettes laissent passer les obstacles qui exercent une « pression » trop grande sur elles.

1.3.1.5 Conclusion sur les méthodes de calcul d'un déplacement

Les approches hybrides, et en particulier les approches itératives basées sur l'extension de méthodes locales d'évitement d'obstacles, offrent actuellement le meilleur compromis entre réactivité et prise en compte des contraintes de tâche, y compris pour des environnements dynamiques et incertains. Des méthodes locales plus adaptées à ces environnements restent néanmoins à être développées, afin de mieux anticiper les collisions. Nous apportons notre contribution à ce problème par le biais du concept de $\mathcal{V}$*-Obstacle Non-Linéaire* présenté au chapitre 2.

1.3.2 Calcul de la commande en présence d'incertitudes

Lorsqu'un déplacement désiré a été calculé, il reste à trouver la commande qui permettra de le réaliser : ceci est un problème de contrôle qui implique que nous possédions une modélisation du système à contrôler. Dans le cas d'un robot mobile, cela suppose un modèle du robot lui-même (contraintes mécaniques et cinématiques), ainsi que de ses interactions avec l'environnement (géométrie du robot et des obstacles pour tester les collisions, et contraintes dynamiques pour calculer les glissements par exemple). Deux types de modèles peuvent être définis (figure 1.7), selon l'utilisation souhaitée :

- le modèle *direct* (encore appelé *fonction de transfert*), qui permet d'obtenir le déplacement du robot correspondant à l'exécution d'une commande,
- le modèle *inverse*, qui permet de calculer quelle commande doit être appliquée pour obtenir un déplacement souhaité.

La principale difficulté pour obtenir ces données est due au nombre important de paramètres qui entrent en jeu et à l'impossibilité de les mesurer précisément, voire même de les identifier.

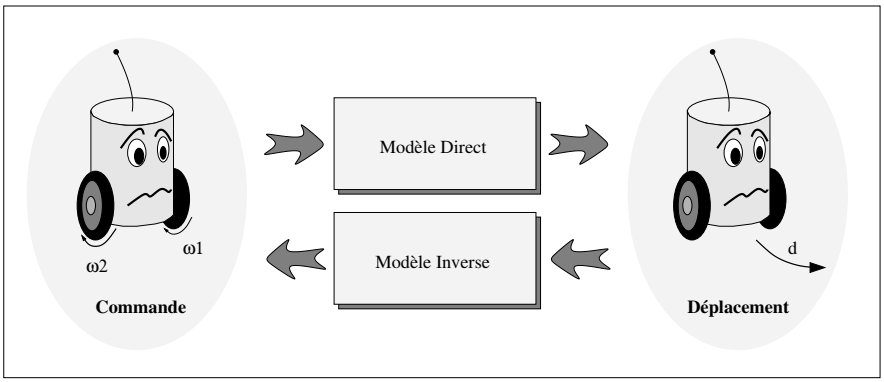

FIG. 1.7 – *Modélisation d'un robot* Les modèles direct et inverse représentent les relations entre les déplacements d'un robot et les commandes envoyées à sa partie motrice (effecteurs).

Nous citerons l'exemple des glissements des roues d'un robot tel que le Cycab : Ils sont dépendants, entre autre, de l'architecture matérielle du robot (le Cycab possède par exemple, de par sa conception mécanique, deux centres de giration instantanée théoriques distincts, impliquant des glissements inévitables. Voir annexe C), de l'état des roues (niveau d'usure, angle de braquage, vitesse angulaire, pression de gonflage), du chargement du robot (répartition du poids du robot sur chaque roue), du sol (nature, état, topologie) et des conditions climatiques (présence ou non de vent, de pluie, de verglas). De plus, les capteurs permettant de mesurer ou d'estimer ces données peuvent eux aussi être défectueux, mal calibrés, bruités, ou simplement ne pas exister.

Dans ces conditions, nous admettrons ici qu'un modèle n'est jamais conforme à la réalité et ne fait qu'en proposer une approximation grossière mais suffisante pour assurer un contrôle satisfaisant du robot. Lorsque les paramètres de ce dernier évoluent au cours du temps, le modèle doit lui aussi évoluer : Il est dit dynamique. Le contrôle devient alors adaptatif, c'est-à-dire que ses paramètres s'adaptent pour prendre en compte les variations du modèle.

Le contrôle adaptatif doit ses débuts en automatique à l'armée, avec l'étude de vols supersoniques et de missiles balistiques dans les années 50. Un aperçu des méthodes proposées à l'époque se trouve dans [Aseltine et al., 58] et [Gregory, 59]. Plus tard, des approches de l'intelligence artificielle, basées sur l'observation du vivant, ont donné lieu aux contrôleurs flous ([Mandani, 74][Lee, 90][Garnier, 95]) de la logique floue ([Zadeh, 65]), aux contrôleurs à base de réseaux de neurones artificiels ([Agarwal, 97][Gauthier, 99]), et plus récemment aux contrôleurs probabilistes ([Lebeltel, 99]).

1.3.2.1 Approche de l'automatique

Du point de vue de l'automaticien, un modèle est un système d'équations mathématiques, dont la forme et les paramètres ont été définis manuellement grâce à une étude

poussée du système contrôlé. L'automaticien dispose alors d'une panoplie d'outils permettant d'apprécier la qualité d'une méthode au travers de critères tels que la stabilité et la convergence du contrôle (Voir [Hermosillo, 03] pour un rappel des notions de base).

Dans le cas d'un RMA, le modèle est imparfait pour les nombreuses raisons citées précédemment. Par conséquent, son utilisation pour générer une commande entraîne des écarts inévitables entre le déplacement initialement prévu et celui effectivement réalisé par le robot.

La mesure de ces écarts permet de corriger le contrôle. On parle alors de contrôle avec retour, ou de contrôle en boucle fermée (le contrôle en boucle ouverte étant l'application des commandes calculées sans vérification de l'effet obtenu).

Deux approches sont possibles pour réduire les écarts de suivi :

- l'ajustement des paramètres du modèle afin de mieux approcher la réalité,
- l'utilisation d'une fonction correctrice appelée loi de commande.

L'identification des paramètres d'un modèle et la façon de les ajuster étant difficiles, la seconde méthode, plus « extérieure » au modèle, est la plus couramment employée. Elle permet d'obtenir un système fiable, dont la stabilité et les performances peuvent être quantifiées avec précision. Néanmoins, elle présente les inconvénients de toute méthode d'automatique, à savoir :

- la nécessité d'effectuer une étude spécifique et complète de chaque système contrôlé
- l'impossibilité de porter facilement une application d'un système vers un autre

1.3.2.2 Approche de l'intelligence artificielle (IA)

L'approche de l'intelligence artificielle consiste à apprendre des relations de cause à effet d'un système en se basant uniquement sur l'observation extérieure de son comportement. Les informations sont représentées sous forme de règles de vérité (« Si observation Alors action ») selon l'approche de la logique floue, de probabilités (« Probabilité De action Sachant observation ») selon l'approche bayésienne ou encore de connexions entre neurones artificiels selon l'approche cogniticienne. Toutes sont inspirées du fonctionnement du cerveau humain ou animal.

La démarche adoptée par l'IA est typiquement celle d'un conducteur humain qui associe une orientation du volant, avec une trajectoire souhaitée. L'association se fait progressivement par apprentissage ou à l'aide de règles résultant d'une observation préalable du système, mais sans connaissance particulière des éléments mis en œuvre et de la façon dont ils interagissent.

Cela donne un attrait certain à l'approche IA, dont nous pouvons citer quelques avantages :

- elle ne nécessite pas une étude mathématique complexe du système : Une identification des « causes » et des « effets » du système, ainsi que des relations qui les unissent est suffisante. Un apprentissage automatique de ces dernières est éventuellement possible.
- elle facilite le portage d'une application d'un robot vers un autre de modèle semblable, ou encore permet l'adaptation des paramètres de contrôle sur un même robot, en raison du niveau élevé de généricité des informations manipulées
- elle permet de traiter des cas non-appris en les rapprochant de cas déjà appris.

En contrepartie, l'IA ne possède pas les outils de validation de l'automatique et estimer la qualité d'un contrôleur défini selon ces principes est difficile autrement que par expérimentation. Deux problèmes fréquents sont :

- maintenir la cohérence de la base de règles pour la logique floue ou l'approche probabiliste,
- s'assurer d'un apprentissage suffisant pour les réseaux de neurones en particulier.

1.4 Notre approche de la navigation autonome

Nous avons dans un premier temps décomposé le problème général de la navigation en trois fonctionnalités de priorités différentes, devant être exécutées en temps-réel :

1. ***Modélisation des déplacements instantanés sûrs***.
   La fonctionnalité principale du robot est d'assurer sa sécurité immédiate en toute circonstance. Il doit donc être capable de décider rapidement d'un déplacement sûr, y compris en présence d'obstacles mobiles.
2. ***Choix d'un déplacement sûr***
   Ensuite et s'il lui reste suffisamment de temps, il peut modifier ou affiner son choix dans le but de remplir une mission (comme atteindre un lieu le plus rapidement possible, ou le plus directement possible). Nous parlerons de prise en compte de contrainte de tâche.
3. ***Apprentissage du modèle du robot***
   Enfin, sa dernière fonctionnalité consiste à exécuter au mieux le déplacement calculé, à l'aide notamment d'un modèle informatique de lui-même et des interactions avec son environnement. L'objectif est alors de maintenir ce modèle aussi réaliste que possible.

Dans ce mémoire, nous présentons nos principales contributions sur chacun de ces thèmes. Nous avons proposé des solutions (compatibles entre elles) qui puissent fonctionner pour tout type d'environnement (y compris dynamique) et qui soient assez rapides pour faire face aux imprévus, tant sur le robot (eg. dégonflement d'une roue) que sur l'environnement (eg. apparition d'un nouvel obstacle, changement de direction brusque).

1.4.1 Modélisation des déplacements instantanés sûrs

Nous avons proposé un formalisme, baptisé *$\mathcal{V}$-Obstacle Non-Linéaire* (abrégé en *NLVO*), pour représenter un obstacle et sa trajectoire dans l'espace des vitesses linéaires instantanées du robot. À chaque obstacle correspond un *NLVO* (figure 1.8). L'union des *NLVO* permet d'identifier les vitesses du robot entraînant une collision sur une période de temps donnée, le premier obstacle percuté, et le temps auquel cela se produira.

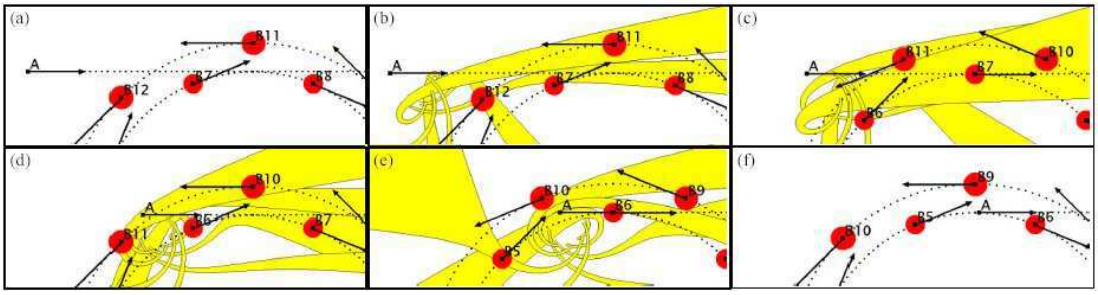

FIG. 1.8 – *Exemple d'application des NLVO* Dans un virage, des obstacles circulent sur deux files de sens opposé. Le robot ($\mathcal{A}$) désire s'insérer dans la file intérieure. Il doit pour cela couper la file extérieure et arriver dans un trou de la file intérieure. À chaque obstacle ($\mathcal{B}_i$) correspond un *NLVO* (en jaune). Il représente les vitesses entraînant une collision avant une période de temps donnée. En b) la vitesse est choisie à l'extérieur des *NLVO* et maintenue constante sur cette période : la séquence permet de vérifier que le robot s'insère correctement, sans entrer en collision avec aucun des obstacles.

En donnant une expression analytique simple d'un *NLVO*, nous en avons rendu possible le calcul en temps-réel. Les apports principaux de la méthode par rapports aux approches existantes sont :

- la prise en compte des trajectoires des obstacles sur un intervalle de temps donné (permettant une meilleure anticipation des collisions)
- la prise en compte de la cinématique et de la dynamique du robot dans une certaine mesure (voir les extensions de la méthode de base au chapitre 4)
- la simplicité des calculs. Elle entraîne des temps de réponse très courts, qui peuvent être réduits encore davantage par le biais d'une carte de rendu graphique 3D, sur laquelle l'approche géométrique du calcul des *NLVO* est facilement implantable (voir page 99).

1.4.2 Choix d'un déplacement sûr

La navigation implique non seulement de sélectionner un déplacement sur des critères de sécurité mais également de le choisir en vue de permettre au robot d'atteindre son but.

Nous avons proposé deux méthodes dérivées des NLVO :

- ***Méthode locale d'évitement d'obstacles***
  Un risque de collision est associé à chaque trajectoire grâce aux NLVO, et un intérêt pour chaque trajectoire est calculé selon l'état courant du robot et la position relative du but à atteindre. La combinaison de ces deux grandeurs dans une fonction de coût à minimiser, décide du meilleur choix de vitesse pour le robot. Comme pour toutes les méthodes de ce type, la fonction de coût peut présenter des minima locaux qui conduisent le robot à des impasses. Cependant, les propriétés d'anticipation intrinsèques aux NLVO permettent de limiter ce phénomène et produisent des trajectoires plus cohérentes en présence d'obstacles rapides, par rapport aux approches courantes.
- ***Méthode itérative de planification de trajectoire sûres***
  Les NLVO sont utilisés comme opérateur d'expansion d'un arbre de recherche. La construction de l'arbre étant incrémentale, la recherche peut être stoppée à tout instant : la méthode retourne alors la meilleure trajectoire parmi celles déjà explorées. Un mécanisme permet éventuellement de ne pas recommencer une exploration complète de l'arbre à chaque décision du robot, mais de la reprendre où elle s'était arrêtée précédemment (en tenant compte des changements survenus entre-temps sur l'environnement).
  Dans le meilleur des cas, cette méthode permet le calcul en temps-réel d'une trajectoire globale sûre du robot en environnement dynamique. Dans le pire des cas, elle assure la survie du robot de façon identique à la méthode d'évitement précédente.

1.4.3 Apprentissage du modèle du robot

Nous avons considéré un véhicule Cycab, dont nous possédons une représentation analytique simplifiée du modèle cinématique. Une loi de commande classique de type Kanayama ([Kanayama et al., 91]) lui permet de suivre une trajectoire de référence. En raison des glissements importants mais non modélisés des roues sur ce véhicule (voir annexe C), des écarts importants apparaissent entre cette dernière et la trajectoire réellement suivie, en particulier à vitesse élevée ou dans les virages.

Dans ce contexte, nous avons proposé de faire apprendre ces glissements, et d'une manière plus générale l'ensemble des paramètres non-modélisés, par un réseau de neurones à partir des données réelles collectées sur le robot. Les approches classiques de ce type (Cf. [Gauthier, 99] pour un aperçu des approches existantes) présentent cependant l'inconvénient de ne pas garantir la cohérence des résultats en cas d'apprentissage insuffisant ou erroné (à cause de données trop bruitées notamment). Nous introduisons ici une sécurité pour maintenir un contrôle satisfaisant du robot lorsque ces cas se présentent.

Nous avons pour cela défini trois entités :

Modèle de référence Le modèle de référence correspond au modèle d'origine ou à tout modèle suffisamment stable et réaliste pour permettre un contrôle satisfaisant du robot en suivi de trajectoire.

Modèle actualisé Le modèle actualisé est synthétisé par un réseau de neurones artificiels dont nous avons mis à profit les capacités d'apprentissage et de généralisation pour approximer le modèle physique réel du robot. Ce modèle est entraîné en ligne à partir de données lues sur les capteurs du robot.

Arbitre L'arbitre s'assure que le niveau d'apprentissage du modèle actualisé, utilisé par défaut, est suffisant. Dans le cas contraire, il force l'utilisation du modèle de référence et évite ainsi les problèmes d'incohérence des résultats obtenus normalement avec un système neuronal.

L'ensemble composé de ces trois entités prend la place du modèle existant et offre ainsi les avantages suivants :

- l'utilisation d'un réseau de neurones limite les problèmes d'identification et d'ajustement des paramètres du système,
- dans le pire des cas (apprentissage erroné ou insuffisant), le modèle équivalent est transparent et se comporte comme le modèle d'origine,
- rapidement, le modèle équivalent admet une erreur moyenne inférieure à 10% par rapport au robot réel, permettant un meilleur contrôle (figure 1.9),
- le modèle équivalent est capable de s'adapter rapidement aux changements (roue dégonflée, sol différent).
- la méthode est très peu invasive et se met en place simplement, en lieu et place d'un modèle existant.

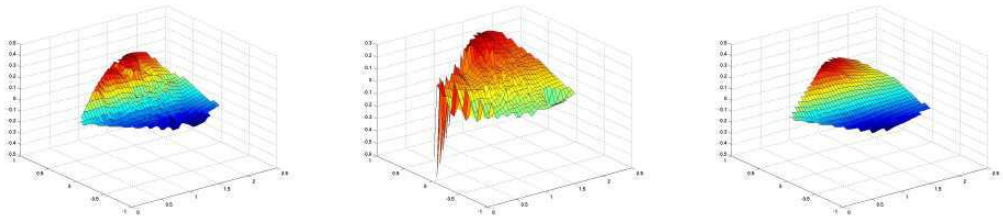

FIG. 1.9 – *Approximation d'un modèle* Les figures montrent l'angle de braquage d'un Cycab en fonction du déplacement désiré, pour une vitesse de rotation des roues donnée. De gauche à droite : modèle réel (données expérimentales), modèle approché (de référence), modèle actualisé (réseau de neurones) : Le modèle actualisé (appris) est plus proche de la réalité que le modèle de référence (modèle analytique d'origine) dont les imprécisions de calculs posent des problèmes aux limites.

Chapitre 2

Concept de $\mathcal{V}$-*Obstacle*

Naviguer en environnement dynamique et incertain demande avant tout de définir un espace de modélisation et un formalisme adapté dans cet espace, afin de pouvoir modéliser efficacement les déplacements libres du robot. L'espace des états-temps est le plus approprié pour représenter à la fois des contraintes cinématiques et dynamiques liées au robot, et des contraintes géométriques liées aux obstacles et à leurs trajectoires. Néanmoins, il n'existe pas de méthode générale de modélisation dans cet espace et sa dimension élevée rend impossible tout calcul de déplacement libre en temps-réel. Nous lui avons préféré un espace de dimension plus réduite à savoir l'espace des vitesses $\mathcal{V}$ ([Burke, 85]), où sont représentées ensembles les positions des objets (robot et obstacles) et leurs vitesses linéaires instantanées. Il permet de prendre en compte les contraintes citées précédemment sous certaines conditions, et sa dimension réduite (2 ou 3) autorise l'utilisation de méthodes géométriques classiques, suffisamment performantes pour le temps-réel.

Dans ce chapitre, nous présentons le concept de $\mathcal{V}$-*Obstacle* (VO) défini dans l'espace des vitesses. Un $\mathcal{V}$-*Obstacle* représente l'ensemble des vitesses linéaires instantanées du robot (donc des trajectoires rectilignes constantes associées) qui entraînent une collision future avec un obstacle avant un *Horizon de Temps* donné. Il prend en compte les trajectoires connues ou estimées des obstacles, et dans une certaine mesure, les caractéristiques cinématiques et dynamiques du robot (Cf. page 53). Nous en proposons une expression analytique et une extension dans l'espace $\mathcal{V} \times \mathcal{T}$ où la dimension $\mathcal{T}$ correspond au temps à collision associé à chaque vitesse de $\mathcal{V}$. Cette donnée sera utilisée au chapitre 3 pour estimer le risque de collision associé à une vitesse particulière du robot en environnement dynamique.

Afin de poser les fondements théoriques des $\mathcal{V}$-*Obstacles* et d'en valider l'approche, nous les étudions ici dans un contexte particulier décrit sous la forme d'hypothèses sur le robot (circulaire et de trajectoire rectiligne constante) et sur les obstacles (circulaires et de trajectoires connues). Leur généralisation à des environnements plus réalistes (quelconques) en vue d'une application pratique de navigation sera l'objet du chapitre 4.

2.1 Hypothèses

Le but recherché ici est de maîtriser la complexité et permettre la modélisation d'un environnement dynamique dans un espace de dimension réduite où s'appliquent les outils classiques de la géométrie algorithmique. Plus précisément, nous désirons représenter l'ensemble des déplacements sûrs du robot dans un tel espace. Nous avons choisi l'espace des vitesses $\mathcal{V}$ ([Burke, 85]), qui est une extension de l'espace de travail, auquel a été ajouté pour chaque objet, sa vitesse linéaire instantanée, représentée par un vecteur $\vec{v}$ attaché à son point de référence. Afin de conserver une faible dimension à cet espace, nous allons poser quelques hypothèses préalables.

Notations L'espace de travail $\mathcal{W}$ du robot est un plan ($\mathcal{W} \equiv \mathbb{R}^2$). Son espace des configurations est noté $\mathcal{C}$ et l'espace des vitesses est noté $\mathcal{V}$ ($\mathcal{W} \equiv \mathcal{V} \equiv \mathbb{R}^2$). Sauf précision contraire, le repère attaché à ces espaces est centré sur le robot. Nous désignons ce dernier par $\mathcal{A}$, et un obstacle par $\mathcal{B}$. $\mathcal{A}(t)$ et $\mathcal{B}(t)$ désignent respectivement $\mathcal{A}$ et $\mathcal{B}$ à l'instant t dans $\mathcal{W}$. Leurs configurations respectives à l'instant t sont notées $c_A(t)$ et $c_B(t)$. Leurs vitesses linéaires instantanées respectives sont représentées par les vecteurs $\overrightarrow{v_A}$ et $\overrightarrow{v_B}$, attachés respectivement à $c_A(t)$ et $c_B(t)$. Nous notons Ω l'environnement peuplé des obstacles $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_n$, et $\mathcal{B}_i$ un obstacle particulier de l'environnement.

Notion d'*Horizon de Temps* Pour des raisons pratiques, il n'est pas possible de modéliser les trajectoires des obstacles mobiles, ni celles du robot, sur une durée infinie. De plus, nous ne désirons pas que le robot considère des collisions potentielles au delà d'un temps raisonnable pour la recherche de déplacements sûrs. Il est donc nécessaire de fixer une borne supérieure aux temps considérés par les calculs. Nous appelons *Horizon de Temps* cette limite et la notons TH. Nous notons t_0 l'instant auquel le déplacement du robot doit débuter. La période de temps considérée par les calculs pour la modélisation des déplacements libres et/ou en collision est donc l'intervalle $[t_0,\ TH]$.

Géométrie du robot et des obstacles Nous posons $\mathcal{A}$ et $\mathcal{B}$ de forme circulaire dans $\mathcal{W}$. Cela se traduit par $\mathcal{W} \equiv \mathcal{C} \equiv \mathcal{V} \equiv \mathbb{R}^2$. c_A et c_B sont leurs centres respectifs pris pour références. Par conséquent :

1. La représentation du robot $\mathcal{A}$ dans $\mathcal{C}$ et dans $\mathcal{V}$ est le point c_A.
2. La représentation d'un obstacle $\mathcal{B}$ dans $\mathcal{C}$ et dans $\mathcal{V}$ est le disque $\mathcal{CB}$ de centre c_B et de rayon r_B, où r_B vaut la somme du rayon de $\mathcal{A}$ et de celui de $\mathcal{B}$ dans $\mathcal{W}$.

Cinématique du robot et des obstacles Le robot (respectivement un obstacle) se déplace selon une trajectoire γ_A (respectivement γ_B) définie par sa position $c_A(t)$ (respectivement $c_B(t)$) et sa vitesse linéaire instantanée $\overrightarrow{v_A}\ (t)$ (respectivement $\overrightarrow{v_B}\ (t)$) à l'instant t. Pour le robot, nous supposerons un mouvement rectiligne constant ($\overrightarrow{v_A}\ (t) = \overrightarrow{v_A}$) sur $[t_0,\ TH]$. Pour les obstacles, les trajectoires sont quelconques mais supposées connues ou estimables sur l'intervalle de temps $[t_0,\ TH]$.

$$\begin{array}{rccl} \gamma_A : & [t_0,\ TH] & \longmapsto & \mathcal{V} \\ & t & \longrightarrow & \left(c_A(t) = c_A(t_0) + (t - t_0) \cdot \overrightarrow{v_A};\ \overrightarrow{v_A}\ (t) = \overrightarrow{v_A}\right) \end{array}$$

$$\begin{array}{rccl} \gamma_B : & [t_0,\ TH] & \longmapsto & \mathcal{V} \\ & t & \longrightarrow & \left(c_B(t);\ \overrightarrow{v_B}\ (t)\right) \end{array}$$

Condition initiale de non-collision Nous supposons le robot en collision avec aucun obstacle à l'instant t_0 :

$$\forall \mathcal{B}_i \in \Omega, \qquad \mathcal{A}(t_0) \cap \mathcal{B}_i(t) = \emptyset$$

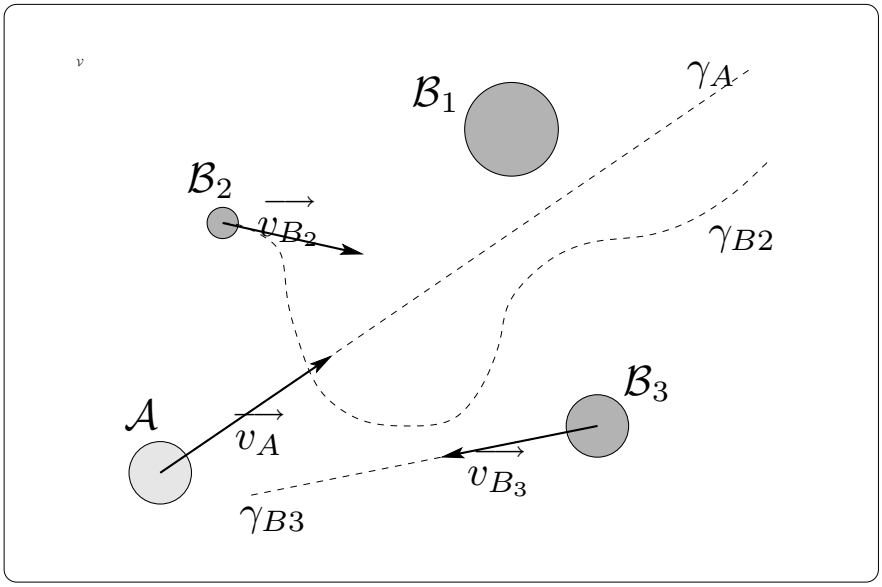

FIG. 2.1 – *Hypothèses* Le robot circulaire $\mathcal{A}$, de mouvement rectiligne constant γ_A, doit éviter les obstacles circulaires $\mathcal{B}_i$, dont les trajectoires γ_{Bi} sont quelconques mais connues.

2.2 Notion de $\mathcal{V}$-*Obstacle* (VO)

2.2.1 Concept général

D'un point de vue général, le concept de $\mathcal{V}$-*Obstacle* est à l'espace $\mathcal{V}$ des vitesses linéaires instantanées du robot ce qu'est le concept de $\mathcal{C}$-*Obstacle* à l'espace $\mathcal{C}$ des configurations : Il permet de représenter dans $\mathcal{V}$ l'ensemble des vitesses linéaires instantanées du robot (et donc ses trajectoires rectilignes constantes) qui entraînent une collision avec un obstacle donné avant un temps borné, appelé l'*Horizon de Temps*.

Le concept de *$\mathcal{V}$-Obstacle* a été introduit dans [Fiorini et Shiller, 93] pour le cas simple d'obstacles se déplaçant en ligne droite et à vitesse constante. Fiorini utilise les *$\mathcal{V}$-Obstacles* pour construire une première approximation d'un déplacement libre d'une voiture dans un contexte autoroutier. Les deux inconvénients principaux de son approche sont un domaine d'application limité en raison de l'hypothèse réductrice sur les déplacements des obstacles, et la forte dépendance des résultats avec le choix de l'*Horizon de Temps*. Nous allons ici en proposer une expression analytique (page 39) permettant de l'étendre à des trajectoires quelconques des obstacles (page 42), et de remplacer l'information booléenne de collision ou non-collision par le temps à collision associé à chaque vitesse (page 45), ce qui permet de ne pas écarter de solution mais au contraire d'ajouter de l'information et réduire ainsi la dépendance au TH.

2.2.2 Définitions

2.2.2.1 *$\mathcal{V}$-Obstacle* VO

Soit un robot $\mathcal{A}$ et un obstacle $\mathcal{B}$. Soit TH un *Horizon de Temps*. Le *$\mathcal{V}$-Obstacle* de $\mathcal{A}$ associé à $\mathcal{B}$, est l'ensemble noté VO des vitesses linéaires constantes appliquées à $\mathcal{A}$ à l'instant t_0, qui entraînent une collision future entre $\mathcal{A}$ et $\mathcal{B}$ sur l'intervalle de temps $[t_0,\ TH]$. Il est défini dans l'espace des vitesses $\mathcal{V}$.

$$VO = \bigcup \left\{ \overrightarrow{v_A} \in \mathcal{V} \;\middle|\; \exists t \in [t_0,\ TH],\ \mathcal{A}(t) \cap \mathcal{B}(t) \neq \emptyset \right\}$$

Cela s'écrit également :

$$VO = \bigcup \left\{ \overrightarrow{v_A} \in \mathcal{V} \;\middle|\; \exists t \in [t_0,\ TH],\ c_A(t) \in \mathcal{CB}(t) \right\}$$

Notion d'élément temporel de VO L'élément temporel d'un VO noté $VO(t)$ est l'ensemble des vitesses linéaires du robot telles que le robot et l'obstacle sont en collision au temps t.

$$\begin{aligned} VO(t) &= \bigcup \left\{ \overrightarrow{v_A} \in \mathcal{V} \;\middle|\; \mathcal{A}(t) \cap \mathcal{B}(t) \neq \emptyset \right\} \\ &= \bigcup \left\{ \overrightarrow{v_A} \in \mathcal{V} \;\middle|\; c_A(t_0) + (t - t_0) \cdot \overrightarrow{v_A} \in \mathcal{CB}(t) \right\} \end{aligned}$$

Un *$\mathcal{V}$-Obstacle* est l'union de ses éléments temporels sur $[t_0,\ TH]$:

$$VO = \bigcup_{t=t_0}^{TH} VO(t)$$

D'un point de vue géométrique, la représentation dans $\mathcal{V}$ de l'élément temporel $VO(t)$ est l'image de $\mathcal{CB}(t)$ par l'homothétie de centre $c_A(t_0)$ et de rapport $1/(t - t_0)$.

Dans le cas présent, il s'agit par conséquent du disque de centre $c_{VO}(t)$ et de rayon $r_{VO}(t)$ (figure 2.2) définis par :

$$VO(t) \begin{cases} c_{VO}(t) &= c_A(t_0) + \frac{1}{(t-t_0)} \cdot \overrightarrow{AB} \\ r_{VO}(t) &= \frac{1}{(t-t_0)} \cdot r_B \end{cases}$$

où $\overrightarrow{AB}$ est le vecteur attaché en $c_A(t_0)$ et dont l'extrémité est $c_B(t)$.

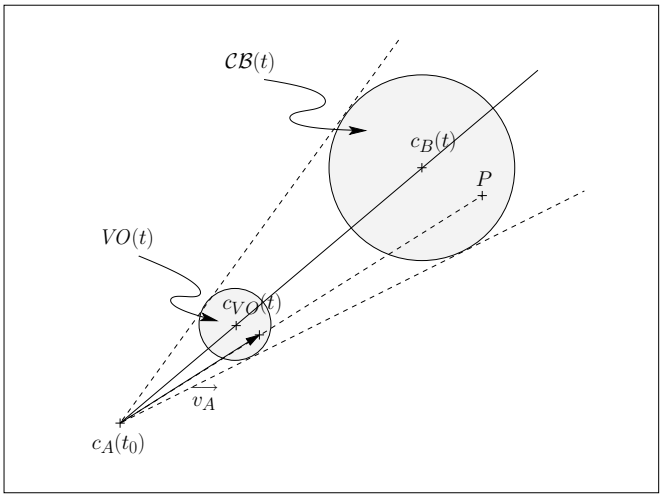

FIG. 2.2 – *Élément temporel de VO* $VO(t)$ est l'image de $\mathcal{CB}(t)$ par l'homothétie de centre $c_A(t_0)$ et de rapport $1/(t-t_0)$. À tout point P appartenant à $\mathcal{CB}(t)$ correspond une vitesse de collision $\overrightarrow{v_A}$ appartenant à $VO(t)$, qui entraîne $\mathcal{A}(t) \cap \mathcal{B}(t) \neq \emptyset$.

$\mathcal{V}$-Obstacle VO_Ω associé à plusieurs obstacles Nous avons jusqu'alors considéré le cas d'un seul obstacle. Soit VO_i le VO de $\mathcal{A}$ associé à un obstacle $\mathcal{B}_i$ particulier de l'environnement Ω ($i \in \{1, \ldots, n\}$). L'ensemble de toutes les vitesses linéaires qui entraînent au moins une collision entre le robot et un obstacle de Ω sur l'intervalle de temps $[t_0, TH]$ est l'union de tous les VO_i.

$$VO_\Omega = \bigcup_{i=1}^{n} VO_i = \bigcup_{i=1}^{n} \bigcup_{t=t_0}^{TH} VO_i(t)$$

2.2.2.2 *Vitesses de Collision V_{col}* et *Vitesses de Contact $V_{contact}$*

Parmi les vitesses d'un VO (i.e. qui entraînent des collisions), nous ferons la distinction entre :

- l'ensemble des **Vitesses de Collision** (notées V_{col})
  Les trajectoires du robot qui correspondent à ces vitesses terminent dans l'obstacle ou le traversent.

– l'ensemble des **Vitesses de Contact** (notées $V_{contact}$)
Les trajectoires du robot qui correspondent à ces vitesses et la trajectoire de l'obstacle sont tangentes et le robot ne fait que frôler l'obstacle.

Soit le robot $\mathcal{A}$ et l'obstacle $\mathcal{B}$ définis dans $\mathcal{W}$. Nous notons $\delta\mathcal{B}$ la frontière de $\mathcal{B}$ et $\overset{\circ}{\mathcal{B}}$ l'obstacle $\mathcal{B}$ privé de sa frontière. Soit VO le $\mathcal{V}$-*Obstacle* de $\mathcal{A}$ associé à $\mathcal{B}$ pour la période $[t_0,\ TH]$, et $\overrightarrow{v_A}$ une vitesse de $\mathcal{A}$ appartenant à VO, nous pouvons écrire que.

$$V_{col} = \bigcup \left\{ \overrightarrow{v_A} \in VO \;\middle|\; \exists t \in [t_0,\ TH],\ \mathcal{A}(t) \cap \overset{\circ}{\mathcal{B}}(t) \neq \emptyset \right\}$$
$$V_{contact} = \bigcup \left\{ \overrightarrow{v_A} \in VO \;\middle|\; \forall t \in [t_0,\ TH],\ \mathcal{A}(t) \cap \overset{\circ}{\mathcal{B}}(t) = \emptyset \right\}$$

Les deux ensembles sont exclusifs pour une même valeur de t et nous avons :

$$VO(t) = V_{col}(t) \cup V_{contact}(t) \qquad et \qquad V_{col}(t) \cap V_{contact}(t) = \emptyset$$

2.2.2.3 Ensemble des vitesses libres sur TH : V_{free}

Après avoir défini l'ensemble des vitesses qui entraînent une collision, nous allons maintenant définir son complément dans $\mathcal{V}$, l'ensemble des vitesses libres.

Pour un obstacle unique Nous appelons ensemble des vitesses libres, l'ensemble complémentaire de VO dans $\mathcal{V}$. Il est noté V_{free} et représente l'ensemble des vitesses linéaires instantanées de $\mathcal{A}$ qui garantissent que le robot ne percutera pas l'obstacle associé à VO sur la période de temps $[t_0,\ TH]$.

$$V_{free} = \overline{VO} = \left\{ \overrightarrow{v_A} \in \mathcal{V} \;\middle|\; \overrightarrow{v_A} \notin VO \right\}$$

Pour plusieurs obstacles Soit Ω l'environnement peuplé des obstacles $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_n$ et VO_Ω l'ensemble de toutes les vitesses linéaires qui entraînent au moins une collision entre le robot et au moins un obstacle $\mathcal{B}_i$ sur la période $[t_0,\ TH]$. V_{free_Ω} représente l'ensemble des vitesses linéaires instantanées qui n'appartiennent pas à VO_Ω.

$$V_{free_\Omega} = \overline{VO_\Omega} = \left\{ \overrightarrow{v_A} \in \mathcal{V} \;\middle|\; \overrightarrow{v_A} \notin VO_\Omega \right\}$$

Propriété Par définition, V_{free_Ω} représente l'ensemble des vitesses linéaires instantanées qui n'entraînent pas de collision avec les obstacles de Ω sur l'intervalle de temps $[t_0,\ TH]$.

$$V_{free_\Omega} = \left\{ \overrightarrow{v_A} \in \mathcal{V} \;\middle|\; \forall t \in [t_0,\ TH],\ \forall \mathcal{B}_i \in \Omega,\ \mathcal{A}(t) \cap \mathcal{B}_i(t) = \emptyset \right\}$$

Sélectionner une vitesse du robot extérieure à V_{free_Ω} garantit dans ces conditions que le robot évitera tous les obstacles sur l'intervalle de temps $[t_0,\ TH]$, sous réserve que les trajectoires des obstacles soient conformes aux prédictions utilisées pour les calculs. C'est cette propriété que nous allons exploiter plus tard pour éviter les obstacles (figure 2.3).

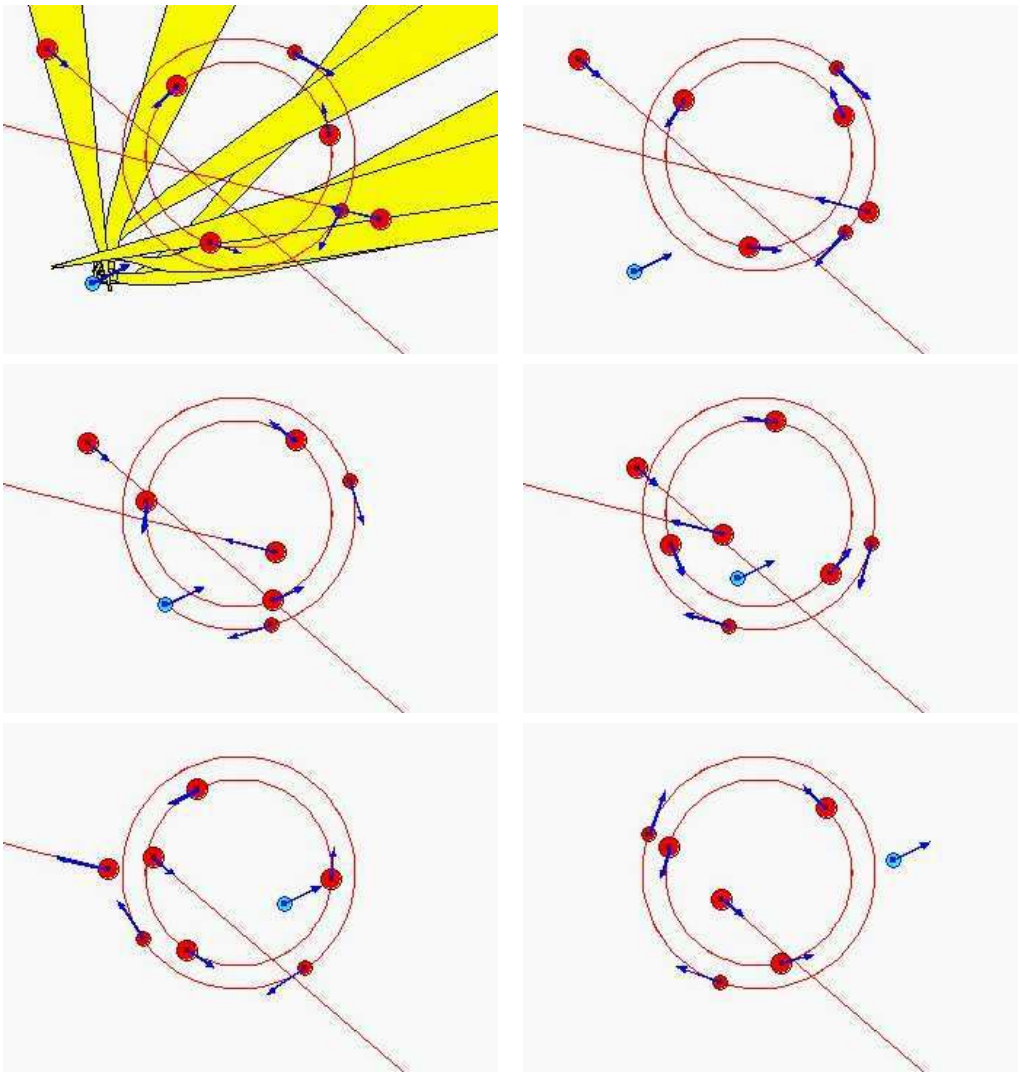

FIG. 2.3 – *Propriété de* V_{free_Ω} Sélectionner une vitesse extérieure aux *NLVO* (en jaune) donc appartenant à V_{free_Ω}, garantit que le robot n'entre pas en collision sur l'intervalle de temps considéré par les calculs, sous réserve que les trajectoires des obstacles aient été correctement estimées sur ce même intervalle.

2.3 Construction de VO pour $\overrightarrow{v_B}(t)$ constant (LVO)

Notion de $\mathcal{V}$-*Obstacle Linéaire* (LVO) Pour le cas particulier d'un obstacle se déplaçant en ligne droite à vitesse $\overrightarrow{v_B}$ constante ($\overrightarrow{v_B}(t) = \overrightarrow{v_B}$), nous allons montrer que le VO correspondant prend la forme d'un cône dans $\mathcal{V}$. La pointe du cône est « tronquée » selon la valeur de TH choisie. Ce type particulier de $\mathcal{V}$-*Obstacle* est dit

$\mathcal{V}$-Obstacle Linéaire et sera noté *LVO*.

$$LVO = VO \Longleftrightarrow \forall t \in [t_0,\ TH],\ \overrightarrow{v_B}\ (t) = \overrightarrow{v_B}$$

2.3.1 Méthode générale de construction

La construction d'un *LVO* dans $\mathcal{V}$ se fait en trois étapes (figure 2.4) :

- Construire le cône issu de $c_A(t_0)$ et dont les bords sont tangents à $\mathcal{CB}(t_0)$. Cet ensemble représente l'ensemble des vitesses linéaires relatives de $\mathcal{A}$ par rapport à $\mathcal{B}$ qui entraînent une collision future[1] avec $\mathcal{B}$. Il est appelé *Cône de Collision* et noté *CC*.
- Translater *CC* de $\overrightarrow{v_B}$ pour obtenir *LVO*.
- Supprimer les vitesses qui entraînent des collisions après *TH*. Elles constituent la « pointe » du cône située « après » $VO(TH)$.

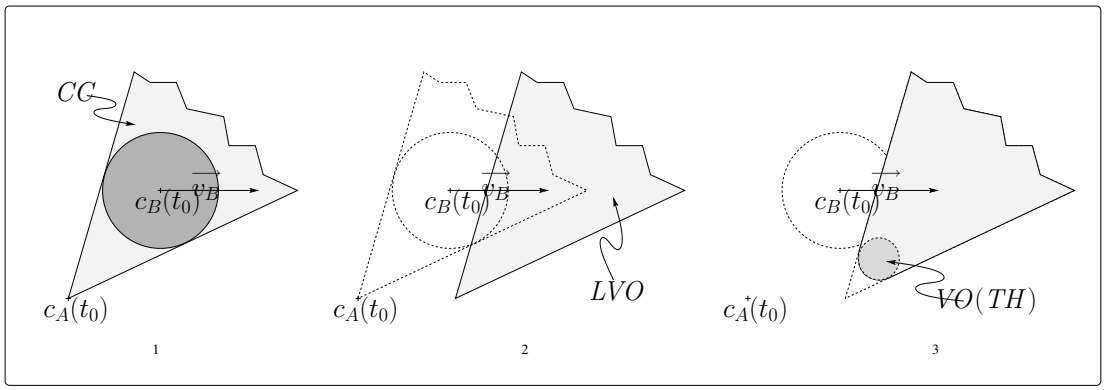

FIG. 2.4 – *Construction de LVO* 1. Construction du *Cône de Collision*, 2. Translation de $\overrightarrow{v_B}$, 3. Suppression des $VO(t)$ pour $t > TH$.

2.3.2 Détail des étapes successives et justifications

Construction du *Cône de Collision* *CC* Soit un obstacle $\mathcal{B}$ en mouvement rectiligne constant de vitesse $\overrightarrow{v_B}$. Nous notons $\overrightarrow{v_{A/B}}$ la vitesse linéaire relative du robot $\mathcal{A}$ par rapport à l'obstacle $\mathcal{B}$ telle que :

$$\overrightarrow{v_{A/B}} = \overrightarrow{v_A} - \overrightarrow{v_B}$$

Soit $\gamma_{A/B}$ la trajectoire relative de $\mathcal{A}$ par rapport à $\mathcal{B}$ définie par :

$$\begin{array}{rccl} \gamma_{A/B} : & [t_0,\ \infty[& \longmapsto & \mathcal{V} \\ & t & \longrightarrow & \left(c_A(t) = c_A(t_0) + (t - t_0) \cdot \overrightarrow{v_{A/B}};\ \overrightarrow{v_A}\ (t) = \overrightarrow{v_{A/B}}\right) \end{array}$$

[1] la collision aura lieu à un temps $t \in\ ]t_0,\ \infty[$

Considérer les vitesses/trajectoires relatives de $\mathcal{A}$ par rapport à $\mathcal{B}$ donne l'illusion que l'obstacle $\mathcal{B}$ est statique. Dans ces conditions, l'ensemble des trajectoires rectilignes relatives de $\mathcal{A}$ par rapport à $\mathcal{B}$ qui entraînent une collision future avec l'obstacle sont les demi-droites issues de $c_A(t_0)$ et qui intersectent $\mathcal{CB}(t_0)$ ou lui sont tangentes. Elles forment un cône issu de $c_A(t_0)$ dont les bords sont les trajectoires tangentes à $\mathcal{CB}(t_0)$ (figure 2.5).

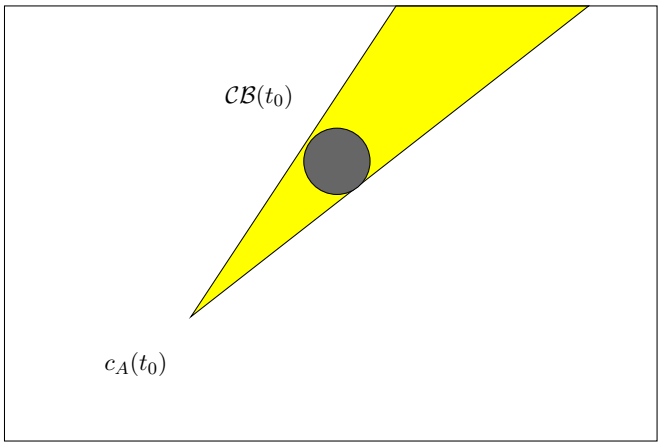

FIG. 2.5 – *Trajectoires rectilignes du robot $\mathcal{A}$ en collision avec un obstacle statique $\mathcal{B}$.*

Puisque l'espace de modélisation est $\mathcal{V}$ et d'après nos hypothèses, les trajectoires correspondent à des vitesses. Par conséquent, le cône que nous avons défini représente aussi l'ensemble des vitesses linéaires relatives du robot qui entraînent des collisions futures entre le robot et l'obstacle. Nous appellerons ce cône *Cône de Collision* et le noterons CC.

$$CC = \bigcup \left\{ \overrightarrow{v_A} \in \mathcal{V} \;\middle|\; \forall t \in [t_0, \infty[,\ c_A(t_0) + t \cdot \overrightarrow{v_A} \in \mathcal{CB}(t_0) \right\}$$

Remarque CC permet de modéliser les vitesses relatives en collision pour un couple robot-obstacle donné et non pour l'ensemble des obstacles. Il est nécessaire pour cela de travailler dans l'espace des vitesses absolues du robot, indépendant des obstacles. Nous allons donc identifier les vitesses absolues du robot qui correspondent aux vitesses relatives de CC.

Construire LVO en translatant CC de $\overrightarrow{v_B}$ Par définition, la vitesse absolue du robot s'obtient simplement à partir de la vitesse de $\mathcal{B}$ et de la vitesse relative de $\mathcal{A}$ par rapport à $\mathcal{B}$:

$$\overrightarrow{v_A} = \overrightarrow{v_{A/B}} + \overrightarrow{v_B}$$

Ceci est vrai en particulier pour toute vitesse relative $\overrightarrow{v_{A/B}}$ appartenant à CC. Par conséquent, il est possible de construire simplement l'ensemble des vitesses absolues du

robot qui correspondent aux vitesses relatives de CC en translatant CC de $\overrightarrow{v_B}$. Nous appellerons *$\mathcal{V}$-Obstacle Linéaire* l'ensemble obtenu, abrégé en LVO .

$$\begin{aligned} LVO &= CC + \overrightarrow{v_B} \\ &= \left\{ \overrightarrow{v_A} \in \mathcal{V} \;\middle|\; (\overrightarrow{v_A} - \overrightarrow{v_B}) \in CC \right\} \end{aligned}$$

Suppression des vitesses en collision après TH Outre l'impossibilité d'effectuer des calculs sur un intervalle de temps infini, ne pas limiter la valeur de TH peut s'avérer contraignant dans certains cas. Prenons l'exemple de la figure 2.6. Quelque soit la trajectoire suivie par le robot, elle est en collision avec l'un des murs. Ne pas borner la valeur de TH implique alors $V_{free_\Omega} = \emptyset$, qui signifie l'absence de déplacement libre. Au contraire, en bornant cette valeur, nous choisissons de ne pas considérer les vitesses qui entraînent des collisions après TH, ce qui laisse apparaître des solutions. Il s'agit des vitesses de faible amplitude qui constituent les « pointes » des LVO associés aux différents obstacles.

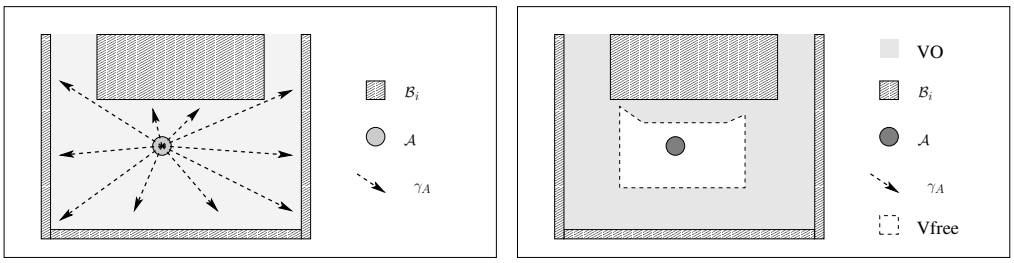

FIG. 2.6 – $V_{free_\Omega} = \emptyset$ *pour* $TH \rightarrow \infty$ Bien que le robot admette des vitesses solutions qui lui permettent de sortir de la pièce modélisée, elles ne sont pas « visibles » à l'aide des VO si $TH \rightarrow \infty$ (à gauche). En revanche, avec un TH borné, alors $V_{free_\Omega} \neq \emptyset$. Le robot peut se déplacer.

Soit l'intervalle de temps $[t_0,\ TH]$. Soient $VO(t_0)$ et $VO(TH)$ les éléments temporels associés aux bornes de cet intervalle. Les vitesses du LVO qui entraînent une collision dans l'intervalle de temps $[t_0,\ TH]$ constituent la portion du LVO délimitée par ces deux disques (frontières des disques comprises).

Par définition, $VO(t_0)$ représente l'ensemble des vitesses de $\mathcal{A}$ qui, appliquées à l'instant t_0, entraînent une collision à ce même instant. L'amplitude de ces vitesses ne peut être qu'infinie (car nous supposons $\mathcal{A}(t_0) \cap \mathcal{B}(t_0) = \emptyset$) et il n'est donc pas nécessaire de considérer cette borne. En conséquence, seules les vitesses qui entraînent une collision après TH sont à supprimer, d'où la suppression seulement des $\overrightarrow{v_A} \in VO(t)$ pour $t > TH$ et telles que $\overrightarrow{v_A} \notin VO(t)$ pour $t \leq TH$.

2.3.3 Expression analytique du LVO

L'approche géométrique de la construction d'un VO vue précédemment permet de déduire une expression analytique simple des « bords » du VO, paramétrée par le temps à collision.

Ces « bords » sont constitués des *Vitesses de Contact*. En effet, par construction, ils correspondent aux « bords » du cône CC, qui correspondent eux-mêmes aux vitesses de $V_{contact}$. Il s'agit des deux demi-droites issues de $c_A(t_0)$ et tangentes à $\mathcal{CB}(t_0)$ aux points que nous noterons cc_g et cc_d.

Nous utilisons les notations complexes pour représenter chaque vitesse de $\mathcal{V}$ dans un repère centré en $c_A(t_0)$. Cela nous donne :

$$vo_g(t) = \frac{cc_g}{(t-t_0)} + \overrightarrow{v_B}$$

$$vo_d(t) = \frac{cc_d}{(t-t_0)} + \overrightarrow{v_B}$$

Où $vo_g(t)$ et $vo_d(t)$ représentent les points qui constituent les « bords » du LVO, et qui correspondent à des vitesses entraînant une collision tangentielle de $\mathcal{A}$ avec $\mathcal{B}$ à l'instant t.

Remarque Les points $vo_d(t)$ (respectivement $vo_g(t)$) sont alignés pour $t \in [t_0,\ TH]$. Par conséquent, le calcul de $vo_d(t)$ (respectivement de $vo_g(t)$) pour 2 valeurs différentes de t (et en particulier pour $t = t_0$ et $t = TH$), suffit à définir les bords du LVO. Chaque $\delta VO(t)$ est scindé en deux arcs de cercle par $vo_d(t)$ et $vo_g(t)$. Pour $t = TH$, le plus petit arc constitue la frontière du LVO. Pour $t = t_0$, la frontière du LVO est le plus grand arc, situé à l'infini (figure 2.7).

2.4 Construction de VO pour $\overrightarrow{v_B}\,(t)$ non-constant ($NLVO$)

Notion de $\mathcal{V}$-*Obstacle Non-Linéaire* ($NLVO$) Un *$\mathcal{V}$-Obstacle Non-Linéaire* est une généralisation du VO pour le cas où la direction et/ou l'amplitude de $\overrightarrow{v_B}\,(t)$ ne sont pas constantes au cours du temps.

$$NLVO = VO \Longleftrightarrow \exists (t_1,\, t_2) \in [t_0,\ TH]^2 \;\Big|\; t_1 \neq t_2 \text{ et } \overrightarrow{v_B}\,(t_1) \neq \overrightarrow{v_B}\,(t_2)$$

2.4.1 Méthode générale de Construction

La construction du $NLVO$ n'est pas directe comme pour le LVO. La trajectoire d'un obstacle étant quelconque, il n'est pas possible de connaître a priori la forme du *$\mathcal{V}$-Obstacle* correspondant et sa construction doit être itérative. L'approche géométrique consisterait à construire l'union des $VO(t)$ pour différentes valeurs de t prises

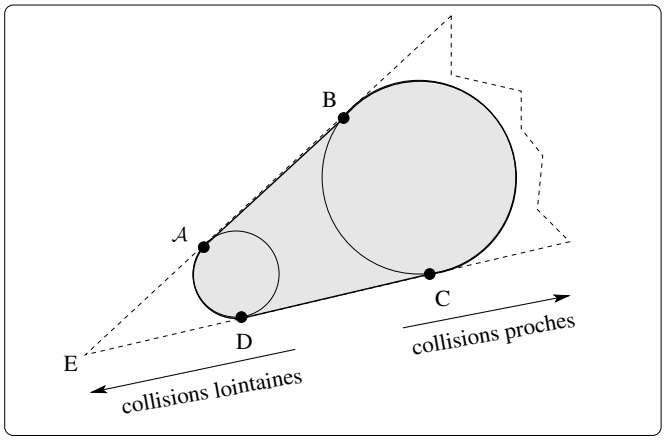

FIG. 2.7 – *Traits remarquables d'un LVO.* Les points A et D correspondent respectivement à $vo_g(TH)$ et $vo_d(TH)$. Les points B et C correspondent respectivement à $vo_g(t_0)$ et $vo_d(t_0)$. Les segments [AB] et [DC], le petit arc de cercle $\widehat{AD}$ et le grand arc de cercle $\widehat{BC}$, suffisent à définir les contours d'un *LVO*. Pour remarque, B et C sont situés à l'infini.

sur l'intervalle de temps $[t_0,\ TH]$. Cela demanderait trop de calculs, et nous présenterons directement une approche moins coûteuse, basée sur l'expression analytique des « bords » du *NLVO* obtenue par analogie avec les *LVO*.

Le principe général de la méthode de construction est le suivant : Pour chaque valeur de t prise dans l'intervalle de temps $[t_0,\ TH]$, nous calculons les points $vo_g(t)$ et $vo_d(t)$ selon la méthode suivante (figure 2.8) :

- Considérer que l'obstacle $\mathcal{B}(t)$ effectue un déplacement rectiligne uniforme à vitesse $\overrightarrow{v_B}\ (t)$ constante et calculer, sous ces hypothèses, une estimation de $\mathcal{B}(t)$ à $t = t_0$. Cette estimation est notée $\mathcal{B}_\circ$.
- Calculer le *LVO* associé à $\mathcal{B}_\circ$ noté $LVO_\circ$.
- Calculer l'élément temporel de $LVO_\circ$ associé au temps t. Il est noté $VO_\circ(t)$
- Calculer les points de tangence $vo_g(t)$ et $vo_d(t)$ de $VO_\circ(t)$ selon la méthode décrite pour les *LVO*. Ce sont également les bords du *NLVO* correspondant à une collision tangentielle de $\mathcal{A}$ avec $\mathcal{B}$ à l'instant t.

2.4.2 Justification de la méthode de construction

Nous nous appuyons ici sur le fait qu'un *LVO* peut être vu comme l'approximation d'ordre 1 du *NLVO* à l'instant t, tout comme la vitesse $\overrightarrow{v_B}\ (t)$ de $\mathcal{B}(t)$ est l'approximation d'ordre 1 de la trajectoire de $\mathcal{B}$ à l'instant t.

Calcul de $\mathcal{B}_\circ$ Soit $\mathcal{B}(t)$ l'obstacle $\mathcal{B}$ de vitesse linéaire instantanée $\overrightarrow{v_B}\ (t)$ à l'instant t. L'obstacle $\mathcal{B}_\circ$ est tel que si $\overrightarrow{v_B}\ (t)$ était constante sur la période $[t_0,\ t]$, alors l'obstacle $\mathcal{B}_\circ$ représenterait l'obstacle $\mathcal{B}$ à l'instant t_0 :

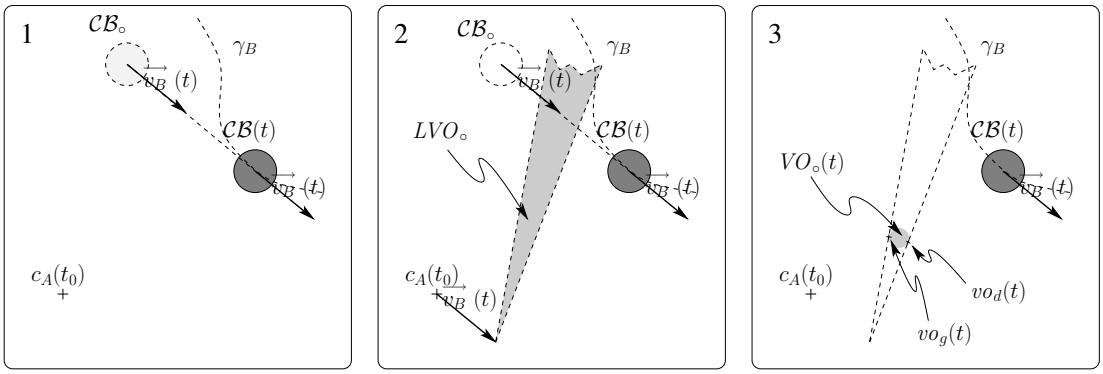

FIG. 2.8 – *Construction de NLVO* 1. Construction de l'obstacle virtuel $\mathcal{B}_\circ$ et de son image $\mathcal{CB}_\circ$ dans $\mathcal{V}$, 2. Calcul du *LVO* associé à $\mathcal{B}_\circ$ et noté $LVO_\circ$, 3. Calcul des points de tangence $vo_g(t)$ et $vo_d(t)$.

Il est obtenu en translatant $\mathcal{B}(t)$ de $-(t-t_0)\cdot\overrightarrow{v_B}(t)$. Le rayon $\mathcal{B}$ étant constant, le rayon de $\mathcal{B}_\circ$ est r_B. Sa vitesse, supposée constante, est $\overrightarrow{v_B}(t)$. Enfin, le centre de $c_{B_\circ}$ est noté $c_{B_\circ}$. Il est obtenu par :

$$c_{B_\circ} = c_B(t) - (t-t_0)\cdot\overrightarrow{v_B}(t)$$

Le *LVO* associé à $\mathcal{B}_\circ$, noté $LVO_\circ$, est construit selon l'approche décrite précédemment.

Recherche des points de tangence entre $\mathcal{A}$ et $\mathcal{B}$ Afin de valider l'approche, nous devons vérifier que les points $vo_g(t)$ et $vo_d(t)$ situés sur $LVO_\circ$ représentent bien les *Vitesses de Contact* du *NLVO*.

Par définition de $VO_\circ(t)$, nous pouvons déduire que $VO_\circ(t) = VO(t)$ pour $t \in [t_0,\ TH]$:

$$\begin{array}{rcl} VO_\circ(t) & = & \left(\left(\mathcal{CB}(t) - (t-t_0)\cdot\overrightarrow{v_B}(t)\right)\cdot\frac{1}{(t-t_0)}\right) + \overrightarrow{v_B}(t) \\ & = & \frac{1}{(t-t_0)}\cdot\mathcal{CB}(t) \\ & = & VO(t) \end{array}$$

Or, par définition également, les *Vitesses de Contact* de $VO_\circ(t)$ telles que les trajectoires de $\mathcal{A}$ et de $\mathcal{B}$ soient tangentes à l'instant t sont représentées par les points $vo_g(t)$ et $vo_d(t)$. Cela est vrai car la relation de tangence étant d'ordre 1, seules les vitesses instantannées du robot et de l'obstacle au point de tangence importent et non leurs trajectoires.

Par conséquent, les points de tangence de $VO(t)$ dans un *NLVO* s'écrivent comme pour le *LVO* associé à $\mathcal{B}_\circ$:

$$vo_g(t) = \frac{cc_{g_\circ}}{(t-t_0)} + \overrightarrow{v_B}$$

$$vo_d(t) = \frac{cc_{d_\circ}}{(t-t_0)} + \overrightarrow{v_B}$$

où $cc_{g_\circ}$ et $cc_{d_\circ}$ sont les points de tangence entre les demi-droites issues de $c_A(t_0)$ et $\mathcal{CB}_\circ$.

2.4.3 Singularité : $vo_g(t)$ et $vo_d(t)$ inexistants quand $c_A(t_0) \in \mathcal{CB}_\circ$

La méthode de construction des *NLVO* décrite précédemment implique la construction de $\mathcal{CB}_\circ$ pour chaque valeur de $t \in [t_0,\ TH]$. Il peut cependant arriver que le cas $c_A(t_0) \in \mathcal{CB}_\circ$ se présente (figure 2.9). La construction de $LVO_\circ$ n'est alors pas possible. En effet, le calcul d'un $\mathcal{V}$-*Obstacle* n'a de sens et n'est possible que pour $\mathcal{A}(t_0) \cap \mathcal{B}(t_0) = \emptyset$ puisque dans le cas contraire, $\mathcal{A}$ et $\mathcal{B}$ étant déjà en collision à l'instant t_0, toutes les vitesses ou trajectoires de $\mathcal{A}$ entraîneraient une collision et par conséquent nous aurions $V_{free} = \emptyset$.

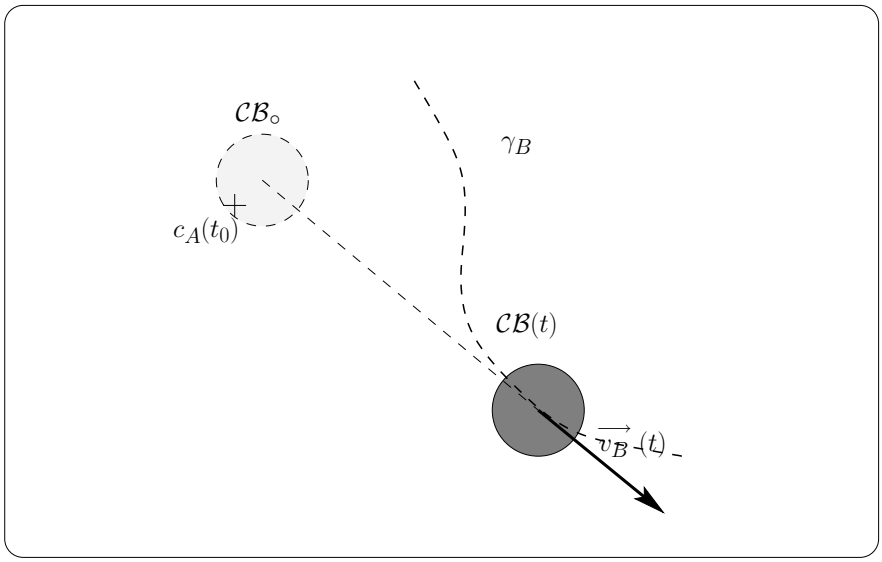

FIG. 2.9 – *Cas où* $c_A(t_0) \in \mathcal{B}_\circ$ La construction de $LVO_\circ$ qui suppose $c_A(t_0) \notin \mathcal{CB}_\circ$ n'est pas possible.

En réalité, par hypothèse $\mathcal{A}(t_0) \cap \mathcal{B}(t_0) = \emptyset$ et donc $\mathcal{B}_\circ \neq \mathcal{B}(t_0)$. Par conséquent, l'élément temporel $VO(t)$ existe bien et reste calculable par la méthode jusqu'alors utilisée (introduite page 32). Seuls les points $vo_g(t)$ et $vo_d(t)$ ne peuvent pas être obtenus.

2.4.3.1 Explication du phénomène et interprétation

L'explication se trouve dans la forme du $\mathcal{V}$-*Obstacle* aux points de singularité : Pour tout t, l'élément temporel $VO(t)$ est, par construction, l'image de $\mathcal{CB}_\circ$ par l'homothétie

de centre $c_A(t_0)$ et de rapport $\frac{1}{(t-t_0)}$. Or aux points de singularité, $c_A(t_0)$ est situé à l'intérieur de $\mathcal{CB}_\circ$, et par conséquent, les éléments temporels ne sont pas sécants, mais inclus les uns dans les autres (figure 2.10).

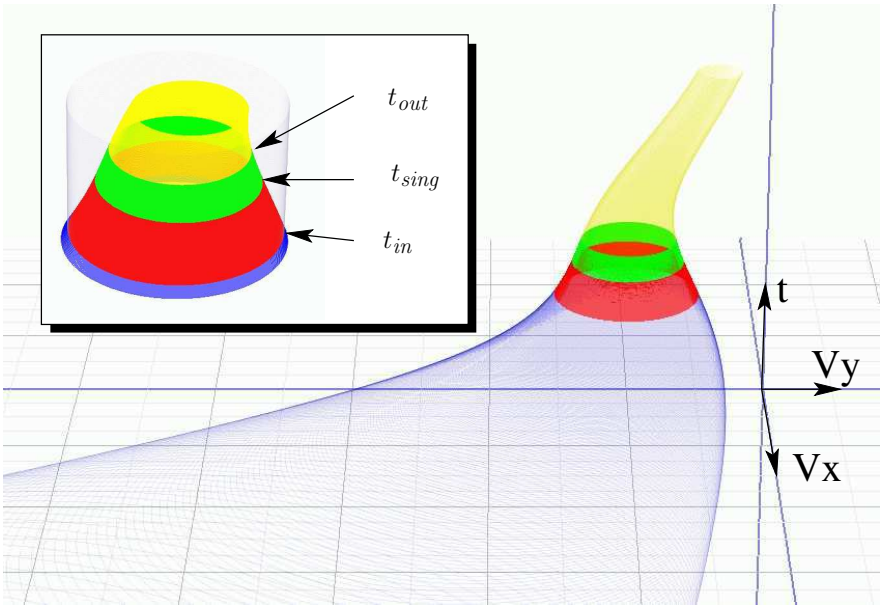

FIG. 2.10 – *Représentation d'un cas de singularité dans $\mathcal{V} \times \mathcal{T}$* Afin de mieux comprendre ce qui se passe, nous avons représenté un *$\mathcal{V}$-Obstacle Non-Linéaire* dans l'espace $\mathcal{V} \times \mathcal{T}$, où la dimension de temps correspond au temps à collision associé aux vitesses. t_{sing} est tel que $\mathcal{A}(t_{sing}) = \mathcal{B}_\circ(t_{sing})$. t_{in} et t_{out} sont respectivement le temps où $c_A(t)$ « entre » et « sort » de $\mathcal{CB}_\circ$ (avec $t_{in} < t_{out}$). En projetant orthogonalement dans $\mathcal{V}$, nous observons que les $VO(t)$ pour $t \in]t_{in},\ t_{out}[$ sont tous inclus dans $VO(t_{in})$ (représenté par le cylindre translucide dans le médaillon en haut à gauche).

Nous considérons les temps t_{in} et t_{out} définis de la manière suivante :

$$\begin{cases} \text{Pour } t \in]t_{in},\ t_{out}[, & c_A(t_0) \in \mathcal{CB}_\circ \\ \text{Pour } t = t_{in} \text{ ou } t = t_{out}, & c_A(t_0) \notin \mathcal{CB}_\circ \end{cases}$$

Sur l'intervalle de temps $]t_{in},\ t_{out}[$ (voir figure 2.10), toute vitesse permettant à $\mathcal{A}$ de frôler les bords de $\mathcal{B}$ à l'instant t (vitesses appartenant à $\delta\, VO(t)$) entraîne une collision avec $\mathcal{B}$ à un instant t_1 antérieur à t :

$$\forall t \in]t_{in},\ t_{out}[, \qquad \overrightarrow{v_A} \in \delta\, VO(t) \Longrightarrow \exists t_1,\ t_1 \in [t_0,\ t[\ \Big|\ \overrightarrow{v_A} \in VO(t_1)$$

Pour tout t tel que $c_A(t_0) \in (\mathcal{CB}(t) - (t - t_0) \cdot \overrightarrow{v_B}\ (t))$, nous pouvons affirmer qu'il n'existe pas de trajectoire rectiligne uniforme de $\mathcal{A}$ qui frôle $\mathcal{B}$ à l'instant t sans l'avoir frôlé/percuté auparavant. Ceci explique l'impossibilité de calculer des *Vitesses de Contact* pour ces valeurs de t, puisqu'elles n'existent pas.

Il est cependant nécessaire de calculer les « bords » du *$\mathcal{V}$-Obstacle* pour ces valeurs de t. $\delta VO(t_{in})$ sera pris comme « bord » du *$\mathcal{V}$-Obstacle* pour l'ensemble des valeurs de t prises dans l'intervalle de temps $[t_{in},\ t_{out}]$ et t_{in} sera le temps à collision associé aux vitesses qui le constituent. Nous tirerons parti de cette propriété pour implémenter la construction d'un *NLVO* dans $\mathcal{V} \times \mathcal{T}$, décrite au chapitre 4 (page 102).

2.4.3.2 Définition de points de tangence virtuels

Afin de conserver une expression analytique des bords $vo_g(t)$ et $vo_d(t)$ du *NLVO* pour toute valeur de t, nous proposons de calculer des remplaçants virtuels de ces points. Nous considérons le fait que l'orientation de l'axe central d'un *$\mathcal{V}$-Obstacle* est continue pour toute valeur de t sauf lorsque la trajectoire de l'obstacle n'est pas continue ou que $c_A(t_0)$ et le centre de $\mathcal{CB}_\circ$ sont strictement confondus. Pour ces valeurs singulières de t notées t_{sing}, nous procéderons par interpolation : nous définissons ϵ_t tel que $\epsilon_t \to 0$, et interpolons linéairement[2] les points de tangence à partir de ceux à $(t_{sing} - \epsilon_t)$ et $(t_{sing} + \epsilon_t)$. Pour les autres cas $(t \neq t_{sing})$, nous considérons les temps t_{in} et t_{out} tels que décrits précédemment. Pour toute valeur de t prise dans l'intervalle de temps $]t_{in},\ t_{sing}[\cup]t_{sing},\ t_{out}[$, la méthode consiste à construire (voir figure 2.11) les deux tangentes à $VO(t)$, parallèles à la droite passant par le point $c_A(t_0)$ et le centre du $\mathcal{CB}_\circ$ associé au temps t (qui s'écrit $c_B(t) - (t - t_0) \cdot \overrightarrow{v_B}\ (t)$). Cette droite représente une approximation d'ordre 1 de l'axe du *NLVO* au temps t considéré. Les points de tangence entre ces droites et le $VO(t)$ sont les points virtuels.

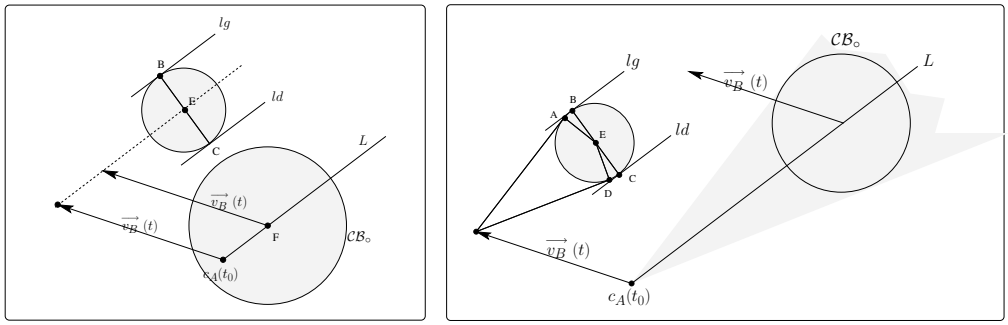

FIG. 2.11 – *Approximation des points de tangence* Pour $t \in]t_{in},\ t_{sing}[\cup]t_{sing},\ t_{out}[$, les points de tangence sont approximés grâce à une approximation d'ordre 1 de l'axe central du *NLVO* (la droite L). Ses parallèles (lg et ld) et tangentes à $VO(t)$ permettent d'obtenir les points B et C. Bien que cette approximation ne soit utile et utilisée que pour les valeurs de t telles que $c_A(t_0) \in \mathcal{CB}_\circ$ (à gauche), la figure de droite permet d'apprécier cette construction pour le cas nominal où A et D représentent les vrais points de tangence.

Justification Nous notons $\psi_{VO}(t)$ le vecteur attaché en $c_A(t_0)$ et dont l'extrémité est le point $c_{B_\circ}$ associé au temps t ($c_{B_\circ} = c_B(t) - (t - t_0) \cdot \overrightarrow{v_B}\ (t)$). La droite portée par $\psi_{VO}(t)$ est, par construction, une approximation d'ordre 1 de la trajectoire décrite par

[2] ϵ_t est choisi suffisamment petit pour que l'erreur due à cette interpolation soit négligeable.

l'axe $c_{VO}(t)$ du $\mathcal{V}$-*Obstacle*. Les points $vo_g(t)$ et $vo_d(t)$ étant situés sur $\delta VO(t)$, ils sont toujours compris entre les deux droites parallèles à $\psi_{VO}(t)$ et tangentes à $VO(t)$.

2.5 Caractérisation des temps à collision

À chacune des vitesses d'un VO correspond une trajectoire en collision et un temps à collision associé. Nous savons maintenant construire un $\mathcal{V}$-*Obstacle* à partir de l'expression analytique de ses bords, paramétrée par le temps à collision. Par conséquent nous connaissons le temps à collision associé à toute vitesse appartenant au bord d'un $\mathcal{V}$-*Obstacle*. Nous allons nous intéresser dans la suite du chapitre au temps avant collision des vitesses situées à « l'intérieur » du $\mathcal{V}$-*Obstacle*. Si nous traçons l'union des éléments temporels dans l'espace $\mathcal{V} \times \mathcal{T}$, dont la dimension de temps $\mathcal{T}$ correspond au temps à collision associé à chaque $VO(t)$ défini dans $\mathcal{V}$, alors nous observons que les temps à collision sont plus petits pour les vitesses prises à l'intérieur du $\mathcal{V}$-*Obstacle* par rapport à celles des « bords » (figure 2.12). Cela parait intuitif puisqu'en effet, si une vitesse permet au robot d'atteindre un point p situé à l'intérieur d'un obstacle au temps t_2, alors c'est que cette même vitesse a entraîné une collision entre le robot et les bords de l'obstacle à un temps antérieur à t_2 et que nous noterons t_1. Nous allons chercher à caractériser ici le rapport existant entre t_1 et t_2, connaissant la distance minimale qui existe entre cette vitesse et le bord du VO. Plus précisément, nous cherchons à calculer le temps à collision associé à une vitesse du VO connaissant la vitesse du bord la plus proche, le temps à collision associé à cette dernière, ainsi que la distance euclidienne séparant les deux vitesses dans $\mathcal{V}$.

2.5.1 Formulation du problème

Soit une vitesse $\vec{v}$ quelconque de VO. Nous appelons $\overrightarrow{v_{proj}}$ la vitesse du bord du VO la plus proche dans $\mathcal{V}$, et $d_{v/v_{proj}}$ la norme de $(\vec{v} - \overrightarrow{v_{proj}})$. $Tc(\vec{v})$ désigne le temps à collision associé à la vitesse $\vec{v}$ et $Tc(\overrightarrow{v_{proj}})$ celui associé à la vitesse $\overrightarrow{v_{proj}}$.

Le problème (illustré par la figure 2.13) consiste à calculer le rapport k tel que :

$$k = \frac{Tc(\vec{v})}{Tc(\overrightarrow{v_{proj}})} = f(d_{v/v_{proj}})$$

Nous avons montré que ce rapport, dont un exemple est illustré par la figure 2.14

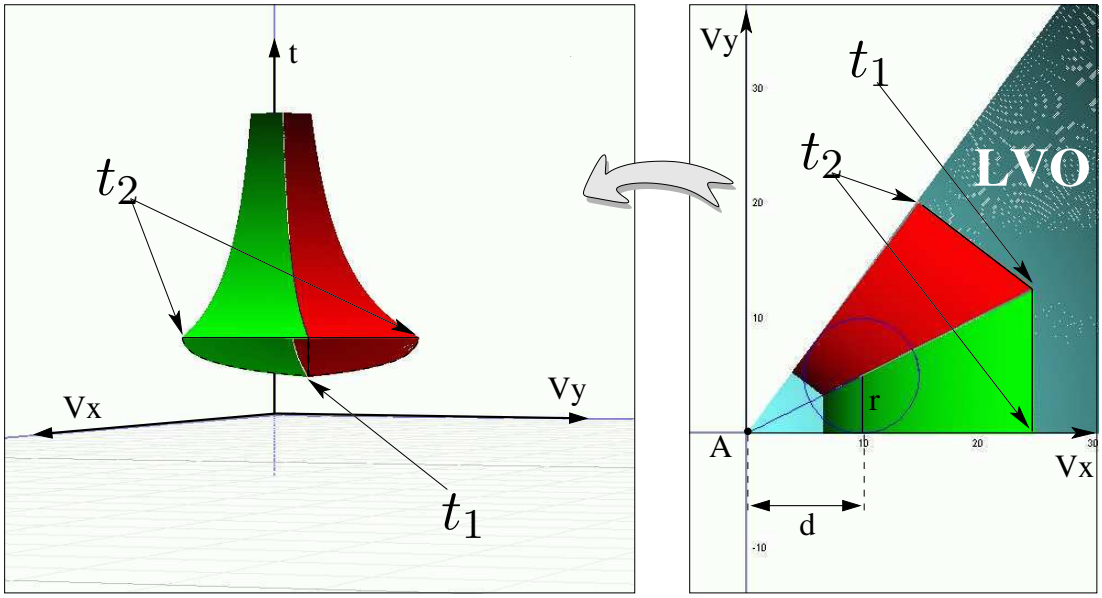

FIG. 2.12 – *Coupe d'un LVO* La figure de droite représente un LVO dans $\mathcal{V}$. Une portion est représentée à gauche dans l'espace $\mathcal{V} \times \mathcal{T}$. Nous voyons qu'une vitesse prise sur l'axe du LVO entraîne une collision au temps t_1, antérieur au temps t_2 de collision des bords. Pour information, $r = 5m$, $d = 10m$ et t est pris entre 0.5s et 1.5s.

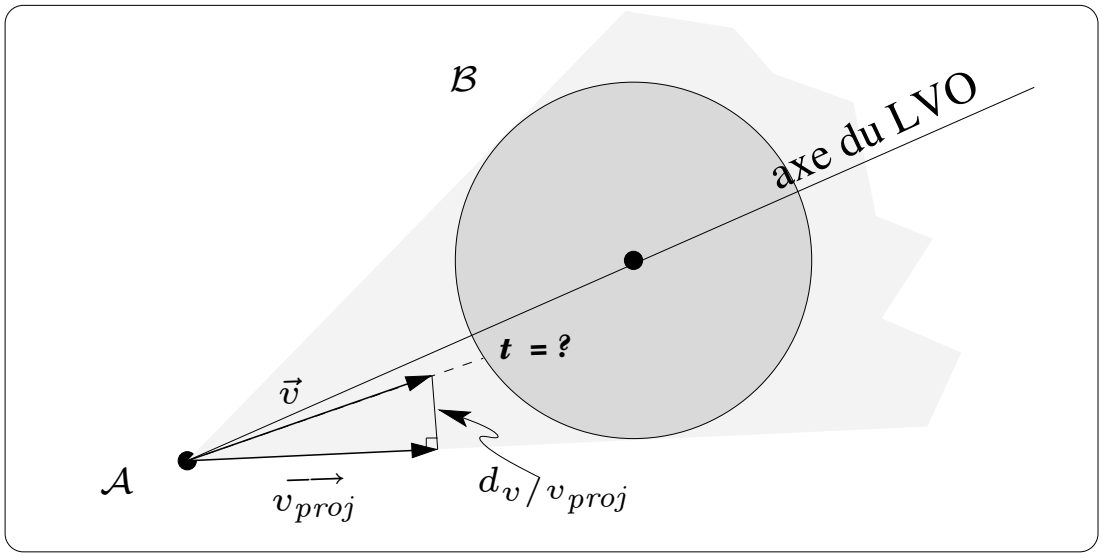

FIG. 2.13 – *Caractérisation des temps à collision* Nous cherchons à calculer le temps à collision de $\vec{v}$ connaissant sa distance minimale aux bords du $\mathcal{V}$-*Obstacle*. Nous disposons comme information supplémentaire, du temps à collision associé à la vitesse la plus proche du bord, notée $\overrightarrow{v_{proj}}$, ainsi que du rayon et de la vitesse de $\mathcal{B}$ à ce même temps.

vaut :

$$k = \sqrt{\frac{(x-d)^2 + (y+r)^2}{d^2 + {d_{v/v_{proj}}}^2}} \qquad \text{avec} \begin{cases} r_B = \text{rayon de } \mathcal{B} \\ \overrightarrow{v_B}\,(t) = \text{vitesse de } \mathcal{B} \text{ au temps } t \\ d_{v/v_{proj}} = \left\| \overrightarrow{v_{proj}} - \vec{v} \right\| \\ r = \dfrac{r_B}{Tc(\overrightarrow{v_{proj}})} \\ a = \dfrac{-d_{v/v_{proj}}}{d} \\ b = d_{v/v_{proj}} - r \\ c = 1 + a^2 \\ d = \left\| \overrightarrow{v_{proj}} - \overrightarrow{v_B}\,(Tc(\overrightarrow{v_{proj}})) \right\| \\ x = \frac{-a \cdot b + \sqrt{c \times r^2 - b^2}}{c} \\ y = ax + b \end{cases}$$

Remarque : La vitesse linéaire instantanée de l'obstacle au temps $Tc(\overrightarrow{v_{proj}})$ est nécessaire au calcul de k. Cette donnée étant nécessaire à la construction du LVO, elle est généralement connue. Cependant, dans le cas contraire, il est possible de la retrouver à partir d'une reconstruction du $LVO_\circ$ associé au temps $Tc(\overrightarrow{v_{proj}})$. Soit D la pointe de ce dernier, il s'agit du vecteur issu de $c_A(t_0)$ et dont l'extrémité est en D. Géométriquement, D est l'intersection de deux droites dans $\mathcal{V}$: La tangente au bord du VO et la tangente à son axe central, pour un temps à collision égal à $Tc(\overrightarrow{v_{proj}})$. La figure 2.15 permet de mieux comprendre cette construction.

2.5.2 Justification du calcul de k

La démonstration est de nature géométrique et un peu longue. Elle est donnée en annexe B afin de ne pas alourdir l'exposé. Le principe consiste tout d'abord à calculer une expression de k pour le cas d'un *$\mathcal{V}$-Obstacle* associé à un obstacle statique. Son expression étant dépendante uniquement de longueurs, elle n'est pas affectée par la translation qui permet de considérer des obstacles en mouvement rectiligne uniforme. Nous généralisons ensuite le cas à des obstacles ayant des trajectoires quelconques en nous appuyant sur l'hypothèse qu'un *$\mathcal{V}$-Obstacle Non-Linéaire* admet un *$\mathcal{V}$-Obstacle Linéaire* pour approximation d'ordre 1 pour chaque valeur de t. Par conséquent, l'expression de

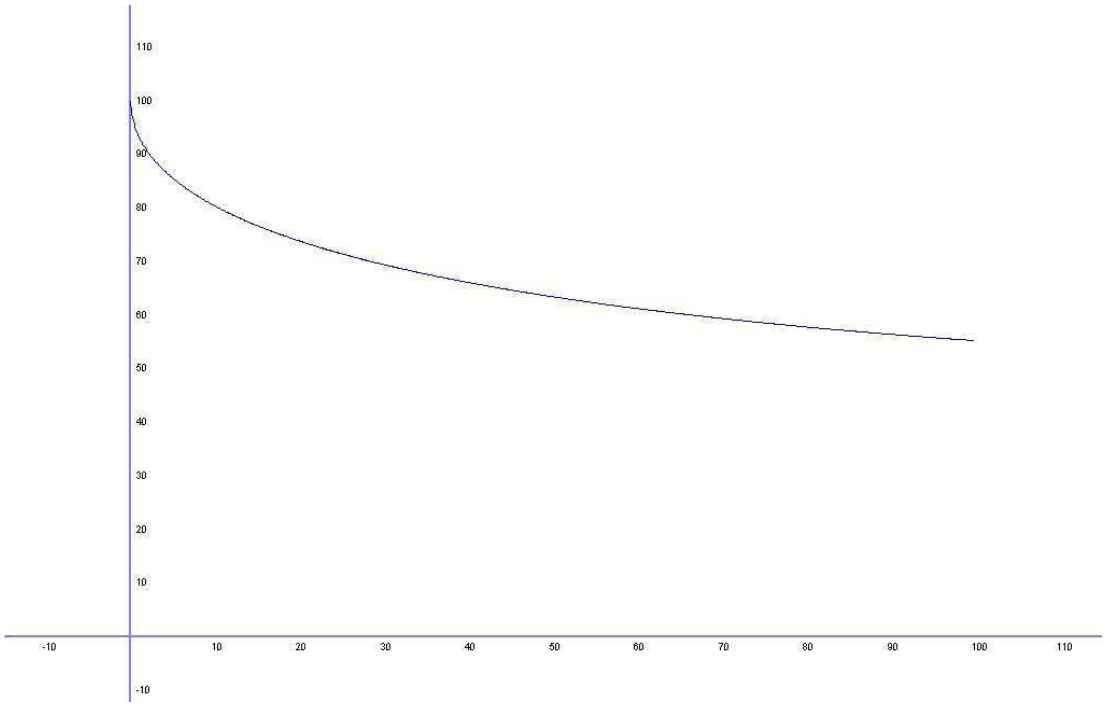

FIG. 2.14 – *Temps avant collision fonction de la distance au bord* La figure représente le temps avant collision $Tc(\vec{v}) = f(d_{v/v_{proj}})$, exprimé en pourcentage de la distance maximale au bord (0=sur le bord, 100=sur l'axe central) pour les valeurs suivantes : $r_B = 10m$, $\left\|\overrightarrow{v_{proj}}\right\| = 20m$ et $Tc(\overrightarrow{v_{proj}}) = 10s$ (Voir texte pour la signification de ces valeurs). Pour l'exemple choisi, nous voyons qu'une vitesse située sur l'axe central (100 en abscisse) entraînera une collision dans un peu moins de 60% du temps à collision de la vitesse du bord la plus proche dans $\mathcal{V}$.

k donnée pour le *LVO*, peut être vue comme une approximation de l'expression de k pour un élément temporel de *NLVO* donné.

2.5.3 Implémentation pratique coûteuse

L'expression précédente permet d'associer toute vitesse d'un $\mathcal{V}$-*Obstacle* à un temps à collision connaissant l'expression de ses bords dans $\mathcal{V} \times \mathcal{T}$. Son intérêt est essentiellement théorique, car d'un point de vue pratique, elle nécessite trop de calculs pour pouvoir être utilisée dans un contexte temps-réel. Une approche plus adaptée à une implémentation pratique sera présentée au chapitre 4. Elle consiste à calculer directement une approximation discrétisée du temps à collision associé aux vitesses situées à l'intérieur d'un $\mathcal{V}$-*Obstacle*.

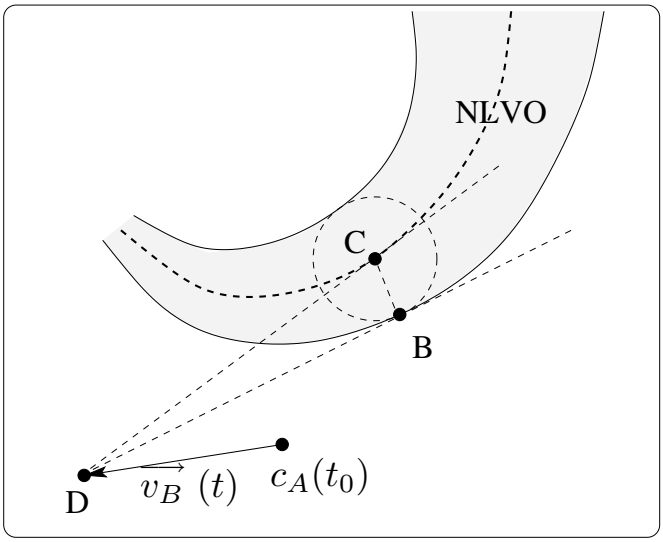

FIG. 2.15 – *Retrouver $\overrightarrow{v_B}\ (t)$ par construction* Pour connaître $\overrightarrow{v_B}\ (t)$, il suffit de trouver la vitesse du bord dont la collision aura lieu à l'instant t. Sur la figure il s'agit du point B. La normale au bord au point B coupe l'axe central du *NLVO* au point C. Il correspond au centre de $VO(t)$. La tangente au bord en B et la tangente à l'axe central en C, se coupent en un point D. Le vecteur attaché en $c_A(t_0)$ et dont l'extrémité est en D est $\overrightarrow{v_B}\ (t)$.

2.6 Conclusion partielle sur les $\mathcal{V}$-*Obstacles*

Dans ce chapitre, nous avons présenté le concept de $\mathcal{V}$-*Obstacle* comme l'ensemble des vitesses linéaires instantanées du robot qui entraînent une collision avec des obstacles statiques ou mobiles. Nous avons supposé le robot circulaire et en mouvement rectiligne constant, ainsi que des obstacles eux-aussi circulaires mais en mouvement quelconque connu. Ces hypothèses nous ont permis de donner une expression analytique simple d'un $\mathcal{V}$-*Obstacle* et du temps à collision associé à chacune de ses vitesses.

L'intérêt principal du concept de $\mathcal{V}$-*Obstacle* est la dimension de l'espace dans lequel il s'exprime, à savoir 2 ou 3. Cela nous permet de nous ramener à une approche purement géométrique, facilement implémentable pour le temps-réel. En contre-partie, cela impose de faire des hypothèses notamment sur la trajectoire du robot. Le chapitre 4 propose des solutions pour permettre l'utilisation des $\mathcal{V}$-*Obstacles* dans un cas plus général.

Avant cela, le chapitre suivant représente deux fonctions de navigation en environnement dynamique dérivées des $\mathcal{V}$-*Obstacles*.

Chapitre 3

Évitement d'Obstacles et Planification itérative avec les *NLVO*

Ce chapitre propose deux fonctions dérivées des *NLVO* pour la navigation de robots mobiles :

- Un module d'évitement réactif des obstacles prenant en compte des contraintes de tâche comme un but à atteindre (section 3.1)
- Un module de planification itérative de trajectoires sûres en environnement dynamique (section 3.2)

Toutes deux sont présentées ici pour des robots et obstacles circulaires. Ces méthodes pourront néanmoins être généralisées à des robots et obstacles de forme quelconque à l'aide des extensions proposées au chapitre 4.

3.1 Évitement réactif d'obstacles mobiles

3.1.1 Problématique et approche

Le problème de l'évitement réactif des obstacles consiste à calculer le prochain mouvement du robot à partir des informations locales de perception. Nous avons suivi une approche classique comprenant deux étapes successives :

1. ***Modéliser l'ensemble des déplacements admissibles libres :***
   Il s'agit de calculer l'ensemble des déplacements du robot immédiatement réalisables et sans collision, compte tenu des caractéristiques du robot et de la position des obstacles.

2. ***Choisir le déplacement libre qui satisfait au mieux les critères de tâche :***
   Il s'agit de définir une fonction numérique, qui permette d'associer un coût à chaque déplacement admissible libre en fonction des contraintes de sécurité et

de celles imposées par la tâche (e.g. atteindre une position le plus directement possible). Le déplacement de coût minimal est considéré comme étant le meilleur et est appliqué au robot.

Nous faisons le choix de ne considérer ici que des trajectoires du robot rectilignes constantes ou assimilées comme telles (i.e. de courbure très faible)[1] sur l'intervalle de temps considéré par les calculs. Cela nous permet d'utiliser les $\mathcal{V}$-*Obstacles* pour modéliser l'ensemble de ces trajectoires en collision et connaître les temps à collision qui leur sont associés. Soit dt la période du contrôleur d'exécution : Les données perçues en début d'itération à l'instant t, sont utilisées pour calculer les vitesses qui seront appliquées au début de l'itération suivante, à l'instant $t + dt$. Dans ces conditions, nous prendrons $t_0 = dt$ et donc $TH > dt$ pour le calcul des VO.

La phase de modélisation des méthodes classiques a pour but d'effectuer un prétraitement sur les données de perception, afin de réduire le nombre de solutions potentielles à envisager. Cette tâche peut aisément être réalisée avec les $\mathcal{V}$-*Obstacles* : Les déplacements libres du robot s'obtiennent alors à partir de l'ensemble des vitesses admissibles, duquel ont été enlevées les vitesses qui entraînent une collision sur un intervalle de temps donné. Cette approche ne nécessite pas le calcul des temps à collision mais uniquement celui des « bords » des VO, ce qui a pour conséquence de ne demander que des calculs simples et donc rapides. En contre-partie, comme pour toutes les approches de ce type, réduire l'ensemble des solutions potentielles peut entraîner une perte totale de solutions.

Nous avons préféré une approche différente qui consiste à ne pas chercher à écarter de solution pour réduire le nombre de choix possibles, mais au contraire à ajouter de l'information pour faciliter ces choix. En effet, l'intérêt principal des $\mathcal{V}$-*Obstacles*, outre la rapidité des traitements, est la possibilité d'obtenir cette information de plus haut niveau sur l'environnement (les temps à collision associés à chaque vitesse/trajectoire) sans écarter aucune solution potentielle.

Pour le calcul des $\mathcal{V}$-*Obstacles*, nous faisons l'hypothèse ici que la position du robot dans son environnement et les trajectoires des obstacles sur l'intervalle de temps $[t_0,\ TH]$ sont connues ou prédictibles. Nous verrons au chapitre 4 comment les incertitudes peuvent être modélisées dans le formalisme des $\mathcal{V}$-*Obstacles*, notamment en faisant varier le rayon du robot ou celui des obstacles.

3.1.2 Modélisation des déplacements admissibles libres

Nous allons procéder en quatre étapes :

1. ***Calculer l'ensemble*** V_{adm} ***des vitesses admissibles***, compte-tenu des caractéristiques mécaniques du robot.

[1] Les erreurs d'approximation peuvent facilement être compensées par un grossissement du robot (voir page 104). Néanmoins, en cas de courbure importante ou d'environnement très contraint, la méthode de planification itérative présentée dans la suite du chapitre sera mieux adaptée.

2. ***Choisir*** *TH* pour le calcul de VO_{Ω}.
3. ***Calculer*** VO_{Ω}.
4. ***Calculer l'ensemble*** V_{poss} ***des vitesses admissibles libres.***

Nous allons détailler chacune de ces étapes, à l'exception du calcul de VO_{Ω} qui ne présente aucune difficulté notoire.

3.1.2.1 Calcul des vitesses admissibles V_{adm}

Toute vitesse n'est pas applicable immédiatement au robot en raison de ses contraintes cinématiques et dynamiques. Il est donc nécessaire de définir, dans un premier temps, l'ensemble V_{adm} ($V_{adm} \subset \mathcal{V}$) des vitesses admissibles du robot compte-tenu de ces contraintes. Nous avons considéré deux classes cinématiques de robots et identifié les paramètres principaux s'y rattachant :

- ***les robots holonomes***, dont les contraintes sont la vitesse maximale autorisée notée $vmax_A$ et l'accélération maximale notée $amax_A$. Nous supposons ces valeurs applicables à toutes les directions (la vitesse tangentielle maximale et la vitesse normale maximale du robot sont égales, tout comme l'accélération tangentielle maximale est égale à l'accélération normale maximale).
- ***les robots non-holonomes***, dont les contraintes sont le rayon minimal de courbure des trajectoires $rmin_A$, la vitesse tangentielle maximale $vmax_A$, la vitesse angulaire maximale ωmax_A, l'accélération tangentielle maximale $amax_A$ et l'accélération angulaire maximale $\dot{\omega} max_A$.

Pour chacun de ces robots, nous donnons ici la forme générale de V_{adm}. Pour une implémentation pratique, nous aurons recours à des approximations polygonales de ces dernières, dont un exemple est proposé au chapitre 4. En plus des contraintes citées ci-avant, nous supposerons disponibles les informations suivantes : la période dt du contrôleur d'exécution, et la vitesse linéaire instantanée courante du robot notée $\overrightarrow{v_{A_\circ}}$. Nous avons choisi l'ensemble de ces paramètres car ils suffisent à modéliser la plupart des robots.

V_{adm} pour un robot holonome L'ensemble des vitesses $\overrightarrow{v_A}$ du robot, applicables en un laps de temps $\epsilon_t < dt$, est tel que :

$$\forall \overrightarrow{v_A} \in \mathcal{V}, \qquad \overrightarrow{v_A} \in V_{adm} \Longleftrightarrow \left\{ \begin{array}{lcll} \left\| \overrightarrow{v_A} \right\| & \leq & vmax_A & (1) \\ \left\| (\overrightarrow{v_A} - \overrightarrow{v_{A_\circ}}) \right\| & \leq & (\epsilon_t \cdot amax_A) & (2) \end{array} \right.$$

La relation (1) définit un disque dans $\mathcal{V}$ centré sur le robot en $c_A(t_0)$, et de rayon $vmax_A$. Nous le noterons $\mathfrak{C}_v$. La relation (2) définit quant à elle un disque de centre

$(c_A(t_0) + \overrightarrow{v_{A_\circ}})$ et de rayon $(\epsilon_t \cdot amax_A)$. Nous le noterons $\mathfrak{C}_a$. V_{adm} s'exprime donc simplement comme l'intersection de ces deux disques (figure 3.1).

$$V_{adm} = \mathfrak{C}_v \cap \mathfrak{C}_a \qquad avec \begin{cases} \mathfrak{C}_v & \left| \begin{array}{l} c_A(t_0) \\ vmax_A \end{array} \right. \\ \mathfrak{C}_a & \left| \begin{array}{l} c_A(t_0) + \overrightarrow{v_{A_\circ}} \\ \epsilon_t \cdot amax_A \end{array} \right. \end{cases}$$

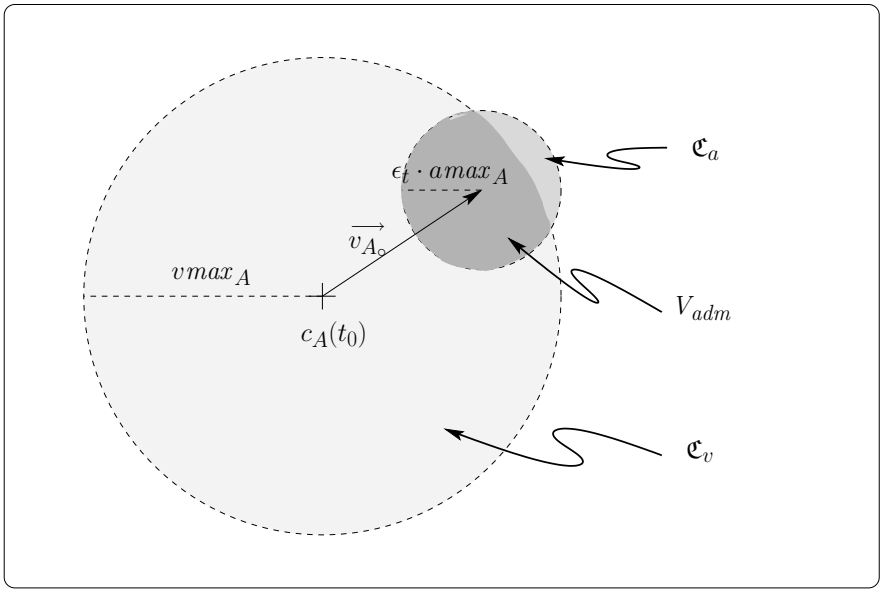

FIG. 3.1 – *Ensemble V_{adm} pour un robot holonome* Il s'agit de l'intersection du disque $\mathfrak{C}_v$ représentant les vitesses qui vérifient la contrainte sur la vitesse maximale $vmax_A$, avec le disque $\mathfrak{C}_a$ représentant les vitesses qui vérifient la contrainte sur l'accélération maximale $amax_A$.

V_{adm} pour un robot non-holonome Pour un robot non-holonome, les trajectoires sont à courbure continue et les seules vitesses linéaires instantanées admissibles sont tangentes à la trajectoire du robot en absence de glissement. Nous posons comme hypothèse que pour un petit déplacement, la courbe décrite par le robot peut être approximée par un segment de droite. Dans ces conditions, une vitesse linéaire admissible sera une vitesse permettant de réaliser ce petit déplacement rectiligne.

La représentation dans $\mathcal{V}$ des vitesses qui satisfont toutes ces contraintes n'est pas aussi directe que pour le cas d'un robot holonome. La forme obtenue peut cependant être approximée par une construction simple (figure 3.2).

La déviation latérale maximale autorisée par le robot est définie par les contraintes ωmax_A et $\dot{\omega} max_A$. Elles permettent de calculer l'angle $\delta\theta max_A$ maximal que peut faire

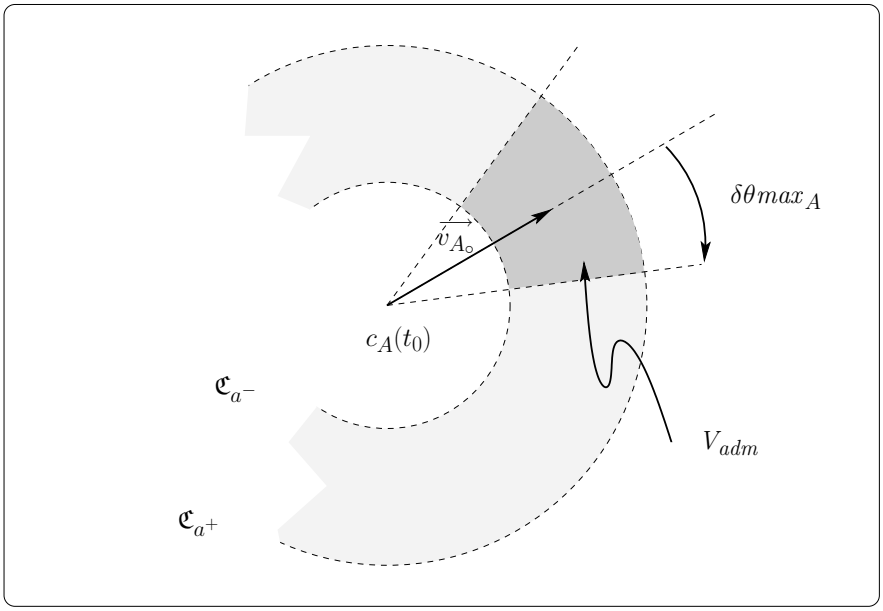

FIG. 3.2 – *Ensemble V_{adm} pour un robot non-holonome* Il s'agit de l'intersection de la portion de couronne définie par les cercles $\mathfrak{C}_{a^-}$ et $\mathfrak{C}_{a^+}$, avec le cône délimité par les droites sécantes en $c_A(t_0)$ formant un angle $\delta\theta max_A$ de part-et-d'autre de $\overrightarrow{v_{A_\circ}}$.

le prochain vecteur vitesse $\overrightarrow{v_A}$ avec le vecteur vitesse courant $\overrightarrow{v_{A_\circ}}$.

$$\delta\theta max_A = \begin{cases} \epsilon_t \cdot (\omega_{A_\circ} + \epsilon_t \cdot \dot{\omega} max_A) \ si \ \|(\omega_{A_\circ} + \epsilon_t \cdot \dot{\omega} max_A)\| \leq \omega max_A \\ \epsilon_t \cdot \omega max_A \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad sinon \end{cases}$$

Cet angle définit à son tour une géométrie dans $\mathcal{V}$ à laquelle les vitesses de V_{adm} peuvent appartenir. Nous appellerons $Vlat_{adm}$ cet ensemble. Il est composé de deux cônes alignés $Vlat_{adm^-}$ et $Vlat_{adm^+}$ contenant respectivement des vitesses de marche arrière et de marche avant du robot.

Les contraintes $vmax_A$ et $amax_A$ permettent de fixer la borne supérieure de la norme des vecteurs vitesse. Cette limite est matérialisée dans $\mathcal{V}$ par le disque $\mathfrak{C}_{a^+}$ de centre $c_A(t_0)$ et de rayon r_{a^+} avec

$$r_{a^+} = \left| \begin{array}{ll} (\overrightarrow{v_{A_\circ}} + \epsilon_t \cdot amax_A) & si \ (\overrightarrow{v_{A_\circ}} + \epsilon_t \cdot amax_A) \leq vmax_A \\ vmax_A & sinon \end{array} \right.$$

Nous définissons alors l'ensemble $\mathcal{V}_{poss^+} = Vlat_{adm^+} \cap \mathfrak{C}_{a^+}$.

De même, $vmin_A$ et $amin_A$ permettent de fixer la borne inférieure, matérialisée par le disque $\mathfrak{C}_{a^-}$, de centre $c_A(t_0)$ et de rayon $|r_{a^-}|$ ($r_{a^-} < 0$ signifie que V_{adm} comporte des vitesses correspondant à des manœuvres du robot en marche arrière) :

$$r_{a^-} = \left| \begin{array}{ll} \overrightarrow{v_{A_\circ}} + \epsilon_t \cdot amin_A & si \ \overrightarrow{v_{A_\circ}} + \epsilon_t \cdot amin_A \geq |vmin_A| \\ vmin_A & sinon \end{array} \right.$$

Nous définissons l'ensemble $\mathcal{V}_{poss^-} = Vlat_{adm} \cap \mathfrak{C}_{a^-}$.
Nous distinguons alors deux cas :
- les vitesses de V_{adm} permettent au robot de reculer ($r_{a^-} < 0$)
- les vitesses de V_{adm} ne permettent pas au robot de reculer ($r_{a^-} \geq 0$)

Selon le cas, l'approximation de V_{adm} sera :

$$V_{adm} \approx \begin{cases} Vlat_{adm^+} \cap \overline{Vlat_{adm^-}} & si\ (r_{a^-} < 0) \\ Vlat_{adm^+} \cup Vlat_{adm^-} & sinon \end{cases}$$

Remarque $rmin_A$ permet de représenter par exemple une contrainte sur l'angle maximale de braquage des roues pour un véhicule de type voiture. Il suppose $(\overrightarrow{v_{A_\circ}}/\omega_{A_\circ}) \geq rmin_A$ vrai à tout instant. Sa conséquence sur la définition de V_{adm} telle que nous venons de le voir sera la limitation de $\delta\theta max_A$ en fonction de $\overrightarrow{v_{A_\circ}}$, ou la limitation de r_{a^+} en fonction de $\omega_{A_\circ}$. Cela nous permet de disposer d'une approximation de V_{adm} simple mais suffisante pour contrôler le robot. Il n'est pas utile ici d'obtenir une représentation plus précise, du fait qu'il subsistera toujours des erreurs dues à l'hypothèse (nécessairement fausse) sur les déplacements linéaires du robot.

3.1.2.2 Choix de TH pour le calcul des $\mathcal{V}$-*Obstacles*

Conséquence de TH sur les performances des $\mathcal{V}$-*Obstacles* Pour rappel, TH représente le temps après lequel les collisions ne sont plus prises en compte. Le choix d'un TH lointain permet une meilleure anticipation des collisions. En revanche, cela implique davantage de calculs donc une réactivité plus faible du robot. De plus, nous pouvons nous poser la question de l'utilité d'anticiper les collisions sur un temps très long, dans un environnement dynamique et incertain où la connaissance sur les obstacles peut être bouleversée à chaque instant. Prendre un TH trop court n'est cependant pas une solution puisque cela augmente la sensibilité de la méthode aux minima locaux. Le choix d'un « bon » TH est donc une tâche difficile car basée sur des critères subjectifs dépendants du contexte. Nous avons fait un choix afin de proposer une valeur de TH qui puisse convenir à la plupart des applications.

Critères de choix du TH Nous avons identifié trois critères qui peuvent influer directement sur le choix de la valeur de TH :

- La limite des connaissances sur les trajectoires des obstacles mobiles
- La limite de la puissance de calcul ou du temps disponible pour le calcul des $\mathcal{V}$-*Obstacles*.
- Le temps nécessaire pour effectuer une manœuvre d'urgence qui puisse accroître la sécurité du robot.

Les deux premiers points sont des réflexions de bon sens, qui nous disent respectivement que si nous ne possédons des informations certaines[2] sur l'environnement que

[2]La certitude sur les informations dont il est question ici fait référence à un niveau de confiance

pour les n_1 secondes à venir, il n'est pas utile de planifier au delà et de même que si la puissance de calcul disponible ne nous permet, dans le temps imparti, que de planifier pour les n_2 prochaines secondes, il n'est pas utile (et même ici pas possible) de planifier au delà. Ces deux observations permettent donc chacune de définir des majorants de TH.

$$TH \leq t_0 + n_1 \qquad et \qquad TH \leq t_0 + n_2$$

Au moment où nous rédigeons ce document, la carte de l'environnement n'est pas fonctionnelle et il ne nous est pas possible de connaître les valeurs expérimentales de n_1. Pour les tests en simulation, nous avons pris les cas $n_1 = 0$, $n_1 = 2$ secondes et $n_1 \longrightarrow \infty$.

Le troisième point vise à estimer un minorant de TH. Pour cela, nous considérons une « manœuvre d'urgence » adaptée à un contexte particulier. Il peut s'agir par exemple d'un freinage brusque jusqu'à arrêt complet du robot ou d'un changement de file suivi d'un arrêt. L'idée consiste à calculer le temps t_{safe} nécessaire au robot pour effectuer cette manœuvre si besoin était et d'utiliser ce temps comme minorant de la valeur de TH. En mode nominal, le robot recherche une trajectoire libre à l'aide des $\mathcal{V}$-*Obstacles* et l'applique. En l'absence de solution, le robot entame une procédure d'urgence.

En pratique, la connaissance sur l'environnement est très incertaine et il peut arriver que $n_1 \leq t_{safe}$. Dans ce cas, nous ne tenons pas compte de t_{safe} car toute anticipation sans un minimum d'information sur l'environnement n'a pas de sens.

Exemple de manœuvre d'urgence et temps associé La manœuvre d'urgence que nous avons considérée est un freinage maximal permettant au robot d'atteindre une vitesse nulle le plus rapidement possible, tout en conservant une trajectoire rectiligne. Ceci est suffisant pour garantir la sécurité du robot en environnement statique mais pas en environnement dynamique. Néanmoins, si les obstacles ne sont pas agressifs, les chances d'éviter les collisions sont plus importantes. Ce type de manœuvre d'urgence pourrait être utilisé par exemple pour un robot de service.

Soit $t_{stop}(\overrightarrow{v_A},\ dec_{A_{max}})$ le temps permettant au robot de s'arrêter, compte-tenu de sa vitesse instantanée $\overrightarrow{v_A}$ à l'instant t_0, et de sa décélération maximale $dec_{A_{max}}$ possible :

$$t_{safe} = t_{stop}(\overrightarrow{v_A},\ dec_{A_{max}}) = -\frac{\left\|\overrightarrow{v_A}\right\|}{dec_{A_{max}}} \qquad avec\ dec_{A_{max}} < 0$$

Dans ces conditions, la procédure de freinage d'urgence est déclenchée par l'évènement $V_{free} = \emptyset$ pour $TH \geq t_0 + t_{safe}$, avec $t_{safe} = t_{stop}(\overrightarrow{v_A},\ dec_{A_{max}})$.

Deux approches ont été envisagées :

- ***une approche optimiste :*** nous supposons qu'une manœuvre d'urgence sûre existe toujours, quelque soit la vitesse choisie parmi celles de V_{adm}.

– ***une approche prudente :*** seules les vitesses qui permettent au robot de s'arrêter en urgence sans entrer en collision avec un obstacle (avant l'arrêt complet) sont conservées. Nous les appellerons les *Vitesses d'Urgence* ($V_{urgence}$).

L'approche prudente implique de pouvoir identifier les *Vitesses d'Urgence*. Une approche discrète comparable aux fenêtres dynamiques est possible. Elle consiste à discrétiser V_{adm} et vérifier la validité de chaque discrétisation, par un calcul. Cela nous ferait perdre l'intérêt de l'approche géométrique des $\mathcal{V}$-*Obstacles*. Nous avons préféré, dans le cadre de cette étude, utiliser une approximation conservative simple de $V_{urgence}$ basée sur les $\mathcal{V}$-*Obstacles* : toute vitesse $\overrightarrow{v_A}$ permet un arrêt sans collision du robot si le segment qui relie $c_A(t_0)$ à l'extrémité de $\overrightarrow{v_A}$ (t_0) est inclus entièrement dans V_{free_Ω} pour $TH = t_0 + t_{safe}$ (figure 3.3). En effet, la vitesse du robot sur l'intervalle de temps $[t_0,\ TH]$ pour $TH \geq t_0 + t_{safe}$ décroît régulièrement de $\overrightarrow{v_A}$ (t_0) à $\vec{0}$.

Cette approche écarte de nombreuses solutions potentielles (du fait que la condition précédente est suffisante mais pas nécessaire) et l'ensemble des vitesses solutions est souvent réduit à un nombre limité de vitesses proches du vecteur nul, ce qui explique les résultats donnés ci-après.

Les deux approches ont été testées sur deux types d'environnements dynamiques : l'un où les obstacles mobiles essaient d'éviter les collisions avec le robot (obstacles non-agressifs), l'autre où les obstacles ont des trajectoires aléatoires ne tenant pas compte des collisions (obstacles agressifs). Le robot et les obstacles sont holonomes, circulaires, leur vitesse linéaire et leur accélération sont bornées (respectivement 10m/s et $5\text{m}/s^2$). L'espace de travail est un rectangle de 60m x 40m, dans lequel le but du robot et celui de chaque obstacle est généré aléatoirement puis déplacé dès qu'il est atteint. Les chiffres du tableau suivant représentent le nombre total de collisions du robot sur une période de 30 minutes d'utilisation, suivi du nombre de collisions du robot qui, parmi les précédentes, ont eu lieu à vitesse nulle (moins de 0.1m/s). Les résultats médiocres obtenus avec l'approche prudente en présence d'obstacles agressifs s'expliquent par le fait que les vitesses autorisées pour le robot sont peu nombreuses et souvent proches de zéro : les bornes sur les accélérations ne permettent alors pas au robot de s'échapper en présence d'obstacles agressifs. Le deuxième chiffre permet de confirmer cette hypothèse et le fait qu'une meilleure approximation de $V_{urgence}$ est nécessaire. Les collisions à vitesse non-nulle sont dues quant à elles à des changements de trajectoires imprévus des obstacles proches du robot (notamment lorsque plusieurs obstacles entourent le robot qui se retrouve alors « coincé » par ses adversaires).

	Obstacles non-agressifs	**Obstacles agressifs**
Approche optimiste	2/0	12/3
Approche prudente	0/0	273/259

TAB. 3.1 – Collisions selon l'approche et l'environnement

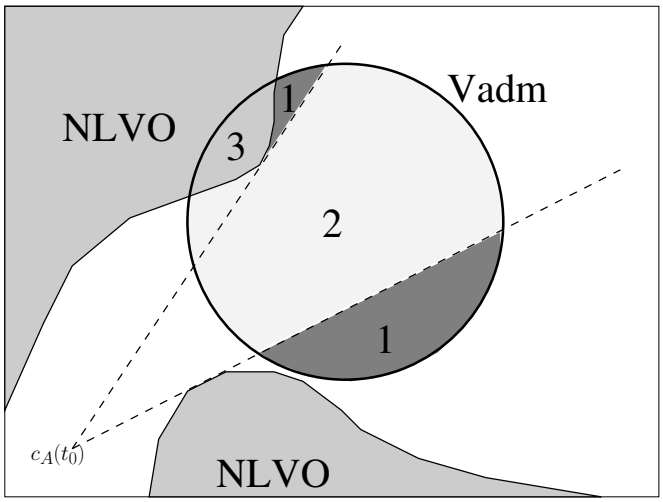

FIG. 3.3 – *Recherche de manœuvres d'urgence libres.* Si le segment qui relie une vitesse à $c_A(t_0)$ est libre, alors le robot est garanti de pouvoir s'arrêter sans entrer en collision pour toute vitesse prise sur ce segment. Ceci permet d'identifier une approximation très conservatrice des vitesses de V_{adm} qui permettent un arrêt sûr (2), celles qui entraînent des collisions (3), et celles sur lesquelles le formalisme choisi ne permet pas de conclure (1).

3.1.2.3 Calcul de V_{poss}

Nous avons fait le choix de n'écarter aucune solution potentielle et par conséquent, $V_{poss} = V_{adm}$. L'ajout d'information provient du calcul de VO_Ω, qui fournit le temps à collision associé à chaque vitesse de V_{adm} et donc de V_{poss}. Le temps à collision associé à une vitesse n'appartenant pas à VO_Ω est positionné à TH.

3.1.3 Choix d'un déplacement libre intégrant des contraintes de tâche

Les approches classiques ajustent le comportement du robot à l'aide d'une fonction de coût global favorisant à des degrés différents, l'alignement du robot avec son but, les vitesses élevées, les vitesses n'entraînant pas de collision. Dans notre cas, le coût fait intervenir deux notions :

- la notion de risque de collision associé à une trajectoire
- la notion de distance au but

3.1.3.1 Notion de Risque

Il est difficile de définir la notion de risque associé à une vitesse sans considérer une application ou un environnement particulier. D'une façon générale, nous pouvons cependant émettre les propositions suivantes (remarques de « bon sens ») :

- Des vitesses impliquant des collisions à court terme sont plus dangereuses que celles impliquant des collisions à long terme.

- Les vitesses entraînant des collisions tangentielles (prises sur le bord des $\mathcal{V}$-*Obstacles*) sont moins dangereuses que celles percutant les obstacles de front (prises à « l'intérieur » des $\mathcal{V}$-*Obstacles*). En effet, si de telles vitesses doivent être prises par le robot, elles ont plus de chance d'être transformées en vitesses libres au cours de la manœuvre avant que l'impact n'ait lieu. Cette propriété peut cependant être rapprochée de la précédente du fait que le temps à collision associé à une vitesse sera d'autant plus petit que cette vitesse est prise près de l'axe du $\mathcal{V}$-*Obstacle*.
- Des vitesses libres sont d'autant plus risquées qu'elles sont proches de vitesses en collision. En effet, de faibles changements sur l'environnement peuvent les transformer en vitesses en collision.

« Traduites » dans le formalisme des $\mathcal{V}$-*Obstacles*, ces observations permettent de définir deux fonctions de risque normalisées :

- $Cost_{tc}(\overrightarrow{v_A})$ dont le but est de favoriser les vitesses entraînant des collisions dans un temps lointain voire infini
- $Cost_{dist}(\overrightarrow{v_A})$ dont le but est de favoriser les vitesses éloignées des $\mathcal{V}$-*Obstacles*.

Définition de $Cost_{tc}(\overrightarrow{v_A})$ Soit $Tc(\overrightarrow{v_A})$ le temps à collision associé à la vitesse $\overrightarrow{v_A}$ si $\overrightarrow{v_A} \in VO$. La fonction $Cost_{tc}(\overrightarrow{v_A})$ est une normalisation du temps à collision permettant de faire correspondre un coût de 1 à une vitesse entraînant une collision immédiate, et un coût de 0 à une vitesse entraînant une collision ayant lieu à l'instant TH ou à un temps ultérieur (voire jamais). Elle est définie par :

$$Cost_{tc}(\overrightarrow{v_A}) = \begin{cases} ((TH - Tc(\overrightarrow{v_A})) * t_0)/(Tc(\overrightarrow{v_A}) * (TH - t_0)) & si\ \overrightarrow{v_A} \in VO \\ 0 & sinon \end{cases}$$

Définition de $Cost_{dist}(\overrightarrow{v_A})$ Soit $d_{VO}(\overrightarrow{v_A})$ la distance euclidienne dans $\mathcal{V}$ entre la vitesse $\overrightarrow{v_A}$ et le « bord » du VO le plus proche. La fonction $Cost_{dist}(\overrightarrow{v_A})$ est une normalisation de cette distance permettant de faire correspondre un coût de 1 à une vitesse appartenant à un $\mathcal{V}$-*Obstacle*, et un coût de 0 à la vitesse la plus éloignée des $\mathcal{V}$-*Obstacles*. Elle est définie par :

$$Cost_{dist}(\overrightarrow{v_A}) = \begin{cases} 1 - \frac{d_{VO}(\overrightarrow{v_A})}{dmax_{VO}} & si\ \overrightarrow{v_A} \notin VO \\ 1 & sinon \end{cases}$$

où $dmax_{VO}$ est le diamètre du plus petit cercle englobant l'ensemble des vitesses de V_{poss} dans $\mathcal{V}$.

3.1.3.2 Notion de distance au but

Nous avons considéré deux types de déplacements géométriques possibles du robot pour atteindre un but (figure 3.4) :

- tourner au maximum pour aligner sa trajectoire avec le but puis avancer vers le but en ligne droite (solution privilégiée)
- avancer tout droit jusqu'à ce que le but soit atteignable en tournant au maximum, puis tourner au maximum vers le but (solution choisie lorsque la première n'est pas applicable)

De plus, en raison de la dynamicité des environnements considérés ici et des contraintes d'accélération du robot, le mettant en danger lorsqu'il évolue à faible vitesse, nous avons favorisé les vitesses élevées le long de ces chemins.

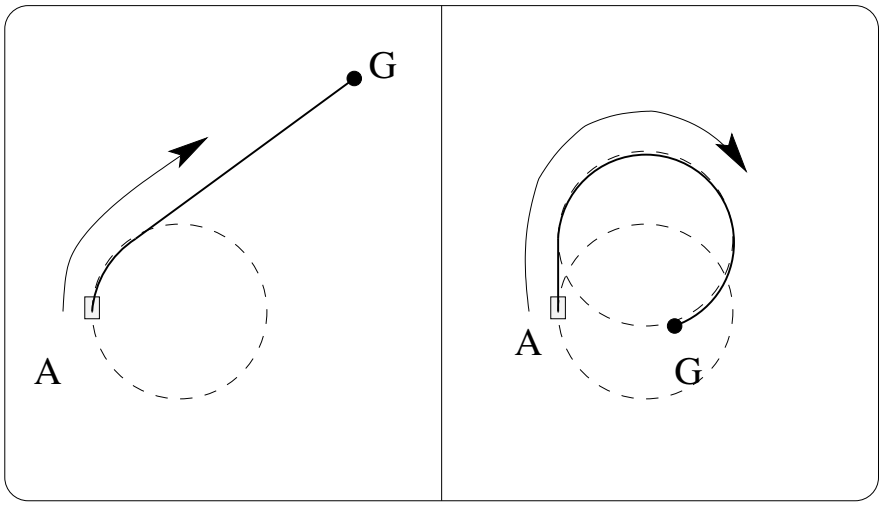

FIG. 3.4 – *Trajectoires considérées* La figure de gauche montre le type de trajectoire considérée en priorité : le robot tourne au maximum pour s'aligner avec son but puis se dirige vers ce dernier en ligne droite. Lorsque le but est situé à l'intérieur du cercle minimal décrit par le robot, le robot commence par avancer jusqu'à ce que le but sorte de ce cercle, puis le robot tourne pour atteindre le but (figure de droite).

Ces choix permettent au robot de se déplacer en ligne droite le plus souvent possible, afin de mieux tirer parti des $\mathcal{V}$-*Obstacles* : Par défaut, la première trajectoire est sélectionnée. Cependant, lorsque le but se situe à l'intérieur du cercle de rayon minimal $rmin_A$ décrit par le robot, il ne peut pas être atteint avec une telle trajectoire. Le robot avance donc jusqu'à ce que le but soit sur le cercle, auquel cas il peut l'atteindre directement en tournant au maximum. Cela permet ainsi de calculer simplement une estimation de la distance au but, notée $d_{but}(\overrightarrow{v_A})$, associée à la vitesse $\overrightarrow{v_A}$. Un profil de vitesse de type « accélération maximale-vitesse maximale constante-décélération maximale » permet d'estimer le temps de parcours $T_{but}(\overrightarrow{v_A})$ associé à cette trajectoire. Ces deux valeurs étant majorées par des bornes supérieures notées respectivement $dmax_{but}$ et $tmax_{but}$, il est possible de définir un coût reflétant l'optimalité, respectivement en distance ou en temps, de la trajectoire possible à partir de la vitesse $\overrightarrow{v_A}$ (0 représentant une trajectoire plus optimale que 1). Ce choix dépend de la contrainte de tâche à

satisfaire en priorité. Soit $Cost_{opt}(\overrightarrow{v_A})$ ce coût, nous aurons donc au choix :

$$Cost_{opt}(\overrightarrow{v_A}) = \begin{cases} 1 - \frac{d_{but}(\overrightarrow{v_A})}{dmax_{but}} & si\ d_{but}(\overrightarrow{v_A}) \leq dmax_{but} \\ 1 & sinon \end{cases}$$

ou bien

$$Cost_{opt}(\overrightarrow{v_A}) = \begin{cases} 1 - \frac{T_{but}(\overrightarrow{v_A})}{tmax_{but}} & si\ T_{but}(\overrightarrow{v_A}) \leq tmax_{but} \\ 1 & sinon \end{cases}$$

Remarque : utilisation d'une fonction NF1 Lorsque la composante statique de l'environnement est connue a priori, l'estimation de la distance au but est obtenue par le biais d'une fonction de navigation de type NF1 ([Latombe, 90]). Celle-ci consiste à propager une vague à partir du but (figure 3.5). Cette approche a récemment été suivie par d'autres méthodes locales, ainsi étendues pour prendre en compte des informations d'ordre plus global ([Minguez et al., 02][Brock et Khatib, 99]). Dans notre cas, la fonction $Cost_{opt}(\overrightarrow{v_A})$ est alors calculée sur le même principe que précédemment mais avec $d_{but}(\overrightarrow{v_A}) = NF1(\overrightarrow{v_A})$. Les contraintes cinématiques du robot ne sont alors plus prises en compte, mais l'information de distance est meilleure et les trajectoires obtenues sont plus directes vers le but notamment en environnement structuré (tel qu'un bâtiment).

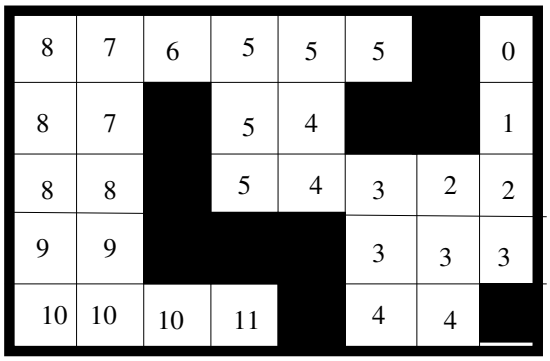

FIG. 3.5 – *Fonction NF1* La valeur nulle est attribuée au but (en haut à droite sur l'exemple). Puis les voisins sont incrémentés de 1 en 1, jusqu'à couverture complète des zones accessibles depuis le but.

3.1.3.3 Définition de la fonction de coût global

La fonction $Cost_{opt}(\overrightarrow{v_A})$ présente l'avantage de mieux prendre en compte les contraintes cinématiques du robot par rapport à une approche classique, basée uniquement sur la vitesse du robot et son alignement avec le but. Les fonctions $Cost_{tc}(\overrightarrow{v_A})$ et $Cost_{dist}(\overrightarrow{v_A})$ permettent quant à elles d'ajuster le comportement du robot et le rendre plus ou moins audacieux.

La fonction de coût globale notée $Cost_{global}(\overrightarrow{v_A})$ est définie par :

$$Cost_{global}(\overrightarrow{v_A}) = \alpha_1 \cdot Cost_{tc}(\overrightarrow{v_A}) + \alpha_2 \cdot Cost_{dist}(\overrightarrow{v_A}) + \alpha_3 \cdot Cost_{opt}(\overrightarrow{v_A})$$

où les α_i représentent des coefficients réels fixés arbitrairement.

En règle générale, nous avons privilégié la sécurité à l'optimalité des trajectoires et avons défini $\alpha_1 > (\alpha_2 + \alpha_3)$. Des exemples de comportements obtenus avec cette méthode sont présentés dans la suite du chapitre.

3.1.4 Résultats expérimentaux

Nous avons testé notre méthode d'évitement d'obstacles en simulation sur deux applications :

- une ***application routière***, visant à permettre à un véhicule de s'insérer sur un rond-point très encombré.
- une ***application de service***, visant à permettre à un robot de ramasser des objets parsemés au milieu d'une foule.

Pour ces exemples, nous avons choisi l'approche optimiste ($V_{poss} = V_{adm}$).

3.1.4.1 Insertion d'un véhicule sur un rond-point encombré

Dans cet exemple, nous allons considérer un rond-point à voie unique, encombré d'obstacles mobiles (matérialisant des véhicules) se suivant à faible distance. Le robot se trouve initialement sur une voie d'accès. Le but est d'adapter la vitesse du robot et sa trajectoire de manière réactive pour lui permettre de s'insérer dans la file de véhicules déjà sur le rond-point. Le robot et les obstacles sont approximés chacun par un disque englobant.

Le robot est non-holonome circulaire, de rayon 2 m, de vitesse maximale 20 m/s, d'accélération maximale[3] 10 m/s^2 et de rayon de giration minimal 3 m. Le rond-point et la voie d'accès sont délimités par des obstacles circulaires statiques juxtaposés les uns à la suite des autres. La largeur de la voie de circulation ne permet pas aux véhicules de se doubler. Les obstacles qui circulent sur le rond-point ont une vitesse constante élevée (15 m/s). Ils sont créés aléatoirement de telle manière que l'espacement avec l'obstacle précédent soit toujours compris entre 4 et 16 mètres (1 à 4 fois la place nécessaire pour le robot).

La figure 3.6 montre une série de captures d'écran de notre simulateur. Elle montre l'insertion du robot dans la file d'obstacles pour le paramétrage suivant du robot : $\alpha_1 = 1.0, \alpha_2 = 0.1, \alpha_3 = 0.5$, $TH = t_{safe} + dt + 1.5s$. Son but est situé sur le rond-point, à la position du « 3 » sur une horloge.

[3]Nous avons pris l'accélération maximale identique à la décéleration maximale du véhicule, d'où sa valeur élevée. Une valeur d'accéleration plus réaliste aurait nécessité une distance de perception plus importante et/ou des vitesses d'obstacles inférieures compte-tenu de la faible distance disponible pour l'insertion.

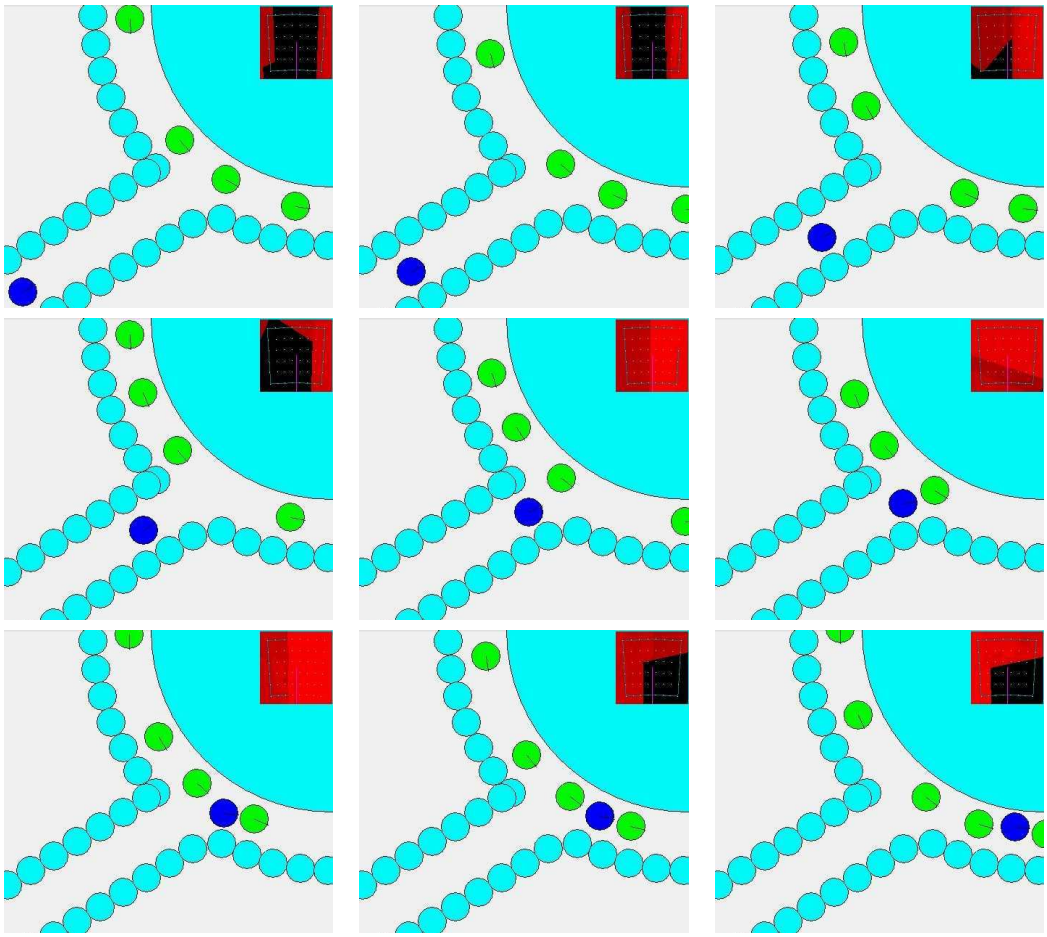

FIG. 3.6 – *Exemple d'insertion d'un véhicule sur un rond-point encombré*

Limitations Il est arrivé durant nos tests que le robot ne parvienne pas à s'insérer et s'arrête sur la voie d'accès juste avant le rond-point. Nous avons pu en identifier la cause : Les $\mathcal{V}$-*Obstacles* permettent de modéliser les trajectoires rectilignes libres du robot, or à son entrée sur le rond-point, le robot effectue un virage serré et les $\mathcal{V}$-*Obstacles* sont alors mal adaptés et ne permettent pas de « voir » toutes les possibilités. Ce nombre limité d'options voire l'absence de solution contraint parfois le robot à s'arrêter.

Concrètement, à son entrée sur le rond-point, le robot est face au « terre-plein » central. S'il continuait tout droit, il devrait s'arrêter au milieu de la voie du rond-point. Cependant la présence des obstacles sur le rond-point et l'anticipation de la collision avec ces derniers fait que le robot s'arrête sur la voie d'accès, juste avant le rond-point. Dans la plupart des cas, un « trou » suffisamment grand est détecté par les $\mathcal{V}$-*Obstacles* entre deux véhicules ce qui permet au robot de ne pas s'arrêter mais au contraire d'accélérer pour s'insérer dans ce trou. Plus le robot s'oriente dans le sens de la file et plus le nombre de solutions potentielles croît (il existe davantage de trajectoires rectilignes libres). Cependant, lorsqu'un obstacle « bloque » les trajectoires rectilignes du robot

qui initieraient le virage, le robot reste face au terre-plein central et est contraint de s'arrêter. Ses accélérations bornées l'empêchent alors de tenter toute insertion future compte-tenu de la vitesse importante des obstacles, et le robot reste en attente. La fréquence du phénomène observé est d'environ 1 essai sur 10.

Nous avons réduit de moitié le nombre de blocages, simplement en plaçant le but du robot à la position du « 6 » d'une horloge, l'incitant davantage à tourner s'il le peut, plutôt que de se diriger tout droit.

La méthode a ainsi permis de valider l'approche des $\mathcal{V}$-*Obstacles* pour le cas complexe d'un robot non-holonome. Dans le pire des cas (situations de blocage citées ci-dessus), la sécurité du robot n'a jamais été inquiétée.

3.1.4.2 Ramassage d'objets au milieu d'une foule

Le second exemple testé en simulation est une application de robot de service. Nous avons imaginé un robot devant récupérer des objets dans une zone occupée par des personnes. Initialement, le robot attend dans un local sécurisé dépourvu d'obstacles mobiles, que nous appellerons sa niche. Il la quitte pour aller chercher différents objets dans un ordre défini, puis retourne à sa niche. Les piétons sont supposés agressifs, c'est-à-dire que leurs trajectoires ne tiennent pas compte du robot. Robots et obstacles sont approximés par des disques englobants.

De même que précédemment, nous avons contraint les déplacements du robot et ceux des obstacles. Tous évoluent sur une aire rectangulaire de $25m \times 25m$. Le robot est circulaire, non-holonome, de rayon 0.5 m, de vitesse maximale 5 m/s, d'accélération maximale 5 m/s^2 et de rayon de giration minimal 2 m. Les objets à ramasser constituent les buts successifs du robot. Chaque but est généré aléatoirement. Le fait d'atteindre un but entraîne la création systématique d'un nouveau but. Après avoir ainsi « ramassé » n objets, le but du robot est positionné sur sa niche. Afin de tester les collisions du robot sur une période suffisante (> 30 min), nous n'avons pas borné la valeur de n durant nos tests et avons laissé le robot en permanence au milieu de la foule. Les trajectoires des obstacles sont générées d'une manière similaire : Chaque obstacle possède un but généré aléatoirement, qui est déplacé dès que l'obstacle l'atteint, et ce sans fin. L'obstacle tâche de s'orienter systématiquement vers son but et privilégie les vitesses élevées. Les obstacles sont holonomes, de rayon compris entre 0.5 m et 1 m, de vitesse maximale 4 m/s et d'accélération maximale 5 m/s^2.

Les figures 3.7 et 3.8 montrent le robot évoluant parmi les obstacles. Le paramétrage choisi est le suivant : $\alpha_1 = 1.0, \alpha_2 = 0.0, \alpha_3 = 0.3$, $TH = t_{safe} + dt + 1.5s$. Le choix de α_2 est motivé par le fait que le calcul de $Cost_{dist}(\vec{v})$ est coûteux et par conséquent nous l'avons supprimé pour gagner en réactivité. En contre-partie, afin d'éviter des trajectoires trop proches des obstacles malgré l'absence de la composante $Cost_{dist}(\vec{v})$, nous l'avons compensée dans les calculs par un grossissement du rayon du robot de 0.2 m.

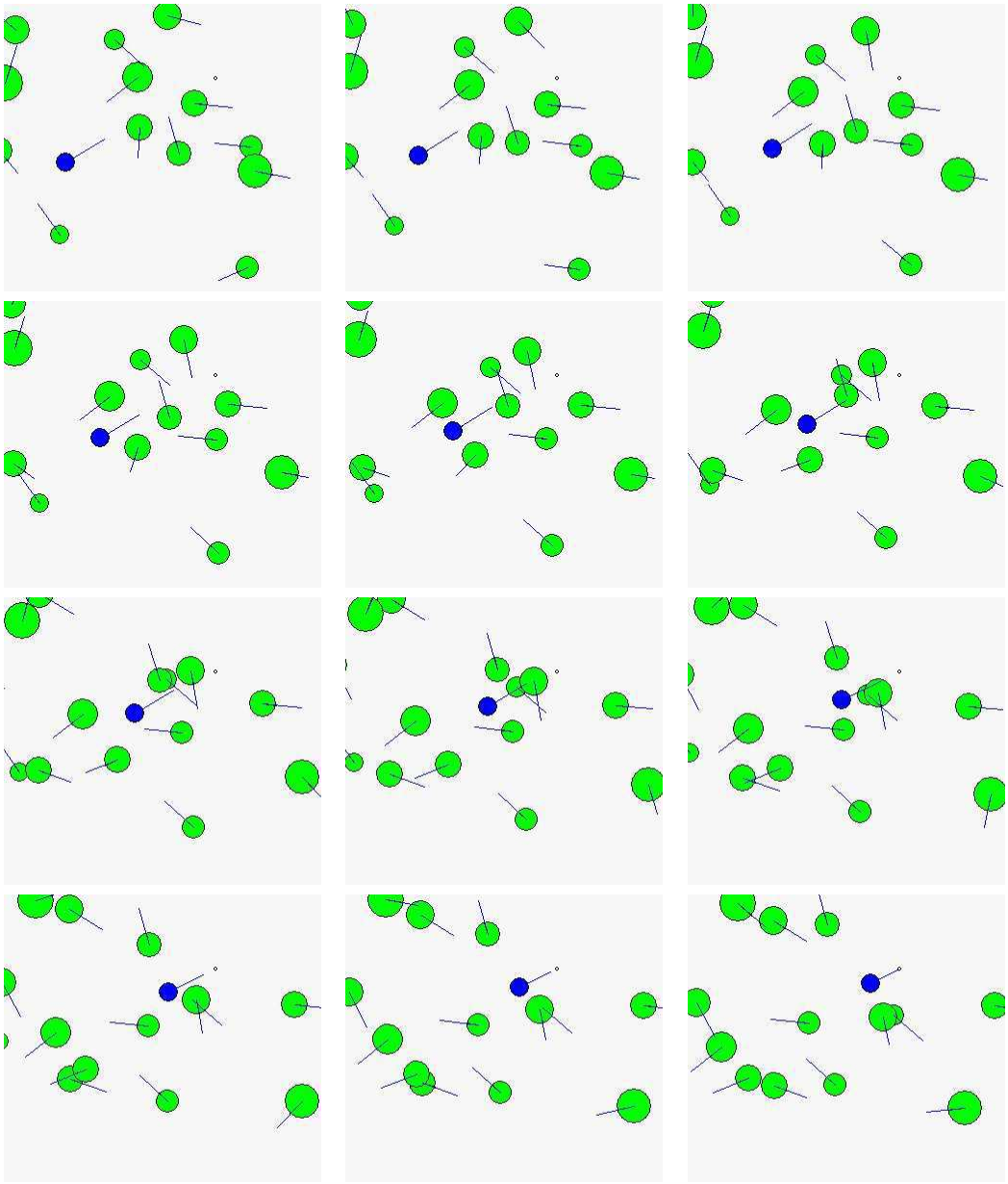

FIG. 3.7 – *Robot de service au milieu d'une foule dense.* Le robot se dirige vers un objet (matérialisé par un petit cercle noir) en évitant les obstacles. Nous pouvons remarquer que l'anticipation permet au robot de « voir » qu'une trajectoire quasiment droite est libre et par conséquent de la suivre, ce qui n'aurait pas été le cas avec une méthode ne prenant pas en compte les trajectoires des obstacles.

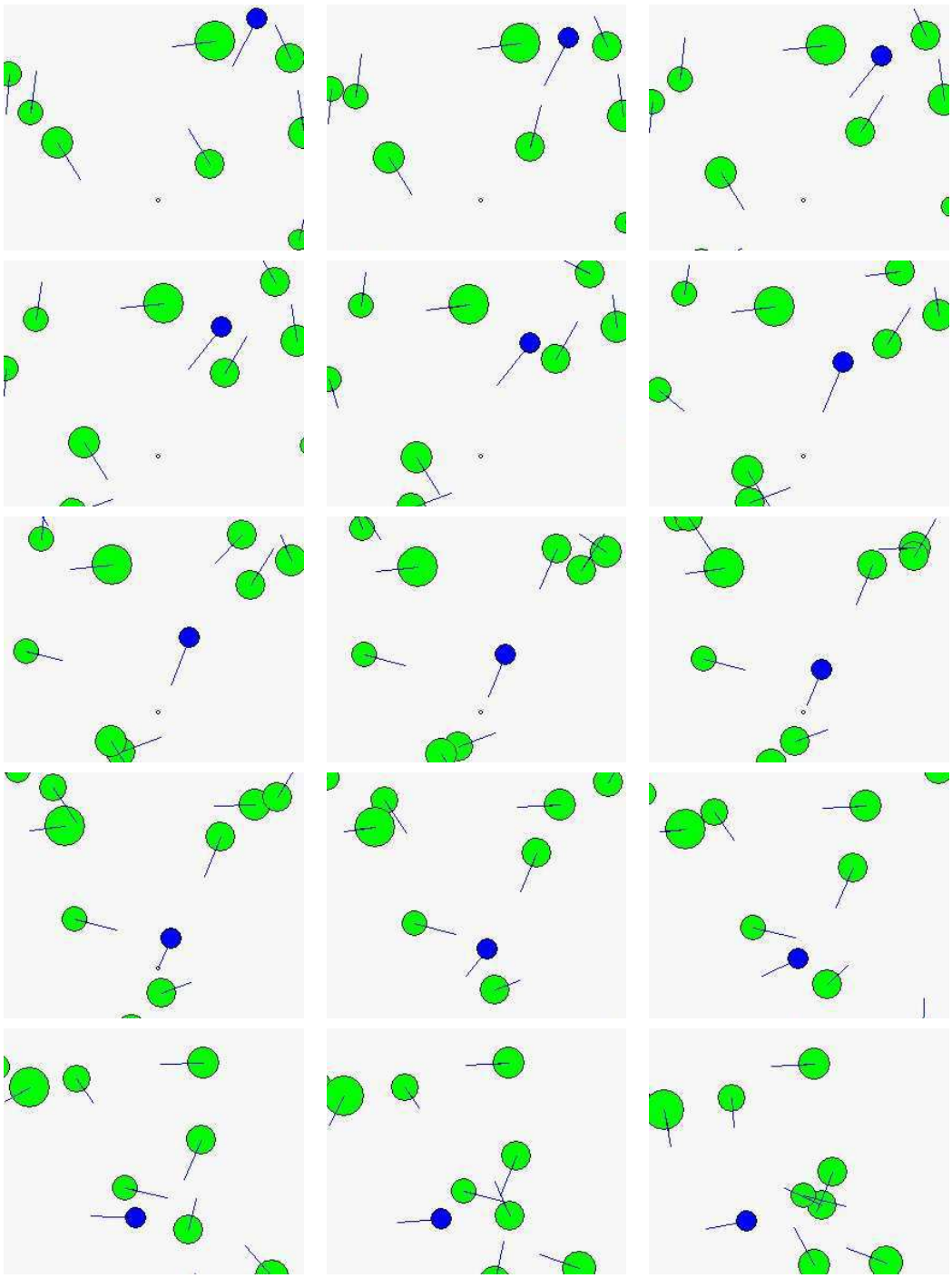

FIG. 3.8 – *Robot de service au milieu d'une foule dense.* Le robot va ramasser l'objet. À l'image 10, il le « prend » et va chercher le suivant.

Limitations Les tests en simulation ont montré que le robot est capable d'éviter la plupart des obstacles avec notre méthode réactive. Néanmoins, nous avons identifié deux situations où la collision ne peut pas être évitée :

- Lorsqu'un obstacle proche du robot change brutalement sa trajectoire. Les contraintes cinématiques et dynamiques du robot peuvent alors ne pas permettre au robot de réagir assez vite.
- Lorsque le robot est à l'arrêt ou quasiment (moins de 0.1 m/s). Toujours en raison des contraintes sur ses déplacements possibles, le robot n'a pas toujours le moyen de s'échapper si plusieurs obstacles se dirigent sur lui.

Dans les deux cas cités, nous pensons que la méthode d'évitement n'est pas la cause de la collision, mais que celle-ci est davantage due au comportement imprévisible des obstacles et aux limitations motrices du robot. Bien qu'il soit difficile de justifier que la collision était inévitable dans ces situations particulières, notre jugement a pu être conforté par deux remarques (voir tableau 3.1) :

- la vitesse du robot au moment de l'impact est nulle ou tend vers zéro : cela montre que le robot n'a d'autre solution que d'essayer de s'arrêter pour minimiser l'impact.
- une seconde expérience avec des obstacles non-agressifs (qui cherchent à éviter le robot) permet de supprimer les collisions.

3.2 Planification itérative de trajectoire en Environnement Dynamique

3.2.1 Motivation et approche proposée

La méthode d'évitement réactif vue précédemment est une méthode locale et par conséquent elle est sujette aux problèmes de minima locaux de sa fonction de coût, malgré l'anticipation des $\mathcal{V}$-*Obstacles* et le recours éventuel à une fonction de type NF1 pour diriger le robot. La solution à ce problème fait appel à des techniques de planification, basées sur l'utilisation d'un arbre de recherche.

Le principe de l'approche consiste à ne pas calculer uniquement le prochain déplacement à l'aide de la méthode locale vue précédemment, mais d'utiliser cette dernière comme opérateur d'expansion de l'arbre pour explorer plusieurs trajectoires possibles et en sélectionner la meilleure. L'exploration de l'arbre se fait de manière incrémentale, et peut être interrompue à tout instant, pour obtenir la meilleure solution parmi les trajectoires déjà explorées. L'espace de recherche étant virtuellement infini, nous avons défini une heuristique afin de converger plus rapidement vers une solution et limiter le nombre de nœuds explorés. L'algorithme utilisé pour cela est du type A^* ([Nilsson, 82]).

3.2.2 Structure de l'arbre de recherche

Nous appellerons Γ l'arbre de recherche constitué des nœuds n_i et des branches $b_{i,j}$. Un nœud n_i représente un état du robot à un instant donné. Il est associé à un coût, défini comme la somme du temps nécessaire au robot pour atteindre cet état et du temps estimé pour atteindre le but depuis cet état. Il existe une branche $b_{i,j}$ entre un nœud n_i et un nœud n_j, s'il existe un déplacement élémentaire[4] possible du robot entre les deux états correspondants. À chaque branche est associée la vitesse linéaire permettant de réaliser le déplacement. Nous notons dt le temps qui sépare deux états consécutifs.

3.2.3 Choix du nœud à explorer

L'ordre dans lequel les nœuds sont explorés a une influence directe sur leur nombre et donc sur le temps et la mémoire nécessaires au calcul d'une trajectoire sûre jusqu'au but.

L'algorithme A^* nécessite la définition d'un coût de la forme $f = g + h$ associé à chaque nœud. g représente le coût réel pour atteindre ce nœud depuis la configuration initiale du robot, tandis que h représente le coût estimé pour atteindre le but depuis le nœud.

Dans notre cas nous avons choisi d'utiliser le temps de parcours comme unité de coût. Soient $\vec{v}_i$ les vitesses portées par les branches successives qui relient le nœud racine de l'arbre à un nœud donné, alors le coût g associé à ce nœud vaut :

$$g = \sum_i \vec{v}_i \cdot dt$$

Le coût h est calculé selon la même méthode que $Cost_{opt}(\vec{v})$ vue précédemment : un chemin géométrique simple composé d'un arc de cercle et d'un segment de droite est calculé. Un profil de vitesse favorisant les vitesses élevées lui est appliqué pour estimer le temps de parcours. Il s'agit d'un profil du type « accélération maximale-vitesse maximale-décélération maximale »

Cette approche permet de favoriser les trajectoires qui atteignent le but le plus rapidement possible (parmi celles explorées).

3.2.4 Exploration d'un nœud

L'opérateur d'expansion de l'arbre permet de calculer les fils d'un nœud n_i, c'est-à-dire de calculer les états atteignables depuis un état donné, sans collision et en un temps dt.

Cette étape est comparable au calcul de V_{poss} décrit précédemment pour la méthode locale d'évitement des obstacles. L'ensemble V_{poss} est discrétisé pour permettre le calcul des coûts associés à chaque vitesse (discrétisation choisie : $32 \times 32 = 1024$ vitesses). Afin de réduire la largeur de l'arbre, seules 5 vitesses sont conservées (figure 3.9). Leur choix est réalisé selon l'algorithme 3.2.

[4] i.e. déplacement rectiligne constant réalisé durant une itération du contrôleur d'exécution

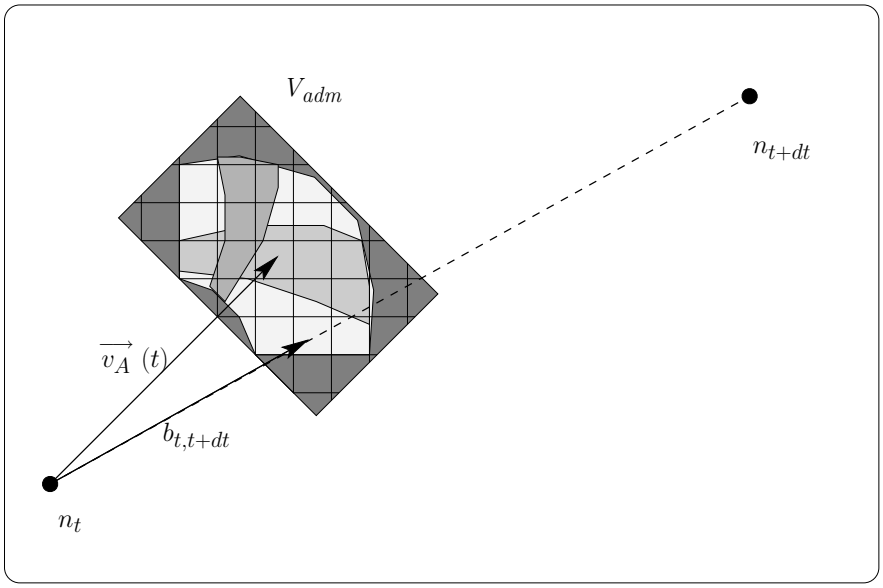

FIG. 3.9 – *Opérateur d'expansion de l'arbre* À chaque nœud correspond un état du robot. Pour explorer un nœud, on commence par calculer l'ensemble (discrétisé) des vitesses admissibles depuis l'état du robot associé au nœud. Parmi elles, 5 sont sélectionnées en fonction de leur temps à collision et de l'estimation du temps au but. Afin d'éviter de prendre des vitesses trop proches les unes des autres, un voisinage est défini autour de chaque vitesse sélectionnée. Les autres vitesses ne seront pas sélectionnées dans le voisinage des vitesses déjà choisies. n_t représente l'état courant et $\overrightarrow{v_A}\ (t)$ la vitesse linéaire instantannée du robot pour cet état. V_{adm} est l'ensemble des vitesses admissibles. $b_{t,t+dt}$ est une des vitesses sélectionnées. Elle permettra d'atteindre un état n_{t+dt}.

NOTATIONS
- Voisinage($\vec{v}$) : retourne les vitesses situées au voisinage de $\vec{v}$ (voir texte)
- $\overrightarrow{v_{opt}}$: vitesse $\vec{v} \in V_{poss}$ de coût $Cost_{opt}(\vec{v})$ minimal
- Tete(*liste*) : retourne le premier élément de *liste*
- listeFils : liste des nœuds fils

DEBUT
- Initialiser listeFils
- Initialiser $\vec{v}$ à $\overrightarrow{v_{opt}}$
- Trier les vitesses $\vec{v}_i$ de V_{poss} sur $Cost_{global}(\vec{v}_i)$ par ordre croissant
- POUR i DE 1 A 5 FAIRE
  - Ajouter $\vec{v}$ à listeFils
  - Enlever Voisinage($\vec{v}$) de V_{poss}
  - Affecter Tete(V_{poss}) à $\vec{v}$
- FIN POUR i
- RETOURNER listeFils

FIN

TAB. 3.2 – Algorithme de sélection des vitesses utilisées pour étendre l'arbre.

Le voisinage d'une vitesse $\vec{v}$, noté *Voisinage($\vec{v}$)*, représente l'ensemble des vitesses situées à une distance de $\vec{v}$ (dans $\mathcal{V}$), inférieure ou égale à une limite donnée. Il permet d'éviter la création de fils trop proches les uns des autres, en particulier lorsque la discrétisation de V_{poss} est trop fine, et permet ainsi une meilleure couverture de l'espace de recherche.

Les 5 vitesses sélectionnées donnent chacune naissance à une branche (étiquetée par cette vitesse) et à un nœud fils.

3.2.5 Mise-à-jour de l'arbre

Malgré l'utilisation d'une heuristique, l'exploration de l'arbre prend du temps et une solution globale n'est pas toujours trouvée dans le temps imparti (une itération). Dans ce cas, la meilleure trajectoire parmi celles explorées est trouvée, cependant rien ne garantit alors que le but sera atteint. Réinitialiser l'arbre à l'itération suivante pour une nouvelle recherche est possible, cependant, il est très probable que l'exploration prenne encore trop de temps et ne garantisse pas que la trajectoire calculée permettent au robot d'atteindre le but.

Afin de réduire les temps de calculs, nous avons cherché à réutiliser l'arbre déjà développé à l'itération précédente et à le mettre à jour simplement plutôt que de le reconstruire entièrement. Pour ce faire, deux étapes sont nécessaires : Il s'agit tout d'abord de ne conserver que le sous-arbre porté par le nœud courant (i.e. de supprimer les autres nœuds). Ensuite, ce nouvel arbre est étendu selon l'approche précédemment décrite. La seule différence réside dans le fait qu'avant d'être exploré, un nœud doit préalablement être validé (en vérifiant que la trajectoire qui mène à ce nœud n'est pas en collision). La découverte d'un nœud intermédiaire invalide entraîne la destruction du sous-arbre qu'il supporte.

L'idée repose sur deux hypothèses :

- le fait que l'environnement évolue peu entre deux itérations consécutives,
- le fait que la validation d'un nœud fils et moins coûteuse que le calcul d'une nouvelle branche et de son fils correspondant.

L'exploration d'un nœud n_i est réalisée selon l'algorithme 3.3.

Cette approche présente des limites lorsqu'un obstacle proche du robot change brusquement de trajectoire et remet en cause les prédictions faites durant les calculs précédents. L'arbre peut alors être invalidé entièrement. Cette situation n'affecte cependant pas la sécurité du robot : Seule l'atteignabilité du but n'est plus garantie si une solution globale ne peut pas être trouvée à temps.

De plus, nous avons pu observer que les trajectoires générées avec notre méthode restent cohérentes d'une itération à l'autre : Alors qu'une réinitialisation complète de l'arbre peut générer une trajectoire très différente de celle précédemment calculée, notre approche au contraire, conserve la même trajectoire tant que celle-ci reste valide. Cette

```
NOTATIONS
    NombreFils( x )  : retourne le nombre de fils du nœud x
    EstValide( x )   : retourne VRAI si x n'entraîne pas de collision
    EstVide( x )     : retourne VRAI si x ne contient aucun élément
    n_i              : nœud exploré
    n_j              : un fils du nœud exploré
DEBUT
    POUR CHAQUE fils n_j de n_i FAIRE
        SI ( NON EstValide( n_j ) )
            Détruire le sous-arbre dont n_j est la racine
        FIN SI
    FIN POUR CHAQUE
    SI ( NombreFils( n_i ) == 0 )
        Calculer V_poss
        Sélectionner 5 vitesses selon l'algorithme 3.2
        Créer les 5 nœuds-fils n_j correspondants
        Ajouter les n_j à l'arbre
    FIN SI
FIN
```

TAB. 3.3 – Nouvel algorithme d'exploration d'un nœud.

trajectoire, moins optimale mais plus sûre, passe par les zones de l'environnement dont l'évolution est conforme aux prédictions. En d'autres termes elle s'éloigne des obstacles dont le comportement est changeant et moins prédictible (et donc plus dangereux).

Dans le pire des cas, cette approche confère au robot un comportement réactif local qui prend tout de même en compte les collisions futures sur plusieurs itérations. Au mieux, une trajectoire complète jusqu'au but est calculée. Son tracé évite les zones dont l'évolution est incertaine et qui peuvent mettre le robot en danger.

Nous avons choisi d'accorder la priorité la plus forte à la sécurité du robot et par conséquent, sommes satisfaits du comportement prudent adopté par le robot. Si un robot plus audacieux est préférable, nous pouvons imaginer de forcer la réactualisation des nœuds de l'arbre en les dotant d'une durée de vie limitée. Cette dernière option n'a cependant pas été implémentée ni testée.

3.2.6 Résultats expérimentaux

Nous avons considéré une application mettant en scène un robot de service devant se déplacer dans un bâtiment dont le plan est connu a-priori (11 murs statiques), et où se déplacent 6 obstacles mobiles. Le but recherché est de permettre au robot d'atteindre un lieu donné le plus rapidement possible (figure 3.10).

Nous avons calculé une fonction *NF1* pour le bâtiment afin de diriger le robot plus rapidement vers son but. La trajectoire la plus rapide calculée par notre méthode comporte un peu moins de 1000 nœuds (moins de 10 secondes de parcours car $dt = 10ms$), pour un total de 2679 nœuds explorés. Ce nombre est bien supérieur à ce que nous aurions pu espérer, notamment avec l'intégration de la fonction *NF1* pour le choix des vitesses à explorer. En réalité, ces résultats sont très satisfaisants si nous considérons

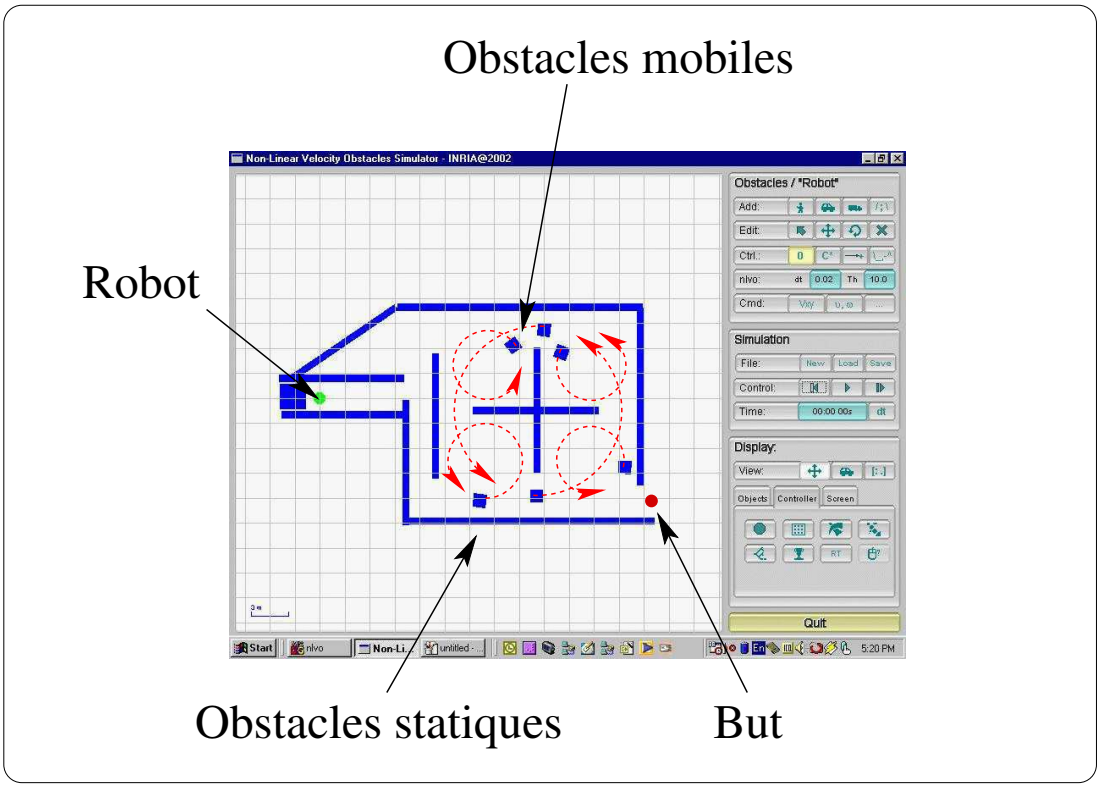

FIG. 3.10 – *Données du problème* La figure est une capture d'écran de notre simulateur, dans lequel le problème suivant a été modélisé : Le robot doit traverser le bâtiment pour sortir par le passage situé en bas à droite. Six obstacles mobiles sont sur son chemin. Ils décrivent des trajectoires circulaires.

le fait que le robot est amené à tourner plusieurs fois, et que les $\mathcal{V}$-*Obstacles* sont mal adaptés à ces manœuvres. La rapidité de calcul des *NLVO* associé au faible nombre de nœuds permet un premier calcul de la trajectoire solution en moins de 10ms.

La figure 3.11 montre quelques captures d'écran de notre simulateur : Après avoir calculé sa trajectoire, le robot tente de la suivre jusqu'au but. Les obstacles conservant leur trajectoire initiale, la première trajectoire calculée reste valide et sa vérification nécessite un temps inférieur à 1 ms.

3.3 Comparaison avec d'autres approches

Une des difficultés majeures pour comparer différentes méthodes entre elles, est que ces dernières ne disposent pas des mêmes informations pour réaliser les traitements. Ainsi, beaucoup d'approches réactives utilisent uniquement la perception locale du robot et nous sommes tentés de nous poser la question de savoir si certaines critiques à leur égard sont toujours justifiées, compte-tenu de la faible quantité d'information disponible. Nous resterons donc assez général ici en insistant surtout sur les algorithmes

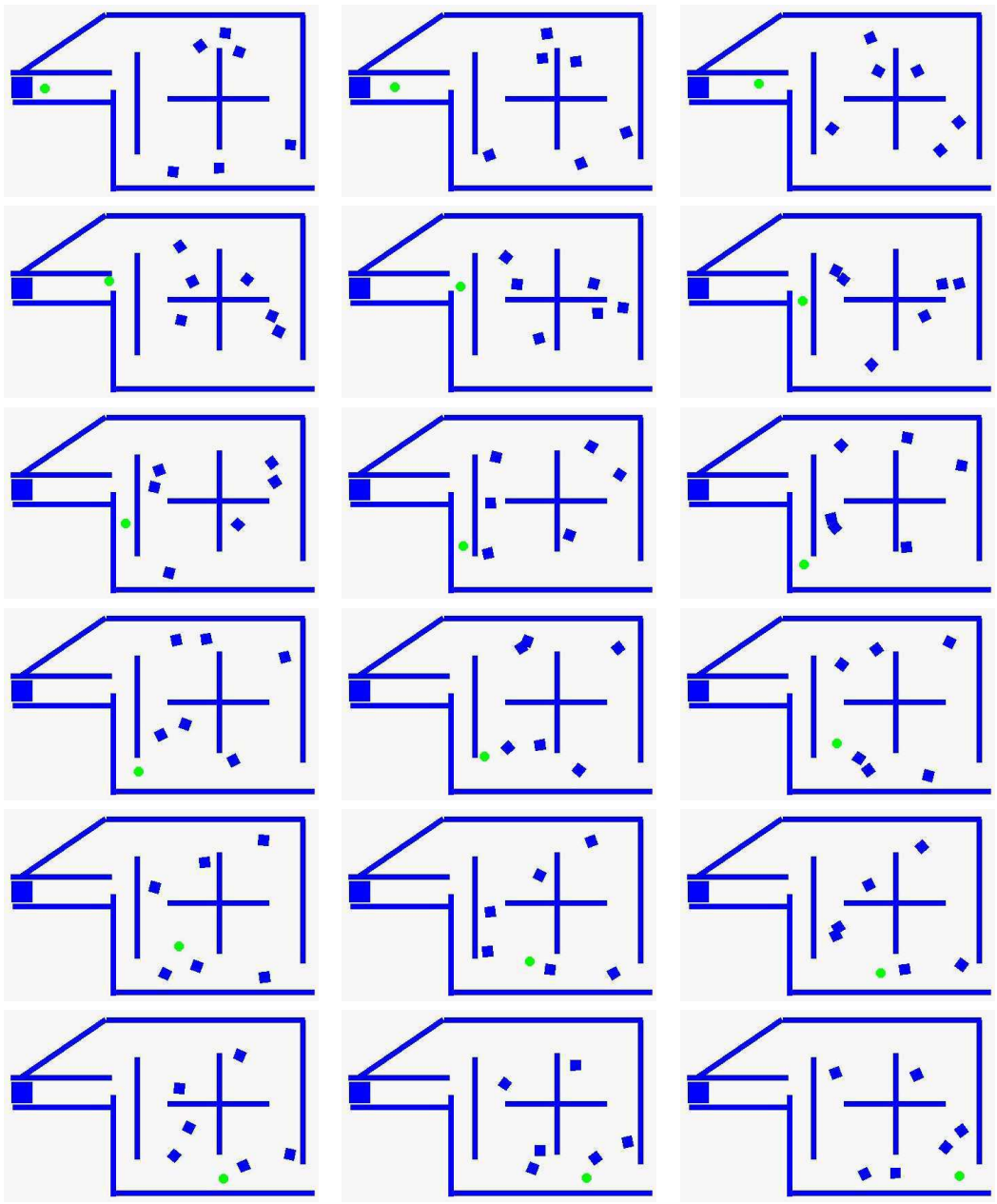

FIG. 3.11 – *Exécution d'une trajectoire planifiée en temps-réel*

employés et leur capacité à satisfaire les contraintes de temps-réel ou à traiter un type d'environnement particulier. Nous avons repris les méthodes qui nous semblent les plus représentatives pour la navigation en environnement dynamique et incertain : les *VFH**, les *GND*, les *GDW* et les bandelettes élastiques.

Bandelettes élastiques Les bandelettes élastiques et notre planificateur itératif ont des propriétés similaires. Tous deux tendent à maintenir une certaine cohérence des trajectoires proposées au cours du temps. Il existe cependant deux différences importantes : La première se trouve dans la manière de traiter les obstacles dynamiques : bien que les bandelettes élastiques soient capables de s'accommoder d'un environnement dynamique (voir l'état de l'art au chapitre 1), elles ne considèrent pas réellement la dynamique de l'environnement. De notre côté, le calcul d'une trajectoire intègre les mouvements des obstacles et, bien que nous n'ayons pas pu le vérifier expérimentalement, nous pensons que cette anticipation est indispensable en milieu très dynamique. La seconde différence porte sur le calcul de la trajectoire. Les bandelettes impliquent l'existence d'une trajectoire globale. Lorsque celle-ci s'invalide, la méthode doit attendre qu'une nouvelle trajectoire soit générée. Notre approche est beaucoup plus flexible, puisqu'elle n'est pas arrêtée par l'invalidation complète de la trajectoire actuelle au but. Simplement, dans le pire des cas, l'atteignabilité du but n'est temporairement plus garantie, cependant, la réactivité du robot à son environnement reste assurée.

VFH* La méthode *VFH** anticipe les déplacements du robot sur un intervalle de temps borné, à la manière des *NLVO*. L'inconvénient que nous pourrions citer par rapport à ces derniers est l'absence d'anticipation et donc une exploration moins efficace des déplacements suivants. Cependant, cela n'est pas systématique en raison des difficultés des *NLVO* pour des trajectoires du robot non-rectilignes. L'absence de prise en compte d'information de plus haut niveau est également difficilement critiquable pour les *VFH**, puisqu'elle est due vraisemblablement à un choix concernant les données disponibles (perception locale uniquement). Sans cela, il ne semble pas y avoir de réelle opposition à une « Global-VFH » sur la même lignée que ses consœurs.

GDW Les GDW présentent l'avantage de pouvoir intégrer toutes sortes de contraintes sur le robot (géométriques, cinématiques et dynamiques) que les *NLVO* ne font qu'approximer dans le meilleur des cas. Pourtant, l'approche discrétisée des GDW implique un nombre conséquent de calculs. La preuve en est l'apparition récente de méthodes bridées qui consistent à ne considérer qu'un sous-ensemble des vitesses admissibles pour réduire les temps de calculs. Nous citerons également le fait que les GDW considèrent des environnements statiques (comme en témoigne la construction incrémentale de la carte des obstacles). Néanmoins, il semble qu'il s'agisse, comme pour les *VFH**, d'un choix. S'agissant de calculs sur des valeurs discrétisées des vitesses, il nous semble que l'intégration des trajectoires des obstacles pourrait être réalisée sans difficulté majeure, sinon sur le choix de l'horizon

de temps à considérer. En conclusion, l'avantage de notre approche est la simplicité des calculs et leur rapidité, tandis que l'avantage des GDW est sa généricité.

GND L'approche GND combine, tout comme notre approche, un niveau global visant à planifier une trajectoire au but, et un niveau réactif qui assure la sécurité du robot. La manière dont se fait le passage de main d'un niveau à l'autre diffère cependant. D'après nos connaissances, l'échec du planificateur entraîne l'activation du niveau réactif, qui perd alors toute capacité d'anticipation. Avec notre approche, le fait de ne pas trouver de solution global n'empêche pas le robot de tirer parti de sa recherche pour choisir un déplacement. Cela permet une meilleure anticipation (i.e. sur un plus long terme). Une autre différence se situe au niveau réactif. Le GND tire ses propriétés des ND, en particulier l'identification d'un contexte (parmi 5) qui permet de déterminer le niveau de sécurité du robot et d'adapter son comportement en conséquence. Ceci rappelle la notion de risque introduite pour les *NLVO*, mais à un niveau d'analyse plus élevé. Cette analyse ne prend cependant pas en compte le mouvement des obstacles et peut être faussée en présence d'obstacles mobiles. De plus, elle demande davantage de calculs que les autres approches citées, d'où une moins grande réactivité du robot aux imprévus (dt=250ms). L'approche GND sera donc préférable pour des environnements peu dynamiques (au sens où les obstacles se déplacent à vitesse lente), tandis que les *NLVO* seront plus adaptés pour évoluer parmi des obstacles rapides.

Chapitre 4

Vers une implémentation pratique sur robots réels

Nous avons jusqu'alors considéré les $\mathcal{V}$-*Obstacles* pour un robot circulaire, parfaitement contrôlé, et des obstacles circulaires dont les trajectoires sont connues. La réalité étant toute autre, ce chapitre présente les structures de données et algorithmes permettant d'appliquer le concept de $\mathcal{V}$-*Obstacle* à un environnement quelconque emprunt d'incertitude, et à un robot réel, de forme non-circulaire et dont les paramètres de contrôle ne sont pas directement calculables : Le Cycab.

Nous présentons dans un premier temps l'application considérée. Celle-ci nous servira de fil conducteur tout au long de ce chapitre pour présenter les différents traitements nécessaires à l'utilisation des $\mathcal{V}$-*Obstacles* dans un contexte réel. Il s'agit notamment d'adapter ces derniers pour un robot et des obstacles de forme quelconques, ou de veiller à ce qu'un déplacement libre désiré soit réalisé au mieux par le robot.

4.1 Présentation de l'application considérée

Un des thèmes de recherche principaux du projet SHARP/CyberMove de l'INRIA Rhône-Alpes, au sein duquel ces travaux ont été réalisés, est l'étude de moyens de transports innovants. Dans ce cadre, nous avons considéré un projet de véhicule autonome, capable de se déplacer seul d'un lieu à un autre sur un parking, tout en évitant les obstacles de forme quelconque, qu'ils soient statiques (voitures en stationnement, murs, bordures de trottoirs) ou en mouvement (autres voitures, vélos, piétons).

Le véhicule utilisé pour les expérimentations est un Cycab. Doté de capteurs et d'organes de communication divers lui permettant de percevoir son environnement, ainsi que de 4 roues motrices orientables pour se mouvoir, le Cycab combine les avantages (et les inconvénients) d'un robot non-holonome et d'un véhicule électrique classique de type voiture de golf. Il est présenté plus en détail à l'annexe C. La particularité du Cycab est sa conception mécanique particulière qui entraîne des glissements des roues inévitables,

de par la présence de deux centres de rotation théoriques du châssis par rapport au sol (Voir détails en annexe). Il en résulte l'impossibilité de modéliser la cinématique exacte d'un tel véhicule, sinon à l'aide d'un modèle simplifié. Ce dernier introduit des erreurs dans le processus de contrôle du robot, qui présente alors des écarts parfois importants entre le déplacement désiré et celui réalisé.

Dans ce contexte, les points que nous allons traiter dans ce chapitre sont les suivants :

- ***représenter un robot de forme quelconque*** (i.e. non-circulaire) à l'aide du concept de $\mathcal{V}$-*Obstacle*
- représenter ***des obstacles de formes quelconques*** (i.e. non-circulaires) à l'aide du concept de $\mathcal{V}$-*Obstacle*
- ***modéliser***, à l'aide du concept de $\mathcal{V}$-*Obstacle*, ***les incertitudes et imprécisions*** sur les trajectoires des obstacles (dues notamment aux hypothèses sur leur cinématique, aux imprécisions des capteurs, aux approximations faites par les traitements de perception, et aux écarts de suivi de trajectoire lors des déplacements du robot).
- ***optimiser les calculs*** des $\mathcal{V}$-*Obstacles* pour le temps-réel
- ***réduire les erreurs*** (écarts) entre la trajectoire calculée et la trajectoire réellement suivie par le robot (en réduisant les différences ***entre le modèle du robot*** utilisé dans le processus de commande ***et la réalité***)

Avant toute chose, nous allons définir quelles sont les données sur l'environnement dont nous disposons.

4.2 Carte de l'environnement

Éviter les obstacles suppose une connaissance suffisante de l'environnement. Dans le cadre de notre étude, nous considérons que la carte de l'environnement nous est donnée sous la forme d'une liste d'obstacles $\Omega = \left\{ \mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_n \right\}$. Chaque obstacle possède un type T qui renseigne sur les caractéristiques a priori de l'obstacle (cinématique, accélérations maximales), une géométrie G qui définit sa forme dans $\mathcal{W}$, et une estimation de sa trajectoire γ_B sur l'intervalle de temps considéré dans les calculs. Une trajectoire est ici une suite datée d'états (configuration du point de référence + vecteur vitesse instantanée) pris par le robot sur l'intervalle de temps étudié.

L'obtention de la carte fait l'objet de plusieurs travaux en cours auxquels l'équipe *Sharp/e-Motion* participe activement, dont le projet *ParkNav*[1]. Deux fonctions ont été définies :

[1] Voir la page web de ce projet pour plus d'informations : *http ://www.inrialpes.fr/sharp/parknav/*

- ***Identifier les obstacles dans l'environnement.*** Cette tâche utilise des outils de vision et consiste à placer des caméras de surveillance en hauteur de manière à avoir une vue semi-aérienne d'un site donné (le parking de l'INRIA Rhône-Alpes dans un premier temps). Un système de détection et de suivi de cibles permet alors de disposer à tout instant de la position des obstacles, de leur taille approximative (ellipse englobante), de leur type (piéton ou véhicule) et d'une approximation de leur vitesse instantanée par dérivation des positions dans le temps.

- ***Estimer les trajectoires futures des obstacles.*** Cette fonction utilise des outils de l'intelligence artificielle, et plus particulièrement une approche bayésienne, pour « apprendre » à estimer les trajectoires futures des obstacles en fonction de leur déplacement actuel, en observant les données fournies par la première fonction. Une probabilité d'erreur ou un indice de confiance peuvent éventuellement être calculés pour chaque trajectoire.

Ces projets ne sont pas encore assez avancés pour prétendre à ce jour à une quelconque utilisation de leurs données. Néanmoins, les informations sur l'environnement que nous pouvons espérer et que nous avons définies pour la simulation sont les suivantes :

$$\Omega = \left\{ \mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_n \right\}$$

$$\mathcal{B}_i = \left| \begin{array}{l} \text{type } T \\ \text{géométrie } G \\ \text{trajectoire future } \gamma_B \end{array} \right.$$

$$T \in \left\{ PIETON, VEHICULE \right\}$$

$$G = \left| \begin{array}{ll} \text{orientation de l'ellipse } \theta_B & \text{(en rad)} \\ \text{petit rayon de l'ellipse } r1_B & \text{(en m)} \\ \text{grand rayon de l'ellipse } r2_B & \text{(en m)} \\ \text{précision de l'approximation } \mathcal{P}_g & \text{(en m)} \end{array} \right.$$

$$\gamma_B = \left\{ s_B(t_1), \ldots, s_B(t_2) \right\}$$
où t_1 désigne le temps actuel
et t_2 la borne supérieur de l'intervalle de temps considéré dans les calculs

$$s_B(t) = \left| \begin{array}{ll} \text{position du centre de l'ellipse au temps } t \; c_B(t) & \text{(en m)} \\ \text{vitesse linéaire instantanée de } \mathcal{B}_i \text{ au temps } t \; \overrightarrow{v_B}\,(t) & \text{(en m/s)} \\ \text{confiance en l'estimation } \mathcal{P}_\gamma \text{ (dans [0.0 ;1.0] avec 1.0=certitude maximale)} & \end{array} \right.$$

4.3 Traitement des robots non-circulaires

Le calcul des NLVO fait l'hypothèse que la forme du robot dans $\mathcal{W}$ est un disque. Une approximation par des disques est donc nécessaire pour le Cycab. Nous proposons deux approches selon que les rotations instantanées du robot sont négligeables ou ne le sont pas :

- rotations négligeables : méthode de fusion des *NLVO*
- rotations non-négligeables : méthode d'encadrement par un *NLVO* englobé et un *NLVO* englobant

Nous considérerons les rotations négligeables lorsque les trajectoires du robot seront très lissées et pourront être approximées localement par des segments de droites. Cette situation se rencontre dans un contexte autoroutier par exemple : Les trajectoires d'une voiture lancée à grande vitesse vérifient ces propriétés. L'orientation du véhicule change de manière très progressive (en mode nominal) et la vitesse angulaire devient négligeable dans ce cas.

4.3.1 Fusion des *NLVO*

Pour tout objet ayant un mouvement de translation pure (vitesse angulaire nulle), tout point de cet objet est animé du même mouvement de translation et admet donc la même vitesse linéaire instantanée. C'est cette propriété que nous allons utiliser pour approximer le *NLVO* d'un robot non-circulaire, dont le mouvement peut être assimilé à une translation pure.

4.3.1.1 Principe de l'approche

L'approche consiste à approximer la forme réelle du robot $\mathcal{A}$ par un ensemble de disques. Chaque disque est associé à un robot virtuel $\mathcal{A}_i$ dont la vitesse linéaire instantanée est celle de $\mathcal{A}$. De même que l'union des $\mathcal{A}_i$ constitue une approximation du robot réel, nous allons montrer que l'union du *NLVO* de chaque $\mathcal{A}_i$ représente une approximation du *NLVO* du robot réel.

4.3.1.2 Description des étapes

La méthode comprend trois phases (figure 4.1) :

- Approximer la forme du robot à l'aide de disques (robots virtuels).
- Calculer les *NLVO* des robots virtuels.
- Fusionner les *NLVO* des robots virtuels.

Approximer la forme du robot par des disques La littérature propose plusieurs méthodes génériques pour approximer une forme quelconque par des disques ([O'Rourke and Badler, 79] [Featherstone, 97]). Dans notre cas, l'utilisation de ces méthodes n'est pas justifiée puisque nous connaissons à l'avance la forme du robot

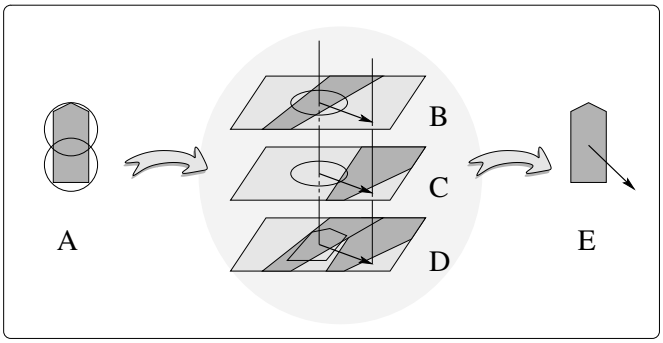

FIG. 4.1 – *Méthode de fusion* Le robot est ici approximé par deux robots virtuels (A). Leurs NLVO respectifs sont calculés (B et C) puis fusionnés (D) pour former une approximation du NLVO associé au robot réel. Une vitesse sûre peut alors être sélectionnée (E).

et que, par conséquent, une approximation « manuelle » donnera de meilleurs résultats. Selon les besoins de l'application, nous minimiserons le nombre de cercles employés ou le grossissement du robot résultant de l'approximation (figure 4.2). La seule contrainte à respecter est le recouvrement total du robot par les disques. En d'autres termes, tout point du robot réel doit appartenir à au moins un disque (nous parlerons de pavage conservatif). Chaque disque constitue un robot virtuel de forme circulaire.

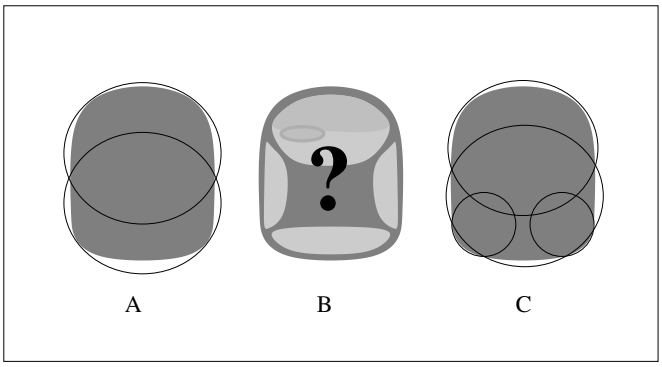

FIG. 4.2 – *Pavage d'un robot* La figure B représente un robot de type voiture électrique. Selon l'application, il est possible de choisir un pavage qui minimise en priorité le nombre de disques (A) ou le grossissement du robot (C).

Calculer les *NLVO* des robots virtuels Les robots virtuels précédemment crées sont circulaires et leur vitesse linéaire instantanée est connue : il s'agit de celle du robot réel, puisque le mouvement de ce dernier est une translation pure. Le calcul de leurs *NLVO* ne pose donc aucun problème particulier.

Fusionner les *NLVO* des robots virtuels Nous allons dans un premier temps considérer qu'il existe un pavage exact du robot par des disques. Dans ce cas, toute

vitesse qui entraîne la collision du robot entraîne aussi celle d'au moins un robot virtuel et inversement. En d'autres termes, soit $\vec{v}$ une vitesse, *NLVO* le *NLVO* du robot réel et $NLVO_i$ le *NLVO* d'un robot virtuel $\mathcal{A}_i$:

$$\vec{v} \in NLVO \qquad \Longleftrightarrow \qquad \exists \mathcal{A}_i \mid \vec{v} \in NLVO_i$$

Ce qui implique :

$$NLVO = \bigcup_i NLVO_i$$

Si nous considérons maintenant un pavage conservatif du robot par des disques. La collision d'un robot virtuel n'implique plus nécessairement celle du robot réel, et l'union des $NLVO_i$ devient une approximation conservative du *NLVO* du robot réel.

$$NLVO \subseteq \bigcup_i NLVO_i$$

La construction du *NLVO* du robot réel est réalisée en exprimant l'ensemble des $NLVO_i$ dans un repère unique de $\mathcal{V}$. Nous l'avons choisi centré en $c_A(t_0)$, le point de référence du robot réel, pour rester cohérent avec les robots circulaires. La figure 4.1 illustre la méthode.

4.3.1.3 Discussion : Avantages et inconvénients de la méthode

L'intérêt de cette approche réside en 3 points :

1. Le surcoût de calculs engendré par la méthode est parfaitement maîtrisé. Il s'agit de calculer n NLVO au lieu d'un seul (où n est le nombre de cercles utilisés pour approximer la forme du robot).
2. L'approximation du robot par des cercles est réalisée une seule fois. Elle définit le nombre de calculs supplémentaires (comme stipulé ci-dessus) et la précision de l'approximation du *NLVO*.
3. L'union des $NLVO_i$ se fait de la même manière que l'union des *NLVO* d'un robot circulaire en présence de plusieurs obstacles. Il n'implique donc aucun algorithmique supplémentaire.

L'inconvénient majeur de la méthode porte sur le fait que le robot doit avoir des mouvements assimilables à des translations pures. La méthode présentée maintenant nécessite davantage de calculs mais s'affranchit de cette contrainte.

4.3.2 Encadrement par *NLVO* englobé et *NLVO* englobant

La méthode présentée ici reprend une technique classique de planification. Il s'agit d'encadrer la forme du robot par deux disques concentriques : Un disque englobant (dit maximal) et un disque englobé (dit minimal). La comparaison des trajectoires libres obtenues pour chaque disque permet de décider s'il existe une solution pour le robot réel.

4.3.2.1 Principe de la méthode

Nous définissons tout d'abord deux cercles centrés sur le robot, tels que :

- Le diamètre du premier cercle est la largeur minimale de la trace laissée par le robot lorsqu'il est translaté. Ce cercle définit un premier robot virtuel circulaire que nous appellerons le robot minimal (Cf. figure 4.3).
- Le diamètre du second cercle est la largeur maximale de cette trace. Ce cercle définit un second robot virtuel circulaire, que nous appellerons le robot minimal (Cf. figure 4.3).

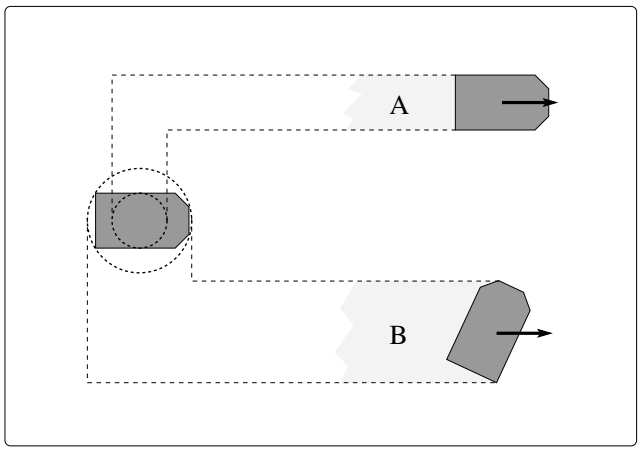

FIG. 4.3 – *Robot minimal et robot maximal* La forme réelle du robot est "encadrée" par deux cercles. Le diamètre du premier correspond à la trace minimale que peut faire le robot (A) en translatant, tandis que celui du second correspond à sa trace maximale (B).

L'idée consiste à rechercher des déplacements élémentaires libres pour chacun des robots virtuels (robot minimal et robot maximal). Trois cas peuvent se présenter (Cf. figure 4.4). Afin de faciliter la lecture des figures, nous les avons représentées pour un environnement statique. Les méthodes décrites ici s'appliquent cependant également aux environnements dynamiques.

Cas 1 Il existe un mouvement libre pour le robot maximal : cette solution est aussi valable pour le robot réel (mais pas nécessairement la meilleure compte-tenu de l'approximation grossière du robot).

Cas 2 Seul le robot minimal admet une solution : ce cas nécessite des calculs supplémentaires pour décider de la validité de la solution. Pour chaque lieu où le robot minimal « passe » mais pas le robot maximal, il est nécessaire d'utiliser la géométrie exacte du robot réel pour vérifier s'il y a effectivement collision ou non.

Cas 3 Les traitement n'admettent pas de solution pour le robot minimal : Cela signifie que le robot minimal ne peut pas « passer » (et donc que le robot réel non plus). Il n'y a pas de solution pour le robot réel.

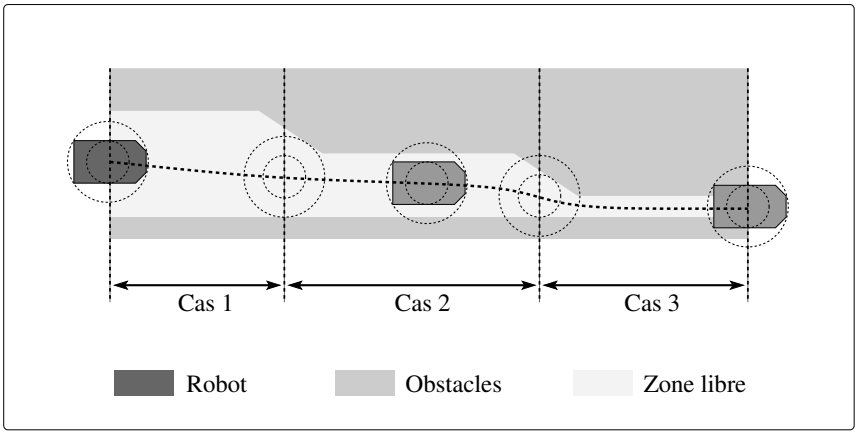

FIG. 4.4 – *Cas possibles* Cas 1 : le robot maximal peut passer et donc le robot réel également. Au cas 3, le robot minimal ne passe pas et donc le robot réel non plus. Au cas 2, seul le robot maximal passe : il n'est pas possible de donner une réponse sans examiner les collisions entre le robot réel et les obstacles.

La trajectoire libre du robot réel est construite à l'aide du planificateur itératif présenté au chapitre 3, par concaténation des mouvements successifs libres issus des cas 1 et 2. Nous allons nous intéresser maintenant aux performances de la méthode selon les mouvements explorés en priorité.

4.3.2.2 Algorithmes : Optimalité vs. Efficacité.

La méthode générale décrite précédemment peut se décliner en plusieurs algorithmes selon l'ordre dans lequel sont effectués les calculs et/ou les comparaisons des mouvements pour les deux robots virtuels.

Nous avons considéré deux types d'algorithmes de planification de trajectoires basés sur la méthode de l'encadrement :

- L'un permet de trouver plus rapidement des trajectoires solutions (éventuellement non-optimales).
- L'autre permet de trouver des trajectoires plus optimales, c'est-à-dire répondant à des critères de tâche (distance parcourue ou temps de parcours minimaux).

Recherche rapide de solutions En environnement peu contraint, il est très probable de trouver une trajectoire libre jusqu'au but pour le robot maximal. Sans nécessiter de calcul supplémentaire, cette dernière sera nécessairement valable pour le robot réel. Il est donc préférable de rechercher ce type de solution en premier.

- En cas de réussite (cas 1), cette solution est valide pour le robot réel, mais n'est pas nécessairement optimale en temps ni en distance. Le grossissement induit par l'approximation impose une trajectoire plus éloignée des obstacles. En revanche, elle est plus sûre pour ces mêmes raisons et ne nécessite pas plus de calculs que pour un robot circulaire vu précédemment.
- En cas d'échec, cela dénote soit la présence de passages étroits (cas 2), soit l'absence totale de solution (cas 3). Pour en décider, il est nécessaire de chercher des solutions locales valables pour le robot minimal, puis, si elles existent, de les valider (ou invalider) pour le robot réel.

Le problème qui se pose est de décider quand et à partir de quel nœud les passages étroits doivent-ils être recherchés. Nous proposons de vérifier l'existence de passage pour chaque nœud de l'arbre : L'exploration d'un nœud engendre des nœuds fils de deux types : les nœuds minimaux, solutions pour le robot minimal uniquement, et les nœuds maximaux, solutions pour le robot maximal (et donc implicitement pour le robot minimal). Un classement des nœuds est réalisé d'abord par catégorie (nœuds maximaux puis nœuds minimaux) puis par distance[2] au but (plus proche au plus éloigné) au sein d'une même catégorie. Ce tri permet de développer en priorité les nœuds maximaux. Dans un cas favorable, seuls ces derniers sont explorés. Sinon, il faut développer des nœuds minimaux et donc en tester préalablement la validité, ce qui induit des temps de calculs supérieurs. Cela nous donne l'algorithme simplifié 4.1.

Recherche de solutions optimales La seconde approche envisagée vise à privilégier les trajectoires optimales au détriment de la sécurité (bien que sans collision, la trajectoire du robot passera plus près des obstacles) et des temps de calculs. Son principe consiste à développer systématiquement le meilleur nœud ouvert du moment (le plus proche du but par exemple), indépendamment qu'il s'agisse d'un nœud minimal ou maximal. Par rapport à l'algorithme 4.1, les modifications sont minimes puisque seul le classement des nœuds nécessite d'être modifié. Ici, le seul critère de tri sera donc la distance au but.

4.3.2.3 Choix a priori de l'algorithme en fonction de l'environnement.

Nous avons jusqu'alors énoncé l'application visée et les contraintes qu'elle suggère (optimalité en temps de la trajectoire ou recherche rapide d'une trajectoire plus sécurisée) pour motiver le choix de l'un ou l'autre des algorithmes proposés. Nous allons

[2]la distance évoquée ici est celle définie à la section 3.1.3.2

```
NOTATIONS
    EstMaximal( x )  : retourne VRAI si x est un nœud solution du robot maximal
    EstMinimal( x )  : retourne VRAI si x est un nœud solution du robot minimal
    EstValide( x )   : retourne VRAI si x est un nœud solution du robot réel
    ouverts          : liste triée des nœuds ouverts (non explorés)
                       (Ordre de Tri : nœuds maximaux puis nœuds minimaux,
                       du plus proche au plus loin du but)
    explores         : liste des nœuds déjà explorés
    s_init           : état courant du robot
DEBUT
    Construire un nœud pour s_init et l'ajouter à ouverts
    TANT QUE ( EstNonVide ( ouverts ) )
        n ← premier nœud de ouverts
        Enlever n de ouverts
        SI ( EstButAtteint( n ) )
            Reconstruire trajectoire de s_init à n
            RETOURNER 'SUCCES'
        SINON
            SI ( EstMaximal( n ) OU ( EstMinimal( n ) ET EstValide( n ) ) )
                Ajouter les nœuds maximaux fils de n à ouverts
                Ajouter les nœuds minimaux fils de n à ouverts
                Ajouter n à explores
            FIN SI
        FIN SI
    FIN TANT QUE
    RETOURNER 'ECHEC'
FIN
```

TAB. 4.1 – Algorithme privilégiant la rapidité de la recherche.

maintenant observer l'influence de l'environnement sur les performances de ces derniers et modifier nos critères de choix en conséquence.

Nous avons identifié trois situations particulières :

Environnement très contraint La méthode de l'encadrement présentée ici ne convient pas à ce type d'environnement, où il est très facile de la mettre en défaut : Considérons un robot de base rectangulaire se déplaçant dans un couloir, tel que sa diagonale soit plus grande que la largeur du couloir (figure 4.5). Il en résulte que le robot maximal ne pourra jamais passer dans le couloir et donc que tous les déplacements devront être calculés à partir de la géométrie réelle du robot. Les deux algorithmes cités précédemment nécessiteront, dans ce cas précis (le pire), plus de calculs que si nous avions effectué directement la recherche d'une solution avec le robot réel.

Environnement "aéré" Dans un environnement aéré (ie. ne comportant pas de passage étroit), une solution possible sera une trajectoire du robot maximal, constituée uniquement de nœuds maximaux. La validation de cette trajectoire pour le robot réel est immédiate et ne nécessite aucun calcul supplémentaire. Sa construction est donc équivalente à la recherche d'une trajectoire pour un robot circulaire. Le premier algorithme, qui privilégie ce type de solution, sera alors préféré au second pour ce type d'environnement.

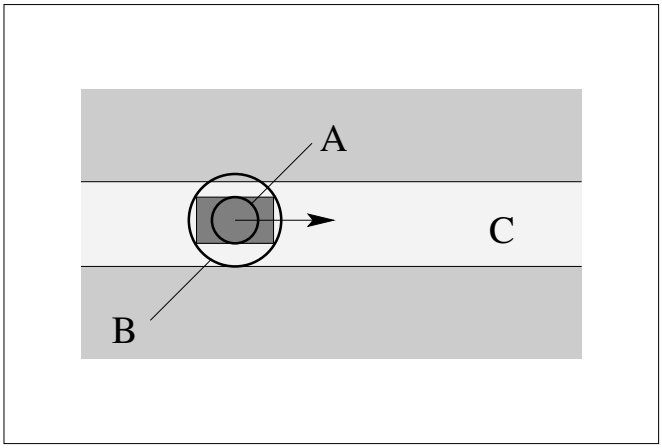

FIG. 4.5 – *Environnement contraint* La forme rectangulaire du robot est encadrée par le robot minimal A et le robot maximal B. Dans le couloir C, le robot minimal peut passer mais le robot maximal est en collision permanente avec les murs. Pour savoir si le robot réel est lui-aussi en collision, la forme réelle du robot devra être utilisée.

Environnement avec passage étroit obligatoire Dans un environnement aéré mais où la trajectoire vers le but passe nécessairement par un passage étroit (passage de porte entre deux pièces par exemple), le premier algorithme explore systématiquement tous les nœuds maximaux possibles situés « avant » le passage étroit, quelque soit leur nombre et l'intérêt qu'ils représentent. Selon la topologie, le nombre de nœuds ainsi calculés inutilement par le premier algorithme peut être très supérieur à celui des nœuds calculés par le second. La figure 4.6 illustre un tel cas.

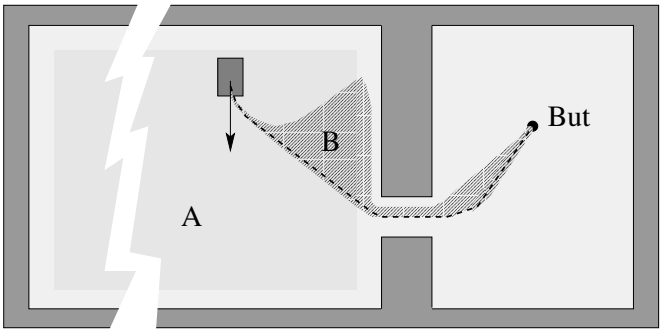

FIG. 4.6 – *Espace exploré selon l'algorithme choisi* Avec le premier algorithme, tous les nœuds maximaux possibles sont explorés avant les minimaux, ce qui peut conduire à un arbre de recherche important (A). Avec le second, seuls les « meilleurs » nœuds du moment sont développés, réduisant ainsi la taille de l'arbre (B). La figure illustre l'avantage du second algorithme (en terme de nœuds développés), pour un environnement vaste comprenant un passage étroit obligatoire pour atteindre le but.

Le second algorithme, en revanche, privilégie les "meilleurs" nœuds et minimise ainsi la taille de l'arbre de recherche. Cependant, une validation des nœuds mini-

maux est nécessaire : Elle entraîne un surcoût de calculs (d'autant plus important que la forme réelle du robot est complexe).

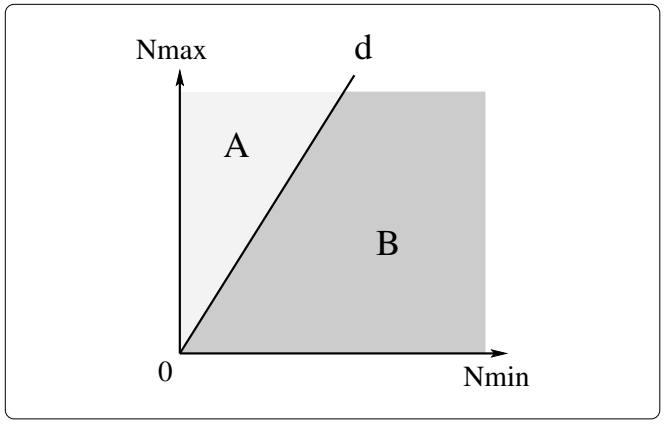

FIG. 4.7 – Nmin et Nmax désignent respectivement le nombre de nœuds minimaux et maximaux développés. Si l'exploration d'un nœud minimal prend m fois le temps nécessaire à l'exploration d'un nœud maximal, alors m est le coefficient directeur de la droite d de la figure, passant par l'origine. Elle coupe le plan en 2 demi-plans A et B. Pour tout couple n(n1,n2), il est possible de savoir s'il est plus intéressant d'utiliser l'algorithme 1 (n ∈ B), l'algorithme 2 (n ∈ A), ou si cela est sans importance (n ∈ d). En d'autres termes, l'algorithme 1 sera préférable chaque fois que $n2/n1 < m$.

Dans ces conditions, il n'est pas possible de déterminer a priori quel algorithme demandera le moins de temps pour trouver une solution, a moins de connaître parfaitement l'environnement, ce qui n'est pas l'hypothèse que nous avons envisagée. Dans ce cas, nous nous orienterons vers une sélection automatique de l'algorithme telle que proposée maintenant.

4.3.2.4 Choix automatique d'un algorithme

Nous proposons ici de contrôler quel algorithme est le meilleur (le moins coûteux en temps), au fur et à mesure du développement de l'arbre. Au lieu de choisir un algorithme particulier et de l'appliquer jusqu'à l'obtention d'une solution ou d'un échec, il n'est appliqué que pour l'exploration du prochain nœud. Au nœud suivant, l'évaluation du meilleur algorithme est réitérée.

Nous supposons connus le temps t_{min} (respectivement t_{max}) nécessaire pour explorer un nœud minimal (respectivement maximal). Cette information peut être évaluée simplement à partir de l'horloge système, en développant plusieurs nœuds fictifs.

Nous supposons également connu le nombre n_{min} (respectivement n_{max}) de nœuds minimaux (respectivement maximaux). De simples compteurs suffisent à obtenir ces données.

Soit k un coefficient réel positif ou nul dont nous verrons la signification plus loin. Nous le considérons égal à 1 dans un premier temps. Intuitivement, nous pouvons alors définir les règles suivantes :

- choix de l'algorithme 1 si $n_{max}/n_{min} < \frac{k \times t_{min}}{t_{max}}$,
- choix de l'algorithme 2 si $n_{min}/n_{max} > \frac{k \times t_{max}}{t_{min}}$,
- choix indifférent si $n_{max}/n_{min} = \frac{k \times t_{min}}{t_{max}}$ (par défaut, l'algorithme 1 est employé).

En modifiant la valeur de k, nous pouvons noter que cela a pour effet d'autoriser des trajectoires plus ou moins optimales, en forçant le choix de l'un ou l'autre des algorithmes de la manière suivante :

- si $k_{\to\infty}$ alors priorité à l'algorithme 1 (trajectoires plus sécurisées),
- si $k_{\to 0}$ alors priorité à l'algorithme 2 (trajectoires plus optimales),
- si $k = 1$ alors priorité à l'algorithme nécessitant le moins de temps de calculs.

Le coefficient k peut donc se lire comme « le nombre de fois que le temps accordé aux nœuds maximaux est supérieur au temps accordé aux nœuds minimaux ». Ainsi en reprenant les valeurs précédentes, $k = 1$ signifie que nous autoriserons le même temps pour les deux types de nœuds. $k = 0$ signifiera que le temps accordé aux nœuds minimaux est nul et inversement que tout le temps est accordé aux nœuds minimaux si $k_{\to\infty}$.

Mise en œuvre Le changement de méthode à la volée est simple puisqu'il s'agit uniquement de changer le critère de tri de la liste des nœuds ouverts. Avec une liste unique, changer de critère nécessiterait de retrier entièrement cette liste et prendrait un temps non négligeable. Pour contourner ce problème, nous avons dupliqué la liste pour disposer à tout instant de deux "copies", contenant les mêmes nœuds, mais triés selon des critères différents. Chaque nœud conserve un lien sur son jumeau afin de faciliter les suppressions. Les listes étant implémentées avec des arbres rouge-noir ([Cormen et al., 90]), le temps moyen d'insertions/suppressions reste très inférieur à celui d'un nouveau tri (voir page 106).

4.3.2.5 Avantages et inconvénients de la méthode

La méthode d'encadrement de la forme du robot par deux disques concentriques est simple à mettre en œuvre, et permet des gains de temps considérables dans un cas pas trop contraint, en particulier pour un robot de forme complexe. En revanche, pour un environnement très contraint, nous avons pu constater que cette méthode n'est pas efficace. Cela provient de l'approximation trop grossière du robot par le disque englobant.

4.3.3 Vers une méthode hybride

Par manque de temps, l'approche de la fusion et celle de l'encadrement ont été testées séparément. Nous pensons néanmoins que leur combinaison permettrait de meilleurs résultats en raison de leur complémentarité, ou tout au moins une plus grande généricité

vis-à-vis du type d'environnement et du déplacement du robot. L'idée serait d'appliquer l'algorithme le plus adapté au contexte courant, à savoir la fusion tant que les rotations sont négligeables, et l'encadrement le reste du temps.

4.4 Traitement des obstacles non-circulaires

Le problème consiste à trouver dans un premier temps une méthode permettant d'approximer une forme quelconque par des disques. Contrairement au pavage du robot, cette approximation ne peut pas se faire "manuellement" puisque les formes des obstacles ne sont pas connues à priori. Il faut donc disposer d'une méthode automatique, suffisamment simple pour ne pas alourdir les traitements, et suffisamment efficace pour donner un bon compromis entre précision et nombre de cercles générés

Ensuite, il s'agira d'attribuer des trajectoires à ces obstacles virtuels à partir de celle de l'obstacle d'origine. Nous devrons donc faire des hypothèses quant au mouvement de l'obstacle.

4.4.1 Obstacles de forme rectangulaire

Pour des raisons d'efficacité, nous avons choisi de limiter notre étude à des obstacles de forme rectangulaire. Le choix des rectangles (une ligne ou un carré sont des rectangles particuliers et entrent dans cette catégorie) est motivé par le fait que toute forme peut être inscrite entièrement dans un rectangle de taille minimale, appelée sa boite englobante orientée (figure 4.8), couramment utilisée dans les techniques de test de collisions en réalité virtuelle et en robotique.

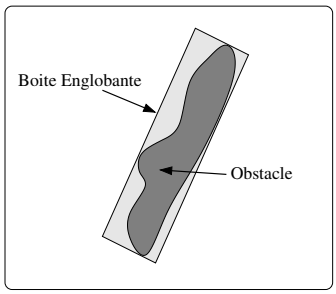

FIG. 4.8 – *Boite englobante orientée associée à un obstacle de forme quelconque.*

4.4.2 Pavement conservatif d'un rectangle par des disques

Il existe une infinité de manières de couvrir un rectangle par des disques. Les paramètres à considérer sont le nombre de disques utilisés, leurs rayons, et l'erreur d'approximation résultante.

Nous avons choisi de ne pas faire appel aux méthodes génériques telles que celles employées dans [O'Rourke and Badler, 79] ou [Featherstone, 97], car nous les avons jugé trop coûteuses pour le pavage d'une forme simple telle qu'un rectangle. Nous leur avons préféré une approche plus add'hoc.

Soit un rectangle de longueur L et de largeur l. Le principe retenu consiste, dans un premier temps, à paver ce dernier par des carrés identiques de côté l, dont les arêtes sont parallèles (et perpendiculaires) à celles du rectangle. Le nombre minimal de carrés nécessaires vaut $n = round(L/l)$, où $round(x)$ est la fonction qui retourne l'entier immédiatement supérieur à x. Dans un second temps, chaque carré est remplacé par son cercle minimal englobant (i.e. passant par ses sommets et de diamètre $l\sqrt{2}$). L'erreur d'approximation maximale résultante est un grossissement de l'obstacle compris entre 0 et $(l\sqrt{2})/2 - l/2 \approx 0.2l$ soit environ 20% de la largeur du rectangle. Selon le nombre de disques autorisés ou l'erreur maximale autorisée, il est possible de scinder le rectangle en deux sous rectangles de dimensions L par l/2.

Cette méthode de pavage présente l'avantage d'être très simple à mettre en œuvre et peu coûteuse en calculs, tout en permettant un contrôle total du nombre de disques générés et de l'erreur d'approximation engendrée.

4.4.3 Estimation des vitesses instantanées des obstacles virtuels

Nous avons envisagé deux cas, selon la valeur de la vitesse angulaire instantanée de l'obstacle, notée ω_B :

- $\omega_B = 0$: l'obstacle se déplace selon une trajectoire rectiligne et par conséquent les obstacles virtuels qui l'approximent se déplacent selon la même vitesse $\overrightarrow{v_B}$. Cela est aussi valable pour les obstacles statiques.
- $\omega_B \neq 0$: l'obstacle admet un centre de rotation instantané unique, noté O. Soit B le point de référence de l'obstacle $\mathcal{B}$, alors $OB = \left\|\overrightarrow{v_B}\right\|/\omega_B$ et $\overrightarrow{OB} \cdot \overrightarrow{v_B} = 0$. Tout point P de $\mathcal{B}$ vérifie cette relation en particulier les centres des obstacles virtuels $\mathcal{B}_i$. L'obstacle étant un solide monobloc, nous supposons ω_B constant pour tout point de $\mathcal{B}$. Cela nous donne le vecteur vitesse instantanée associé à chaque $\mathcal{B}_i$.

Remarque : Méthode alternative Nous avions dans un premier temps approximé les vitesses instantanées de chaque obstacle virtuel en dérivant dans le temps ses positions successives. Cette solution n'a cependant pas été retenue car elle est très dépendante de la discrétisation en temps des trajectoires des obstacles et peut donner des valeurs erronées lorsque le déplacement de l'obstacle n'est pas continu (changements brusques de direction par exemple).

4.4.4 *LVO* d'un obstacle de forme quelconque

Nous considérons ici le cas particulier d'un obstacle statique, ou d'un obstacle se déplaçant en ligne droite, à vitesse constante et sans rotation sur lui-même. Nous allons

montrer que dans ces cas précis, la construction du *LVO* exact ne nécessite quasiment pas plus de calculs que pour un obstacle circulaire.

La justification est purement géométrique. Elle s'appuie sur la manière dont est construit le cône de collision CC à partir duquel est obtenu le *LVO*. Pour rappel, CC peut être vu comme l'ensemble des vitesses linéaires instantanées du robot qui entraînent des collisions futures avec un obstacle statique. Pour le cas d'un obstacle circulaire, nous avions montré qu'il s'agissait de l'ensemble des vitesses de $\mathcal{V}$ situées dans un cône dont les bords sont les demi-droites issues de $c_A(t_0)$, le point de référence du robot, et tangentes à l'obstacle. De même, pour un obstacle de forme quelconque, nous pouvons construire ces deux demi-droites tangentes extérieurement à l'obstacle. Pour un obstacle polygonale par exemple, les points de tangence seront les sommets P_i et P_j tels que les demi-droites issues de $c_A(t_0)$ et passant respectivement par P_i et P_j, n'intersectent aucun côté de l'obstacle, autre que ceux issus respectivement de P_i et de P_j (voir figure 4.9).

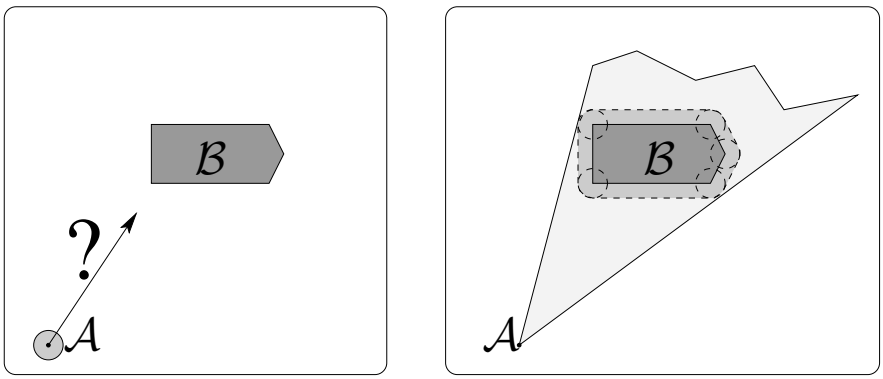

FIG. 4.9 – *CC d'un obstacle de forme quelconque.* Les deux tangentes les plus extérieures à $\mathcal{CB}$ forment les bords de CC.

Cette méthode de construction permet un calcul simple des bords du *LVO* associé à un obstacle de forme quelconque. Le problème qui se pose alors est le calcul du rapport k caractérisant les temps à collision associés aux *Vitesses de Collision* (situées à « l'intérieur » du $\mathcal{V}$-*Obstacle*). La forme quelconque de l'obstacle ne permet pas de proposer une expression générique comme pour le cas d'un disque. Ces temps devront par conséquent être approximés par la construction approchée du $\mathcal{V}$-*Obstacle* dans $\mathcal{V} \times \mathcal{T}$, telle que décrite dans la suite du chapitre (voir « Approximation polygonale d'un $\mathcal{V}$-*Obstacle* » page 95).

4.5 Algorithmes et structures de données pour le temps-réel

D'un point de vue pratique, les méthodes décrites au chapitre 2 sont difficilement exploitables telles qu'elles. D'une part, elles nécessitent toutes une discrétisation du temps et de l'espace. D'autre part, certaines constructions doivent être optimisées afin de réduire les temps de calculs nécessaires. Nous allons présenter ici les principales techniques que nous avons utilisées pour implémenter les $\mathcal{V}$-*Obstacles* et leurs fonctions dérivées.

4.5.1 Représentation de $\mathcal{V} \times \mathcal{T}$ et de V_{adm} par un tableau à 2 dimensions

Seules les vitesses de V_{adm} ont besoin d'être représentées. Par conséquent, nous avons dans un premier temps calculé une approximation polygonale de cet ensemble (voir l'exemple figure 4.10 pour un robot holonome), puis nous en avons calculé une boîte englobante orientée. Cette boîte a ensuite été implémentée dans un tableau de réels de dimension 2 : Chaque case du tableau correspond à un sous-ensemble de V_{adm}. Son contenu représente le temps à collision le plus faible associé aux vitesses de ce sous-ensemble. Par défaut, les temps à collision sont initialisés à $TH + dt$, signalant l'absence de collision sur l'intervalle de temps $[t_0,\ TH]$. Un temps à collision nul est affecté aux cases dont les vitesses appartiennent à la boîte englobante de V_{adm}, mais pas à V_{adm} (voir figure 4.11).

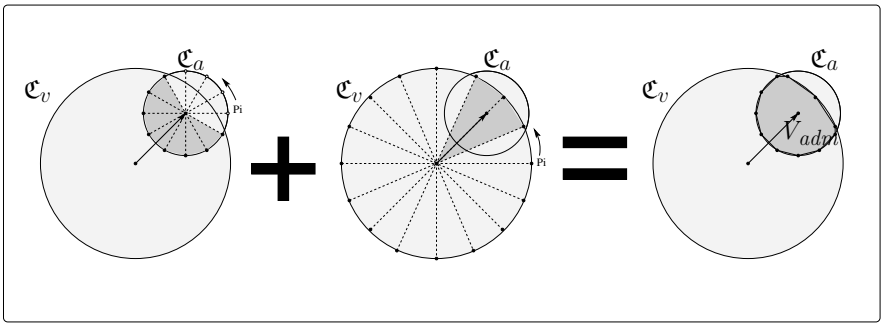

FIG. 4.10 – *Exemple d'approximation polygonale de* V_{adm}. La figure illustre la méthode de construction d'une approximation polygonale de V_{poss} pour un robot holonome, telle que définie à la page 53. Le principe consiste à approximer le disque $\mathfrak{C}_a$ par un polygone régulier. Seuls les points P_i situés à l'intérieur de $\mathfrak{C}_v$ ($P_i \in \mathfrak{C}_v$) sont conservés. Si $\mathfrak{C}_a \subseteq \mathfrak{C}_v$, l'algorithme s'arrête car l'approximation polygonale de V_{poss} est celle de $\mathfrak{C}_a$. Sinon, nous calculons une approximation polygonale de $\mathfrak{C}_v$. Seuls les P_j situés dans $\mathfrak{C}_a$ ($P_j \in \mathfrak{C}_a$) sont conservés. L'approximation polygonale de V_{adm} est obtenue en reliant les P_i entre eux, puis les P_j entre eux, en « tournant ».

A titre d'exemple, pour un robot tel que le Cycab, dont les vitesses sont comprises entre 0m/s et 5m/s, nous avons choisi une discrétisation de V_{adm} comprise entre 16×16 et 128×128, selon la puissance de calcul disponible.

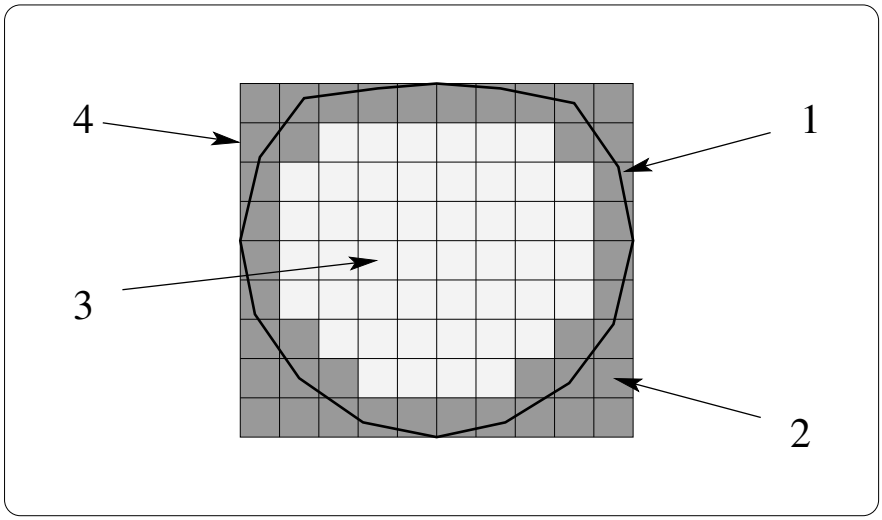

FIG. 4.11 – *Discrétisation de V_{poss}.* Les cellules qui ne sont pas incluses dans V_{poss} ont leur temps à collision (et donc celui de l'ensemble des vitesses qu'elles représentent) initialisé à 0. 1) représente l'approximation polygonale de V_{poss}, 2) représente les cellules initialisées à 0, 3) les cellules initialisées à $TH + dt$ et 4) la grille qui contient toutes les vitesses de V_{poss} (dont les bords correspondent à la boîte englobante orientée de V_{poss}).

4.5.2 Tracé 2.5D d'un $\mathcal{V}$-*Obstacle* dans V_{adm}

La représentation que nous avons choisie pour V_{adm} peut être vue comme une image rectangulaire, dont chaque pixel correspond à une vitesse et dont la couleur de chaque pixel correspond au temps à collision associé à la case. Cette analogie laisse présager de l'approche envisagée pour construire les $\mathcal{V}$-*Obstacles*, à savoir l'utilisation de primitives graphiques 2D. Nous avons utilisé le terme 2.5D pour dénoter le fait que les $\mathcal{V}$-*Obstacles* sont dessinés dans un plan, mais comportent néanmoins une information temporelle.

La méthode comprend trois étapes :

- le ***calcul de la matrice de transformation*** permettant d'effectuer les calculs dans un repère centré en haut à gauche de la boîte englobante (première case du tableau).

- le ***calcul d'une approximation polygonale des $\mathcal{V}$-Obstacles*** dans $\mathcal{V} \times \mathcal{T}$, sous la forme de triangles définis dans $\mathbb{R}^3$ (la dimension Z est le temps à collision associé à chaque sommet). La méthode est décrite juste après.

- le ***« tracé » des triangles*** par ordre décroissant des Z (i.e. des temps à collision), afin que chaque triangle dont les vitesses impliquent une collision dans un futur plus proche soit dessiné par dessus les autres triangles. Ainsi, lorsqu'une vitesse entraîne des collisions à des temps différents, cela permet de n'en conserver que le plus proche. Nous nous sommes inspirés pour l'écriture de ces fonctions des librairies graphiques pour jeux en 2D sous DOS, disponibles gratuitement sur l'Internet.

4.5.2.1 Approximation polygonale d'un $\mathcal{V}$-*Obstacle*

Nous avons donné au chapitre 2 l'expression analytique des bords d'un $\mathcal{V}$-*Obstacle* paramétrée par le temps à collision. La discrétisation du temps que nous avons choisie par défaut est de dt secondes, soit la période du contrôleur d'exécution du robot. Nous calculons les points des bords gauche et droit du $\mathcal{V}$-*Obstacle* pour chaque valeur de t, tel que $t = t_0 + i.dt$ avec $i \in [0, \ldots, round(\frac{(TH - t_0)}{dt})]$. Ces points sont notés $vo_g(t)$ et $vo_d(t)$.

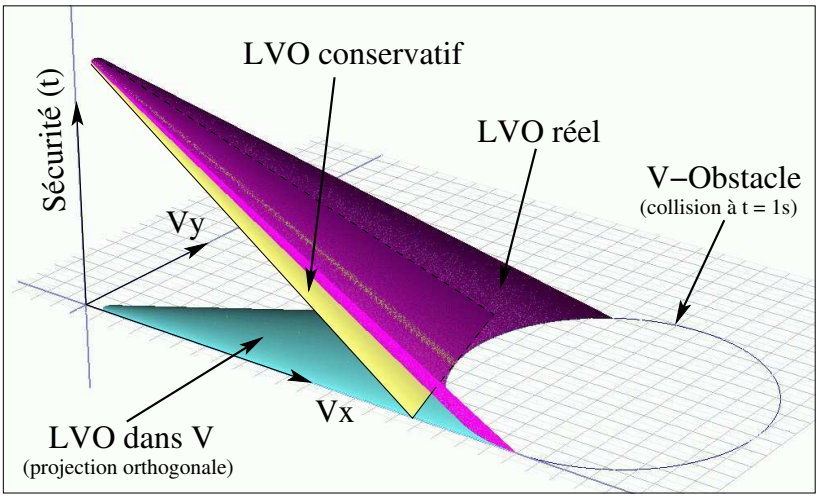

FIG. 4.12 – *Estimation conservative du temps à collision.* Le tracé d'un *LVO* linéarisé (voir section 4.5.3.2 page 97) dans $\mathcal{V} \times \mathcal{T}$ permet de mieux se rendre compte de la différence entre le temps à collision associé à une vitesse prise sur l'axe du *LVO* et une vitesse prise sur son bord. Si nous désirons utiliser uniquement les bords du *VO* pour l'approximer, alors le temps à collision associé à une « tranche » doit être le temps à collision minimal de l'ensemble des vitesses que contient cette tranche. Le résultat est dessiné ici dans $\mathcal{V} \times \mathcal{T}$ (sous la dénomination « *LVO* conservatif ») pour mieux se rendre compte de la construction obtenue.

En prenant les paires consécutives de points 2 par 2 ($vo_g(t)$, $vo_d(t)$, $vo_g(t + dt)$ et $vo_d(t + dt)$), nous pouvons construire une paire de triangles pour chaque valeur de t et obtenir ainsi une première représentation polygonale du $\mathcal{V}$-*Obstacle* dans $\mathcal{V} \times \mathcal{T}$. Il faut cependant considérer le fait que ces triangles contiennent des vitesses situées sur l'axe central du $\mathcal{V}$-*Obstacle* dont le temps à collision est inférieur à t. Par conséquent, le temps à collision n'est pas t mais $t - \frac{r_B}{D}$ (figure 4.12) où r_B est le rayon de l'obstacle et D est la distance dans $\mathcal{V}$ entre le centre du robot et le centre de $VO(t)$ (figure 4.13)).

Au chapitre précédent, nous avons caractérisé le temps à collision associé à une vitesse en fonction de sa distance minimale aux bords du $\mathcal{V}$-*Obstacle*. Cela permet de calculer une information plus approchante que celle obtenue ici. Cette expression est cependant coûteuse en calculs et n'est pas adaptée aux contraintes temps-réel. Nous allons proposer une méthode alternative, permettant de construire une approximation polygonale de l'intérieur d'un $\mathcal{V}$-*Obstacle* et des temps à collision associés.

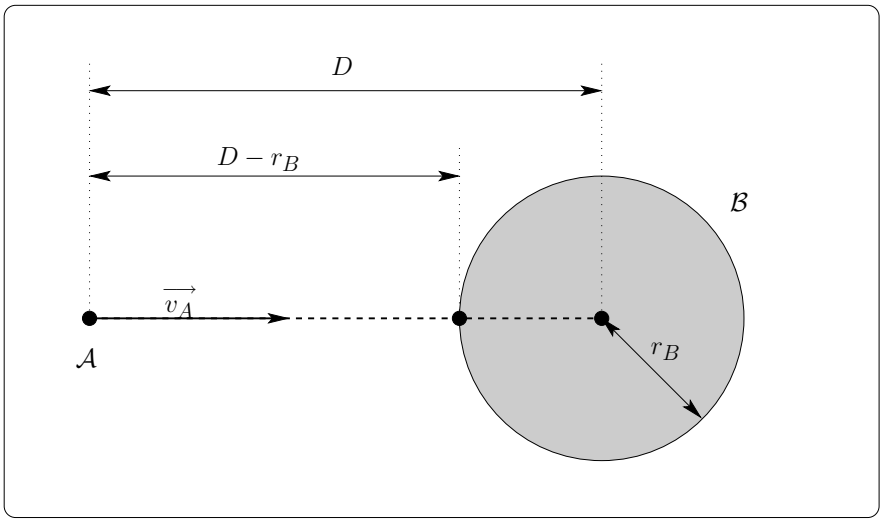

FIG. 4.13 – *Temps à collision des vitesses situées sur l'axe central du VO.* Par définition, si une vitesse $\overrightarrow{v_A}$ permet au robot d'atteindre le centre de l'obstacle au temps t_2, alors elle permettra d'atteindre son bord au temps $t_1 = t_2 - \frac{r_B}{D}$. En effet, la première proposition nous donne $\left\|\overrightarrow{v_A}\right\| = D/t_2$, et la seconde $\left\|\overrightarrow{v_A}\right\| = (D - r_B)/t_1$. Nous en déduisons que $D/t_2 = (D - r_B)/t_1$, soit $t_1 = t_2 - \frac{r_B}{D}$.

Le principe est illustré par la figure 4.14 : Il consiste à approximer par une ligne brisée, le plus petit arc de cercle de $\delta VO(t)$ situé entre les deux points de tangence $vo_g(t)$ et $vo_d(t)$. Ces points correspondent tous à des vitesses impliquant des collisions à l'instant t.

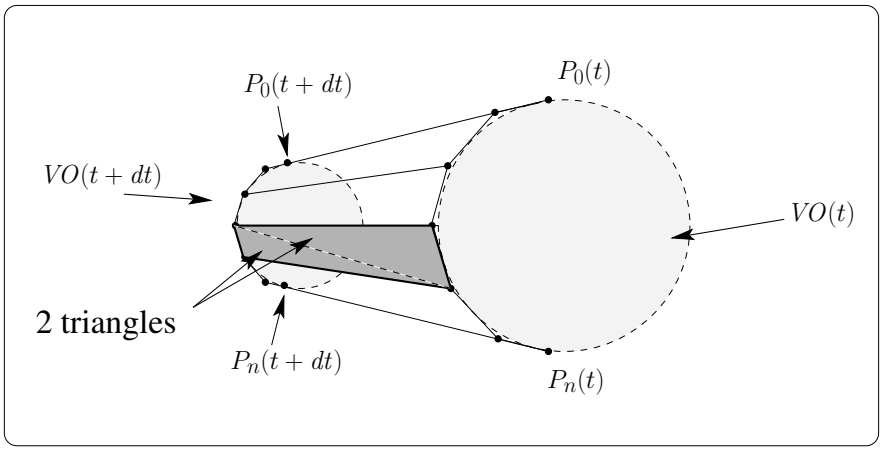

FIG. 4.14 – *Approximation 2.5D d'un* $\mathcal{V}$-Obstacle *dans* $\mathcal{V} \times \mathcal{T}$. Après approximation conservative des $\delta VO(t)$ par des segments (ici $n+1$), les sommets adjacents sont regroupés par 4 pour former 2 triangles. Le temps à collision qui leur est associé est le plus petit de leurs sommets (ici t).

Soient $P_i(t)$ ($i \in [0, n]$) les sommets de cette ligne, avec $P_0(t) \approx vo_g(t)$ et $P_n(t) \approx$

$vo_d(t)$. Le temps à collision associé à chaque $P_i(t)$ est t. Afin d'avoir un point $P_i(t)$ sur l'axe du $\mathcal{V}$-*Obstacle*, nous prendrons de préférence n impair (dans nos tests, nous avons choisi $n = 7$). Les points adjacents sont pris 4 par 4 pour former des quadrilatères (par exemple, $P_i(t)$, $P_{i+1}(t)$, $P_{i+1}(t + dt)$ et $P_i(t + dt)$) qui sont ensuite scindés en deux triangles.

Cette technique de construction permet d'obtenir une meilleure approximation d'un $\mathcal{V}$-*Obstacle* dans $\mathcal{V} \times \mathcal{T}$ sous forme de triangles 2.5D (triangle 2D associé à un temps). Elle pose néanmoins un problème : Elle attribue le même temps à collision pour toutes les vitesses appartenant à un même triangle, à savoir le plus petit des temps associés à chacun de ses sommets. Son tracé en 3D dans $\mathcal{V} \times \mathcal{T}$ que nous allons voir maintenant apporte une solution à ce problème.

4.5.3 Tracé 3D d'un $\mathcal{V}$-*Obstacle* dans V_{adm}

Dans la méthode précédente, le tracé des triangles est réalisé à l'aide de primitives graphique 2D. Nous allons utiliser ici des techniques issues du dessin en 3D, telles que la technique du Z-buffer ou l'utilisation de librairies spécialisées comme OpenGL et ses dérivées.

4.5.3.1 Technique du Z-buffer

Le Z-buffer est une technique couramment employée en 3D, y compris dans les cartes graphiques, pour gérer l'affichage de scènes 3D. Le principe consiste à définir un tableau de dimension 2 tel que celui utilisé jusqu'alors. Chaque case représente un pixel. Le contenu d'une case représente la coordonnée Z du pixel. Lors du tracé d'un pixel, la coordonnée Z de ce dernier est comparée à la valeur déjà stockée dans la case. Si elle est inférieure, le pixel est dessiné (son Z remplace celui stocké), sinon la valeur stockée reste inchangée (figure 4.15). Par rapport à l'approche 2.5D vue précédemment, cette approche nécessite davantage de calculs, en particulier le test sur chaque pixel, mais aussi un algorithme de remplissage de triangle plus complexe (pour refléter les différences de temps, le remplissage n'est pas plein mais en dégradé). En contre-partie, elle permet une représentation plus approchée (plus fine) des temps à collision.

Remarque : Contrairement à l'approche en 2.5D, les triangles seront dessinés ici dans l'ordre croissant de leur temps à collision. Ceci permet de faire l'économie du dessin des pixels cachés, ce qui représente un nombre considérable d'appels de fonctions et d'affectations mémoire.

4.5.3.2 Linéarisation des temps à collision

Une fois maîtrisé le tracé de triangles en 3D, il est intéressant d'essayer de tracer directement les *LVO* sans avoir à les discrétiser pour toutes les valeurs possibles de t, comme cela était nécessaire en 2.5D. Un rapide examen de la coupe verticale d'un *LVO* selon son axe permet de constater que son expression est de la forme $x(t) = c/t; y(t) = t$

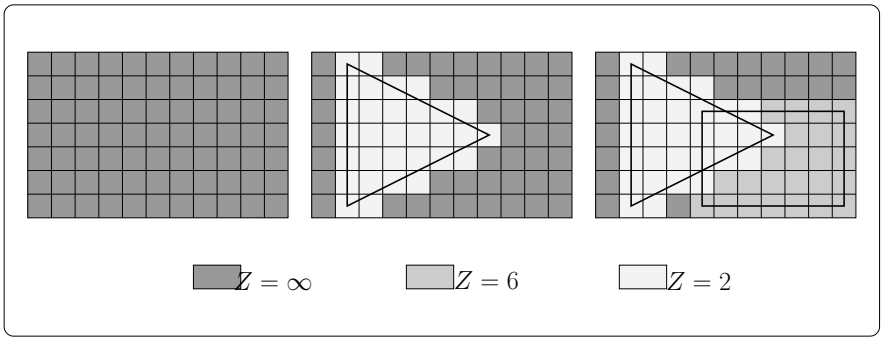

FIG. 4.15 – *Principe du Z-buffer.* La grille est supposée initialisée à une valeur maximale (∞ ici). Un triangle est dessiné à Z=2. Ensuite un rectangle est dessiné à Z=6. Seules les cellules de la grille pour lesquelles le « pixel » dessiné est plus proche que celui existant sont modifiées.

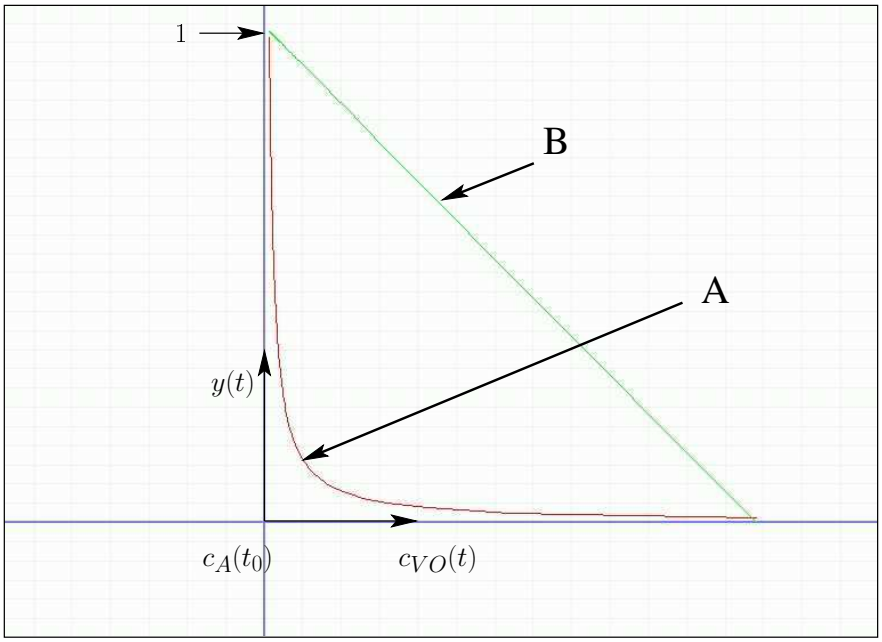

FIG. 4.16 – *Linéarisation de la pente d'un LVO* Nous avons coupé un *LVO* le long de son axe central, perpendiculairement au plan $\mathcal{V}$ de $\mathcal{V} \times \mathcal{T}$. A représente la coupe d'un *LVO* dont l'ordonnée est le temps à collision ($y(t) = t$). B représente ce même *LVO* avec une ordonnée « linéarisée » ($y(t) = \frac{(t-t_0)\cdot TH}{(TH-t_0)\cdot t}$).

où c est une constante et $t \in [t_0,\ TH]$. Nous allons modifier l'expression de $y(t)$ telle que la courbe $y = f(x)$ soit une droite (figure 4.16).

La solution la plus simple est $y(t) = 1/t$. Cependant cela implique $y(t) \to \infty$ pour $t_0 \to 0$, ce qui est difficilement représentable par l'ordinateur. Nous proposons donc de normaliser $y(t)$, tel que :

$$t \in [t_0,\ TH] \qquad \Longrightarrow \qquad y(t) \in [0,\ 1]$$

L'expression de $y(t)$ devient :

$$y(t) = \frac{(t - t_0) \cdot TH}{(TH - t_0) \cdot t}$$

Le temps à collision t, utilisé comme coordonnée en Z des différents points calculés, est donc remplacé par l'expression de $y(t)$. Dans $\mathcal{V} \times \mathcal{T}$, cela permet d'aligner entre eux les $P_i(t)$ de même indice i pour un LVO. Sa construction ne nécessite plus alors que le calcul des $P_i(t)$ pour $t = t_0$ et $t = TH$ (figure 4.17).

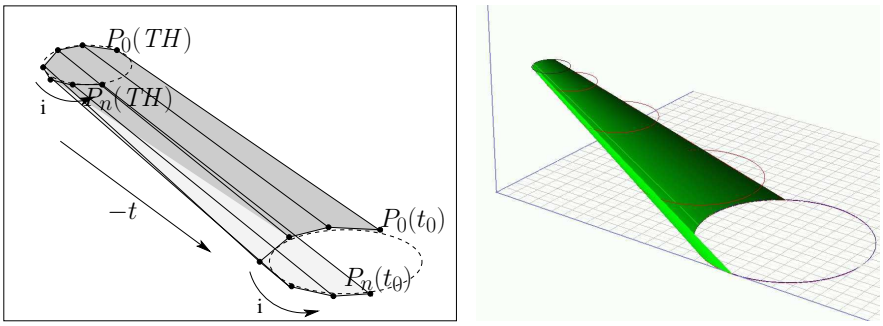

FIG. 4.17 – *Approximation 3D d'un LVO dans* $\mathcal{V} \times \mathcal{T}$. Seul le calcul des points $P_i(t)$ pour $t = t_0$ et $t = TH$ est nécessaire pour représenter le LVO complet dans $\mathcal{V} \times \mathcal{T}$. La figure de droite montre le résultat pour $n = 49$. Quelques $VO(t)$ intermédiaires ont été dessinés pour vérification (cercles horizontaux en rouge léger).

4.5.3.3 Utilisation d'OpenGL et des cartes graphiques

La technique du Z-buffer est simple à mettre en œuvre, mais elle peut s'avérer coûteuse pour une trop grande taille du tableau. Une solution consiste à implémenter cette technique sur un matériel spécifique, tel que les cartes graphiques 3D disponibles dans le commerce. Celles-ci sont en effet capables d'afficher en temps-réel les millions de triangles d'une scène 3D, décrite dans le langage OpenGL[3]. Ce dernier permet notamment la manipulation de matrices de transformation, le tracé de triangles dans $\mathbb{R}^3$ ou encore la gestion de différents types de perspectives.

Dans le cas qui nous intéresse, nous avons utilisé OpenGL de la manière suivante :

[3]Voir http ://www.opengl.org

1. Nous avons dans un premier temps défini une transformation qui annule tout effet de perspective (utilisation de GL_ORTHO en OpenGL).

2. Nous avons ensuite positionné cette vue perpendiculairement à l'axe des Z, telle que l'image observée corresponde à la boîte orientée de V_{adm} et que la « caméra » regarde dans la direction des Z croissants. La taille de l'image de la « caméra » est définie en fonction de la résolution désirée pour V_{adm}.

3. Un masque (stencil en OpenGL) a été placé dans le plan Z=0. Sa forme est le complément de V_{adm}, de manière à cacher tout objet dont la projection dans le plan de la « caméra » n'est pas dans V_{adm} (figure 4.18).

4. Nous avons approximé les $\mathcal{V}$-*Obstacles* sous la forme de séries de triangles adjacents (TRI_STRIP en OpenGL) dans le repère de la « caméra » (figure 4.19). Leur coordonnée en Z représente le temps à collision des vitesses ou sa linéarisation (voir page 97).

5. L'image de la « caméra » et la coordonnée en Z la plus proche de chaque pixel (par l'intermédiaire du Z-buffer OpenGL) peuvent alors être lues. Le résultat obtenu est comparable au tableau précédemment obtenu (figure 4.19).

Nous avons pu obtenir avec cette approche des temps de calculs divisés par un facteur 10 à 100. Cependant, si les temps de calculs d'une scène 3D, par une carte graphique, sont très rapides, en revanche les transferts de données entre le PC et la carte peuvent être un problème, en particulier lors de la lecture du Z-buffer. Ce type de matériel est en effet conçu pour afficher des données et par conséquent envoyer rapidement des données du processeur vers la carte et non l'inverse.

$\mathcal{V}$-*Obstacles* et systèmes embarqués Avec la progression constante de la puissance des processeurs, il est vraisemblable que l'utilisation d'une carte graphique ne soit rapidement plus justifiée, en particulier à cause des taux de transfert de données limités. Leur intérêt pour nous réside cependant surtout dans la démonstration qu'une implémentation matérielle des $\mathcal{V}$-*Obstacles* est envisageable à moindre coût, que ce soit sous la forme d'une carte dédiée, à l'instar des cartes graphiques, ou d'une librairie optimisée pour systèmes embarqués.

4.5.4 Traitement des singularités

4.5.4.1 Approximation du calcul de $VO(t \to t_0)$

Utiliser uniquement les points de tangence pour calculer une approximation polygonale d'un $\mathcal{V}$-*Obstacle* n'est pas suffisant lorsque les tangentes à $\mathcal{CB}_\circ$ forment un angle quasiment plat entre elles, alors que $t \to t_0$. En effet, une partie du $\mathcal{V}$-*Obstacle* n'est pas représentée (voir figure 4.21). Cela est du au fait que les vitesses du $\mathcal{V}$-*Obstacle* situées à l'infini ne peuvent pas être représentées pour cause de dépassement arithmétique.

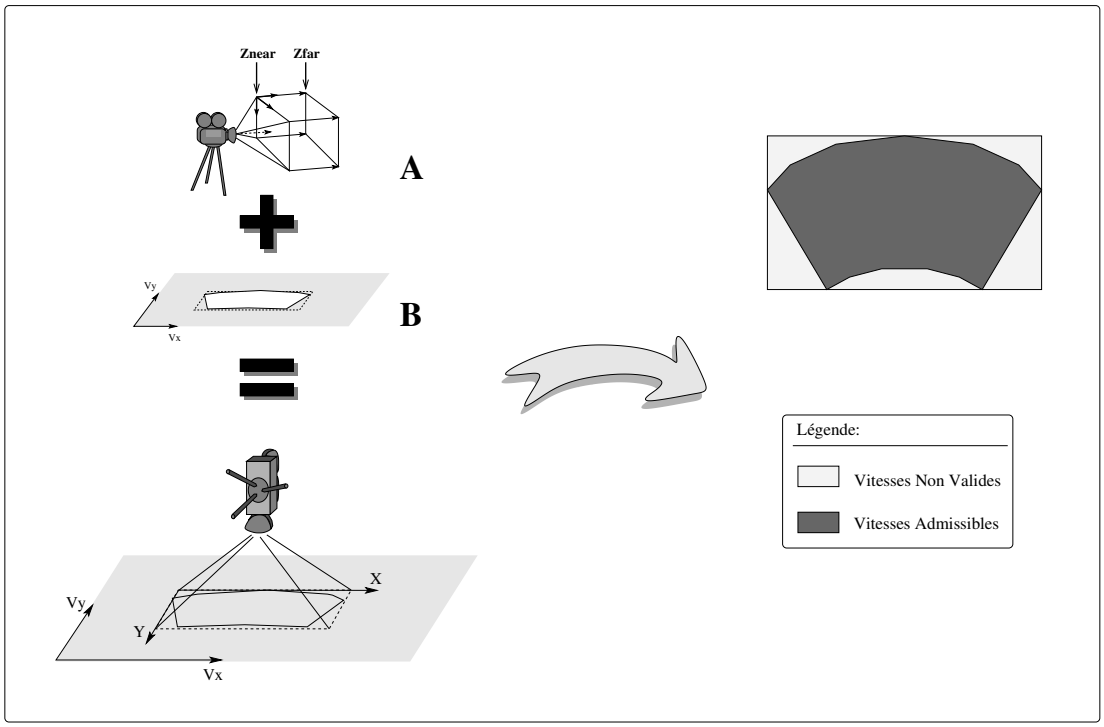

FIG. 4.18 – *Utilisation d'OpenGL : masque pour V_{poss}* Une fenêtre OpenGL est crée telle que ses dimensions en pixels correspondent à la résolution voulue dans l'espace des vitesses (entre 16×16 et 128×128 dans nos expérimentations). Les effets de perspective sont annulés par une projection orthogonale (GL_ORTHO). La fenêtre est placée de manière à coïncider avec la boîte englobante orientée des vitesses admissibles. Perpendiculairement à l'axe de la « caméra » (A), nous plaçons le masque pour cacher les vitesses non admissibles (B).

Pour résoudre ce problème, nous avons recherché le temps t minimal pour lequel le disque $VO(t)$ peut encore être dessiné complètement, sans générer de dépassement arithmétique (figure 4.22).

En notation complexe (chaque point représente un vecteur), nous savons que :

$$c_{VO}(t) = c_A(t_0) + \frac{(c_B(t) - c_A(t_0))}{t} \qquad \text{donc} \qquad \|c_{VO}(t)\| \leq \|c_A(t_0)\| + \frac{\|(c_B(t) - c_A(t_0))\|}{t}$$

Nous notons ρ_{max} la valeur maximale que peut prendre un entier en valeur absolue. $VO(t)$ pourra être tracé complètement si ses coordonnées restent toutes deux inférieures à ρ_{max} en valeur absolue. Par conséquent, nous devons vérifier que :

$$(\|c_{VO}(t)\| + 2 \cdot r_{VO}(t)) < \rho_{max} \qquad \text{avec} \qquad r_{VO}(t) = \frac{r_B}{t}$$

D'où la condition suivante sur t :

$$t > \frac{(\|(c_B(t) - c_A(t_0))\| + 2 \cdot r_B)}{(\rho_{max} - \|cAtt_0\|)}$$

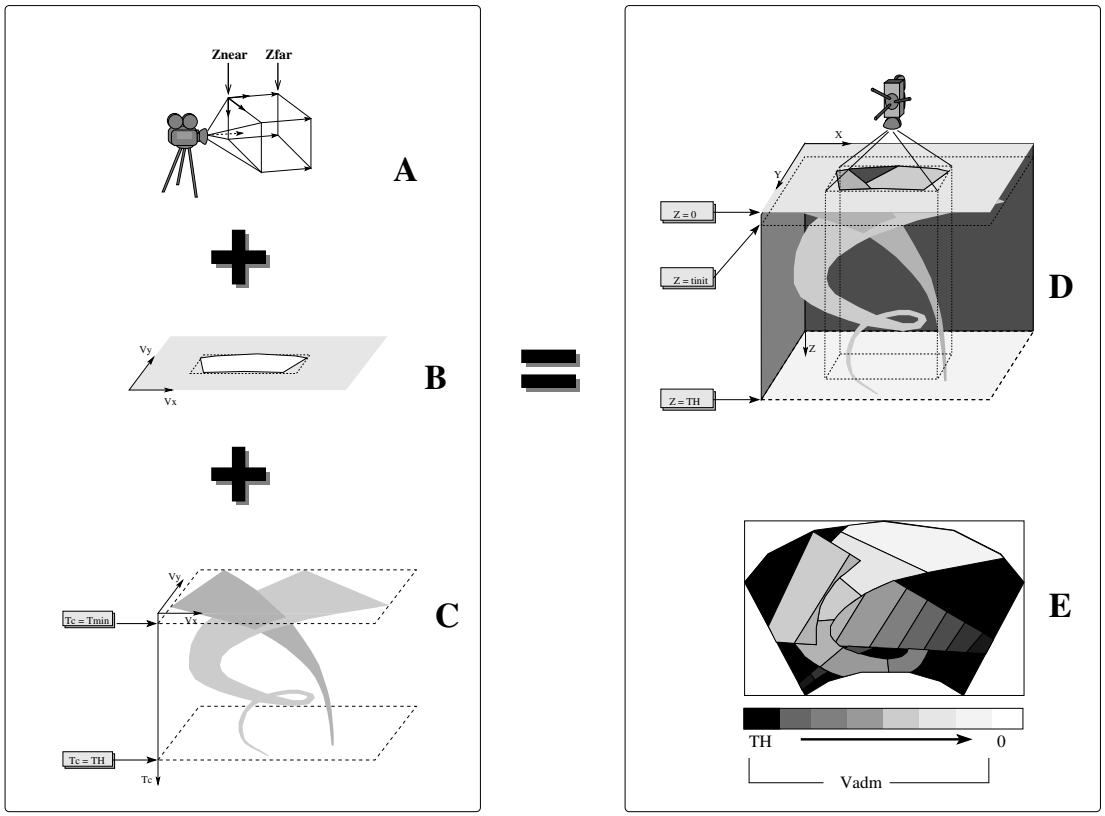

FIG. 4.19 – *Utilisation d'OpenGL : récupération des temps à collision* Les NLVO sont approximés par des triangles (C) puis affichés derrière le masque (D). Ainsi, pour chaque pixel de l'image créée (E), la distance à la caméra reflète le temps à collision de la vitesse associée. Plus la distance est faible et plus les vitesses entraînent des collisions dans un futur proche. Une distance nulle signifie une vitesse non admissible.

La valeur minimale de t est alors utilisée pour approximer $VO(t_0)$. Le rapport très important qui existe entre la dimension de V_{adm} et celle du $VO(t_0)$ approché permet d'obtenir le résultat escompté.

4.5.4.2 Approximation du calcul de $VO(t)$ pour $c_A(t_0) \in \mathcal{CB}_\circ$

Lorsque $c_A(t_0) \in \mathcal{CB}_\circ$, nous avons montré au chapitre 2 que les $VO(t)$ sont inclus les uns dans les autres, et plus précisément que le $VO(t_{in})$ contient tous les autres. Pour rappel, t_{in} représente toute valeur de t telle que (avec $\epsilon_t \to 0$) :

$$\begin{cases} c_A(t_0) \notin \mathcal{CB}_\circ & \text{à } t = t_{in} - \epsilon_t \\ c_A(t_0) \in \mathcal{CB}_\circ & \text{à } t = t_{in} + \epsilon_t \end{cases}$$

Ainsi, d'un point de vue algorithmique et en raison de l'ordre « d'affichage » des triangles, tracer $VO(t_{in})$ est suffisant pour représenter le $\mathcal{V}$-*Obstacle* tant que $c_A(t_0) \in$

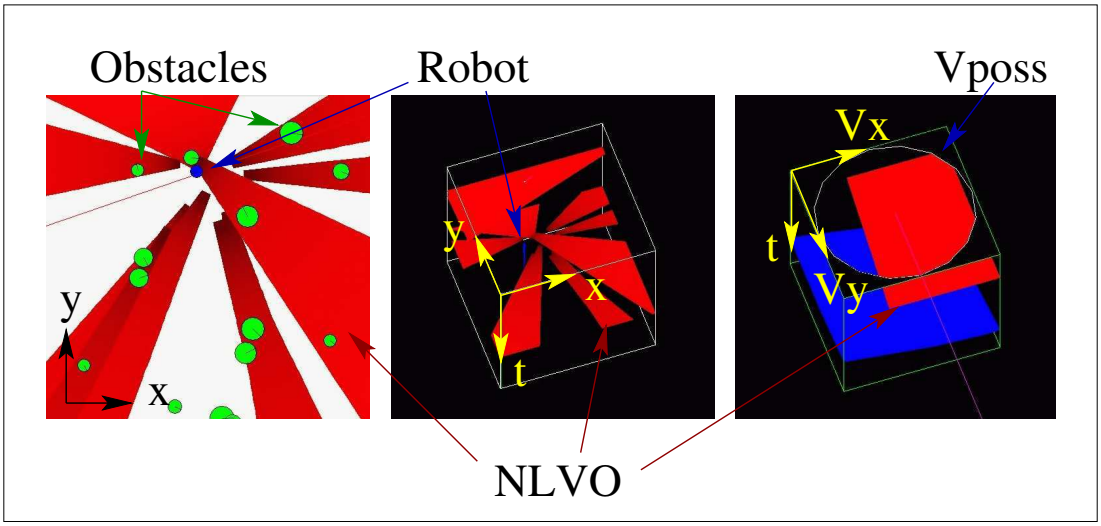

FIG. 4.20 – *Utilisation d'OpenGL : captures d'écran* De gauche à droite, la figure montre un robot se déplaçant parmi des obstacles mobiles et leurs $NLVO$ associés dans $\mathcal{V}$, puis la même scène représentée dans $\mathcal{V} \times \mathcal{T}$, et enfin, une vue de la boîte englobante de V_{poss} et des $NLVO$ dans $\mathcal{V} \times \mathcal{T}$. Ces vues, et notamment la dernière, ont été inclinées pour mieux voir la coordonnée Z du masque par rapport aux $NLVO$. Dans le simulateur, elles sont perpendiculaire à l'axe des Z.

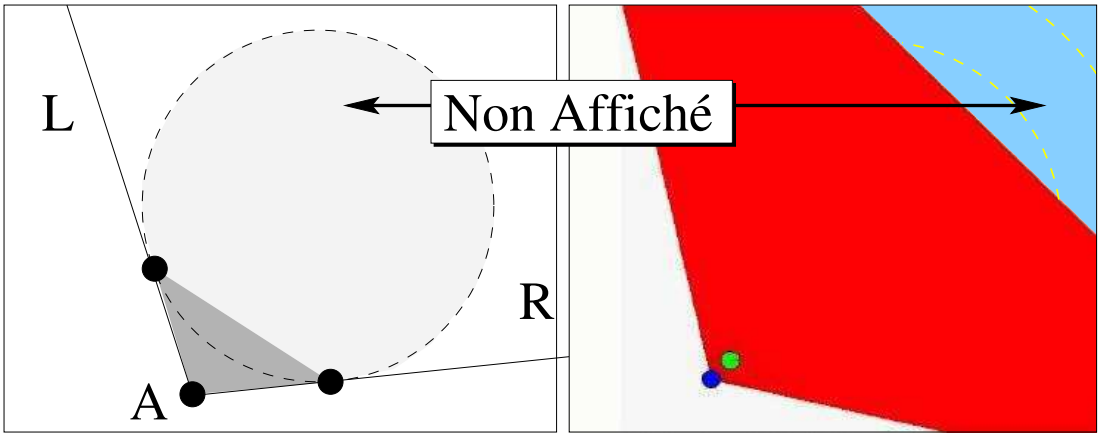

FIG. 4.21 – *Erreur d'approximation de* $VO(t_0)$ Si nous n'utilisons que l'expression analytique des bords du $\mathcal{V}$-*Obstacle* pour le tracer, alors un problème peut survenir pour $t = t_0$: Lorsque l'obstacle et le robot sont proches, l'angle formé par les bords du $\mathcal{V}$-*Obstacle* est très ouvert et le fait de relier les points de tangence n'est pas suffisant. Une partie importante de $VO(t)$ n'est pas « affichée » dans ce cas. La figure de gauche illustre cette idée tandis que celle de droite montre un exemple obtenu en simulation.

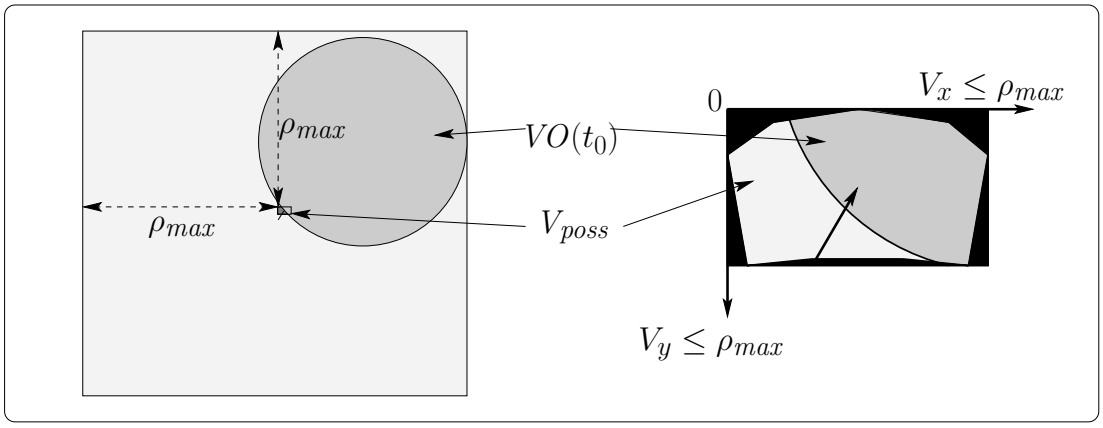

FIG. 4.22 – *Approximation de* $VO(t_0)$ Le repère est centré sur le coin haut gauche de la boîte englobante de V_{poss}. Dans ce repère, les coordonnées de tout point affichable sont inférieures à ρ_{max} en valeur absolue, la limite arithmétique pour les réels. Cela permet de calculer quel peut être le t le plus petit tel que $VO(t)$ soit affichable en totalité (c'est-à-dire sans générer de dépassement arithmétique lors du calcul de ses coordonnées). Ce $VO(t)$ est utilisé comme approximation de $VO(t_0)$.

$\mathcal{CB}_\circ$. Le tracé des $VO(t)$ est alors interrompu et reprend normalement lorsque les valeurs de t sont à nouveau telles que $c_A(t_0) \notin \mathcal{CB}_\circ$.

4.5.5 Traitement des erreurs d'approximation et incertitudes

Les erreurs sont inévitables en raison de l'approximation de la forme du robot et des obstacles, mais également en raison des discrétisations et des approximations polygonales utilisées pour les calculs. À cela peuvent s'ajouter des incertitudes sur les trajectoires des obstacles, ou encore sur le déplacement qu'effectuera réellement le robot. Pour tous ces cas, nous avons envisagé un traitement unique et simple basé sur les $\mathcal{V}$-*Obstacles*.

L'idée consiste à faire varier le rayon du robot et/ou des obstacles. Un simple grossissement constant suffit pour supprimer la majorité des cas d'erreurs dues aux discrétisations et aux approximations. Pour traiter les erreurs sur la trajectoire des obstacles, ce grossissement peut être fonction du temps ou de la distance parcourue par les obstacles. Dans ce cas, le $\mathcal{V}$-*Obstacle* n'est plus nécessairement un cône dont la pointe est en $c_A(t_0)+\overrightarrow{v_B}$ $(t \to \infty)$ et les points de tangence ne correspondent plus aux bords. L'approximation du $\mathcal{V}$-*Obstacle* résultant à partir des points de tangences habituels n'est pas conservative (voir figure 4.23). Ceci n'est cependant pas gênant ici puisque prendre les points de tangence comme bords signifie considérer un rayon virtuel du robot à l'instant t plus petit qu'il n'est mais toujours supérieur à celui de l'instant $t - dt$. La sécurité du robot n'est donc pas remise en question.

Grossir le robot ou les obstacles à l'infini est une approche simple mais il semble normal de se poser la question de son impact sur V_{free}. Pour rappel, trop grossir les

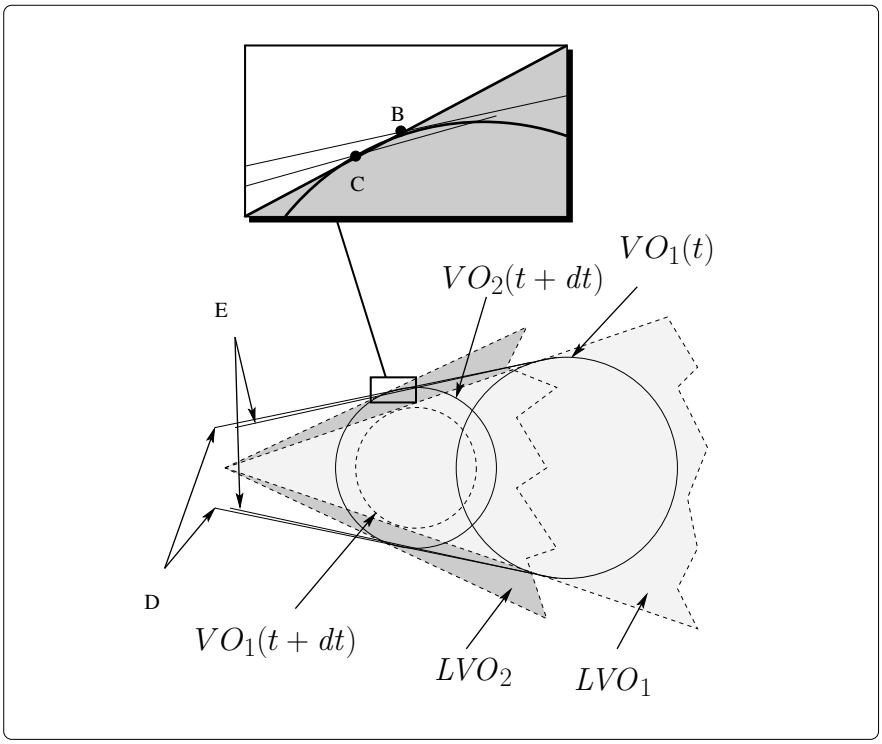

FIG. 4.23 – *Erreur d'approximation pour r_B variable* Afin de faciliter la compréhension du lecteur, nous nous plaçons dans le cas d'un LVO. Dans le cas nominal (rayon constant), deux $VO(t)$ consécutifs ($VO_1(t)$ et $VO_1(t+dt)$) admettent pour tangentes les droites E passant par les points de tangence. Il s'agit des bord de LVO_1. Lorsque le rayon de $\mathcal{A}$ ou de $\mathcal{B}$ augmente au cours du temps, son $VO(t)$ à l'instant t ($VO_2(t+dt)$) est plus grand que dans le cas nominal ($VO_1(t+dt)$). Les bords du VO résultant peuvent être approximés localement par les tangentes D entre $VO_1(t)$ et $VO_2(t+dt)$. Cette construction implique que les droites E sont situées entre les droites D, et donc que l'approximation du VO à l'aide des points de tangence n'est pas conservative. En revanche, cette approximation englobe LVO_1.

$\mathcal{V}$-Obstacles ne signifie pas systématiquement supprimer des solutions, mais seulement les déconseiller en réduisant leur temps à collision. Par conséquent, l'ensemble des solutions n'est pas réduit. Seules les priorités entre celles-ci sont modifiées.

4.5.6 Optimisations des structures de données

Nous pourrions citer plusieurs techniques courantes d'optimisation des structures de données, indépendantes des *$\mathcal{V}$-Obstacles*. Nous avons par exemple fait appel à des tables précalculées des fonctions trigonométriques, ou à des bibliothèques de calcul avec un nombre de décimales fixe. Les deux citées maintenant, bien que tout aussi simples, sont particulièrement intéressantes pour les *$\mathcal{V}$-Obstacles*.

4.5.6.1 Élimination des obstacles trop éloignés du robot

Quelque soit la méthode employée, modéliser un environnement dynamique en un temps raisonnable n'est possible que si le nombre d'obstacles à prendre en compte est raisonnable. Afin de réduire ce dernier, nous avons défini une procédure permettant de ne pas considérer les obstacles jugés inoffensifs. Il s'agit des obstacles dont la distance au robot est telle que le robot et l'obstacle ne peuvent matériellement pas entrer en collision sur l'intervalle de temps $[t_0,\ TH]$, compte-tenu des caractéristiques de chacun.

Le principe consiste à estimer dans un premier temps la vitesse maximale $vmax_B$ qu'un obstacle peut atteindre. Cet obstacle parcourra donc une distance maximale de $(TH - t_0) \times vmax_B$ mètres sur l'intervalle de temps $[t_0,\ TH]$. De son côté, la distance maximale que peut parcourir le robot sur cet intervalle est de $(TH - t_0) \times vmax_A$. Cela nous permet d'écarter tout obstacle situé à une distance supérieure à $(TH - t_0) \times (vmax_A + vmax_B)$.

$$\|c_{B_i} - c_A\| > (TH - t_0) \cdot (vmax_A + vmax_B) \Longleftrightarrow VO_i = \emptyset$$

Nous avons utilisé cette méthode lors de nos tests en simulation afin de traiter de vastes environnements non-structurés.

4.5.6.2 Implémentation de l'arbre de recherche avec un arbre rouge-noir

Toute méthode de planification est basée sur l'utilisation d'un arbre de recherche. Généralement, ces méthodes sont utilisées hors-ligne, et leur temps de réponse n'est pas critique. Dans ces conditions, une liste linéaire chaînée ordonnée suffit pour implémenter l'arbre, ce qui signifie des insertions/suppressions de nœud en $\mathcal{O}(n)$. Dans le planificateur présenté au chapitre 3, nous avons des contraintes de temps-réel fortes. Nous avons donc préféré utiliser un arbre rouge-noir ([Cormen et al., 90]) qui permet de conserver les nœuds de l'arbre triés, et d'en insérer/supprimer avec une complexité en $\mathcal{O}(log\, n)$.

Un arbre rouge-noir est un arbre binaire équilibré. Il tire son nom de la méthode d'équilibrage utilisée, qui consiste à attribuer une couleur (rouge ou noire) à chaque nœud.

4.6 Amélioration du suivi de trajectoire

Faire suivre une trajectoire par un robot est un problème généralement rencontré dans des ouvrages traitant du contrôle, et non de la navigation. La nature des robots que nous désirons utiliser pour expérimenter les $\mathcal{V}$-*Obstacles*, les Cycab (figure 4.24), nous oblige néanmoins à y faire référence. En effet, bien qu'il soit possible de prendre en compte des erreurs lors du calcul d'un déplacement, celles-ci peuvent tout de même obliger le robot à replanifier en permanence sa trajectoire si elles sont importantes. Or, c'est le cas des erreurs de suivi observée sur le Cycab, et qui sont dues essentiellement à des glissements inévitables des roues par rapport au sol (voir annexe C pour plus de détails sur la conception mécanique du Cycab et les effets sur les glissements).

FIG. 4.24 – *Cycab de l'INRIA Rhône-Alpes.* Le Cycab est un véhicule électrique équipé de capteurs divers en vue d'expérimentations robotiques. Il est présenté à l'annexe C.

4.6.1 Problématique

Le problème du suivi de trajectoire est un problème de contrôle classique, consistant à amener un système dans un état désiré. Nous avons vu à la section 1.3.2 qu'il existe différentes approches, à savoir :

Les approches de l'automatique Ce sont les plus fiables mais elles nécessitent un travail de modélisation difficile. Elles acceptent mal les changements de modèle et ne peuvent s'exécuter que sur le système pour lequel elles ont été définies. Elles s'appuient sur l'utilisation conjointe d'un modèle analytique simplifié du robot et d'une loi de commande dont le but est de compenser les erreurs du modèle.

Les approches de l'IA Elles sont plus génériques, plus robustes aux changements et ne nécessitent qu'une étude extérieure du système, basée sur une observation des causes à effets. Mais leur fiabilité n'est pas garantie (notamment en cas d'apprentissage erroné ou insuffisant)

Les études réalisées sur le Cycab au sein du projet *Sharp/e-Motion* (voir annexe C), ont confirmé ces observations générales. Nous avons donc cherché une approche à mi-chemin entre celles-ci, permettant de tirer parti à la fois des capacités d'apprentissage et des facilités de mise en place des techniques d'I.A., et à la fois des propriétés de stabilité des modèles issus de l'automatique.

4.6.2 Approche proposée

Nous avons suivi une approche classique qui consiste à définir un modèle simplifié du robot et de l'utiliser conjointement avec une loi de commande de suivi de trajectoire. Néanmoins, afin d'améliorer les performances du système (réduire les écarts de suivi), nous nous sommes intéressés au moyens de réduire les écarts entre le modèle utilisé et le robot réel.

Il a été démontré que des réseaux de neurones artificiels (RNA) et en particulier ceux à fonctions de base radiales (RBF) sont particulièrement bien adaptés pour modéliser une fonction non-linéaire telle que le modèle cinématique inverse du Cycab. Nous renvoyons le lecteur à l'annexe D et aux travaux de Gauthier ([Gauthier, 99]) pour plus d'informations sur le sujet.

Comme pour tout RNA, nous notons cependant que les performances des RBF sont très dépendantes du niveau et de la qualité de l'apprentissage. Nous proposons donc d'utiliser ce type de RNA mais de le seconder par un modèle de référence, qui le remplacera en cas d'erreur. Ce dernier pourra être un modèle simplifié, mais devra être suffisant pour assurer un contrôle satisfaisant du Cycab conjointement avec la loi de commande.

Le modèle résultant se compose de trois modules (figure 4.25) :

- un ***modèle cinématique inverse de référence***
- un ***modèle cinématique inverse actualisé*** en ligne
- un module de basculement automatique entre les deux modèles précédents que nous appellerons l'***arbitre*** ou le ***sélecteur***.

4.6.2.1 Modèle cinématique inverse de référence

Le module de référence permet de modéliser le robot avec assez de réalisme pour en assurer le contrôle dans de bonnes conditions. De tels modèles ont déjà été étudiés au sein du projet *Sharp/e-Motion*, pour les robots Cycab. Celui que nous avons utilisé est présenté à l'annexe C et rappelé ici (figure 4.26) :

$$\begin{cases} \dot{x} &= v_f \cdot cos(\theta + \phi_f) \\ \dot{y} &= v_f \cdot sin(\theta + \phi_f) \\ \dot{\theta} &= \frac{v_f}{L} \cdot \frac{sin(\phi_f - \phi_r)}{cos(\phi_r)} \end{cases}$$

où v_f est la vitesse linéaire de la roue virtuelle avant par rapport au sol, ϕ_f et ϕ_r respectivement l'angle de braquage de la roue virtuelle avant et arrière, L la distance entre les essieux et (x, y, θ) la configuration du véhicule.

4.6.2.2 Modèle cinématique inverse actualisé

Structure du réseau Nous cherchons à définir le modèle inverse du Cycab de manière à pourvoir calculer les commandes qui provoquent un mouvement désiré de ce dernier.

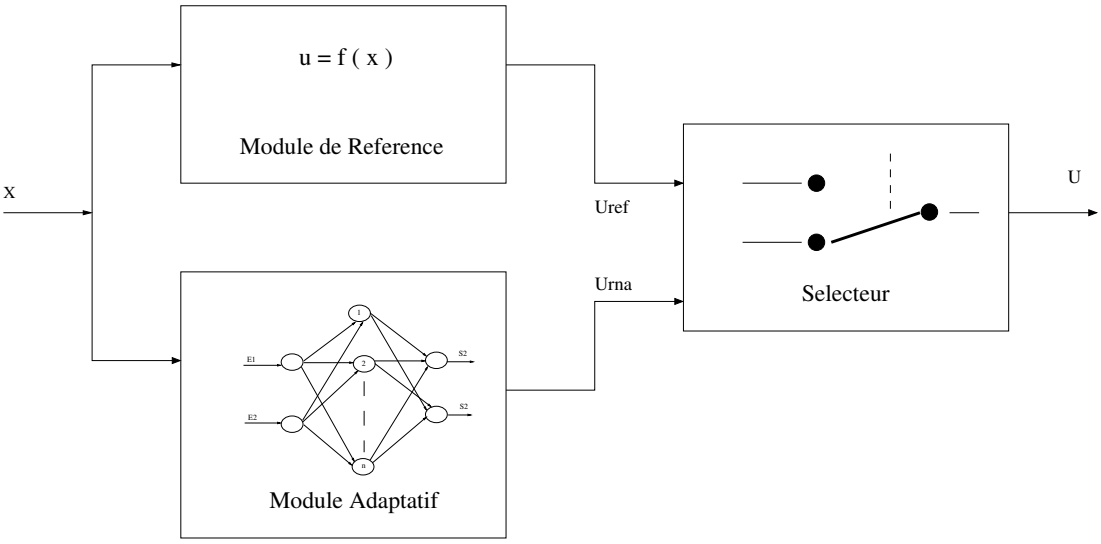

FIG. 4.25 – *Notre approche de la modélisation du Cycab* Le déplacement désiré est présenté à chaque modèle (de référence et actualisé). Chacun calcule une commande correspondante. Le sélecteur choisi parmi celles-ci laquelle sera réellement envoyée au robot.

Huge

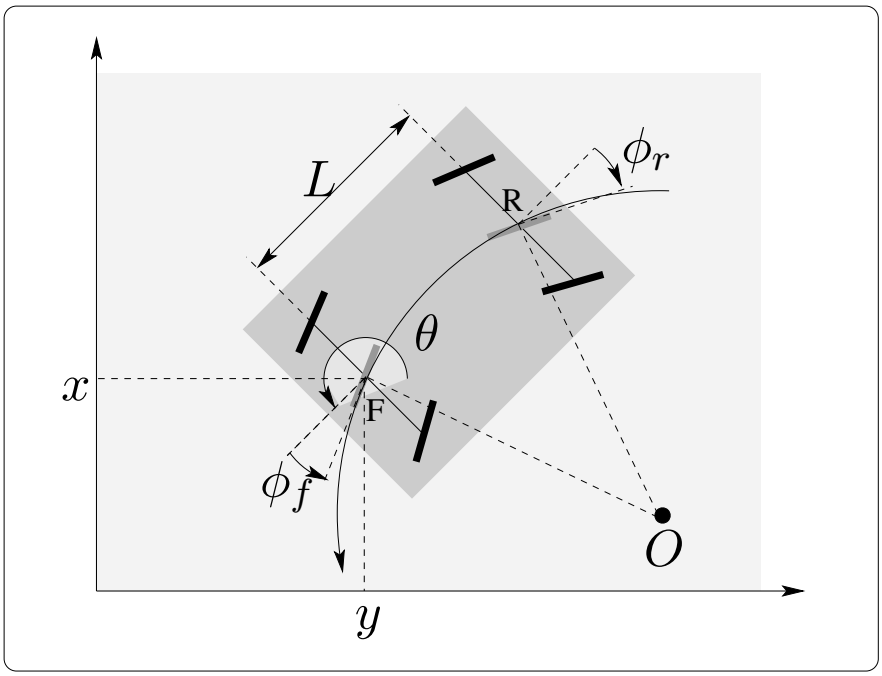

FIG. 4.26 – *Modèle cinématique simplifié du Cycab* Le point de référence est le milieu F de l'essieu avant. La configuration du Cycab est (x,y,θ) où (x,y) est la position de F dans $\mathcal{W}$ et θ est l'orientation du Cycab. les angles ϕ_f et ϕ_r désignent respectivement le braquage de la roue virtuelle avant et arrière. $\overrightarrow{V_f}$ est le vecteur vitesse instantanée du point F. Enfin, L est la distance entre les deux essieux.

Trop de paramètres entrent cependant en jeu pour tous les utiliser en tant qu'entrée de notre RNA. En effet, nous cherchons à minimiser le nombre de ces dernières afin de limiter le nombre d'apprentissages nécessaires qui en découle.

Le modèle appris étant réactualisé en permanence, nous avons mis de côté tout paramètre lié au contexte, tel que la nature du sol. Nous espérons que les effets liés à un changement de contexte de ce type seront pris en compte de manière transparente par le biais d'un nouvel apprentissage le moment venu.

Un point plus subversif concerne la non prise en compte des éléments de dynamique, tels que l'accélération courante (linéaire et angulaire) du robot, qui peuvent avoir une influence non négligeable sur le modèle, en raison notamment de l'inertie. Compte-tenu de la puissance de couple élevée disponible sur chaque roue, ces effets portent essentiellement sur les glissements, eux-mêmes dus essentiellement à la cinématique particulière du Cycab et à sa vitesse. Nous avons donc choisi de ne pas les utiliser comme entrées du RNA.

Finalement, le modèle approximé par le RNA se résume à un modèle cinématique inverse du Cycab. Il comporte donc :

- 2 entrées, correspondant à la vitesse linéaire instantanée du robot et à sa vitesse angulaire dans le repère du véhicule, centré sur son point de référence.
- 5 sorties, représentant les commandes du Cycab (4 vitesses angulaires des roues, et 1 angle de braquage avant, celui arrière étant fonction de l'angle avant).

Le nombre de neurones cachés de la couche intermédiaire et les paramètres des gaussiennes ont été définis de manière à couvrir au mieux l'espace des valeurs d'entrée tout en affectant à chaque unité cachée une zone d'influence suffisamment grande pour ne pas devoir multiplier les apprentissages, et suffisamment petite pour que l'approximation du modèle soit possible. Nous nous sommes inspirés des résultats de [Gauthier, 99] pour définir une structure de base dont nous avons testé des variantes de manière expérimentale. Les meilleurs résultats ont été obtenus pour une discrétisation de l'espace d'entrée de 3×3 à 5×5, et un taux de recouvrement de 67%. La figure 4.27 permet d'apprécier l'influence du nombre de neurones sur la convergence de l'apprentissage.

Pour remarque, nous signalerons le fait que nous avons également envisagé de faire évoluer ce RNA vers une configuration avec 3 entrées (vitesse tangentielle, vitesse normale et vitesse angulaire du robot). Cette solution a été abandonnée pour 2 raisons :

- le nombre de neurones cachés nécessaires, pour les mêmes performances, est supérieur et implique un apprentissage plus long.
- les données nécessaires à l'apprentissage du RNA demandaient des calculs coûteux et imprécis voire erronés (localisation globale du robot).

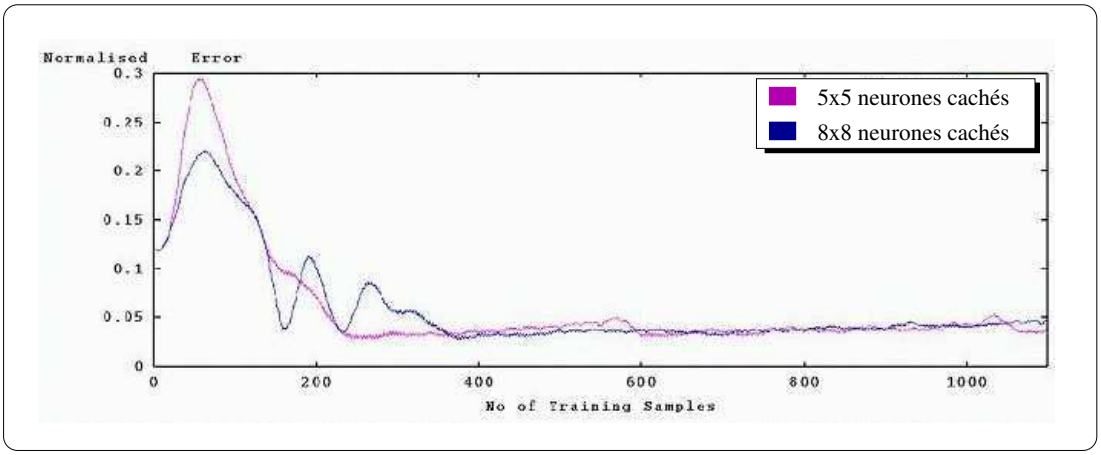

FIG. 4.27 – *Erreurs d'apprentissage obtenues avec différents RNA* Les courbes représentent l'erreur (normalisée) sur la vitesse angulaire moyenne des 4 roues. L'abscisse représente le nombre d'itérations (1 nouveau cas appris à chaque itération). La trajectoire suivie a été obtenue par conduite manuelle du Cycab. Elle correspond approximativement à une sinusoïde suivie à vitesse quasi-constante. Nous pouvons observer que le RNA comportant 25 unités cachées fait une plus grande erreur au début, mais qu'elle converge plus rapidement vers une valeur proche de zéro. Pour des performances comparables, le RNA comportant 64 neurones nécessite davantage de pas d'apprentissage.

Apprentissage du réseau Les trois questions que nous nous sommes posées sont :

1. Quel type d'apprentissage ?
2. Quand apprendre ?
3. Comment apprendre (à partir de quelles données) ?

À la première question nous avons répondu : en mode supervisé, sur les poids et par descente de gradients. Le mode supervisé s'impose de lui-même ici puisque nous désirons limiter le nombre d'apprentissages. De plus, nous allons voir que les données sont disponibles pour permettre un tel apprentissage. Le choix de limiter l'apprentissage à celui des poids vient quant à lui du fait que nous conservons ainsi la même couverture de l'espace d'entrée.

À la seconde question, la réponse la plus triviale, sachant que nous cherchons à disposer d'un modèle actualisé, est : toujours. Néanmoins, un sur-apprentissage du réseau peut l'amener à désapprendre des cas rencontrés que rarement, et ce malgré la propriété de généralisation locale. Ceci vient du fait que même faible, l'influence d'un cas sur l'ensemble des neurones n'est pas nulle, car le niveau d'activation calculé à partir des gaussiennes n'est jamais exactement égal à zéro. Plusieurs chercheurs ont proposé de conserver une base de cas significatifs, qui pourrait être présentée au RNA à chaque nouvel apprentissage. Cette approche n'est pas utilisable ici à cause du temps que demanderait un nouvel apprentissage, et de la difficulté à définir quel sont les cas significatifs.

Nous avons donc choisi une approche plus rapide qui consiste à apprendre tant que l'erreur d'apprentissage constatée est supérieure à un seuil fixé expérimentalement. Nous avons constaté qu'un seuil trop bas entraîne une trop grande spécialisation du RNA qui apprend alors en permanence, y compris des cas erronés, tandis qu'un seuil trop haut ne permet jamais d'atteindre une approximation suffisamment fine. Pour y remédier, nous avons défini deux seuils. L'apprentissage a lieu tant que le seuil bas n'est pas atteint, et il ne reprend que lorsque le seuil haut est dépassé. La zone comprise entre les deux seuils permet des apprentissages ponctuels moins fréquents (figure 4.28). Un exemple d'expérimentation sur le véhicule réel est présenté à la figure 4.33.

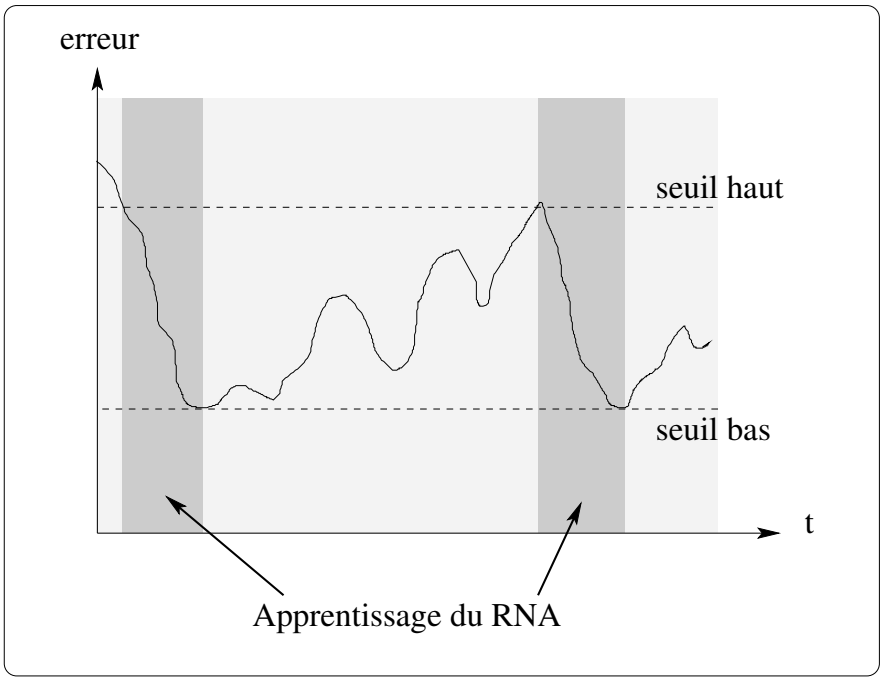

FIG. 4.28 – *Principe d'activation de l'apprentissage*

Enfin, pour de répondre à la dernière question, il convient d'examiner comment notre modèle va s'intégrer dans une application existante, entre en amont un planificateur et une loi de commande, et en aval le robot (figure 4.29).

À chaque instant, la commande envoyée au robot est stockée dans un buffer, lequel permet donc de disposer de la commande précédemment envoyée au robot. Parallèlement à cela, le déplacement effectué par le robot depuis l'application de cette commande peut lui aussi être estimé à partir de différents capteurs de position et des encodeurs des roues. Le RNA dispose donc à la fois d'un déplacement et de la commande qui l'a généré. Ces données sont suffisantes pour entraîner le RNA en mode supervisé.

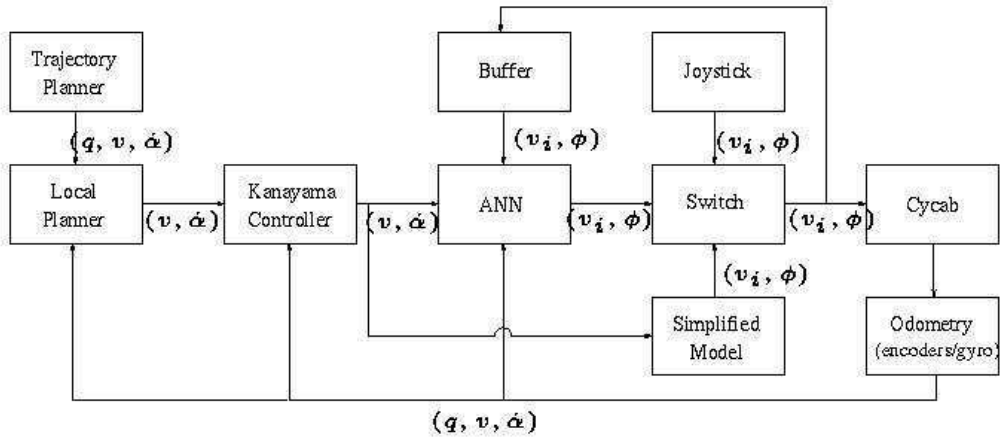

FIG. 4.29 – *Application complète incluant notre modèle actualisé* Un planificateur global fournit une trajectoire de référence (buts intermédiaires) au planificateur local dont le rôle est de calculer le prochain déplacement désiré du robot (à partir d'un but intermédiaire, de la position courante du robot, et éventuellement de la présence d'obstacles). Ce déplacement est ensuite traduit en une commande par le couple loi de commande-modèle. La commande envoyée au Cycab est choisie entre les différents modèle de manière automatique par un sélecteur (ou *switch*). Ce dernier a également la responsabilité de permettre la reprise du contrôle manuel par l'utilisateur à tout instant s'il le souhaite. Les commandes envoyées au Cycab dans ce cas ne sont pas celles des modèles (actualisé ou de référence) mais celle d'un module simple de contrôle par Joystick.

4.6.2.3 Arbitre

Le sélecteur joue un rôle de gardien. Il détecte les cas où le RNA n'est pas en mesure de fournir des résultats corrects. Il sélectionne alors les résultats du modèle de référence. La difficulté réside dans l'identification d'un cas d'erreur.

En fonction des données dont nous disposons, nous avons défini 3 critères permettant de juger du niveau d'apprentissage du RNA :

- le fait qu'il soit ou non en train d'apprendre et sa dernière erreur d'apprentissage (erreur en pourcentage entre la valeur désirée et celle proposée)
- l'écart entre les résultats proposés et ceux fournis par le modèle de référence
- l'évolution des informations précédentes au cours du temps

Nous en avons déduit les règles suivantes :

1. Si l'écart entre les valeurs fournies par le RNA et celles fournies par le modèle de référence est trop grand, le système sélectionne le modèle de référence mais signale une erreur : Cela dénote que le modèle de référence doit être amélioré ou que le RNA doit être initialisé hors-ligne sur le modèle de référence (voir plus loin pour ces techniques).

2. Sinon, si le RNA n'apprend pas alors il est prioritaire

3. Sinon, si son erreur d'apprentissage est inférieure à un seuil 10% sur les 10 derniers apprentissages, il est prioritaire

4. Sinon, si sa dernière erreur d'apprentissage est inférieure à 10% et que cet erreur diminue en moyenne sur les 10 derniers apprentissages, il est prioritaire

5. Sinon, le modèle de référence est sélectionné.

Initialement, le sélecteur envisagé ne comportait qu'une seule règle basée sur l'écart avec le modèle de référence. Cela donnait de bon résultat en simulation, mais suite à des expérimentations sur robot réel (figure 4.31), nous nous sommes aperçu que le modèle de référence était sélectionné quasiment en permanence après quelques minutes d'utilisation seulement. La cause de cela est l'importance des erreurs qui existe entre le modèle de référence (simplifié) et la réalité (et par conséquent le RNA qui l'approxime). Nous avons donc fixé l'écart autorisé à une valeur très importante (60%) et n'avons utilisé ce critère que pour signaler des erreurs graves. Nous allons voir maintenant comment ce cas peut être évité.

4.6.3 Réduire les écarts entre modèle de référence et RNA

Partant de l'hypothèse que le RNA approxime correctement la réalité, la présence de grands écarts entre ce dernier et le modèle de référence ne peut provenir que d'une trop grande simplification de ce dernier.

4.6.3.1 Mise-à-jour du modèle de référence

Il est évident que le modèle de référence, bien que simplifié, doit rester proche de la réalité. Une solution à cela est de le remplacer par une copie du modèle actualisé, dont l'apprentissage aura préalablement été validé et figé.

Actuellement, la validation d'un modèle actualisé en vue de l'utiliser comme modèle de référence, se fait de manière simple et demanderait une étude plus approfondie :
Nous partons de l'idée qu'un changement faible d'une des entrées provoque un changement faible de chaque sortie. L'ensemble des combinaisons de valeurs d'entrées est présenté au RNA, tandis que la variation de la sortie est mesurée :

- quantitativement (elle doit être inférieure à un seuil donné)
- et qualitativement (nous utilisons nos connaissances pour vérifier que la variation d'une entrée dans un sens implique bien une variation des sorties dans le sens escompté).

Les expérimentations ont montré une utilisation du RNA dans quasiment 90% du temps avec un modèle de référence « appris », tandis que le modèle simplifié d'origine laissait moins de 30% du temps au RNA en raison des écarts importants (>60%, figure 4.30).

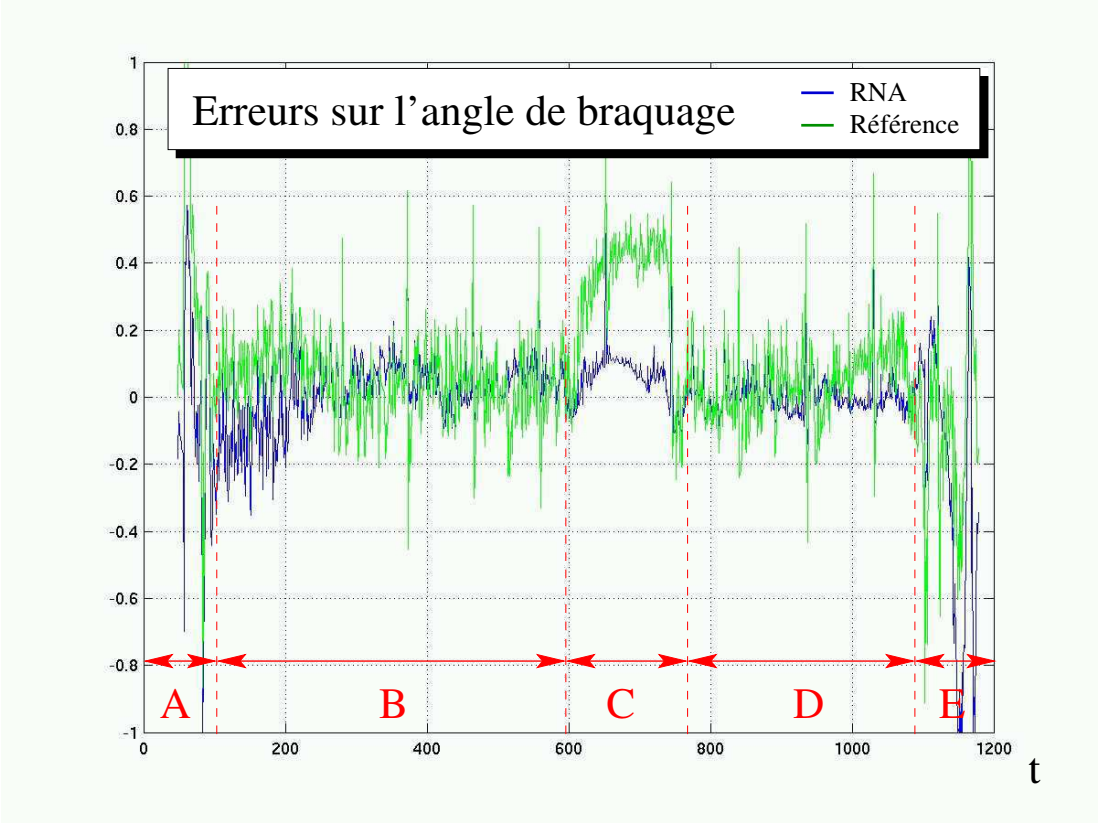

FIG. 4.30 – *Erreur sur le braquage* La figure montre les erreurs sur les valeurs de l'angle de braquage. Alors que celles du RNA sont assez faibles en moyenne, celles du modèle analytique peuvent atteindre plus de 60% dans les virages (en C). Les erreurs importantes en A et E s'expliquent par une vitesse très faible du robot à cet instant : l'influence de l'angle de braquage pour ces vitesses n'est pas suffisante pour permettre un apprentissage correct du RNA. Le modèle analytique souffre quant à lui de problèmes numériques pour ces valeurs d'entrée.

4.6.3.2 Initialiser le RNA

Afin d'éviter des écarts trop importants en début d'apprentissage, nous proposons d'initialiser le RNA à partir du modèle de référence. Ainsi, au début, les erreurs sont suffisamment faibles pour que la première règle de sélection énoncée précédemment ne s'applique pas.

Une autre raison à cela est la difficulté de faire le lien entre le déplacement du Cycab et la vitesse de rotation d'une roue particulière : Une roue peut tourner plus vite que nécessaire sans influer sur la vitesse du Cycab ; elle glissera seulement plus par rapport au sol. Cela conduit le Cycab à apprendre parfois des cas erronés et à ne converger que très lentement vers la solution réelle.

L'apprentissage hors-ligne permet d'initialiser les poids à des valeurs proches de ce qu'elles doivent réellement être et permet ainsi de faciliter l'apprentissage par la suite.

Méthode alternative Une autre approche consisterait à ne faire calculer au RNA qu'une vitesse linéaire (voire deux, une pour les roues gauches et une pour les roues droites) et un angle de braquage. Nous avions d'ailleurs expérimenté cette approche sur véhicule réel en vue d'évaluer l'approche de [Gauthier, 99]. Il est cependant nécessaire dans ce cas de convertir la vitesse linéaire en quatre vitesses angulaires (une pour chaque roue). Or cela se fait via un modèle analytique qui introduit des erreurs supplémentaires (par exemple, le diamètre des roues n'est jamais connu avec précision et peut varier) et surtout réduit l'intérêt du RNA.

4.6.4 Résultats expérimentaux préliminaires

Nos travaux relatifs à l'apprentissage du modèle du Cycab sont les seuls à ne pas être dépendants de la carte de l'environnement. Par conséquent, par rapport aux autres résultats présentés dans ce rapport et qui ont été validés seulement en simulation, ceux-ci ont pu être également expérimentés sur véhicule réel. L'implémentation a été réalisée au moyen d'outils développés par l'INRIA pour l'écriture de contrôleurs temps-réel de robots : Orccad. La figure 4.31 montre une capture d'écran de la première version, développée sous Orccad ([Simon et al., 93]). L'application considérée correspond à celle décrite à la figure 4.29.

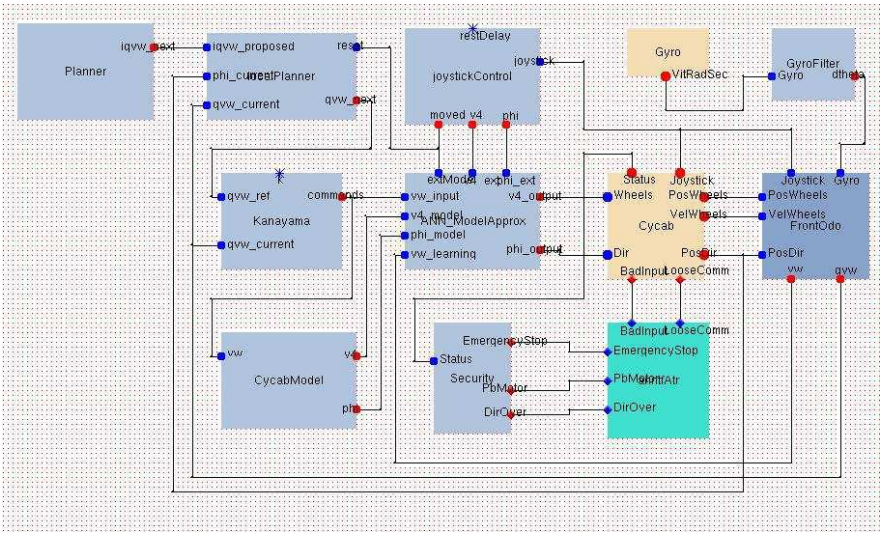

FIG. 4.31 – *Application implémentée avec ORCCAD* L'application présentée à la figure 4.29 a été implémentée dans un premier temps à l'aide du logiciel Orccad ([Simon et al., 93]) développé par l'INRIA, dont le but est la définition de contrôleurs temps-réel pour la robotique. La figure représente une capture d'écran de la tâche principale. L'organisation d'une tâches en modules reliés entre eux par les données qu'ils échangent permet de retrouver le synoptique de la figure 4.29. Pour des raisons pratiques, une nouvelle version a été réalisée ensuite en C++ "pur". Les résultats expérimentaux présentés plus loin proviennent de ces deux implémentations.

Avant chaque expérimentation, nous avons la possibilité de réinitialiser le RNA par apprentissage à partir du modèle de référence, ou de conserver les données acquises

lors d'une précédente expérimentation. Dans tous les cas, le RNA a subi au moins une initialisation pour les raisons évoquées précédemment.

Lors de nos tests, la trajectoire à suivre n'est pas calculée par le planificateur mais imposée par l'utilisateur. Dans un premier temps, ce dernier pilote manuellement le Cycab selon ses désirs. La trajectoire suivie est alors estimée à partir des données fournies par le module de localisation du Cycab puis enregistrée. Elle servira de trajectoire de référence.

La procédure de test est en fait un automate qui comprend 5 étapes :

1. conduite manuelle du Cycab à partir d'une configuration initiale prise pour origine. Cette étape sert à enregistrer la trajectoire de référence (apprentissage en ligne désactivé)
2. retour au point d'origine en conduite manuelle (apprentissage en ligne désactivé)
3. suivi automatique de la trajectoire de référence (apprentissage en ligne activé)
4. retour au point d'origine en conduite manuelle (apprentissage en ligne activé)
5. suivi automatique de la trajectoire de référence (apprentissage en ligne activé)

Cette procédure nous a permis de vérifier les points suivants :
- le RNA apprend correctement le modèle du Cycab (figure 4.32)
- le RNA arrête d'apprendre lorsque son erreur est suffisamment faible (figure 4.33)
- le RNA ne désapprend pas en cas de présentation répétée d'un même cas (figure 4.34)

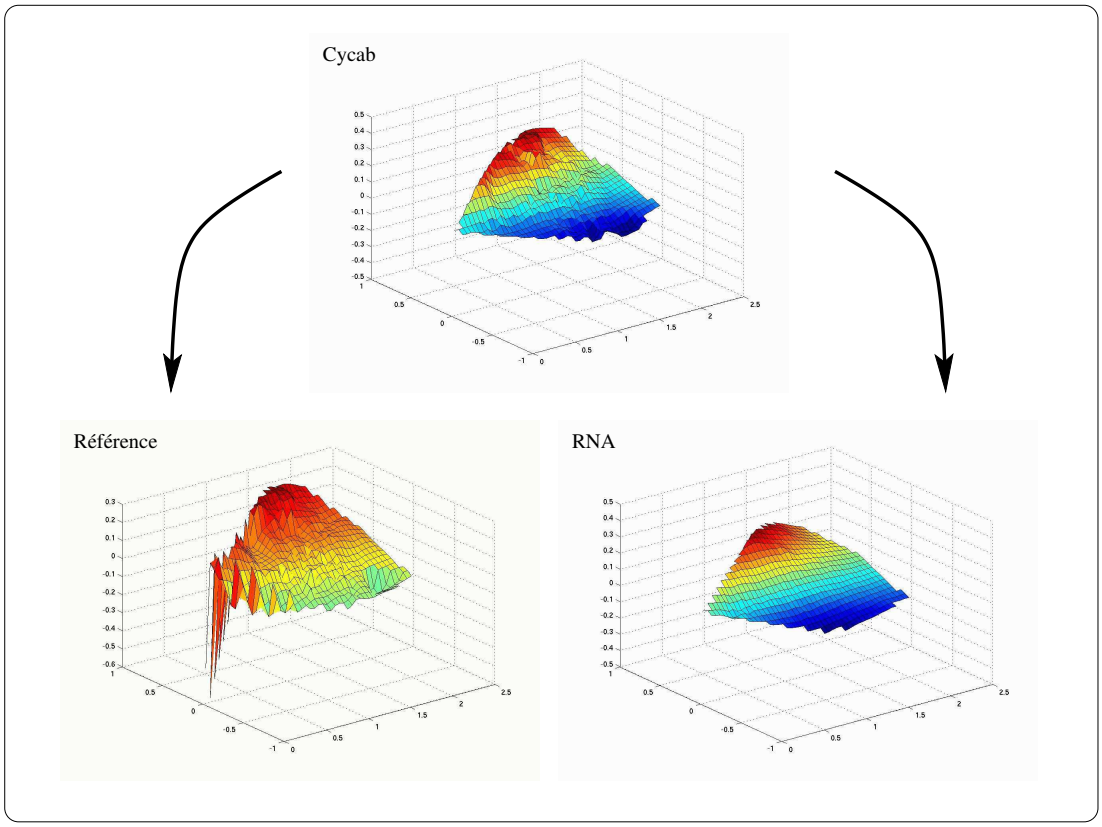

FIG. 4.32 – *Modèle cinématique inverse réel et approximations* La figure en haut au centre est une représentation de la valeur de l'angle de braquage en fonction de la vitesse angulaire (vitesse linéaire fixée). Les figures du bas représentent ses approximations. À gauche, celle obtenue avec le modèle de référence analytique, et à droite celle obtenue avec le RNA entraîné en ligne. Nous pouvons constater que le RNA s'approche davantage de la réalité et qu'il ne présente pas les singularités aux limites que présente le modèle analytique. Ces résultats laisse par ailleurs apparaître une dérive de notre Cycab que le modèle analytique (qui suppose le modèle du Cycab parfaitement symétrique) ne prend pas en compte.

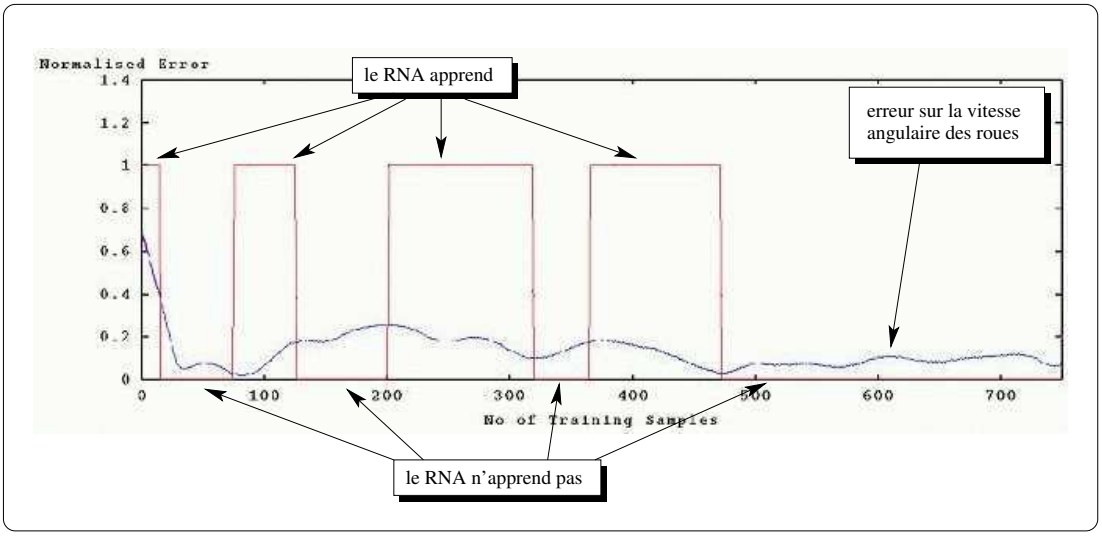

FIG. 4.33 – *Évolution simultanée de l'apprentissage (1=apprentissage, 0=pas d'apprentissage) et de l'erreur sur la vitesse angulaire moyenne des roues.* La figure montre que dans le cas nominal (contexte stable), les apprentissages sont de moins en moins fréquents, alors que l'erreur se stabilise.

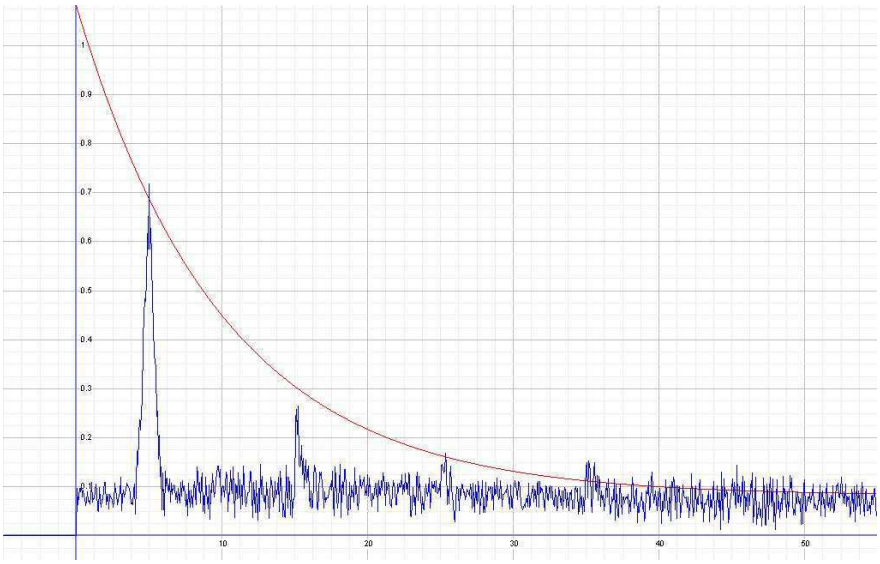

FIG. 4.34 – *Le RNA ne désapprend pas.* Afin de vérifier qu'un RNA entraîné ne désapprend pas les cas déjà appris, nous l'avons forcé à apprendre une valeur erronée de l'angle de braquage, pour une valeur particulière de ses entrées. Nous avons présenté ce cas jusqu'à ce que le RNA l'ait appris avec une erreur inférieure à 1%. Nous avons ensuite utilisé le RNA en situation normale, sur une trajectoire sinusoïdale (afin d'obtenir la valeur d'entrée particulière de manière répétée). La figure montre que l'erreur est restée faible pour toutes les valeurs d'entrées excepté celle pour lequel de RNA a subit l'apprentissage erroné. Néanmoins, nous remarquons également que cette erreur décroit rapidement (Cf. exponentielle superposée à la figure pour comparaison) après quelques présentations de la valeur correcte.

Conclusion et Perspectives générales

Nous nous sommes intéressés, au cours de cette thèse, aux concepts et outils algorithmiques permettant à un robot mobile de se déplacer de manière autonome vers un but, et plus particulièrement nous avons considéré ce problème pour le cas où l'environnement est dynamique et incertain.

Deux tâches sont généralement identifiées pour ce travail : le calcul préalable d'une trajectoire de référence à l'aide de méthodes de planification ; puis l'exécution de cette trajectoire à l'aide d'une méthode réactive qui permet de prendre en compte les petits imprévus et de rattraper les erreurs de suivi.

Dans le cas présent, l'absence de connaissance complète a priori sur l'environnement ne permet pas de garantir la validité de la trajectoire sur un temps indéfini, et celle-ci doit être recalculée régulièrement. Or, la présence d'obstacles mobiles nécessite que le robot reste réactif, et par conséquent que ces calculs soient réalisés dans un temps court et borné. La dimension élevée des espaces manipulés ne le permet pas et seul un déplacement local peut généralement être recherché, sans certitude que le but pourra être atteint.

Nous avons cherché, dans un premier temps, à augmenter la quantité de temps disponible pour les calculs, en proposant un formalisme capable de prendre en compte l'évolution de l'environnement sur une période de temps bornée, et de permettre ainsi le calcul de déplacements valides plus longtemps sur cette période. Nous nous sommes inspirés pour cela du concept de $\mathcal{V}$-*Obstacle*, défini dans l'espace des vitesses linéaires instantanées du robot, et dont la signification est l'ensemble des vitesses qui entraînent une collision future. Dans ce mémoire, nous l'avons étendu pour permettre le calcul rapide du temps à collision associé à chaque vitesse potentiellement applicable au robot, et en avons proposé une expression analytique pour le cas d'un robot et d'obstacles circulaires. Cela nous a permis d'en construire une approximation pour le cas général, selon une approche géométrique simple et efficace, adaptée aux contraintes de temps-réel. Le nouveau formalisme proposé, baptisé $\mathcal{V}$-*Obstacle Non-Linéaire* (*NLVO*), permet le calcul rapide de l'ensemble des vitesses qui entraînent des collisions sur un intervalle de temps borné, ainsi que le calcul des temps à collision qui leur sont associés. Il prend pour paramètres les trajectoires connues ou estimées des obstacles sur cet intervalle.

Dans un second temps, nous avons présenté une méthode d'évitement réactif des obstacles, basée sur ces concepts. L'idée consiste à utiliser les *NLVO* pour estimer un risque de collision associé à chaque déplacement possible du robot, compte-tenu de contraintes cinématiques et dynamiques élémentaires sur ce dernier. Nous nous distinguons des approches existantes par le fait qu'aucune solution n'est écartée par la modélisation de l'environnement, et que celle-ci confère au robot une réelle capacité à anticiper les collisions. Cela rend ses trajectoires plus cohérentes et plus sûres en présence d'obstacles mobiles. Néanmoins, les problèmes rencontrés avec les autres approches de ce type (locales) demeurent, et en particulier l'impossibilité de garantir que le but sera atteint, ni dans combien de temps.

Nous avons alors proposé une méthode de planification de trajectoire globale dans le but d'apporter une solution à ce problème. Les approches classiques de ce type nécessitent la modélisation préalable complète de l'espace libre et la recherche de sa connectivité avant de pouvoir y calculer un déplacement libre. Ceci demande trop de temps pour être utilisable en ligne (i.e. sous contrainte de temps-réel), y compris en utilisant les *NLVO* pour modéliser l'espace libre. Nous avons donc considéré une approche itérative, basée sur une exploration heuristique incrémentale de l'espace de recherche à l'aide de notre méthode locale d'évitement. Le principal avantage de cette approche est qu'elle autorise une interruption de la recherche à tout instant. Parmi l'ensemble des trajectoires déjà explorées, la plus propice à conduire au but est alors retournée. Cette propriété est particulièrement importante puisqu'elle nous permet d'utiliser notre méthode en remplacement de la méthode d'évitement précédente, sans mettre en danger le robot : Dans le meilleur des cas, une trajectoire globale sûre sera calculée, et dans le pire des cas, le robot conservera un comportement réactif. De plus, un système de mise-à-jour partielle de l'arbre a été développé pour éviter une nouvelle exploration complète à chaque prise de décision du robot. Il s'avère particulièrement efficace lorsque l'environnement varie peu par rapport aux hypothèses faites pour le calcul des $\mathcal{V}$-*Obstacles*, et il favorise les trajectoires qui évitent les zones à risques, c'est-à-dire celles où le comportement des obstacles est difficilement prédictible.

Disposer d'une trajectoire sûre ne suffit pourtant pas à garantir la sécurité du robot ; Encore faut-il que ce dernier la suive correctement. Ce problème est particulièrement délicat lorsque l'ensemble des paramètres de contrôle ne sont pas maîtrisés. Nous avons étudié le cas d'un véhicule Cycab, dont les glissements des roues sur le sol sont inévitables, non modélisables et évoluent au cours du temps. De fait, le modèle analytique simplifié utilisé pour le contrôler induit des écarts entre la trajectoire de référence et celle réalisée, malgré l'utilisation d'une loi de commande pour le suivi. Nous nous sommes donc intéressés à la manière d'améliorer ce modèle. Nous avons opté pour l'utilisation d'un réseau de neurones artificiels dont les propriétés de généralisation locale et d'approximateur universel de fonction ont permis de réaliser un apprentissage en ligne, directement à partir des données réelles du robot. Ce type d'approche souffre cependant de performances médiocres en cas de mauvais apprentissage (insuffisant ou erroné). Nous avons donc défini un protocol permettant de détecter ces cas et de remplacer temporairement le réseau par un modèle analytique stable pour maintenir un

contrôle satisfaisant du robot.

Ainsi, l'utilisation conjointe des outils présentés couvre l'ensemble des besoins essentiels d'un robot (à l'exception de la perception) pour se déplacer de manière autonome en présence d'obstacles mobiles.

Tous ces travaux ont été validés en simulation, pour différents types de robots et différents types d'environnements. Nous regrettons cependant que seul l'apprentissage en ligne du modèle ait pu être porté sur robot réel. En effet, les autres méthodes demandent des ressources extérieures non abordées dans ce rapport, dont la perception de l'environnement, qui n'ont pas pu être réunies à temps. Ces expérimentations seront nécessaires pour valider définitivement l'approche des $\mathcal{V}$-*Obstacles* et des fonctions dérivées, cependant, les résultats obtenus sur simulateur pour des environnements artificiels particulièrement hostiles nous rendent optimistes à ce sujet.

D'autre part, nous pensons que les travaux présentés dans ce mémoire ne constituent qu'une introduction aux $\mathcal{V}$-*Obstacles*, dans le but d'en définir les concepts de base et d'en démontrer l'intérêt pour la navigation. Ils devront être poursuivis pour compléter l'exploration des nombreuses possibilités offertes par l'approche. Nous pensons notamment à des représentations différentes telles que le bayésien, ou encore à une étude plus poussée de la connectivité de l'espace des vitesses pour en faire profiter la planification. De plus, l'étude des $\mathcal{V}$-*Obstacles* pour une famille de trajectoires autre que celle présentée ici est souhaitable.

Le frein actuel des $\mathcal{V}$-*Obstacles* en robotique reste la nécessité de posséder une information suffisante sur l'environnement. Ceci est cependant un autre problème qui trouve déjà des solutions par le biais de la vidéosurveillance, pour des cas ponctuels tels que les bâtiments, les parking, ou encore les carrefours, où les $\mathcal{V}$-*Obstacles* pourraient être d'un intérêt certain. Mais d'autres domaines, tels que la réalité virtuelle, disposent de suffisamment d'informations sur l'environnement pour bénéficier d'ores-et-déjà des $\mathcal{V}$-*Obstacles*.

Annexe A

Planification de chemin en environnement statique connu

La planification d'un chemin géométrique en environnement statique consiste en deux étapes consécutives :

- une étape de modélisation de l'espace libre dans $\mathcal{C}$
- une étape de construction d'un chemin dans cet espace

La construction du chemin se résume généralement à un parcours d'arbre et ne présente pas un intérêt particulier pour notre étude. En revanche bien que définies pour des environnements statiques, les méthodes de modélisation permettent d'avoir un aperçu des possibilités de partitionnement d'un espace, et leurs propriétés. Nous n'en citons ici que les principales, classées de la manière suivante (voir [Latombe, 90] pour plus d'informations) :

- Les « Roadmaps »
  - Graphe de visibilité
  - Diagramme de Voronoï
  - Méthode des autoroutes
  - Méthode des silhouètes
- Les méthodes de décomposition cellulaire
  - Exactes
  - Approchées
- Les champs de potentiels

A.1 « Roadmaps »

Le terme « roadmap » désigne une carte des routes que le robot peut emprunter pour se rendre d'un endroit à l'autre de son espace de travail. Cela permet de ne considérer qu'un nombre limité de possibilités de chemins entre deux points. L'information est représentée sous forme de graphe. Les nœuds sont les « carrefours » et les branches les « routes » libres entre ces derniers. Chaque carrefour est associé à une région (un sous-espace de $\mathcal{C}_{free}$) et réciproquement. Étant donnée une position de départ et une d'arrivée, le problème consiste à trouver les deux carrefours qui leur sont associés, puis à rechercher la route qui les relie.

A.1.1 Graphe de visibilité

La première méthode apparue porte sur des espaces de configuration de dimension 2 seulement. Cela signifie un robot mobile circulaire, ou un robot ayant une orientation fixe ([Lozano-Perez, 81]). L'algorithme consiste à construire un graphe reliant deux-à-deux tous les sommets des obstacles polygonaux qui peuvent être connectés entre-eux par une ligne ne traversant pas un autre obstacle (figure A.1). Le graphe obtenu porte le nom de graphe de visibilité ([Nilsson, 69]) du fait qu'il permet de connaître les sommets visibles depuis un sommet donné.

Dans le cas d'un robot simple comme un polyèdre se déplaçant uniquement en translation, Lozano-Pérez a montré que cette construction pouvait se faire par la différence de Minkowski avec une complexité de $O(mnlog(mn))$, avec m et n les nombres de sommets respectifs du robot et des obstacles.

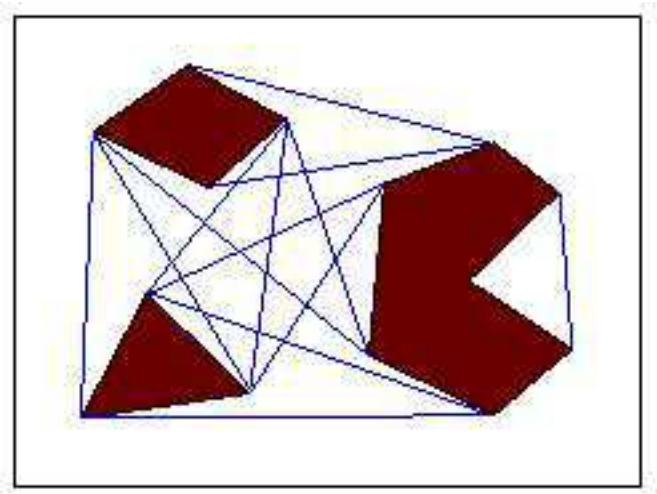

FIG. A.1 – Diagramme de Visibilité

Cette approche présente cependant l'inconvénient de modéliser des déplacements frôlant les obstacles. De tels déplacements, libres en théorie, peuvent poser problème en

pratique à cause des erreurs d'exécution (de suivi) ou des incertitudes sur la position des obstacles et du robot. Par mesure de sécurité, il est donc prudent de « grossir » les obstacles, au risque de masquer des solutions.

A.1.2 Méthode des autoroutes

La méthode des autoroutes ([Brooks, 83]) a une approche opposée de la précédente dans la mesure où elle consiste à passer le plus loin possible des obstacles. Pour ce faire, un couloir correspondant à une zone libre, est calculé pour chaque couple de polygones. L'axe de ce couloir ainsi que sa largeur sont conservés. Les intersections des axes correspondent à des changements de direction éventuels du robot, et matérialisent la connectivité de deux couloirs (comme des carrefours). Ces points constituent les nœuds d'un graphe dont les arêtes représentent les couloirs et donc les déplacements autorisés. (figure A.2)

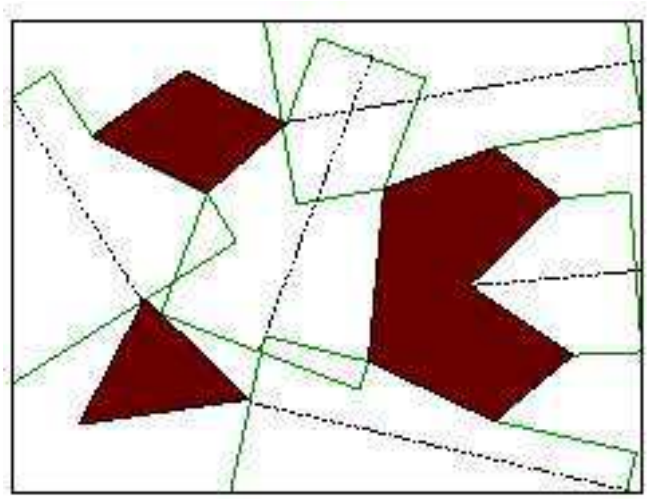

FIG. A.2 – Autoroute

Cette approche permet au robot de naviguer le plus loin possible des obstacles. Elle serait donc une bonne candidate pour limiter les conséquences des incertitudes. En contre-partie, cela conduit à des trajectoires plus longues et pas toujours très naturelles.

A.1.3 Méthode des silhouètes

La méthode des silhouètes ([Canny et al., 88]) s'applique à tout espace de travail pouvant être représenté par un ensemble mathématique semi-algébrique. Il s'agit du seul algorithme de planification de chemin ayant une complexité en temps exponentielle dans la dimension de $\mathcal{C}$. Le principe de la méthode consiste à balayer l'espace des configurations avec un hyperplan de dimension m-1, pour créer une « silhouète » de $\mathcal{C}$.

Ce principe est appliqué récursivement puis les « silhouètes » obtenues sont connectées entre elles dans un graphe pour représenter la connectivité de l'espace libre.

A.1.4 Graphe de Voronoï

Selon le même principe que la méthode des autoroutes, le diagramme de Voronoï, (figure A.3) ([O'Dunlaing et Yap, 82]) consiste à s'éloigner des obstacles, et en l'occurence, en trouvant la courbe unidimensionnelle correspondant aux positions équidistantes des obstacles. En dimension 2, Yap a montré que le calcul de ce diagramme peut se faire en $O(nlog(n))$ où n est le nombre total de sommets des obstacles. La frontière (à égale distance d'au moins deux objets) des cellules obtenues constitue un squelette, matérialisant les points de passage privilégiés pour le robot.

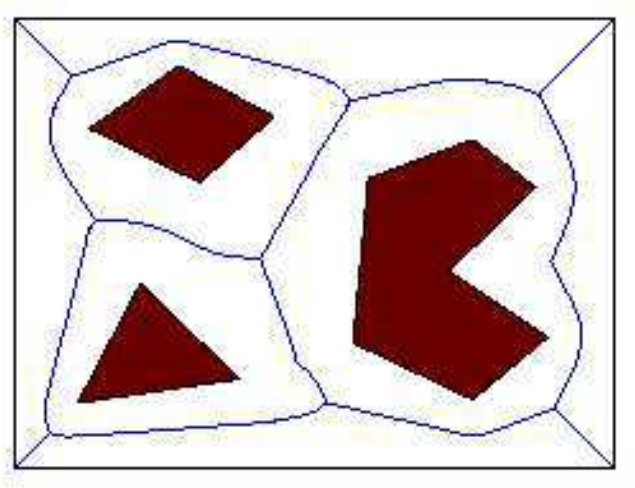

FIG. A.3 – Voronoi

Cette méthode est la plus couramment utilisée des méthodes polygonales. Les diagrammes de Voronoï sont des constructions classiques et bien maîtrisées.

A.1.5 Discussion

Les inconvénients des roadmaps sont leur grande dépendance au nombre total d'arêtes constituant les obstacles et la nécessité de travailler dans des espace de petite taille.

Leur avantage réside dans le fait qu'une pré-interprétation de l'espace libre est réalisée par ces méthodes, puisque les informations représentées sont déjà des déplacements libres du robot. Cela peut également être un inconvénient du fait de la restriction importante des possibilités.

D'une manière générale, ces approches seront surtout utiles pour proposer un déplacement de référence du robot, calculé hors ligne à partir des informations connues a priori.

A.2 Décompositions cellulaires

A.2.1 Exactes

Principe La décomposition cellulaire exacte a pour but d'obtenir une représentation polygonale aussi exacte que possible de l'espace libre, sous la forme d'un ensemble de cellules polygonales. La connectivité des cellules est exprimée à l'aide d'un graphe appelé *graphe de connectivité.*

Le principe général consiste à découper l'espace ($\mathcal{C}$) peuplé de polygones (les *C-obstacles*), en primitives géométriques telles que des triangles ou des trapèzes.

Discussion Les techniques de décomposition cellulaire exacte sont généralement purement géométriques (triangulation de Delaunay par exemple) et possibles dans des espaces de faible dimension (2 ou 3). Pour des dimensions plus élevées, une méthode générale ([Schwartz et Shark, 83]) existe pour calculer ces ensembles, mais en pratique, son coût oblige à se limiter à des espaces de dimension 2, correspondant par exemple à l'espace des configurations d'un robot circulaire ou ayant une orientation figée (voir [Avnaim et al., 88]).

D'une manière générale, les calculs associés à ces techniques sont complexes et ne permettent pas de modéliser des environnements complexes[1] dans des temps raisonnables pour permettre la modélisation de l'environnement en ligne.

En revanche, lorsque cela n'est pas nécessaire (lorsque l'environnement est connu), la modélisation peut être réalisée hors ligne avec l'une de ces approches.

Les représentations polygonales des obstacles permettent alors de conserver une résolution importante de $\mathcal{C}_{free}$ dans $\mathcal{C}$.

A.2.2 Approchées

L'idée première consiste à casser la complexité des décompositions cellulaires exactes ([Schwartz et Shark, 83]) ou des roadmaps en considérant une discrétisation plus grossières de l'espace ([Brooks et Lozano-Perez, 85]). Cette discrétisation, matérialisée par un découpage de l'espace en cellules de même forme, peut être figée (cellules de dimension fixe) ou adaptative (cellules de dimension variable).

Grilles homogènes La version la plus simple de grille d'occupation est la grille homogène. Son principe consiste à diviser l'espace en cellules rectangulaires (ou carrées) régulières. Un contenu booléen permet de savoir si un obstacle occupe la zone correspondante de l'espace, et donc son appartenance à $\mathcal{C}_{col}$ ou $\mathcal{C}_{free}$.

L'avantage principal de cette méthode est sa simplicité, tant sur le principe que sur les calculs mis en oeuvre. En revanche son principal défaut est sa forte dépendance

[1] La complexité de l'environnement dépend pour ces approches géométriques, du nombre total de segments constituant les obstacles.

vis-à-vis de la résolution de la grille : des cases trop grandes dans un environnement contraint peuvent aboutir à $\mathcal{C}_{free} = \phi$ (figure A.4). Il faut dans ce cas augmenter le nombre de cases, ce qui signifie augmenter la quantité de calculs et faire face à des problèmes de taille de stockage des informations.

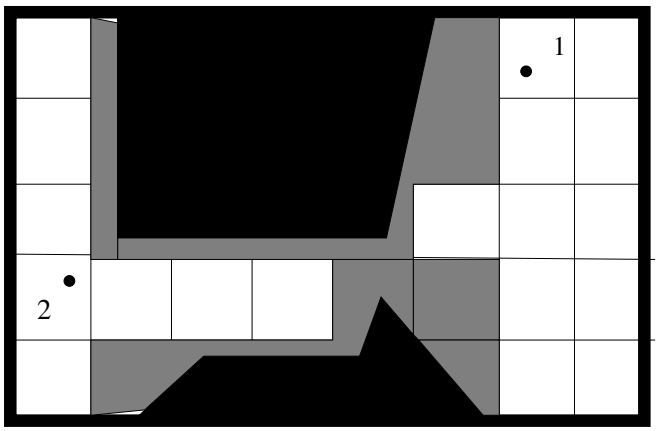

FIG. A.4 – $\mathcal{C}_{free} = \emptyset$

Décomposition adaptative Au lieu de considérer des cellules de même taille, il est également possible de travailler avec des cases de taille variable en fonction de la topologie des obstacles. Cela permet d'éviter une taille de cellule trop petite et l'explosion du nombre de cellules qui l'accompagne, lorsque l'environnement comporte des passages étroits.

L'algorithme le plus répandu est appelé quadtrees. Il s'agit d'un découpage récursif d'une cellule rectangulaire en quatre sous-cellules de même dimension :

- soit jusqu'à ce qu'une cellule corresponde à un espace tout en collision ou tout libre,
- soit jusqu'à ce que la condition précédente soit fausse, mais que la cellule ait atteint la taille minimale donnée.

Cette méthode permet une discrétisation hiérarchique de l'environnement, permettant de choisir le niveau de discrétisation auquel l'on désire travailler.

Comme pour toutes les méthodes basées sur des grilles, l'inconvénient de la méthode porte sur le nombre de cellules pour des environnements très contraints, mais dans une mesure moindre.

Une approche semblable consiste à représenter dans un arbre, une hiérarchie de boîtes englobantes des obstacles (alignées sur les axes principaux ou ceux des objets). Cette approche est séduisante en particulier pour des « boîtes » de forme circulaire en 2D ou sphériques en 3D ([Quinlan, 94]). Elle permet en effet d'approximer l'environnement par un ensemble d'obstacles circulaires, avec une résolution variable (en fonction du

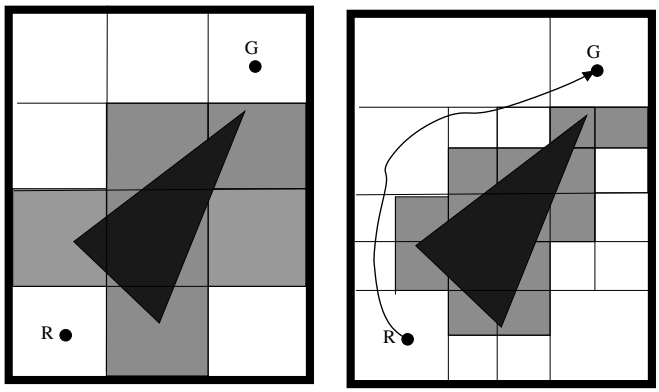

FIG. A.5 – Différents Niveaux de Resolution

niveau de profondeur choisi dans l'arbre). Les tests de collisions sont alors réduits à des calculs de distance entre le robot (représenté par un point dans $\mathcal{C}$) et le centre des *C-obstacles*.

Discussion sur l'utilisation de ces approches pour la navigation En raison de la faible dimension des espaces manipulés, les méthodes de décomposition cellulaire approchées, et en particulier les approches adaptatives hiérarchiques, permettent une représentation suffisamment précise de $\mathcal{C}_{free}$ sans avoir la complexité des approches exactes.

Ces approches sont parfaitement utilisables par une méthode de navigation pour la plupart des environnements de la vie courante. De plus, en cas de modification d'une partie de l'environnement, la structure hiérarchique de cette représentation permet de ne pas avoir à tout recalculer mais seulement la région modifiée.

A.3 Champs de potentiels

La méthode des champs de potentiels associe des forces attractives au but et des forces répulsives aux obstacles pour créer des champs de forces. En suivant les valeurs décroissantes de ce dernier le robot se dirige vers son but en évitant les obstacles.

L'inconvénient majeur de cette approche est le fait que la fonction de potentiel qui permet de diriger le robot vers le but, n'est pas strictement décroissante vers ce but, et admet des minima locaux.

L'approche initiale ([Khatib, 80] [Khatib, 86]) est sujette aux problèmes de minima locaux et ne garantit donc pas que le robot sera en mesure d'atteindre son but en suivant la courbe des potentiels décroissants. La première solution envisagée pour y remédier vise à se sortir des minima en appliquant des déplacements aléatoires lorsque ces derniers sont

atteints ([Barraquand et Latombe, 89]). La seconde solution est de garantir l'absence totale de minima locaux. Des fonctions de navigation ont donc été étudiées pour satisfaire cette contrainte : La première propose de séparer le gradient menant vers le but de celui associé aux obstacles (figure A.6). La fusion des deux conserve une pente vers le but en tout point, et minimise les minima locaux ([Koditschek, 87] [Rimon et Koditschek, 92]) sans toutefois les supprimer totalement. L'autre solution, meilleure mais nécessitant des calculs plus complexes, fait appel aux fonctions harmoniques ([Connolly et al., 90]) et se présente comme une analogie avec les fluides ([Feder et Slotine, 97]) ou l'électricité ([Connolly et Grupe, 94]).

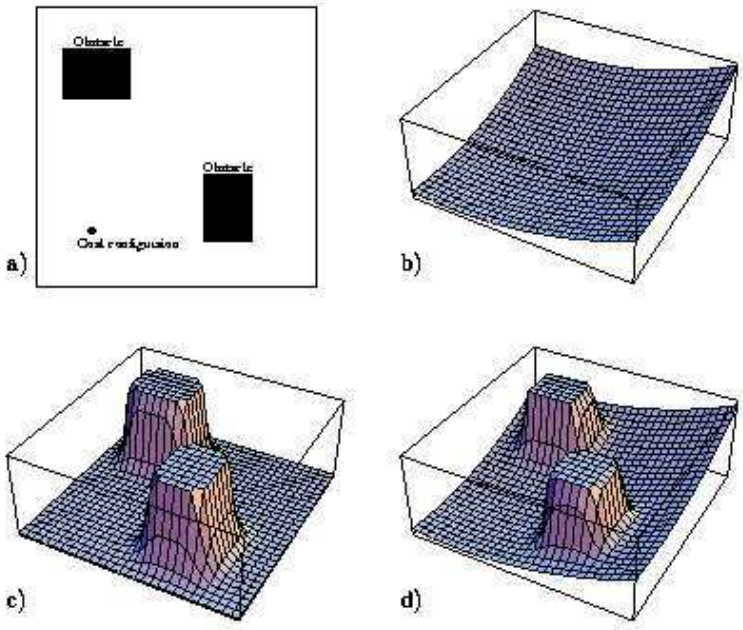

FIG. A.6 – Champs de Potentiel

A.3.1 Discussion sur l'utilisation de ces approches pour la navigation

Pour des raisons d'efficacité, le calcul des potentiels est généralement discrétisé selon une grille régulière définie dans l'espace de travail du robot ou celui des configurations. Par commodité, les obstacles peuvent être discrétisés selon la même grille selon le principe des grilles homogènes. Des versions analytiques non discrétisées ont également été proposées notamment par Faverjon, pour permettre une meilleure résolution.

Dans ces conditions et pour des environnements de taille raisonnable, les champs de potentiels peuvent en théorie être utilisés en ligne par une méthode de navigation. Ils

présentent de plus l'avantage de fournir un déplacement éloigné des obstacles à l'image des autoroutes ou du diagramme de Voronoï cités précédemment.

En pratique, cela dépendra de notre capacité à s'affranchir des problèmes de minima locaux, à l'aide d'une méthode appropriée de choix d'un déplacement.

Annexe B

Caractérisation des temps à collision

À chacune des vitesses d'un VO correspond une trajectoire en collision et un temps à collision associé. Nous nous intéresserons ici au temps avant collision des *Vitesses de Collision* situées à « l'intérieur » du LVO.

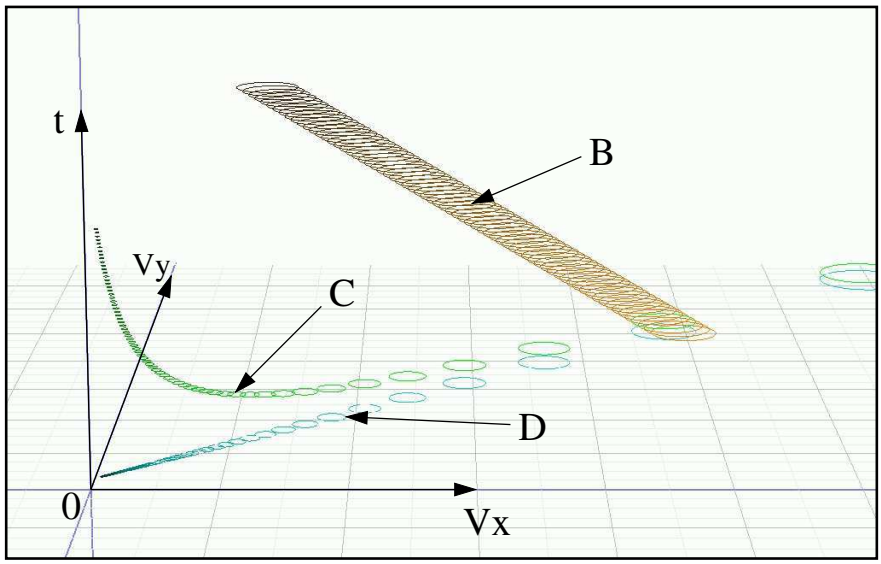

FIG. B.1 – *Projection orthogonale d'un LVO dans* $\mathcal{V}$ Le robot est localisé à l'origine. La trace B représente la position du robot dans $\mathcal{C}$ au cours du temps. La trace C est le LVO correspondant. Il est volontairement discrétisé de façon très grossière pour mieux apprécier la répartition des éléments temporels "le long" du LVO. La trace D représente la projection othogonale du LVO dans $\mathcal{V}$. Ce sont les "bords" de cette projection dont nous avons donné une expression analytique.

La figure B.2 nous permet d'apprécier une coupe d'un LVO exprimé dans $\mathcal{V} \times \mathcal{T}$, le long de son axe central, perpendiculairement à $\mathcal{V}$. Nous allons en considérer un élément temporel particulier. Si nous prenons la vitesse située en son centre, nous constatons que

celle-ci entraîne une collision avec l'obstacle à un temps antérieur. Ceci s'explique simplement par le fait qu'avant « d'atteindre » le centre de l'obstacle, le robot rencontrera son bord.

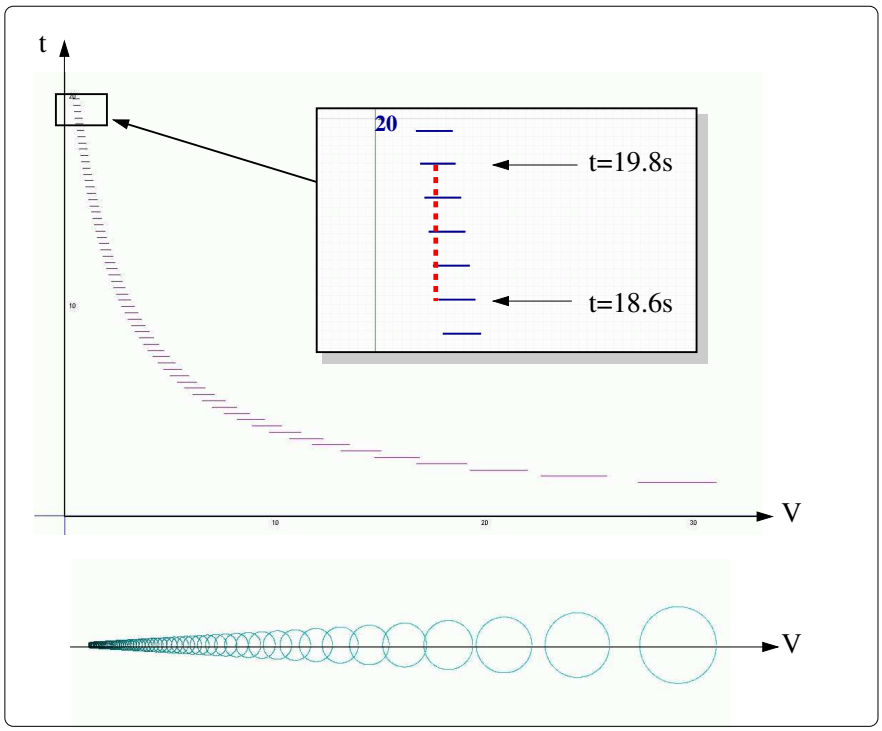

FIG. B.2 – *Profil d'un LVO* Le LVO de la figure B.1 est vu ici de profil (en haut) et de dessus (en bas). Nous considérons l'élément temporel pour t=19.8s et plus précisément la vitesse correspondant à son centre. Le robot va donc être tangent à l'obstacle à t=19.8s s'il maintient cette vitesse constante depuis t=0s. Ce temps correspond au temps avant collision des bords du LVO. Pour le centre en revanche, nous voyons que cette vitesse aura entraîné une collision avec l'obstacle dès t=18.6s. Les vitesses du centre du LVO et celles du bord ne génèrent pas une collision en même temps.

Nous allons chercher ici à caractériser le temps avant collision d'une vitesse en fonction de sa distance aux bords du *LVO*. Cela revient à considérer un élément temporel particulier. Par construction, cette démonstration s'appliquera donc aussi dans chaque $LVO_\circ$ qui approxime un *NLVO*. Cependant, du fait qu'un $LVO_\circ$ n'est qu'une approximation d'ordre 1 du *NLVO* à un instant donné, le temps à collision obtenu dans le $LVO_\circ$ ne sera lui aussi qu'une approximation du temps réel à collision dans le *NLVO*. Sans information plus précise sur les trajectoires des obstacles, il ne nous sera pas possible d'obtenir une expression plus exacte, sinon par discrétisation (voir méthode page 95).

B.1 Collisions de front avec obstacle statique

B.1.0.1 Hypothèses

Soit $\mathcal{A}$ un robot-point et $\mathcal{B}$ un obstacle circulaire statique définis dans $\mathcal{C}$ et tels que $\mathcal{A} \cap \mathcal{B} = \emptyset$.

Soit le LVO de $\mathcal{A}$ associé à $\mathcal{B}$. $\vec{v}$ est une vitesse de $\mathcal{A}$ appartenant au LVO représentée par un point V dans l'espace des vitesses immédiates du robot. Le projeté orthogonal de V dans cet espace, sur le bord du LVO le plus proche, est noté V'. Sa vitesse associée est $\overrightarrow{v'}$.

Nous appelons h la distance séparant V du bord du LVO (par définition, h est la longueur du segment [VV']).

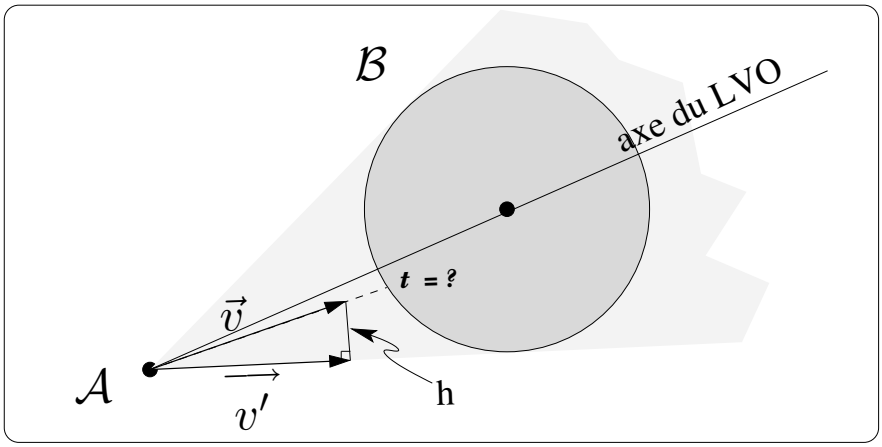

FIG. B.3 – *Illustration du problème* Connaissant les vitesses $\vec{v}$ et $\overrightarrow{v'}$, la distance h séparant ces vitesses dans $\mathcal{V}$, et t' le temps à collision associé à $\overrightarrow{v'}$, nous cherchons à calculer le temps à collision t associé à $\vec{v}$.

Nous appelons t (respectivement t'), le temps durant lequel le robot peut maintenir sa vitesse égale à $\vec{v}$ (respectivement $\overrightarrow{v'}$) sans entrer en collision avec $\mathcal{B}$.

Nous cherchons à définir le temps t en fonction de t' et de h (figure B.3).

Remarque : En raison de la symétrie de la figure par rapport à l'axe central du LVO, prendre une vitesse $\vec{v}$ à gauche ou à droite de cet axe ne change rien car le problème est symétrique. Nous choisirons d'étudier le cas d'une vitesse $\vec{v}$ située sur ou à gauche de l'axe du LVO.

B.1.0.2 Démonstration

Pour l'ensemble de la démonstration, nous nous plaçons dans l'espace des vitesses immédiates $\mathcal{V} = \mathbb{R}^2$ du robot.

Nous considérons le cercle δB de centre B et de rayon r_B représentant l'obstacle et le point A représentant le robot (figure B.4).

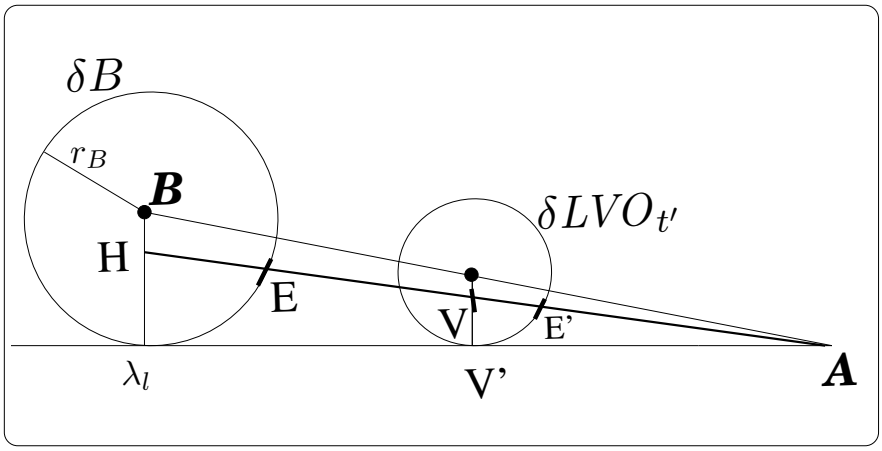

FIG. B.4 – *Illustration du problème* Voir texte pour explications.

À chaque vitesse v définie dans $\mathcal{V}$, on associe le vecteur $\vec{v}$ issu de A et dont l'extrémité est V.

Soit λ_l le bord gauche du LVO de $\mathcal{A}$ associé à $\mathcal{B}$.
Soit $\text{LVO}_{t'}$ l'élément temporel du LVO de $\mathcal{A}$ associé à $\mathcal{B}$ au temps t', où t' est tel que défini précédemment. Le contour de $\text{LVO}_{t'}$ est un cercle noté $\delta\text{LVO}_{t'}$.

Par définition, le point V' décrit précédemment (projeté othogonal dans $\mathcal{V}$ de V sur λ_l)est tel que :

$$\{V'\} = \lambda_l \cap \delta\text{LVO}_{t'}$$

Nous définissons le point H tel que $\vec{H} = t' \cdot \vec{v}$.
Puisque $V \in$ LVO, par définition, $\vec{H}$ représente une trajectoire rectiligne du robot entraînant une collision (i.e. terminant dans l'obstacle $\mathcal{B}$). Nous avons donc :

$$H \in \mathcal{B}$$

Le point auquel le robot percute l'obstacle en premier, s'il suit la trajectoire définie par $\vec{H}$, est noté E. Ce point est l'intersection du segment [AH] et du cercle δB. Nous avons alors :

$$\{E\} = [\mathrm{AH}] \cap \delta B$$

Nous savons que $\vec{H} = t' \cdot \vec{v}$. De même, il existe un temps t, tel que :

$$\vec{E} = t \cdot \vec{v}$$

Les relations précédentes nous donnent (si $\| \cdot \|$ désigne la norme d'un vecteur) :

$$\|\vec{H}\| = t' \times \|\vec{v}\|$$

et

$$\|\vec{E}\| = t \times \|\vec{v}\|$$

Nous en déduisons :

$$\|\vec{v}\| = \frac{\|\vec{H}\|}{t'} = \frac{\|\vec{E}\|}{t}$$

D'où la relation suivante :

$$t = \frac{\|\vec{E}\|}{\|\vec{H}\|} \times t'$$

Considérons maintenant $\mathcal{H}$, l'homothétie de centre A et de rapport $1/t'$. Par définition, $\mathrm{LVO}_{t'}$ est l'image de $\mathcal{B}$ par $\mathcal{H}$. Nous notons E' l'image de E par $\mathcal{H}$.

Nous pouvons en déduire que :

$$\{E'\} = [\mathrm{AV}] \cap \delta\mathrm{LVO}_{t'}$$

L'homothétie conservant les rapports entre les longueurs, nous avons également :

$$t = \frac{\|\vec{E'}\|}{\|\vec{V}\|} \times t'$$

Nous connaissons t' et $\|\vec{V}\|$. Il nous manque $\|\vec{E'}\|$ mais nous savons que E' est l'intersection d'un cercle connu et d'un segment connu.

Le problème se redéfinit donc en un problème classique de géométrie analytique : Trouver l'intersection d'un cercle et d'un segment.

Soit $r_{\mathrm{LVO}_{t'}}$ le rayon du cercle $\delta\mathrm{LVO}_{t'}$, nous avons :

$$r_{\mathrm{LVO}_{t'}} = \frac{r_B}{t'}$$

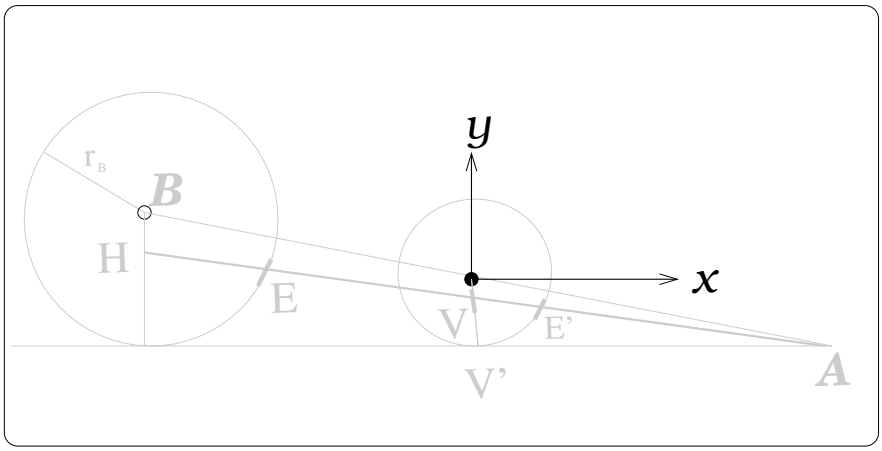

FIG. B.5 – *Changement de repère* Voir texte.

Pour faciliter les calculs, nous nous plaçons maintenant dans un repère orthonormé dont l'origine est le centre de ce cercle. Nous conservons tout de même la même convention quant aux vecteurs associés aux vitesses de $\mathcal{V}$. L'orientation du repère est telle que λ_l est horizontale (figure B.5).

Dans ce repère, l'équation du cercle $\delta\text{LVO}_{t'}$ de centre $O(0\,;\,0)$ et de rayon $r_{\text{LVO}_{t'}}$ s'écrit :

$$\delta\text{LVO}_{t'} : x^2 + y^2 = {r_{LVO_{t'}}}^2$$

Nous posons :

$$a = \frac{-h}{\|\overrightarrow{V'}\|}$$

et

$$b = h - r_{\text{LVO}_{t'}}$$

L'équation de la droite passant par $A(\|\overrightarrow{V'}\|\,;\; -r_{LVO_{t'}})$ et $V(0\,;\,h - r_{\text{LVO}_{t'}})$ s'écrit alors :

$$(\text{AV}) : y = ax + b$$

Trouver les coordonnées de E' revient donc à résoudre le système suivant :

$$\begin{cases} x^2 + y^2 = {r_{\text{LVO}_{t'}}}^2 \\ y = ax + b \end{cases}$$

Par substitution, la première équation devient :

$$x^2 + (ax + b)^2 = {r_{LVO_{t'}}}^2$$

C'est une équation du second degré dont le déterminant Δ est :

$$\Delta = {r_{LVO_{t'}}}^2 \times \left(1 + a^2\right) - b^2$$

Nous savons que cette équation admet une solution unique si $h = 0$ ($\Delta = 0$) et deux dans le cas contraire ($\Delta > 0$). Ces équations sont de la forme :

$$x = \frac{-ab \pm \sqrt{\Delta}}{(1 + a^2)}$$

Or, dans le repère choisi nous savons que E' sera tel que $x \geq 0$. Donc, nous obtenons :

$$x = \frac{-ab + \sqrt{\Delta}}{(1 + a^2)}$$

Nous posons :

$$c = 1 + a^2$$

Dans ces conditions, x s'écrit maintenant :

$$x = \frac{-ab + \sqrt{{r_{LVO_{t'}}}^2 \times c - b^2}}{c}$$

Nous en déduisons les coordonnées complètes du point E' :

$$\begin{cases} x_{E'} = \frac{-ab + \sqrt{{r_{LVO_{t'}}}^2 \times c - b^2}}{c} \\ \\ y_{E'} = a \times x_{E'} + B \end{cases}$$

Connaissant ces coordonnées, il est maintenant possible de calculer $\|\vec{E'}\|$ (la longueur du segment [AE']) :

$$\|\overrightarrow{E'}\| = \sqrt{\left(x_{E'} - \|\overrightarrow{V'}\|\right)^2 + \left(y_{E'} + r_{\mathrm{LVO}_{t'}}\right)^2}$$

Nous pouvons également calculer $\|\vec{V}\|$ (la longueur du segment [AV]) :

$$\begin{aligned} \|\vec{V}\| &= \sqrt{\left(-\|\overrightarrow{V'}\|\right)^2 + \left(h - r_{\mathrm{LVO}_{t'}} + r_{\mathrm{LVO}_{t'}}\right)^2} \\ &= \sqrt{\|\overrightarrow{V'}\|^2 + h^2} \end{aligned}$$

Finalement, nous obtenons la valeur de k :

$$k = \frac{\|\overrightarrow{E'}\|}{\|\vec{V}\|} = \sqrt{\frac{\left(x_{E'} - \|\overrightarrow{V'}\|\right)^2 + \left(y_{E'} + r_{\mathrm{LVO}_{t'}}\right)^2}{\|\overrightarrow{V'}\|^2 + h^2}} \quad \text{avec} \begin{cases} a = \frac{-h}{\|\overrightarrow{V'}\|} \\ b = h - r_{\mathrm{LVO}_{t'}} \\ c = 1 + a^2 \\ x_{E'} = \frac{-ab + \sqrt{{r_{LVO_{t'}}}^2 \times c - b^2}}{c} \\ y_{E'} = a \times x_{E'} + b \end{cases}$$

B.2 Collisions de front avec obstacle de mouvement rectiligne constant

B.2.0.3 Hypothèses

Par définition, les LVO permettent de modéliser non seulement des obstacles statiques, mais également des obstacles mobiles se déplaçant en ligne droite et à vitesse constante.

Nous allons étudier ici quel est l'effet de ce déplacement sur les résultats obtenus précédemment pour le cas d'un obstacle statique.

Le but est de définir le temps avant collision associé à une vitesse appartenant au LVO, selon sa distance par rapport aux "bords" du LVO.

B.2.0.4 Proposition

De même que précédemment, nous considérons un robot-point $\mathcal{A}$ et un obstacle circulaire $\mathcal{B}$, mais la vitesse de ce dernier est maintenant non-nulle (figure B.6). Nous nous plaçons dans l'espace $\mathcal{V}$.

D'un point de vue géométrique, dans $\mathcal{V}$, le passage d'un *LVO* associé à un obstacle statique vers un *LVO* associé à un obstacle mobile de vitesse linéaire $\overrightarrow{v_B}$, est une simple translation de vecteur $\overrightarrow{v_B}$ (Cf. page 36 pour plus de détails).

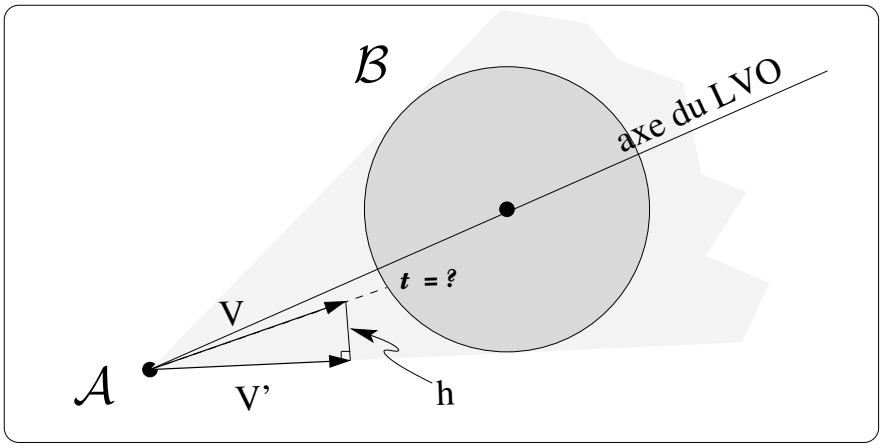

FIG. B.6 – *Rappel du problème*

Le calcul du rapport k pour un obstacle statique résultant uniquement de calculs de géométrie sur les longueurs et la translation conservant les longueurs, nous cherchons à démontrer que ce calcul est toujours valable pour un obstacle mobile.

B.2.0.5 Démonstration

Soit un obstacle $\mathcal{B}$ de vitesse linéaire $\overrightarrow{v_B}$. Soit également $\overrightarrow{v_A}$ une vitesse linéaire immédiate du robot. Nous pouvons nous ramener au cas précédent (obstacle statique) en considérant la vitesse relative $\overrightarrow{v_{A/B}}$ du robot par rapport à l'obstacle (figure B.7).

Définition 1 (Vitesse relative du robot par rapport à l'obstacle) *La vitesse relative $\overrightarrow{v_{A/B}}$ du robot par rapport à l'obstacle est définie par la relation :*

$$\overrightarrow{v_{A/B}} = \overrightarrow{v_A} - \overrightarrow{v_B}$$

où $\overrightarrow{v_A}$ et $\overrightarrow{v_B}$ désignent respectivement la vitesse linéaire instantanée du robot et celle de l'obstacle.

Par cette astuce, nous pouvons faire « disparaître » la vitesse de l'obstacle et nous ramener au cas d'un obstacle statique.

Le LVO associé à l'obstacle et alors remplacé par CC, son *Cône de Collision* correspondant (CC représente l'ensemble de vitesses relatives du robot par rapport à l'obstacle, qui entraînent une collision avec l'obstacle. Voir page 36).

Nous posons :

$$\vec{v} = \overrightarrow{v_{A/B}} = (\overrightarrow{v_A} - \overrightarrow{v_B})$$

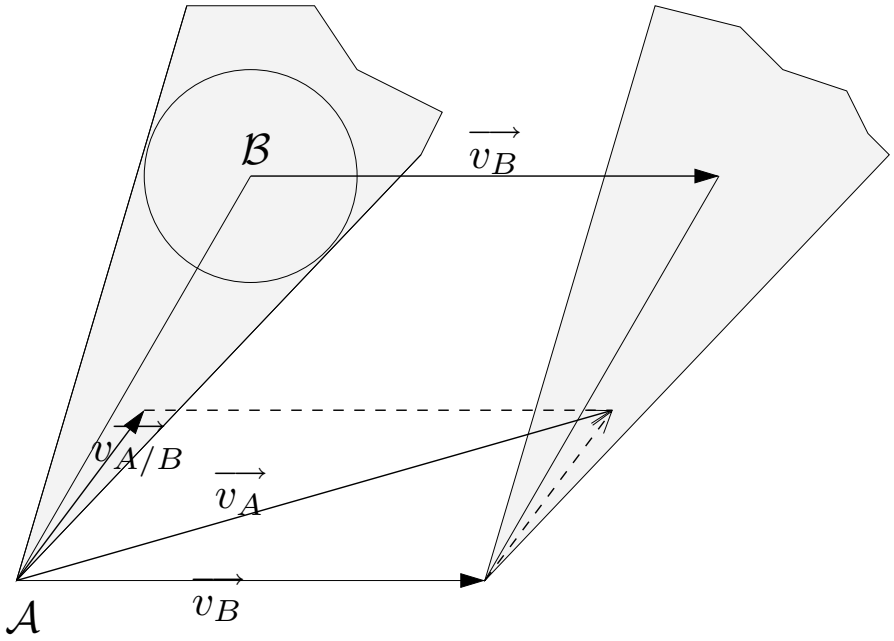

FIG. B.7 – *Passage de CC à LVO*

Soit V_A (respectivement $V_{A/B}$) l'extrémité du vecteur $\overrightarrow{v_A}$ (respectivement $\overrightarrow{v_{A/B}}$) issu de A dans $\mathcal{V}$.

Par définition du CC et du LVO, la position relative de V_A dans le LVO est la même que celle de $V_{A/B}$ dans le CC. En effet, LVO est le translaté de CC par $\overrightarrow{v_B}$, et V_A est le translaté de $V_{A/B}$ par $\overrightarrow{v_B}$.

Nous pouvons en déduire la proposition suivante :

Propriété 1 (Distance de V_A au bord du LVO) *La distance minimale de V_A au bord du LVO, est la distance minimale de $V_{A/B}$ au bord du CC.*

De la propriété précédente, nous pouvons en déduire que les équations valables dans CC le sont aussi dans LVO et vice-versa. Or CC est équivalent au LVO de $\mathcal{B}$ pour $\overrightarrow{v_b} = \vec{0}$, c'est-à-dire pour le cas où $\mathcal{B}$ est statique. Par conséquent, les équations décrites pour le cas d'un obstacle statique et qui sont valables dans CC, le sont aussi dans LVO.

Propriété 2 (Passage de CC à LVO) *LVO étant le translaté de CC par le vecteur $\overrightarrow{v_B}$, les relations valables entre les différents points remarquables du LVO associé à un obstacle statique, sont également valables pour leurs translatés et donc pour les points d'un LVO associé à un obstacle en translation.*

Soit A' le point situé à l'extrémité du LVO. Nous avons :

$$A' = A + \overrightarrow{v_B}$$

À condition de se placer dans un repère centré en A' et non plus en A, l'équation du calcul du rapport k, définie précédemment pour le cas d'un obstacle statique, reste identique.

Nous allons ainsi pouvoir définir ce rapport pour le cas général.

Caractérisation du temps avant collision Soit $\mathcal{A}$ un robot-point et $\mathcal{B}$ un obstacle circulaire de rayon r_B et de vitesse linéaire $\overrightarrow{v_B}$, tels que :

$$\mathcal{A} \cap \mathcal{B} = \emptyset$$

Nous nous plaçons dans l'espace $\mathcal{V}$ des vitesses linéaires immédiates de $\mathcal{A}$, centré en A.

Soit le LVO de $\mathcal{A}$ associé à $\mathcal{B}$, défini dans cet espace.

Nous notons A' l'extrémité du LVO correspondant à la vitesse du robot générant une collision avec $\mathcal{B}$ dans un temps infini. Ce point s'obtient par construction de la manière suivante :

$$A' = A + \overrightarrow{v_B}$$

Soit $\vec{v}$ une vitesse linéaire immédiate de $\mathcal{A}$ appartenant au LVO.

V est l'extrémité du vecteur $\vec{v}$ issu de A.

Le projeté orthogonal de V sur le bord du LVO le plus proche, est noté V'.

t' est le temps avant collision associé au bord du LVO au point V'.

Soient h et d les longueurs respectives des segments [VV'] et [A'V'].

Le temps t_c avant collision, associé à la vitesse $\overrightarrow{v}$, est tel que :

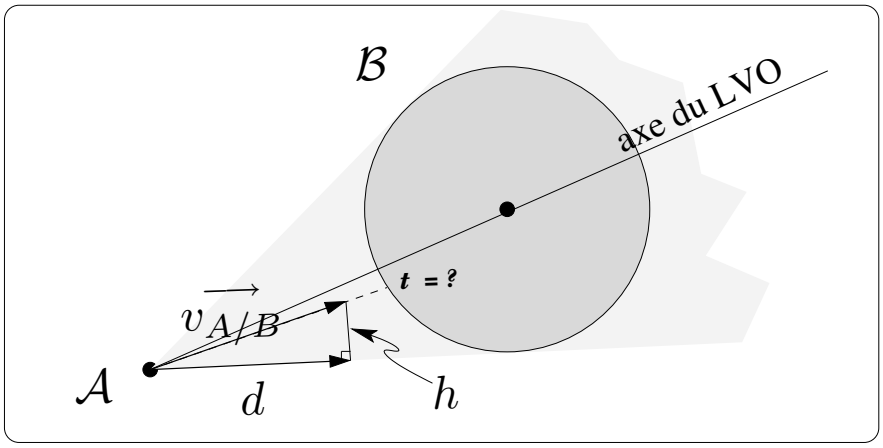

FIG. B.8 – *Calcul de t.*

$$t_c = k \times t' \qquad \text{avec} \begin{cases} k = \sqrt{\frac{(x-d)^2+(y+r)^2}{d^2+h^2}} \\ r = \frac{r_B}{t'} \\ a = \frac{-h}{d} \\ b = h - r \\ c = 1 + a^2 \\ x = \frac{-ab+\sqrt{c \times r^2 - b^2}}{c} \\ y = ax + b \end{cases} \tag{B.1}$$

Annexe C

Le Cycab

C.1 Présentation générale du Cycab

Le robot mobile considéré pour nos expérimentations est un véhicule électrique expérimental non-holonome de type voiture de golf, le Cycab (figure C.1). Conçu par l'INRIA, il est construit et distribué par la société Robosoft. Le braquage des roues avant et arrière est contrôlable, ainsi que la vitesse de rotation de chaque roue. Le pilotage peut être manuel par le biais d'un joystick central, automatique grâce à des unités de calculs et des capteurs embarqués, ou semi-automatique (applications d'assistance à la conduite).

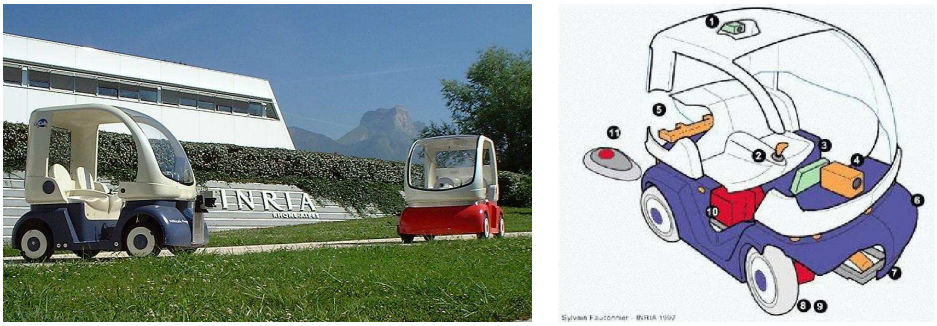

FIG. C.1 – *Cycab de l'INRIA Rhône-Alpes.* Le Cycab est disponible sous différentes configurations. La vue éclatée (à droite) est une configuration possible, comprenant une caméra vidéo (1), un joystick (2), un écran LCD tactile (3), une caméra linéaire infrarouge (4), une cible infrarouge (5), une ceinture de proximètres à ultrasons (6), une direction électrique (7), des freins électriques (8) et un moteur (9) sur chaque roue et un système de recharge des batteries (10-11).

C.2 Particularités mécaniques et cinématiques

Contrainte d'Ackermann (ou épure de Jeantaud) D'un point de vue mécanique, la conception du Cycab vérifie, pour chaque essieu, la contrainte d'Ackermann (épure

de Jeantaud) que l'on retrouve sur les voitures, à savoir le fait que les prolongements des axes des roues gauches et droites d'un même essieu doivent, au braquage, se croiser sur le prolongement de l'autre essieu. L'angle de braquage des roues n'est alors pas le même à gauche et à droite (figure C.2). Dans ces conditions, le terme braquage avant fait référence à l'angle de braquage d'une roue virtuelle, placée au milieu de l'essieu avant, et dont l'orientation respecte la contrainte d'Ackermann. Il en est de même pour le terme braquage arrière.

Modèle cinématique d'une voiture Sur une voiture, l'angle de braquage arrière est contraint à zéro en permanence. En faisant l'hypothèse que les glissements sont négligeables, la contrainte d'Ackermann nous permet de dire que toutes les roues restent alors tangentes à la trajectoire circulaire du véhicule (figure C.2). Le modèle cinématique d'une voiture contrôlée par son angle de braquage avant et la vitesse linéaire instantanée de son point de référence (ici le milieu de l'essieu avant), peut alors s'écrire de la manière suivante ([Barraquand et Latombe,93]) :

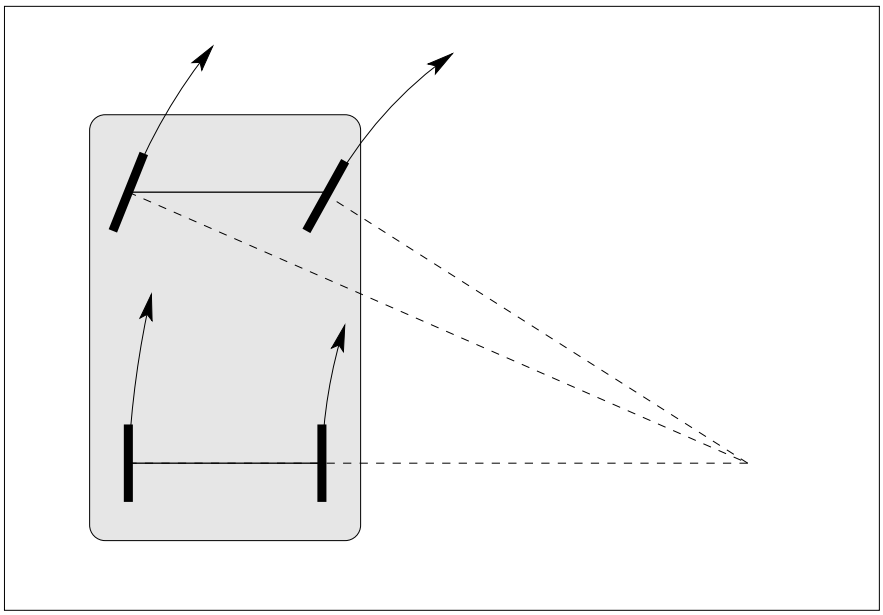

FIG. C.2 – *Contraintes d'Ackermann sur une voiture.* Le prolongement des axes des roues se croisent en un point unique : le centre théorique de rotation du véhicule.

$$\begin{cases} \dot{x} &= v_f \cdot cos(\theta + \phi) \\ \dot{y} &= v_f \cdot sin(\theta + \phi) \\ \dot{\theta} &= \frac{v_f}{L} \cdot sin(\phi) \end{cases}$$

où v_f est la vitesse linéaire de la roue virtuelle avant par rapport au sol, ϕ l'angle de braquage de la roue virtuelle avant, L la distance entre les essieux et (x, y, θ) la configuration du véhicule.

Condition de non-glissement et cas du Cycab Il est admis qu'une roue est supposée ne pas glisser lorsque les forces tangentes à la surface de contact restent inférieures aux forces normales à la surface de contact. Par conséquent, l'absence de glissement sera vérifiée à l'aide d'une inégalité simple sur les accélérations tangentielles et normales du véhicule (application des lois de Newton) :

$$\ddot{x}^2 + \ddot{y}^2 \leq \mu^2 g^2$$

où μ est un coefficient de glissement et g l'accélération de la gravité terrestre.

Cette approche est très simplifiée car les paramètres qui entrent en jeu sont en réalité beaucoup plus nombreux (usure des pneumatiques, surface de contact, nature du sol, force de maintien de la roue au sol...) et d'autres modèles plus complexes ont été développés (voir [Liu et Peng, 96] [Latombe, 00] et [Sienel, 97] pour quelques exemples).

Pour le Cycab, les angles de braquage avant et arrière sont tous deux différents de zéro, et ces méthodes ne s'appliquent plus. En effet, le véhicule n'admet plus un seul centre de rotation théorique, mais deux distincts (figure C.3). Il en résulte des glissements inévitables et non modélisables par une approche analytique.

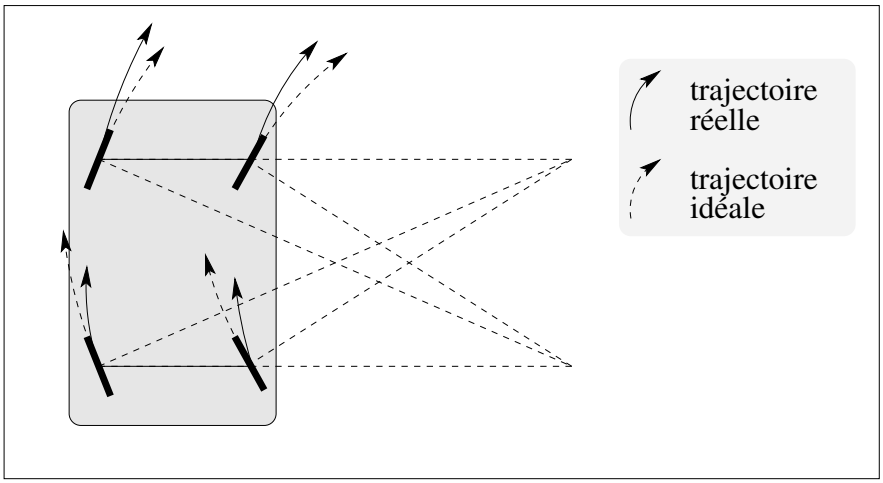

FIG. C.3 – *Contraintes d'Ackermann sur le Cycab.* Le prolongement des axes des roues avant et arrières se croisent en 2 points distincts. Sur la figure, nous avons placé le centre de rotation réel du véhicule à mi-chemin entre ces deux points, afin d'apprécier les écarts entre la trajectoire idéale des roues (en pointillés) minimisant les glissements et la trajectoire réelle (en trait continu) suivie par chaque roue.

Modèle cinématique du Cycab Pour les raisons citées précédemment, la cinématique du Cycab est modélisée par un modèle simplifié, éloigné de la réalité. Il est basé sur deux hypothèses fausses qui sont :

- l'absence de glissement des roues
- la présence du centre de giration à l'intersection des prolongements des axes des 2 roues virtuelles (figure C.4).

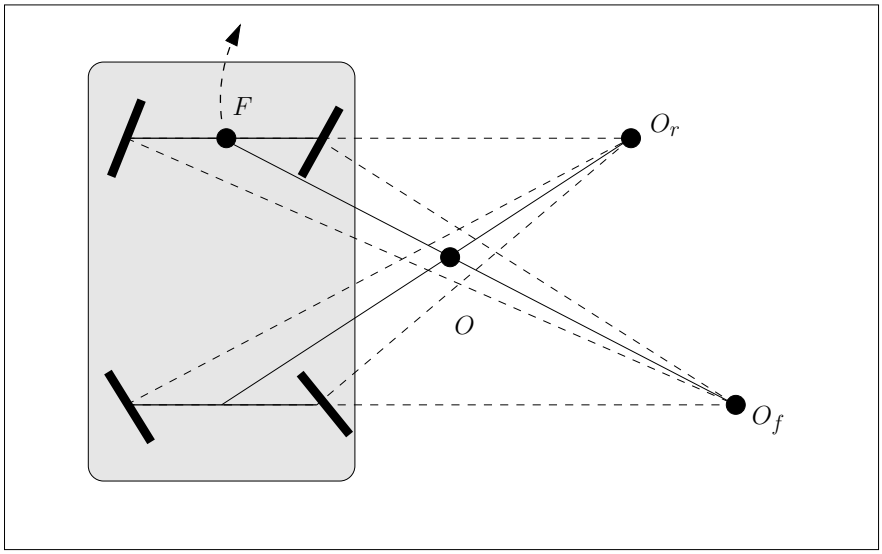

FIG. C.4 – *Approximation du modèle cinématique du Cycab.* Une roue virtuelle est placée au milieu de chaque essieu. Son orientation est telle qu'elle vérifie la contrainte d'Ackermann (les roues de l'essieu avant ont leurs axes qui s'intersectent sur le prolongement de l'essieu arrière en O_f et réciproquement pour les roues arrières en O_r). Nous faisons l'hypothèse que le centre de rotation du véhicule est placé à l'intersection des prolongements des axes des 2 roues virtuelles, au point O. Le point F désigne le point de référence du véhicule (milieu de l'essieu avant).

L'expression analytique d'un tel modèle s'écrit alors :

$$\begin{cases} \dot{x} &= v_f \cdot cos(\theta + \phi_f) \\ \dot{y} &= v_f \cdot sin(\theta + \phi_f) \\ \dot{\theta} &= \frac{v_f}{L} \cdot \frac{sin(\phi_f - \phi_r)}{cos(\phi_r)} \end{cases}$$

où v_f est la vitesse linéaire de la roue virtuelle avant par rapport au sol, ϕ_f et ϕ_r respectivement l'angle de braquage de la roue virtuelle avant et arrière, L la distance entre les essieux et (x, y, θ) la configuration du véhicule.

Suivi de trajectoire avec le Cycab Le suivi d'une trajectoire par le Cycab a été réalisé selon trois approches :

- l'approche automatique : Une loi de commande ([Kanayama et al., 91]) est utilisée pour réduire les écarts entre une trajectoire de référence et la trajectoire actuellement suivie par le robot.
- l'approche RNA ([Gauthier, 99]) : Un réseau de neurones artificiels est entraîné pour apprendre quelles commandes du véhicule permettent de réaliser un déplacement désiré.
- l'approche de la logique floue ([Garnier, 95]) : Un contrôleur flou minimise à l'aide de règles trois critères : l'écart en position, l'écart d'orientation du véhicule et l'écart en vitesse par rapport à une consigne.

En raison des difficultés à modéliser le Cycab, l'approche automatique ne donne pas de bons résultats en particulier dans les virages où les glissements sont très importants. Les approches de l'intelligence artificielle bénéficient de capacités d'apprentissage les rendant plus intéressantes pour le Cycab. En revanche, elles peuvent fournir des réponses fausses en cas d'apprentissage insuffisant ou erroné.

L'approche proposée dans ce rapport est une alternative (voir 4.6).

Annexe D

Réseaux de neurones artificiels à fonction à base radiale (RBFN)

Cette annexe introduit les notions de base nécessaires à la compréhension de la section 4.6, qui met en œuvre un réseau de neurones artificiels à fonctions de base radiale. Le lecteur pourra se reporter à [Haykin, 94],[Hecht-Nielsen, 90] ou encore [Gauthier, 99] pour une description plus détaillée des différents types de réseaux de neurones existants.

D.1 Généralités

Neurone artificiel Le neurone artificiel est né avec l'idée de copier le neurone biologique du cerveau humain. À l'origine, il en propose une version très simplifiée, qui a maintenant évolué vers un automate au fonctionnement très différent du modèle biologique. Un neurone artificiel comprend 4 éléments (figure D.1) :

1. des ***entrées***, auxquelles sont associés des poids pour en modifier l'importance
2. une ***fonction de prétraitement de ces entrées*** (généralement une simple somme pondérée)
3. une ***fonction d'activation***, qui calcule l'état d'activation du neurone à partir de ses entrées. Il peut s'agir d'une valeur réelle ou booléenne
4. une ***sortie***, qui prend généralement pour valeur l'état d'activation du neurone.

Organisation en réseaux La sortie de chaque neurone (aussi appelé unité) peut être reliée à une ou plusieurs entrées d'autres neurones pour former un réseau de neurones artificiels (RNA). Les neurones recevant les données du monde extérieur sont appelés neurones d'entrée et ceux qui lui retournent un résultat sont appelés neurones de sortie. Les neurones intermédiaires éventuels sont dits cachés. Un neurone peut éventuellement être à la fois entrée et sortie.

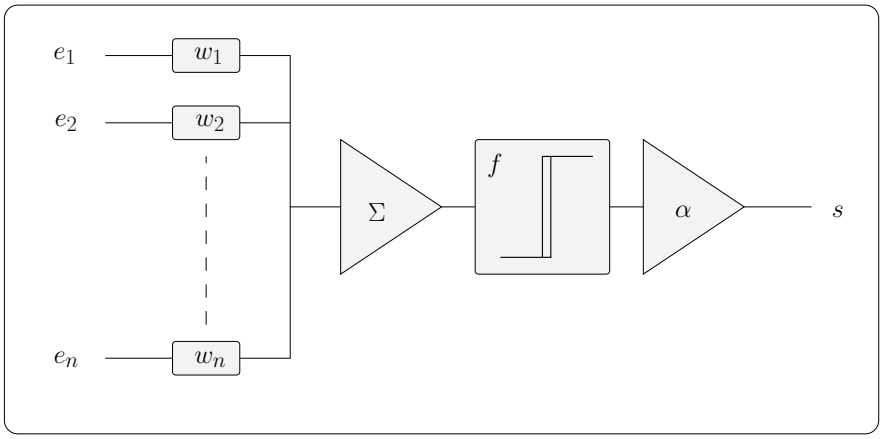

FIG. D.1 – *Neurone formel* Il comprend n entrées pondérées par des poids, puis sommées. Le résultat est présentée à une fonction d'activation. La sortie de celle-ci est amplifiée (pondérée) pour donner la valeur de sortie.

Apprentissage À l'instar du cerveau humain, certains RNA sont capables d'apprendre à modéliser une fonction par apprentissage. Trois types d'apprentissage sont possibles :

1. l'apprentissage supervisé : des couples (entrée ; sortie correspondante) sont présentés au réseau. L'écart entre la sortie proposée par le RNA et la sortie désirée permet de calculer une erreur quantitative utilisée pour adapter les paramètres du RNA.

2. l'apprentissage semi-supervisé (ou par renforcement) : une mesure qualitative permet de décider si la correspondance (entrée ;sortie) proposée par le RNA est correcte ou non. La réponse est utilisée pour adapter le RNA (renforcer ses « croyances » ou les affaiblir).

3. l'apprentissage non supervisé : il n'existe pas de connaissance particulière sur les correspondances (entrée ;sortie). Il s'agit généralement pour le RNA de regrouper les données par catégories (données possédant des propriétés communes).

Technique de descente de gradient La technique de descente du gradient est la plus courante pour adapter les paramètres d'un RNA lors de l'apprentissage. Elle consiste à minimiser l'erreur quadratique définie par :

$$Q = \frac{1}{2}\sum(a_i - s_i)^2 \tag{D.1}$$

où a_i et s_i représentent respectivement l'activation mesurée et l'activation désirée d'un neurone.

La méthode a été introduite dans les années 80 comme une méthode permettant de faire remonter l'erreur entre un état donné et un état désiré vers les niveaux précédents.

Les poids entre les couches de neurones sont modifiés proportionnellement au niveau d'activation des neurones correspondants. L'idée consiste donc à davantage modifier les poids des neurones qui ont le plus contribué à cette erreur.

Propriétés Selon la topologie du réseau les propriétés du RNA sont différentes. Nous citerons les principales :

1. La présence de cycles dans le graphe orienté des connexions entre neurones : Ces RNA sont dits récurrents et permettent de mieux traiter des problèmes comportant un aspect temporel. L'apprentissage de ces RNA est cependant plus complexe, plus long et mal maîtrisé.

2. La manière de stocker (localement ou globalement) l'information : Une information codée localement suppose que chaque neurone soit spécialisé dans le traitement d'une partie spécifique de l'espace d'entrée. Cela permet de modifier une correspondance (entrée;sortie) sans désapprendre les connaissances acquises pour d'autres valeurs d'entrées. Inversement, un codage global fait intervenir l'ensemble des neurones et chaque nouvel apprentissage remet en cause l'ensemble du RNA.

D.2 RNA à fonctions de base radiale (RBF)

D.2.1 Origines

Une fonction à base radiale (RBF), est une fonction Φ dont la sortie est symétrique autour d'un centre μ_c. Ceci donne :

$$\Phi_c(x) = \Phi(||x - \mu_c||)$$

Une fonction gaussienne est un exemple de RBF. Comme la plupart d'entre elles, elle possède un second paramètre pour la définir, σ, qui permet d'ajuster la largeur de la zone d'influence autour de μ_c :

$$\Phi_c(x) = \Phi(||x - \mu_c\sigma||)$$

Il a été montré ([Megdassy, 61] [Powel, 85] [Broomhead et Lowe, 88]) qu'une combinaison linéaire de RBF permet d'approximer ou d'interpréter un grand nombre de fonctions, et des chercheurs ont été tentés d'en proposer des représentations neuronales (RBFN pour Radial Basis Functions Network) ([Lee et Kil, 88] [Moody et Darken, 89] [Pogio et Girosi, 90]). Les propriétés d'approximateurs universels ont ainsi pu être démontrées pour plusieurs configurations de réseaux de neurones ([Kowalski et al., 90] [Park et Sandberg, 91] [Park et Sandberg, 93]) et leurs taux de convergence ont été établis ([Niyogi et Girosi, 96] [Xu et al., 94]). L'apprentissage des paramètres μ_i et σ_j des RBF d'une part, et des poids (w_j) sur les entrées d'autre part, ont été rendus indépendants afin de permettre un apprentissage rapide à partir de données bruitées. Les études

sur la convergence et autres propriétés des RBFN ont permis de s'entourer d'outils formels comme en automatique. Le lien avec cette discipline, et plus précisément avec la théorie de la régulation, a d'ailleurs été mis en avant.

D'une manière synthétique, les RBFN peuvent être vus comme des approximateurs universels de fonctions non-linéaires, dotés d'algorithmes d'apprentissage efficaces et bien maîtrisés, ainsi que d'une structure locale permettant un apprentissage incrémental. Pour ces raisons et leur lien établis avec des techniques de l'automatique notamment, ils ont été utilisés entre-autre en robotique mobile pour le contrôle de robot (voir [Gauthier, 99] pour quelques exemples).

D.2.2 Architecture d'un RBFN

D.2.2.1 Topologie

Les RBFN comportent trois couches de neurones (figure D.2) : la couche d'entrée, la couche de sortie, et entre les deux, une couche cachée. Ils sont dits orientés, c'est-à-dire que les liaisons entre leurs neurones vont des entrées, vers la couche cachée, puis de cette dernière vers les sorties (Les liaisons sont entre neurones de couches différentes et immédiatement adjacentes uniquement). Chaque liaison d'un neurone caché vers un neurone de sortie est pondérée par un coefficient afin de moduler l'influence du premier sur le second. Enfin, chaque neurone caché utilise une RBF pour fonction d'activation. Nous utiliserons des gaussiennes.

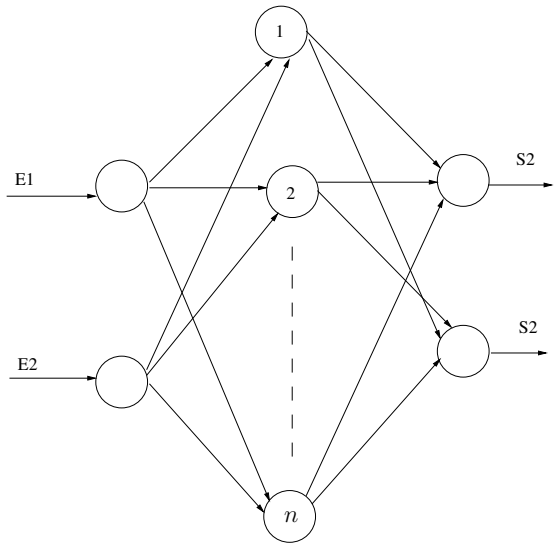

FIG. D.2 – *Exemple de réseau de neurones RBF* Le réseau présenté ici comporte 2 entrées (e_1 et e_2), deux sorties (s_1 et s_2) et n unités cachées.

D.2.2.2 Taux de recouvrement

Les centres et variances des gaussiennes définissent la couverture d'un réseau RBF, c'est-à-dire sa capacité à couvrir l'ensemble des valeurs de son espace d'entrée. Le taux de recouvrement d'un RBFN, noté τ, est définit de la manière suivante :

$$\tau = exp(-\frac{(\frac{\delta}{2})^2}{2\sigma^2}) \tag{D.2}$$

où δ est la distance séparant les centres des deux gaussiennes et σ^2 leur variance.

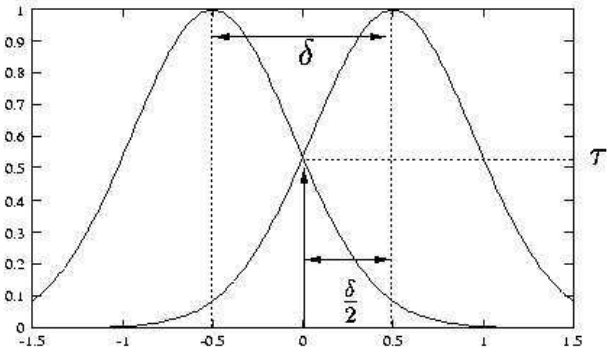

FIG. D.3 – Taux de recouvrement τ

Les valeurs de τ sont généralement comprises entre 60% et 90%, en fonction du nombre d'unités et de la fonction à approximer ([Renders, 95]). Il n'existe cependent pas de méthode exacte et cette valeur doit être déterminée expérimentalement.

D.2.2.3 Activation des unités

Neurones cachés Pour un réseau comportant n entrées et m unités cachées, l'activation des neurones cachés est définie par :

$$a_i = exp(-\frac{1}{2}\sum_{k=1}^{n}\frac{(e_k - \mu_{k,i})^2}{\sigma_{k,i}^2}) = \prod_{k=1}^{n} exp(-\frac{(e_k - \mu_{k,i})^2}{2\sigma_{k,i}^2})$$

où e_k est la $k^{ème}$ entrée, $\mu_{k,i}$ le centre de la gaussienne du $i^{ème}$ neurone caché correspondant à la $k^{ième}$ entrée et $\sigma_{k,i}^2$ sa variance.

Neurones de sortie L'activation d'un neurone de sortie est calculée quant à elle de la manière suivante :

$$a_i = \frac{\sum_{j=1}^{m} w_{i,j} a_j}{\sum_{j=1}^{m} a_j}$$

où a_j représente le niveau d'activation du $j^{ème}$ neurone caché et $w_{i,j}$ le poids entre ce neurone et la sortie i.

D.2.3 Propriétés

Les réseaux RBF présentent deux propriétés essentielles : la représentation locale de la connaissance appelée propriété de généralisation locale, et la capacité à approximer toute fonction continue.

Généralisation locale Chaque neurone caché ne s'active de manière significative que lorsque les valeurs présentées sur les neurones d'entrée sont proches du centre de sa gaussienne. Cette propriété de localité est primordiale. Contrairement à d'autres réseaux tels que les réseaux multi-couches, les réseaux RBF traitent l'information localement et non sur l'ensemble des neurones. Cela signifie que la modification d'un paramètre du réseau n'a qu'une influence locale et que la connaissance préalablement acquise ne va pas être remise en cause. Cette propriété permet au réseau d'apprendre incrémentalement de nouveaux cas.

Nous noterons que cela s'avère particulièrement intéressant pour un apprentissage en ligne tel que nous l'avons envisagé dans ce rapport. Cette propriété n'est par exemple pas vérifiée avec d'autres types de réseaux tels que les réseaux multi-couches entraînés par la méthode de rétropopagation du gradient décrite plus loin. Pour ces derniers, la solution consiste à conserver un échantillon représentatif des cas déjà rencontrés pour réentraîner le réseau à chaque présentation d'un nouveau cas ([Pomerleau, 93]). Cela pose des problèmes sur le choix des cas à conserver, leur nombre, et implique un nouvel apprentissage trop long pour être réalisé en ligne.

Approximateur universel Sous réserve de respecter la contrainte de couverture introduite précédemment, un réseau RBF est capable d'approximer toute fonction continue, c'est-à-dire définie sur l'ensemble de ses valeurs d'entrée. Si cette propriété n'était pas vérifiée, le réseau ne pourrait approximer qu'une version discrète du modèle. Ce serait par exemple le cas avec un réseau de neurones de type **CMAC** : Bien qu'en théorie le niveau de discrétisation puisse être ajusté en jouant sur la taille de ce type de réseau, en pratique l'augmentation du nombre de neurones implique une augmentation du nombre d'exemples d'apprentissages, là encore, non souhaitable dans notre cas.

D.2.4 Apprentissage utilisé dans ce rapport

L'apprentissage d'un réseaux RBF peut se faire selon les trois approches classiques citées précédemment, à savoir, en mode supervisé, semi-supervisé (dit par renforcement) ou non-supervisé. De plus, il peut porter sur les centres des gaussiennes, leurs variances, les poids vers les sorties ou une combinaison des 3.

Dans ce rapport, nous avons réalisé un apprentissage supervisé des poids uniquement (voir [Musawi et al., 92] pour une approche différente). Il y a trois raisons à cela :

- Nous disposons des données nécessaires à un apprentissage supervisé or ce dernier permet au réseau de converger plus vite vers une solution.

- La sortie est linéaire par rapport aux poids entre la couche cachée et la couche de sortie. Ceci a pour effet de réduire les risques de minima locaux (voir « rétropopagation du gradient »), et donc les risques de blocage en cas d'apprentissage des poids.
- Modifier uniquement les poids permet de conserver les centres et variances des gaussiennes, et par conséquent de toujours conserver la même couverture du réseau.

Considérant une couverture correcte, l'apprentissage des poids en mode supervisé, par la technique de descente de gradient, nous donne la fonction suivante de modification des poids :

$$\Delta W_{i,j} = -\eta \frac{\partial Q}{\partial W_{i,j}}$$

où η est une constante positive appelée le pas du gradient. Schématiquement, plus sa valeur est élevée et plus l'apprentissage d'un cas aura de l'importance. Des valeurs trop importantes peuvent conduire à des instabilités du réseau lorsque les valeurs d'apprentissage sont bruitées tel que c'est notre cas. En revanche, de trop petites valeurs ralentissent l'apprentissage. L'ajustement de cette constante se fait généralement par expérimentation, tout comme le taux de recouvrement.

En prenant un facteur de normalisation $R = \sum_{j=1}^{m} a_j$, on obtient la modification des poids suivante :

$$\Delta W_{i,j} = \frac{\partial Q}{\partial a_i} \frac{\partial a_i}{\partial W_{i,j}} = (a_i - s_i) \frac{a_j}{R}$$

Bibliographie

[Ahuactzin, 94] J. Ahuactzin. *Le Fil d'Ariane : Une Méthode de Planification Génerale. Application à la Planification Automatique de Trajectoires.* Ph.D Thesis, Thèse de doctorat. Inst. Nat. Polytechnique de Grenoble. Grenoble(FR), September 1994.

[Ahuactzin et Gupta, 99] Ahuactzin, Gupta. *The kinematic roadmap : A novel motion planning based approach for inverse kinematics of redundant manipulators.* In IEEE Transactions on Robotics and Automation, 15(5), 1999.

[Albus, 75] J. S. Albus. *A new approach to manipulator control : the cerebellar model articulation controller.* Trans. ASME, Journal Of Dynamic Systems, Measurement and Control, 97 :220-227, 1975.

[Arras et al., 02] K. Arras, J. Persson, N. Tomatis, and R. Siegwart. *Real-Time Obstacle Avoidance For Polygonal Robots With A Reduced Dynamic Window.* In Proc. of the IEEE Int. Conf on Robotics and Automation. Washington, DC, May 2002.

[Aseltine et al., 58] J.A. Aseltine, A.R. Mancini, C.W. Sarture *A Survey of adaptive Control Systems.* IRE Transaction on Automatic Control, PGAC-3, pp.102-108, 1958

[Avnaim et Boissonnat, 88] Avnaim, F et J.D.Boissonat. *Polygon placement under translation and rotation.* Technical Report, INRIA, Sophia-Antipolis, France. 1998.

[Avnaim et al., 88] F. Avnaim, J.D. Boissonnat and B.Faverjon. *A practical exact motion planning algorithm for polygonal objects amidst polygonal obstacles.* In Proc. of the IEEE Int. Conf. on Robotics and Automation, volume 3, pp 1656-1661, Philadelpha, USA. 1998.

[Agarwal, 97] Agarwal. *A systematic classification of neural network-based control.* IEEE Control Systems, 17(2) :75-95, 1997

[Barraquand et Latombe, 89] J, Barraquand and J.C. Latombe. *On non-holonomic mobile robots and optimal maneuvering.* Revue d'Intelligence Artificielle, 3(2) :77-103, 1989

[Bessiere et Ahuatzin, 93] P. Bessiere, J. Ahuatzin *The « Adriane's Clew Algoritm » : Global Planning with local methods.* In IEEE/RSJ Conf. on Intelligent Robots and Systems, 1993.

[Borenstein et Koren, 91] Borenstein and Koren, *The vector field histogram - fast obstacle avoidance for mobile robots.* IEEE Transactions on Robotics and Automation, (7)3 :278-288, 1991.

[Bobrow et al., 85] J.E. Bobrow, S. Dubowsky, and J.S. Gibson. *Time-optimal control of robotic manipulators along specified paths* Int. Journal of Robotics Research, 4(3) :3-17, Fall 1985.

[Borenstein et al., 96] Borenstein, Everette, Feng. *Where Am I ?* A.K.Peters Ltd. Wellesley, MA, 1996.

[Brady, 89] Brady, M. *System Development Foundation Benchmark Series.* In Robotics Science, 1989.

[Brock et Khatib, 99] Brock and Khatib. *High Speed Navigation using the global dynamic window approach.* In Proc. Int. Conf. on the Robotics and Automation, 1999.

[Brock et Khatib, 00] Brock and Khatib. *Real-time replanning in high-dimensional configuration spaces using sets of homotopic paths.* In Proc. Int. Conf. on the Robotics and Automation, pages 2328, San Francisco, USA, 2000.

[Broomhead et Lowe, 88] D.S. Broomhead and D. Lowe. *Multivariable functional interpolation and adaptive networks.* Complex Systems, 2 :321-355, 1988.

[Brooks, 83] Brooks, R.A. *Solving the find-path problem by good representation of free-space.* IEEE Transactions on Systems, Man and Cybernetics SMC-13(3), 190-197. 1983.

[Brooks et Lozano-Perez, 85] Brooks, R.A et T.Lozano-Pérez. *A subdivision algorithm in configuration space for findpath with rotation.* Proceedings of the International Conference on Systems, Man and Cybernetics SMC-15(2), 224-233. 1985.

[Burke, 85] Burke. *Applied differential geometry.* Cambridge University Press, 1985.

[Canny, 88] Canny. *The Complexity Of Robot Motion Planning.* Mit Press, Cambridge, MA(88), pp. 1988.

[Canny et al., 88] J. Canny, B. Donald, J. Reif, P. Xavier, *On the complexity of kinodynamic planning.* In Proc. of the IEEE Symp. on the Foundations of Computer Sciences, pages 306-316, White Plains, NY (USA), November 1988.

[Canny et al., 90] J. Canny, A. Rege, and J. Reif *An exact algorithm for kinodynamic planning in the plane.* In Proc. of the ACM Symp. on Computational Geometry, pages 271-280, Berkeley, CA (USA), November 1988.

[Cherif et Vidal, 98] Cherif, Vidal. *Planning handling operations in changing industrial plants.* In IEEE Int. Conf. on Robotics and Automation, 1998.

[Connolly et al., 90] C.I. Connolly, J.B.Burns, and R.Weiss. *Path Planning using Laplace's Equation.* In Proc. of the IEEE Int. Conf. on Robotics and Automation, volume 3, pp 2102-2106, Cincinatti OH, US. 1990.

[Connolly et Grupe, 94] C.I.Connolly, R.A.Grupen. *Nonholonomic path planning using harmonic functions.* Technical Report 94-50, University of Massachussets, Departement of Computer Science, Amherst, USA. 1994.

[Cormen et al., 90] Cormen, C. H., C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms.* McGraw-Hill, 1990.

[Donald et Xavier, 90] Donald and Xavier. *Provably good approximation algorithms for optimal kinodynamic planning for cartesian robots and open-chain manipulators.* In Proc. of the ACM Symp. on Computational Geometry, pages 290-300, Berkeley, CA(USA), 1990.

[Erdmann et Lozano-Perez, 86] M. Erdmann and Lozano-Perez, T. *On Multiple Moving Objects.* Algorithmica, (2) :pp.477-521, 1987, also appeared in Proceedings of IEEE International Conference on Robotics and Automation, San Francisco, CA, April 1986, pp.1419-1424.

[Featherstone, 97] R.Featherstone. *Swept bubbles : A method of representing swept volume and space occupancy.* Technical Report TR-90-069. Philips Laboratories, North Amercia Philips Corporation, Briarcliff Manor, New York 10510, August 21, 1990.

[Feder et Slotine, 97] Feder, H. and J-J.Slotine. *Real-time path planning using harmonic potentials in dynamic environments.* In Proc. of the Int. Conf. on Robotics and Automation. Vol 1, pp 874-811.

[Fiorini et Shiller, 93] Fiorini and Shiller. *Motion Planning in Dynamic Environments Using the Relative Velocity Paradigm.* In IEEE International Conference on Robotics and Automation, Atlanta GA, May 1993.

[Fiorini et Shiller, 96] Fiorini and Shiller. *Time Optimal trajectory planning in dynamic environments.* In Proc. of the IEEE Int. Conf. on Robotics and Automation, volume 2, pages 1553-1558, Minneapolis, Minnesota, April 1996.

[Fox et al., 97] Fox, D, W.Burgard and S.Thrun. *The dynamic window approach to collision avoidance.* IEEE Robotics and Automation Magazine 4(1), 23-33. 1997.

[Fox et al., 98] Fox, Burgard and Thurn. *A hybrid collision avoidance method for mobile robots.* In Proc. of the IEEE Intl. Conf. on Robotics and Automation. Leuven, Belgium, volume 2, pages 1238-43, 1998.

[Fraichard, 93] Fraichard. *Dynamic trajectory planning with dynamic constraints : a « state-time space » approach.* In Proc. of the IEEE-RSJ Int. Conf. on Intelligent Robots and Systems, volume 2, pages 1394-1400, Yokohama (JP), July 1993.

[Fraichard, 99] Th. Fraichard. *Planning in a Dynamic Workspace : a State-Time Space Approach* In Advance Robotics, 13(1) : 75-94, 1999.

[Fujimura et Samet, 89] K. Fujimura and H. Samet. *A hierarchical strategy for path planning among moving obstacles.* IEEE Trans. Robotics and Automation, 5(1) :61-69, February 1989.

[Fujimura et Samet, 90] K. Fujimura and H. Samet. *Motion planning in a dynamic domain* In Proc of the IEEE Int. Conf. on Robotics and Automation, pages 324-330, Cincinnatti OH (USA), May 1990.

[Garnier, 95] Garnier, Ph. *Controle d'exécution réactif de mouvements de véhicules en environnement dynamique et structuré.* Ph.D thesis, Inst. Nat. Polytechnique de Grenoble, Grenoble, FR, 1995.

[Gauthier, 99] E.Gauthier. *Utilisation des Réseaux de Neurones pour la Commande d'un véhicule Autonome.* Ph.D thesis. Inst. Nat. Polytechnique de Grenoble, 1999.

[Goldstein, 80] H. Goldstein, *Classical Mecanics*(2nd edition). Addison-Wesley. 1980.

[Gregory, 59] P.C. Gregory, Gregory ed. Proc. Self adaptive flight control Symposium, Wright air development center, Wright-Patterson air force base, OH, 1959.

[Hacking, 75] I. Hacking, *The emergence of probability.* Cambridge Univ. Press, Cambridge, UK, 1975.

[Haykin, 94] S. Haykin. *Neural Networks : A comprehensive foundation.* Macmillan College Publisher Comp., 1994.

[Haywards, 95] Haywards. *Efficient Collision Prediction Among Many Moving Objects.* The International Journal of Robotics Research, Vol. 14, No 2, pp. 129-143, April 1995.

[Hecht-Nielsen, 90] R. Hecht-Nielsen. *Neurocomputing.* Addison-Wesley, 1990.

[Hermosillo, 03] J. Hermosillo. *Motion Planning and Feedback Control of Bi-Steerable Robots an Approach based on Differential Flatness.* Ph.D. thesis. Inst. Nat. Polytechnique de Grenoble. 2003.

[Hwang et Ahuja, 92] Hwang and Ahuja. *Gross Motion Planning A Survey Advanced Robotics Redundancy and Optimisation.* ACM Computing Surveys, Addision Wesley Publishing Company volume 24, No. 3, September 1992.

[Jacobs et al., 89] P. Jacobs, G. Heinzinger, J. Canny, B. Paden. *Planning guaranted near-time-optimal trajectories for a manipulator in a cluttered workspace.* Research Report ESRC 89-20/RAMP 89-15, Engineering Systems Research Center, Univ of California, Berkeley, CA (USA), Octobober 1989.

[Kanayama et al., 91] Y. Kanayama, Y. Kimura, F. Myazaki, and T. Noguchi. *A stable tracking control method for a non-holonomic mobile robot.* In Proc. of the IEEE-RSJ Int. Workshop on Intelligent Robots and Systems, 6 :1236-1241, Osaka, JP, 1991.

[Kant et Zucker, 86] Kant and Zucker. *Towards efficeient trajectory planning : the path-velocity decomposition.* In Int. Journal of Robotics Research, 5(3) :72-89, Fall 1986.

[Khatib, 80] Khatib,O. *Commande Dynamique dans l'Espace Opérationnel des Robots Manipulators en Présence d'Obstacles.* Ph.D thesis, École Nationale Supérieur de l'Aeronautique et de l'Espace, Toulouse, France. 1980.

[Khatib, 86] Khatib,O. *Real-Time obstacle avoidance for manipulators and mobile robots.* International Journal Robtics Research 5(1), 90-98. 1986.

[Khatib et al., 97] M. Khatib, H. Jaouni, R. Chatila, and J.P. Laumond *Dynamic path modification for car-like non-holonomic mobile robots.* IEEE Int. Conf. on Robotics and Automation, Albuquerque, USA, 1997.

[Krause et Clark, 93] Krause, P. and Clark, D. *Representing Uncertain Knowledge.* Kluwer Academic Press, Dordrecht, NL, 1993.

[Ko et Simmons, 98] N. Y. Ko and R. Simmons. *The lane-curvature method for local obstacle avoidance.* In Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS), 1998.

[Koditschek, 87] Koditschek,D.E. *Exact robot navigation by means of potential functions : Topological considerations.* In Proceedings of the International Conference on Robotics and Automation, pp.1-6. 1987.

[Kowalski et al., 90] J. Kowalski, E. Hartman, and J. Keeler. *Layered neural networks with gaussian hidden units as universal approximators.* Neural Computation, 2 :210-215, 1990.

[Large et al., 00] F. Large, J. Hermosillo, S.Sekhavat, Ch. Laugier. *Using Artificial Neural Networks to Improve Sensor Based Maneuvers for a Car-Like Vehicle.* In Int. Conf. on Intelligent Autonomous Systems. Venezia, pp 1033-1040, 2000.

[Large et al., 00a] F. Large, S. Sekhavat, Ch. Laugier and E. Gauthier. *Towards Robust Sensor Based Maneuvres for a Car-Like Vehicle* IEEE Int Conf on Robotics and Automation. April 22-28. San francisco, US, 2000.

[Large et al., 02a] F. Large, S. Sekhavat, Z. Shiller and Ch. Laugier. *Towards Real-Time Global Motion Plannng in a Dynamic Environment Using the NLVO Concept.* In Proc. of the IEEE-RSJ Int. Conf. on Intelligent Robots and Systems. 30 september-4 october. Lausanne, Switzerland, 2002.

[Large et al., 02b] F. Large, S. Sekhavat, Z. Shiller and Ch. Laugier. *Using Non-Linear Velocity Obstacles to Plan Motions in a Dynamic Environment* Int. Conf. on Control, Automation, Robotics and Vision. 2-5 Dec, Singapore, 2002.

[Latombe, 90] J.C. Latombe. *Robot Motion Planning.* Standford University, 1990.

[Laumond, 98] J.P. Laumond. *Robot Motion Planning and control.* Springber-Verlag, 1998.

[LaValle, 98] LaValle. *Rapidly-Exploring Random Trees : A New Tool for Path Planning.* Technical Report No. 98-11, Dept. of Computer Science, Iowa State University, Oct. 1998.

[Lebeltel, 99] O.lebeltel. *Programmation bayésienne des Robots* Ph.D Thesis, Inst. Nat. Polytechnique de Grenoble, France 1999.

[Lee, 90] C.C. Lee. *Fuzzy logic in control systems : Fuzzy logic controller.* 20(2) :404-435, april 1990.

[Lee et Kil, 88] Sukhan Lee and Rhee M. Kil. *Multilayer feedforward potential function network.* In Proceedings of the Second International Conference on Neural Networks, pages 161-171, 1988.

[Lozano-Perez et Wesley, 79] T. Lozano-Pérez and M. A. Wesley. *An Algorithm for Planning Collision-Free Paths Among Polyhedral Obstacles.* Communications of the ACM (CACM), Volume 22, 1979 CACM 22(10) : 560-570 (1979).

[Lozano-Perez, 81] Lozano-Pérez. *Automatic path planning of manipulator transfer movements.* IEEE Trans. Systems, Man and Cybernetics, 11(10) : 681-698. 1981.

[Lozano-Perez, 83] Lozano-Pérez, T. *Robot Programming* Proceedings of the IEEE, Vol 71, July 1983, pp.821–841 (Invited).

[Mandani, 74] Mandani, EH. *Application of fuzzy algorithms for simple dynamic plants.* Proceedings IEEE, Vol. 121(12) :1585-1588, 1974.

[Megdassy, 61] P. Megdassy, *Decomposition of superposition of distributed functions.* Hungarian Academy of Sciences, Budapest, 1961.

[Minguez et al., 02] *Global nearness diagram navigation* J. Minguez, L. Montano, N. Siméon, R. Alami. In Proc. of the IEEE Int. Conf on Robotics and Automation. Washington, DC, May 2002.

[Minguez et Montano, 00] *Nearness diagram navigation. A new real-time collision avoidance approach.* J. Minguez, L. Montano. IEEE/RSJ Int. Conf on Intellingent Robots and Systems. Takamatsu, JP, 2000.

[Moody et Darken, 89] J. Moody and C. J. Darken. *Fast learning in networks of locally-tuned processing units.* Neural Computation, 1(2) :281-294, 1989.

[Musawi et al., 92] M. T. Musawi, W. Ahmed, K. H. Chan, K. B. Faris, and D. M. Hummels, *On the training of radial basis function classifiers.* Neural Networks, 5(4) :595-603, 1992.

[Nilsson, 69] Nilsson, N.J. *A mobile automaton : An application of artificial intelligence techniques.* In Proceedings of the Iternational Joint Conference on Artificial Intelligence, pp 509-520. 1969.

[Nilsson, 82] Nilsson. *Principles of Artificial Intelligence.* Springer, Berlin, 1982.

[Niyogi et Girosi, 96] P. Niyogi and F. Girosi. *On the relationship between generalization error, hypothesis complexity and sample complexity for radial basis functions.* Neural Computation, 8 :819-842, 1996

[Novalès, 94] C. Novalès. *Pilotage par actions réflèxes et navigation locale de robots mobiles rapides.* Ph.D thesis, Montpellier, FR, 1994.

[O'Dunlaing, 87] O'Dunlaing, C. *Motion planning with inertial constraints.* Algorithmica, 2 :431-475, 1987.

[O'Dunlaing, 88] O'Dunlaing, C. *A Tight Lower Bound For The Complexity Of Path-Planning For A Disc.* IPL(28), 1988, pp. 165-170.

[O'Dunlaing et Yap, 82] O'Dunlaing, C. and C.K. Yap *A retraction method for planning the motion of a disc.* Journal of Algorithms, 1(6) :104-111, 1982

[O'Rourke and Badler, 79] J. O'Rourke and N. Badler. *Decomposition of Three-Dimensional Objects Into Spheres.* IEEE Trans. On Pattern Analysis and Machine Intelligence, PAMI-1(3) :295–305, 417, July 1979.

[Park et Sandberg, 91] J. Park and I.W. Sandberg. *Universal approximation using radial basis function networks.* Neural Computation, 3(2) :246-257, Summer 1991.

[Park et Sandberg, 93] J. Park and I.W. Sandberg. *Universal approximation and radial basis function networks.* Neural Computation, 5(2) :305-316, 1993.

[Pek et al., 02] P. Pek, O. Lebeltel and Ch. Laugier. *Parking a Car using Bayesian Programming.* In Int. Conf. of Robotics, Automation and Computer Vision. Singapore, 2002.

[Pomerleau, 93] D.A. Pomerleau. *Neural network perception for mobile robot guidance.* Kluwer Academic Press, 1993.

[Pogio et Girosi, 90] T. Pogio and F. Girosi. *Networks for approximation and learning.* Proc. IEEE, 78(9) :1481-97, Sept 1990.

[Powel, 85] M.J.D. Powel. *Radial basis functions for multivariable interpolation : A review.* In IMA Conf. on Algorithms for the approximation of functions and data, pages 143-167, 1985.

[Quinlan, 94] Quinlan, S. *Efficient distance computation between non-convex objects* In Proceedings of the International Conference on Robotics and Automation, Volume 4, pp 3324-3329. 1994.

[Quinlan, 94a] Quinlan, S. *Real-time modiication of collision-free paths.* Ph.D thesis, Stanford University, dec. 1994.

[Quinlan et Khatib, 93] Quinlan, S. and Khatib, O. *Elastic Bands : connecting path plannin and control.* 2 :802-807, Atlanta, USA, 1993.

[Renders, 95] J.-M. Renders *Algorithmes génétiques et réseaux de neurones.* Hermes, 1995.

[Reif et Sharir, 85] J. Reif and M. Sharir. *Motion planning in the presence of moving obstacles.* In Proceedings of the IEEE Symp. on the Foundations of Computer Science, pages 144-154, Portland, OR (USA), Octobober 1985.

[Rimon et Koditschek, 92] Rimon, E et D.E.Koditschek. *Exact robot navigation in geometrically complicated but topologically simple spaces.* In Proceedings of the International Conference on Robotics and Automation, pp 1937-1942. 1990.

[Saffiotti, 87] Saffiotti, A. *An AI view of the treatment of uncertainty.* The Knowledge Engineering Review, 2(2) :75-97, 1987.

[Sahar et Hollerbach, 85] Sahar and Hollerbach. *Planning of minimum time trajectories for robot arms.* In Proc. of the IEEE Int. Conf. on Robotics and Automation, pages 751-758, St Louis, MI (USA), March 1985.

[Scheuer, 01] Scheuer, A. *Optimal continous-curvature trajectories for mobile robots.* Rapport de recherche. INRIA, France. 2001.

[Shih et al., 90] C.L. Shih, T.T. Lee, and W.A. Gruver. *Motion Planning with time-varying polyhedra obstacles based on graph search and methematical programming.* In Proc. of the IEEE Int. Conf. on Robotics and Automation, pp. 331-337, Cincinnatti, OH (USA), May 1990.

[Shiller et Dubowsky, 85] Shiller and Dubowsky. *On the optimal control of robotic manipulators with actuatorand end-effector constraints.* In Proc. of the IEEE Int. Conf. on Robotics and Automation, pp. 614-620, Saint Louis, MI (USA), March 1985.

[Shiller et Dubowsky, 88] Shiller and Dubowsky. *Global time optimal motions of robotic manipulators in the presence of obstacles.* In Proc. of the IEEE Int. Conf. on Robotics and Automation, pp. 370-375, Philadelphia, PA (USA), April 1988.

[Shiller et Dubowsky, 89] Shiller and Dubowsky. *Robot path planning with obstacles, actuator, gripper and payload constraints.* In Int. Journal of Robotics Research, 8(6) :3-18, December 1989.

[Shiller et aL., 01] Shiller, Large and Sekhavat. *Motion Planning in Dynamic Environments : Obstacles Moving Along Arbitrary Trajectories.* In Proc. of the IEEE Int. Conf. on Robotics and Automation, pp. 3716-3721, Seoul, Korea, May 2001.

[Shiller et Lu, 90] Z. Shiller and H.H. Lu. *Robust computation of path-constrained time-optimal motion.* In Proc. of the IEEE Int. Conf. on Robotics and Automation, pp. 144-149, Cincinnatti, OH (USA), May 1990.

[Schwartz et Shark, 83] J.T. Schwartz and M. Shark. *On the piano movers' problem : II. The general techniques for computing topological properties of real algebraic manifolds* Adv. in Appl. Math. 4 .1983 pp. 298-351.

[Simeon et al., 97] T. Siméon, S. Leroy et J.P. Laumond. *Computing good holonomic collision-free paths to steer nonholonomic mobile robots.* In Proceedings of the IEEE-RSJ Int. Conf. on Intelligent Robots and Systems, volume 2, pp 1004-1009, Grenoble, France. 1997.

[Simmons, 96] R. Simmons. *The curvature-Velocity Method for Local Obstacle Avoidance.* In Proc. of the IEEE Intl. Conf. on Robotics and Automation. Minneapolis, Minnesota, April 1996.

[Simon et al., 93] D. Simon, B. Espiau, E. Castillo, and K. Kapellos. *Computer aided design of generic robot controller handling reactivity and real-time control issues.* IEEE Trans. Control Systems Technology, pages 213-229, 1993.

[Smets, 95] Ph. Smets. *Probability, possibility, belief : which for what ?* In Procs. of the Wp. on Foundations and Applications of Possibility Theory, pages 20-40, Ghent, BE, 1995. World Scientific.

[Ulrich et Borenstein, 98] Ulrich, Borenstein. *VFH+ : Reliable Obstacle Avoidance for Fast Mobile Robots.* In IEEE Int. Conf. on Robotics and Automation. Leuven, Belgium, 1998.

[Ulrich et Borenstein, 00] I. Ulrich, J. Borenstein. *VFH* : Local obstacle avoidance with look-ahead verification.* In IEEE Int. Conf. on Robotics and Automation. San Francisco, USA, 2000.

[Xu et al., 94] L. Xu, A. Krzyzak, and A. Yuille. *On radial basis function nets and kernel regression : Statistical consistency, convergence rates and receptive field size.* Neural Networks, 7 :4 :609-628, 1994

[Yap, 87] *C. K. Yap. An O(n log n) algorithm for the Voronoi diagram of a set of simple curve segments.* Discrete Comput. Geom., 2 :365-393, 1987.

[Yoshizawa, 96] H. Yoshizawa, M. Wada and S.Mori. *Path Tracking Controle of a Mobile Robot using a Quadratic Curve.* In Proc. of the IEEE Int. Symp. on Intelligent Vehicles, vol 1, pp 58-63. Tokyo, 1996.

[Zadeh, 65] L.A. Zadeh. *Fuzzy Sets.* Information and Control. Vol 8, pg 338-352, 1965.

[Zhu et Latombe, 91] Zhu and Latombe. *New Heuristic Algorithms for Efficient Hierarchical Path Planning.* In Artificial Intelligence, North-Holland, Vol. 27, No. 1, pages 9-20, February 1991.