Navigation autonome en environnement dynamique : une approche par déformation de trajectoire

Vivien Delsart

UNIVERSITÉ DE GRENOBLE

Thèse

pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE

Spécialité : *Mathématiques et Informatique*

préparée au laboratoire LIG et l'INRIA Rhône-Alpes, dans le cadre de l'École doctorale *Mathématiques, Sciences et Technologies de l'Information, Informatique*

présentée et soutenue publiquement par

Vivien Delsart

le 11 octobre 2010

Titre :

Navigation autonome en environnement dynamique : Une approche par déformation de trajectoire

Directeur de thèse :

Thierry Fraichard

Composition du jury :

M.	James Crowley	Président
M.	Philippe Martinet	Rapporteur
M.	Thierry Siméon	Rapporteur
M.	Jacques Blanc-Talon	Examinateur
M.	Thierry Fraichard	Examinateur

Remerciements

Cette thèse est née d'une collaboration entre la Délégation Générale de l'Armement (*DGA*), qui l'a financée, et l'Institut National de Recherche en Informatique et Automatismes (*INRIA*) de Grenoble dans laquelle se sont déroulés mes travaux. Elle n'aurait jamais vu le jour sans le soutien et l'aide de nombreuses personnes à qui je tiens à exprimer toute ma reconnaissance.

Tout d'abord Christian Laugier, directeur de recherche à l'INRIA Grenoble pour m'avoir accueilli dans l'équipe-projet *e-Motion*.

Ensuite Thierry Fraichard, chargé de recherche INRIA, qui m'a encadré durant ces trois années avec efficacité, a toujours su se rendre disponible du premier jour jusqu'au dernier, a su me guider scientifiquement, tout en me laissant libre de mes choix. Même si parfois il était difficile de répondre à ses attentes, je tiens sincèrement à lui exprimer ma gratitude pour sa détermination et son aide, qui m'ont permis à maintes reprises de continuer à avancer.

Je tiens également à remercier les membres de mon jury : James Crowley, professeur à l'INPG pour avoir présider la soutenance de ma thèse. Jacques Blanc-Talon, responsable scientifique DGA, pour ses questions pertinentes sur mes travaux. Philippe Martinet, professeur à l'IFMA et Thierry Siméon, chargé de recherche du CNRS pour leurs remarques sur ce manuscrit.

Je ne voudrais surtout pas oublier de citer ici l'ensemble des membres de l'équipe *e-Motion*, qui m'ont accompagné tout au long de cette aventure. Je pense en particulier à Christophe et Amaury, pour m'avoir fait intégrer l'équipe. A Luis et Manu, pour toutes ces discussions qui m'ont permis d'avancer. A mes collègues de bureau : Stéphanie, Antoine, Frank, Sara, pour leur sympathie qui m'a permis de rendre cette expérience scientifique, mais aussi humaine, fort enrichissante. A Mao, Jorge, Dung, Chiara, Christopher, et tous ceux que j'aurai oublié, pour avoir contribué de près ou de loin à l'avancement de ces travaux.

Enfin je souhaite remercier l'ensemble de ma famille et de mes proches. Je pense particulièrement à mes parents, à Thib, et à Maud, qui m'ont toujours soutenu aussi bien dans les bons moments que dans les difficiles, et qui m'ont donné le courage de mener cette thèse à bien, jusqu'au bout.

Table des matières

1 Introduction à la navigation en robotique mobile 9
1.1 La robotique mobile 9
1.2 Problématique 10
1.3 Approches de navigation existantes 12
1.4 Contribution 14
1.5 Plan du mémoire 16

2 Approches de Navigation en Environnement Dynamique 17
2.1 Introduction 17
2.2 Approches délibératives 18
2.2.1 Définition du problème 18
2.2.2 Méthodes par graphes 20
2.2.3 Méthodes par arbres 28
2.2.4 Complexité des approches délibératives 31
2.3 Approches réactives 32
2.3.1 Approches par champs de potentiel 32
2.3.2 Histogramme de champs de vecteurs 33
2.3.3 Navigation par diagrammes de proximité 35
2.3.4 Méthode de navigation courbure-vélocité 36
2.3.5 Fenêtre dynamique 37
2.3.6 Représentation des obstacles dans l'espace des vitesses 38
2.3.7 Navigation basée sur les états de collisions inévitables . 40
2.3.8 Planification de mouvement partiel 41
2.3.9 Défaut de convergence vers le but 42
2.4 Déformation de mouvement 42
2.5 Conclusion 43

3 Déformation de Trajectoire 45
3.1 Introduction 45
3.1.1 Motivations 45
3.1.2 Contribution 46

3.1.3 Plan du chapitre 46
3.2 Déformation de mouvement : approches existantes 47
3.2.1 Bande élastique de Khatib 47
3.2.2 Carte de route élastique de Brock 50
3.2.3 Déformation variationnelle de Lamiraux 50
3.2.4 Optimisation de gradient de Ratliff & Zucker 51
3.2.5 Planification multi-véhicules par déformation 52
3.2.6 Limitations des approches précédentes 53
3.3 Teddy : Déformation de trajectoire 55
3.3.1 Notation & définitions 55
3.3.2 Principe de la déformation de trajectoire 55
3.3.3 Algorithme de déformation `Teddy` 56
3.3.4 Forces répulsives 58
3.3.5 Forces élastiques 60
3.3.6 Maintien de la connectivité de la trajectoire 61
3.3.7 Rééchantillonnage de la trajectoire 68
3.3.8 Vérification de la validité de la trajectoire 68
3.4 Résultats en simulation 70
3.4.1 Spécifications de l'implémentation 70
3.4.2 Cas d'étude 1 : masse ponctuelle 72
3.4.3 Cas d'étude 2 : robot différentiel 76
3.4.4 Cas d'étude 3 : robot de type voiture 79
3.5 Conclusion et perspectives 81

4 Génération de Trajectoires avec Contrainte sur le Temps Final 83
4.1 Introduction 83
4.1.1 Motivations 83
4.1.2 Contributions 84
4.1.3 Plan du chapitre 85
4.2 Génération de mouvement : Approches existantes 85
4.2.1 Combinaison de primitives géométriques 85
4.2.2 Résolution d'un problème aux limites 88
4.2.3 Recherche d'un contrôle optimal 91
4.2.4 Contrainte sur le temps final dans les approches existantes 93
4.3 Problème de la génération de trajectoires 95
4.3.1 Génération de trajectoires "classique" 95
4.3.2 Ajout de la contrainte sur le temps final 96
4.3.3 Contrainte sur le temps final et atteignabilité 96
4.4 Génération de trajectoire par méthode variationnelle 98
4.4.1 Représentation paramétrique du contrôle 98
4.4.2 Modification itérative de la trajectoire 99

4.5 Tiji : Générateur de trajectoire avec contrainte sur le temps final 100
4.5.1 `Tiji` : Présentation de l'algorithme 101
4.5.2 Détermination des paramètres initiaux de la trajectoire 102
4.5.3 Calcul d'une paramétrisation admissible de la trajectoire 103
4.5.4 Mise à jour de la paramétrisation de la trajectoire . . . 107
4.6 Cas d'étude : Robot différentiel 110
4.6.1 Modèle du système 111
4.6.2 Représentation paramétrique du contrôle 111
4.6.3 Choix des paramètres initiaux 112
4.6.4 Détermination d'une trajectoire faisable 112
4.6.5 Calcul d'une correction sur les paramètres 117
4.7 Résultats en simulation 117
4.7.1 Cas 1 : Etats-temps buts atteignables 117
4.7.2 Cas 2 : Etats-temps buts non atteignables 120
4.8 Autres cas d'étude 122
4.8.1 Véhicule de type voiture 123
4.8.2 Vaisseau spatial 123
4.8.3 Cas d'étude 1 : Etats-temps but atteignables 124
4.8.4 Cas d'étude 2 : Etats-temps buts non atteignables . . . 125
4.9 Conclusion et Perspectives 127

5 Application à une chaise roulante automatisée 131
5.1 Introduction 131
5.2 Chaise roulante automatisée : le système 131
5.2.1 Architecture matérielle 132
5.2.2 Architecture logicielle 133
5.2.3 Modélisation du système robotique 136
5.3 Mise en place des différentes tâches de navigation 137
5.3.1 Localisation et cartographie 137
5.3.2 Détection et suivi des obstacles mobiles 138
5.3.3 Construction d'un modèle prévisionnel de l'environnement 139
5.3.4 Planification de mouvement 140
5.3.5 Evitement d'obstacles réactif 141
5.3.6 Exécution du mouvement 141
5.3.7 Gestion des ressources et ordonnancement des tâches . 142
5.4 Evaluation de l'approche de déformation de trajectoire 143
5.4.1 Conditions expérimentales 143
5.4.2 Scénario 1 : déformation au milieu d'obstacles statiques 145
5.4.3 Scénario 2 : cisaillement 147
5.4.4 Scénario 3 : multiples obstacles 149
5.5 Analyse et discussion des résultats 152

6 Conclusions et Perspectives Générales **155**
6.1 Contributions 155
6.2 Perspectives et travaux futurs 157

Chapitre 1

Introduction à la navigation en robotique mobile

1.1 La robotique mobile

Alors que depuis soixante dix ans, Asimov nous fait rêver de machines d'acier totalement autonomes, dotées de cerveaux positroniques non dépourvus de compassion, peut-être plus intelligents et rationnels qu'un individu lambda, l'état actuel de nos connaissances nous laisse encore bien loin d'imaginer le jour où de tels engins circuleront parmi nous. Et pourtant, de sérieux progrès ont été réalisés durant ces cinquante dernières années.

Un robot tel que nous le définissons aujourd'hui est un dispositif mécanique complexe équipé de capteurs et d'actionneurs conçu pour opérer des tâches complexes de manière autonome ou supervisée. Qu'il s'agisse de tâches pénibles, répétitives, dangereuses ou trop précises pour l'homme, les divers secteurs d'application de la robotique ne tendent qu'à s'élargir : initialement introduite dans l'industrie manufacturière, la robotique s'est étendue aux secteurs spatiaux (robots d'exploration sur mars), militaires (drônes de reconnaissance et de surveillance, véhicules de contre-minage, de transports d'équipement et d'évacuation sanitaire), médicaux (chirurgie de précision), domestiques (aspirateurs, tondeuses automatiques) ou encore des loisirs (robots sociaux). Autant d'enjeux économiques ont permis à la recherche en robotique de se perfectionner durant toutes ces années.

Dans tous les cas, quel que soit le système robotique, la tâche qui lui incombe nécessite l'action et le mouvement. Les premiers robots industriels évoluaient dans des environnements clos, parfaitement connus et totalement contrôlés, évitant ainsi tous risques de collisions. Les nouveaux systèmes tels les véhicules intelligents (Cybercars) sont désormais destinés à naviguer en

environnement dynamique et incertain, au milieu d'obstacles mobiles tels des piétons ou d'autres véhicules. La difficulté est donc d'autant plus importante qu'il est impératif d'assurer la sécurité du mouvement dans de tels environnements.

1.2 Problématique

Cette thèse s'intéresse à la navigation de véhicules autonomes en environnement dynamique et incertain. Pour ce faire, un système robotique se doit de répondre aux trois questions ci-dessous :

- **Où suis-je et qu'y a t-il autour de moi ?**

  Avant même de pouvoir commencer à se déplacer, un robot doit être capable de **se localiser** par rapport à un repère fixe attaché à l'environnement dans lequel il évolue. De plus, lorsque ce dernier est inconnu, le robot doit être capable de **cartographier** les obstacles qui l'entourent afin, d'une part de les éviter, et d'autre part de disposer de points de repère pour améliorer sa localisation. Ces deux étapes sont assurées au cours de la navigation du système robotique à l'aide des capteurs *proprioceptifs* fournissant les informations internes sur l'état du robot (odométrie, gyroscopes, etc), et de capteurs *extéroceptifs* donnant accès à des informations sur le monde extérieur (capteurs de contact, visuels, lasers, GPS, etc).

- **Comment rejoindre mon but à partir de ma position initiale ?**

  Une fois le robot capable de se situer dans son propre environnement, il lui faut déterminer un chemin lui permettant de rejoindre son but. Par exemple si le système évolue dans un labyrinthe ou dans un réseau routier urbain, il lui faut trouver la route à prendre puis adapter ensuite son mouvement au gré des obstacles pouvant se dresser devant lui. La décision du mouvement à entreprendre s'effectue donc généralement, dans le cadre d'un environnement dynamique, en deux temps. Tout d'abord, si le robot dispose d'une connaissance à priori sur le monde qui l'entoure, il peut alors **planifier** un mouvement complet lui permettant de rejoindre son but. Ensuite, au cours de l'exécution du mouvement, il devra être capable d'**éviter les obstacles** inattendus ou dont le comportement varie en adaptant son mouvement.

– **Comment exécuter ce mouvement ?**

Une fois la trajectoire à suivre déterminée, il est nécessaire de pouvoir **contrôler** le robot c'est à dire d'être capable de lui délivrer au cours du temps la commande à envoyer sur ses actionneurs permettant de suivre cette trajectoire. Du fait de l'incertitude qui caractérise aussi bien la localisation du système que la perception des distances aux obstacles ou encore la commande envoyée, celle-ci doit être recalculée à chaque instant afin de s'adapter aux informations perçues sur l'état du système et de l'environnement et à la décision prise par les modules de planification et d'évitement d'obstacles.

Parmi ces différentes tâches, notre problématique est centrée sur la partie décisionnelle de la navigation, à savoir la planification de mouvement et l'évitement d'obstacles. Axée sur la navigation pour un véhicule mobile (par exemple une voiture) en environnement dynamique, le choix d'un mouvement à entreprendre nécessite la considération des contraintes suivantes : tout d'abord le robot doit prendre en compte sa propre **géométrie**. Dans le cas d'une voiture décidant de s'engager dans un passage étroit, ou dont l'objectif est de se garer dans une place de parking, celle-ci devra s'assurer qu'elle est capable de s'insérer au milieu des obstacles qui l'entourent.

Ensuite il est nécessaire de respecter les **contraintes cinématiques et dynamiques** du système. Celles-ci représentent l'ensemble des contraintes restreignant le mouvement du système robotique. Par exemple, une voiture dispose d'une vitesse et d'une accélération maximale, ainsi que d'un angle de braquage borné limitant ses manoeuvres. Si elle dispose d'un temps restreint pour atteindre un point de passage (*e.g.* traverser un carrefour régit par des feux tricolores), elle devra tenir compte de ces contraintes afin de déterminer si elle peut s'engager et passer le carrefour ou si elle doit ralentir et s'arrêter au feu rouge.

Enfin lorsque le système robotique évolue dans un milieu inconnu ou au milieu d'obstacles mobiles, il doit considérer la **dynamique de l'environnement**. Pour cela le robot doit être apte à réagir, lors de la découverte d'obstacles non perçus précédemment, ou de changement de comportement d'un obstacle mobile s'apprêtant à obstruer le passage par lequel le robot souhaitait passer. Le système doit alors pouvoir décider d'un nouvel itinéraire à prendre, ou au moins être capable de se mettre hors de danger, et ne pas être dangereux pour les autres agents de l'environnement, en s'arrêtant ou en changeant de direction. Une **contrainte sur le temps de décision** doit alors être prise en compte : en effet, l'environnement peut changer à tout instant, le système robotique dispose donc d'un temps limité pour adapter

son comportement à celui des obstacles. Cette limite de temps dépend de la vitesse du système et des obstacles ainsi que de leur proximité.

Pour résumer, notre objectif consiste à déterminer rapidement des mouvements sans collision, faisable par le robot considéré, s'adaptant à l'évolution de son environnement, et lui permettant de rejoindre un but prédéfini.

1.3 Approches de navigation existantes

Les approches de détermination de mouvement ont initialement été développées suivant deux modèles bien distincts (*cf.* chapitre 2) : d'un côté, les approches dîtes délibératives consistent à déterminer un mouvement complet vers le but à partir d'un modèle de l'environnement aussi complet que possible. Le problème de telles approches est que leur complexité inhérente les empêche en général de pouvoir être calculées en temps limité. A l'opposé se trouvent les méthodes dîtes réactives calculant à chaque instant le mouvement à appliquer au prochain pas de temps à partir des informations collectées par les capteurs du système. La *convergence vers le but* de telles méthodes peut se révéler problématique. Par exemple, un robot entrant dans une impasse sans issue ne serait pas nécessairement capable de ressortir de cette impasse en ne raisonnant que sur le mouvement à suivre au pas de temps suivant. Afin de palier les défauts de ces deux catégories, de nombreuses méthodes de navigation développées ces dernières années essaient de combiner une planification délibérative avec un évitement d'obstacles réactif.

Parmi ces approches intermédiaires, nous nous sommes intéressés à la *déformation de chemin* introduite par Khatib en 1993 ([QK93]). Un chemin représente la courbe géométrique de l'espace par où passe le robot. Le principe de la *déformation de chemin* est alors le suivant : Un chemin complet jusqu'au but est calculé à priori et fourni au système robotique. Au cours de l'exécution, la partie du mouvement restant à être exécutée est déformée continuellement comme un élastique, en réponse aux informations sur l'environnement récupérées par les capteurs. Le système peut ainsi modifier son parcours en fonction du déplacement d'obstacles ou de l'imprécision et l'incomplétude de sa connaissance de l'environnement. La déformation résulte en général de deux type de forces : des forces externes dues à la proximité des obstacles, et des forces internes destinées à maintenir la faisabilité et la *connectivité du mouvement* (un chemin calculé entre une configuration initiale et une configuration finale est dit connecté s'il lie ces deux configurations par un mouvement continu respectant les contraintes du système robotique). Tant que la connectivité du chemin est maintenue,

la convergence vers le but est assurée.

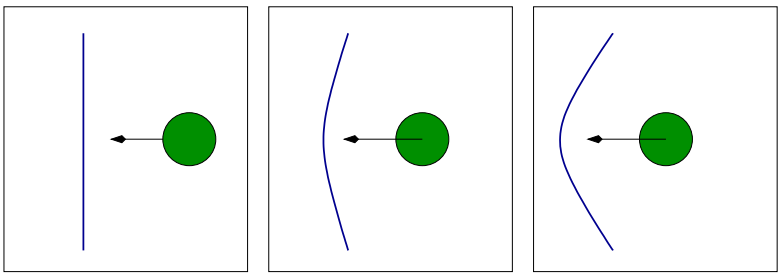


FIGURE 1.1 – Problème lié à la déformation de chemin : En réponse à l'approche d'un objet mobile qui coupe le chemin suivi par le robot, le chemin est continuellement déformé. Si elle est trop importante, une telle déformation risque de rendre le système robotique incapable de suivre le chemin, alors qu'il aurait été beaucoup plus simple de laisser passer l'obstacle.

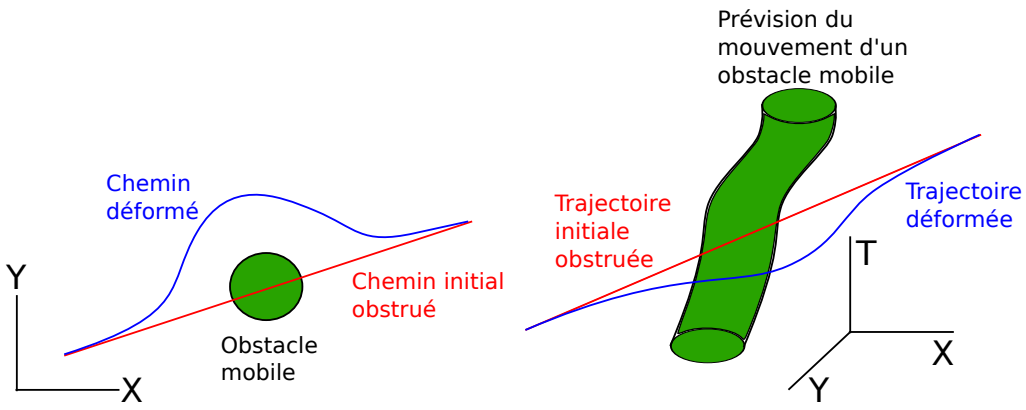
La déformation de chemin souffre néanmoins d'une forte contrainte : les déformations sont limitées à des homotopies (déformations continues) du chemin de départ, pouvant ainsi conduire à des comportements inappropriés. Imaginez par exemple un obstacle souhaitant couper le chemin suivi par le robot. Une simple déformation de chemin déformerait ce dernier comme un élastique (*cf.* Fig. 1.1) alors qu'il serait préférable de jouer sur la vitesse du robot, en ralentissant afin de laisser passer l'obstacle.

Comment obtenir un tel comportement ? Pour ce faire, il est nécessaire non seulement de considérer la géométrie des obstacles, mais également de **raisonner sur le futur** de l'environnement. En considérant la dimension temporelle, l'étude du mouvement du robot et des obstacles au cours du temps peut aider la prise de décision du mouvement à suivre. Modifier le chemin suivi par le robot, non seulement dans l'*espace* mais aussi dans le *temps*, lui permet ainsi d'adapter sa vitesse au cours du temps et de choisir par *où* passer et à quel *instant* afin d'**anticiper** le mouvement des obstacles de l'environnement. Une connaissance à priori du mouvement des différents agents circulant autour du système robotique est cependant nécessaire pour adopter un tel comportement. Sans disposer de dons de prescience il est possible d'inférer un modèle prévisionnel du comportement des obstacles (*cf.* annexe 1), soit à partir d'observations simples (*e.g.* un piétons se trouvant à trois cents mètres du robot ne risque pas de se trouver sous ses roues dans les cinq prochaines secondes), soit à partir d'apprentissage de comportements des agents et du contexte (*e.g.* un piéton souhaitant traverser une chaussée à trois voies empruntera certainement le passage pour piétons pour cela et lorsque son feu sera vert). Même si la prédiction se révèle inexacte ou incertaine,

elle peut néanmoins aider le robot à prendre une décision sur les actions à entreprendre. Un automobiliste dispose de telles connaissances à priori. En munissant un système robotique d'informations similaires, il lui sera possible de choisir plus intelligemment quel comportement adopter face aux agents qui l'entourent.

De plus en plus d'approches de navigation prennent en compte un modèle prévisionnel du comportement des obstacles [HKLR02, LLS05, OM06, MGF09]. Nous souhaitons ici en faire de même dans le cadre de la déformation de mouvement.

1.4 Contribution



(a) Déformation de chemin (b) Déformation de trajectoire

FIGURE 1.2 – Déformation de chemin vs. déformation de trajectoire : alors que la déformation de chemin se contente de déformer la courbe géométrique de l'espace $\mathbb{R}^2 = X \times Y$ suivie par le robot pour éviter un obstacle, la déformation de trajectoire modifie celle-ci également dans le temps T en utilisant une prévision du mouvement des obstacles afin d'anticiper leur futurs mouvements.

Afin de pouvoir anticiper le mouvement d'obstacles mobiles dans le cadre de la déformation de mouvement, il est nécessaire de quitter le paradigme de *déformation de chemin* pour celui de *déformation de trajectoire*. Une *trajectoire* est simplement un chemin géométrique paramétré par le temps. Elle est généralement exprimée dans l'espace d'états-temps, où un *état* représente à la fois la configuration du système (ensemble de paramètres indépendants décrivant de manière unique la position et l'orientation d'un système robotique) à un instant donné, mais également des informations sur son mouvement (sur sa vitesse, par exemple). En exerçant des forces similaires aux

approches précédentes sur la trajectoire à partir du modèle prévisionnel du futur des obstacles, il est alors possible de modifier cette trajectoire à la fois dans l'*espace* et dans le *temps* (*cf.* Fig. 1.2). De telles déformations permettent alors de faire adopter au robot de bien plus nombreux comportements que le schéma de déformation de chemin initial : le temps auquel les configurations successives du robot sont atteintes et la vitesse le long de cette trajectoire pouvant être altérés, le robot peut alors simplement contourner un obstacle pour garder une certaine distance de sécurité vis à vis de lui, accélérer devant un obstacle avant qu'il ne croise son chemin, ou au contraire décélérer afin de laisser passer l'obstacle et reprendre sa course derrière lui.

Une des principales difficultés de notre approche de déformation de trajectoire est le *maintien de la connectivité* de cette trajectoire (maintien d'une trajectoire continue respectant les contraintes sur le mouvement du système robotique et liant son état initial à son état but). La trajectoire étant librement modifiée dans l'espace-temps, les contraintes dynamiques du robot risquent de ne plus être respectées le long de celle-ci. Afin de s'assurer du maintien de la connectivité de la trajectoire déformée et de sa restauration lorsqu'elle a été perdue, cette dernière est discrétisée en une séquence finie d'états-temps (états du système définis à des instants fixés). La connectivité est alors assurée si chaque état-temps est *atteignable* par son prédécesseur. La caractérisation de l'*espace atteignable* à partir d'un état-temps donné, *i.e.* de l'ensemble de tous les mouvements faisables à partir de celui-ci est cependant bien trop coûteuse pour un système robotique complexe. Nous aimerions alors au moins vérifier s'il existe un mouvement faisable liant chaque couple d'état-temps successifs de la trajectoire déformée. Pour cela, un *générateur de trajectoire* nous a été nécessaire. Le but de celui-ci est d'une part d'essayer de déterminer une trajectoire faisable entre deux états-temps de la trajectoire déformée, et d'autre part de trouver une trajectoire respectant toutes les contraintes sur le mouvement du système et se "rapprochant autant que possible" de l'état-temps but dans le cas contraire. A notre connaissance, un tel générateur de trajectoire n'existait pas dans la littérature. Nous avons alors développé un algorithme, basé sur une méthode variationnelle de génération de mouvement afin de répondre à cette attente.

Pour résumer, les contributions de cette thèse se constituent de :

- Une méthode de déformation de trajectoire nommée `Teddy` (Trajectory Deformer) prenant en compte une estimation à-priori du mouvement futur des obstacles mobiles.
- Une approche de génération de trajectoire nommée `Tiji` (Trajectory Generator) permettant de déterminer un mouvement entre deux états donnés à des temps fixes ou bornés, respectant toutes les contraintes

du système robotique et s'arrêtant "aussi près que possible" du but lorsqu'il n'est pas atteignable.
– Enfin, la méthode de déformation a été intégrée à une architecture de navigation complète et expérimentée sur une chaise roulante automatisée.

1.5 Plan du mémoire

Ce mémoire est donc organisé comme suit : Un état de l'art non exhaustif des approches de détermination de mouvement en environnement dynamique est tout d'abord proposé au Chapitre 2. L'approche de déformation de trajectoire `Teddy` est ensuite présentée au Chapitre 3 et suivie du générateur de trajectoire `Tiji` au Chapitre 4. L'intégration à une architecture complète de navigation et les expérimentations sur une chaise roulante automatisée sont détaillées au Chapitre 5. Enfin une discussion des résultats obtenus et des perspectives envisageables concluent ce manuscrit au Chapitre 6.

Chapitre 2

Approches de Navigation en Environnement Dynamique

2.1 Introduction

Nos travaux s'intéressent donc à la navigation pour un robot mobile en environnement dynamique. Soit un système robotique équipé de capteurs lui permettant de percevoir son environnement et d'actionneurs lui permettant de se déplacer. Notre but consiste alors à déterminer un mouvement du robot qui :

- respecte les contraintes sur le mouvement du système robotique.
- mène à un but prédéterminé.
- évite toutes collisions avec les obstacles au cours de la navigation.

La détermination du mouvement pour un robot automatisé a été largement abordée au cours de ces cinquante dernières années. On distinguait habituellement deux grandes catégories d'approches : les approches délibératives et les approches réactives. Le principe des approches délibératives est de déterminer un mouvement complet (un chemin ou une trajectoire) du robot entre une position initiale et une position finale à partir d'un modèle de l'environnement dans lequel évolue le système aussi complet que possible. Toutefois, la complexité inhérente à ces approches peut empêcher le calcul du mouvement en temps réel. Les approches réactives quant à elles calculent uniquement le mouvement à appliquer au prochain pas de temps à partir de données capteurs récupérées par le système robotique à chaque instant. Une représentation de l'environnement est ainsi construite au fur et à mesure du déplacement : la navigation est donc possible en environnement incertain comme en environnement dynamique.

La convergence vers le but du système est néanmoins difficile à garantir. En effet ces approches sont sujettes à des minima locaux dont il peut être difficile de sortir sans connaître le chemin global permettant de rejoindre le but.

Au regard des progrès réalisés dans le domaine de la navigation durant ces dernières années tel le DARPA Challenge 2007, la nécessité de disposer à la fois des capacités des approches réactives et délibératives s'est confirmée. De nombreux travaux actuels combinent désormais un planification globale et un évitement d'obstacle local afin de palier les défauts de chacune de ces deux catégories d'approches.

Afin de pouvoir introduire l'approche de déformation de trajectoire constituant le sujet de ce manuscrit, une classification standard (délibérative/réactive) a été conservée (Section 2.2 et 2.3). Néanmoins nous soulevons les efforts opérés ces dernières années afin de combler le manque de convergence des approches réactives et l'efficacité des approches délibératives. Cet état de l'art ne se veut pas exhaustif, néanmoins il est constitué des travaux ayant pu inspirer les nôtres ou ayant servi d'outils nécessaires à la mise en place de nos expérimentations. Le lecteur peut se référer aux ouvrages [Lat91, Lav06] pour un état de l'art plus complet des méthodes de navigation.

2.2 Approches délibératives

2.2.1 Définition du problème

Les approches dîtes délibératives consistent à résoudre un problème de *planification de mouvement*. La planification de mouvement est la détermination à priori d'une stratégie de mouvement entre une position initiale et une position finale du robot à partir d'un représentation de l'environnement dans lequel il évolue.

Initialement motivée par l'utilisation de bras manipulateurs dans l'industrie manufacturière (*cf.* Fig. 2.1), la planification pour de tels systèmes robotique disposant de multiples degrés de liberté cherchait à déterminer la séquence de poses (ou configurations) du robot permettant à son élément terminal (partie du robot destinées à "manipuler" d'autres objets afin de remplir sa tâche) de rejoindre une position but sans entrer en collision avec les objets se trouvant à portée du robot. Formellement, une *configuration* q est définie par l'ensemble des paramètres indépendants définissant de manière unique la position et l'orientation de chaque partie du robot. L'*espace de configurations* $\mathbf{C}$ défini l'ensemble de toutes les configurations possibles du système

FIGURE 2.1 – Bras manipulateurs dans une industrie automobile

robotique.

Parmi toutes les configurations constituant $\mathbf{C}$, certaines d'entre elles caractérisent une pose du robot telle que l'un des éléments qui le composent se trouve en collision avec les obstacles. On note généralement $\mathbf{C}_{obs}$ l'ensemble de toutes les configurations en collision et son complément $\mathbf{C}_{libre} = \mathbf{C} \setminus \mathbf{C}_{obs}$ l'ensemble des configurations libres. Le problème basique de planification de mouvement consiste alors à déterminer un *chemin* $\Psi_{\mathbf{C}}$ (*i.e.* une séquence continue de configurations entre une configuration initiale q_{init} et une configuration finale q_{but}), tel que toutes configurations q de $\Psi_{\mathbf{C}}$ appartiennent à $\mathbf{C}_{libre}$.

Souhaitant déterminer un mouvement sans collisions pour un robot évoluant dans un environnement dynamique caractérisé par des obstacles mobiles, une simple *planification de chemin* ne suffit pas : en effet, la position de ces obstacles évoluant au cours du temps, il est nécessaire de planifier une *trajectoire i.e.* un chemin paramétré par le temps afin de s'écarter des obstacles au cours du temps. Celle-ci indique donc par quelles *configurations* le système passe, mais également à quels *instants* et à quelles *vitesses*. Une *trajectoire* est généralement exprimée dans l'espace d'*états-temps* où un *état* s représente à la fois la configuration prise par le robot à cet instant, mais également sa vitesse instantanée. Une représentation courante de l'espace d'états est l'espace de phase donné par $(q, \dot{q})$.

Un robot est néanmoins soumis à un certain nombre de contraintes. Tout d'abord, les *contraintes cinématiques* limitent le mouvement du

système robotique. On distingue généralement deux types de contraintes cinématiques :

- Les *contraintes holonomes* limitent l'ensemble des configurations pouvant être prises par le robot. Par exemple, un bras manipulateur tenant un verre rempli d'eau et ayant pour tâche de ne pas le renverser ne pourra pas passer par une configuration à laquelle le verre se retrouve à l'envers.
- Les *contraintes non-holonomes* limitent le mouvement instantané du système mais pas l'ensemble des configurations qu'il peut occuper. Par exemple une voiture disposant d'un angle de braquage maximal ne pourra pas se translater sur le côté. Par contre, en effectuant une manoeuvre, elle pourra néanmoins rejoindre cette position.

Ensuite les *contraintes dynamiques* sont des contraintes sur l'état du système robotique et limitent l'évolution de son mouvement au cours du temps. Par exemple une voiture à l'arrêt dispose d'une accélération maximale l'empêchant d'atteindre sa vitesse maximale instantanément.

Le problème étant posé, nous présentons ici un bref aperçu des principales approches délibératives ayant retenu notre attention. Elles peuvent principalement être regroupées en deux catégories :

- méthodes d'exploration d'un graphe de recherche.
- méthodes incrémentales de construction d'un arbre de recherche.

Celles-ci sont décrites ci-dessous.

2.2.2 Méthodes par graphes

Le principe de ces méthodes est de tenter de capturer la topologie de l'espace de recherche (espace de configuration ou espace d'état du système robotique) dans le but de simplifier le problème à une recherche dans un graphe. Elles sont donc constituées de deux étapes :

- Construction du graphe dans l'espace de recherche approprié
- Parcours du graphe dans le but de déterminer un chemin ou une trajectoire entre les configurations initiale et finale

Le parcours du graphe s'effectue la plupart du temps de la même manière : un algorithme heuristique tel A* [HNR68] est utilisé dans le but d'éviter l'exploration complète de l'espace de recherche. La construction du graphe peut, quant à elle, fortement varier : alors que les premiers travaux de planification s'intéressaient à trouver un chemin pour des systèmes dépourvus de contraintes sur leur mouvement ou pouvant se déplacer dans toutes les directions (systèmes holonomes), les recherches actuelles prennent en compte les contraintes cinématiques et dynamiques des robots étudiés, et planifient

des trajectoires dans l'espace-temps afin de considérer le mouvement futur des obstacles mobiles. La représentation de l'espace de recherche, s'est en conséquence adaptée à ces évolutions.

Une notion importante permettant d'évaluer la "qualité" d'un algorithme de planification a été proposée : il s'agit de la *complétude*.

Def. 1 (Complétude) *Un algorithme est dit complet si quelque soit l'entrée, il rapporte correctement si une solution existe ou non. De plus, si une solution existe, il doit être capable de la trouver en un temps fini.*

Nous présentons ci-dessous les principales méthodes de représentation de ces espaces de recherches pour des systèmes et environnement de plus en plus contraints, en soulignant le degré de complétude de chaque algorithme.

Construction d'un graphe complet

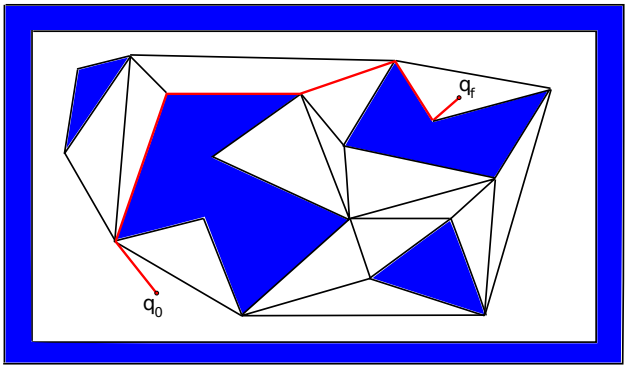


FIGURE 2.2 – Chemin déterminé entre deux configurations q_0 et q_f à partir d'un graphe de visibilité.

Lorsque la complexité du problème est assez faible (faible dimensionalité de l'espace de recherche), des approches combinatoires garantissant ainsi la complétude de l'algorithme de planification, peuvent être utilisées. La construction d'un **graphe de visibilité** [Nil69], qui se trouve peut-être d'ailleurs être la toute première approche de planification de chemin connue, fait partie de celles-ci. Un graphe de visibilité (Fig. 2.2) consiste à considérer chaque sommet des enveloppes convexes d'obstacles polygonaux, et à relier chacun de ces sommets à tout autre sommet visible de cet ensemble. On obtient ainsi un graphe dans lequel peut être effectuée une planification après avoir relié les positions initiale et finale aux sommets de cet ensemble les plus proches. Notez que cette technique autorise les configurations de

contact entre le système mobile $\mathcal{A}$ et les obstacles, c'est l'une des raisons pour lesquelles elle est relativement peu utilisée.

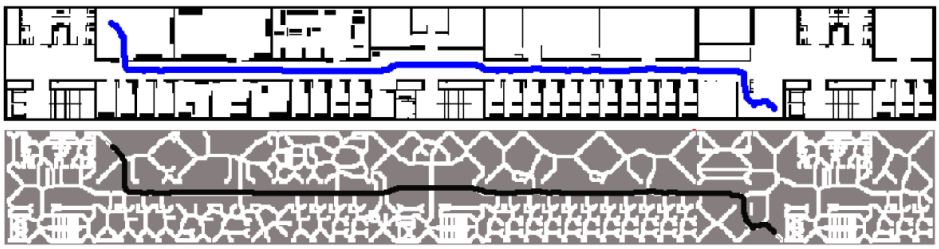


FIGURE 2.3 – Diagramme de Voronoï calculé sur une carte de l'environnement statique d'un système robotique et chemin entre deux configurations données déterminé sur le diagramme (source [GM06]).

Afin d'éviter le contact avec les obstacles, une seconde option combinatoire (complète) consiste à construire un **diagramme de Voronoï** [Aur91] dans l'espace de configurations $\mathbf{C}$ dans lequel évolue le système. On appelle région de Voronoï associée à un élément $p \in \mathbf{C}$ l'ensemble des points de $\mathbf{C}$ qui sont plus proches de p que de tout autre point de $\mathbf{C}$. Pour deux points a et b de $\mathbf{C}$, l'ensemble $\Pi(a, b)$ des points équidistants de a et b est un hyperplan affine (un sous-espace affine de co-dimension 1). Cet hyperplan est la frontière entre l'ensemble des points plus proche de a que de b, et l'ensemble des points plus proches de b que de a. On définit ainsi le diagramme de Voronoï d'un espace $\mathbf{C}$ comme étant l'ensemble des hyperplans équidistants des obstacles les plus proches pour chacune des régions de Voronoï de l'espace libre.

Le diagramme de Voronoï (fig 2.3) étant défini, on dispose donc du graphe induit où chaque noeud correspond à un sommet des arêtes du diagramme et chaque liaison correspond aux arêtes elles-mêmes. On peut alors comme précédemment effectuer une planification par recherche dans ce graphe. Notez cependant que ce diagramme est assez facilement définissable dans un espace de dimension 2 mais qu'il devient très complexe à calculer dès que l'on passe en dimension supérieure.

Un dernier exemple d'approches complètes consiste en une **décomposition cellulaire** de l'espace de configuration $\mathbf{C}$. A partir d'une représentation simple de l'espace de configuration $\mathbf{C}$, cette approche consiste à diviser cet espace en un nombre fini de sous-espace convexes. Par exemple, dans un espace de configuration $\mathbf{C} = \mathbb{R}^2$ dont la représentation des obstacles est polygonale (*cf.* Fig. 2.4), une décomposition cellulaire verticale [Cha87]

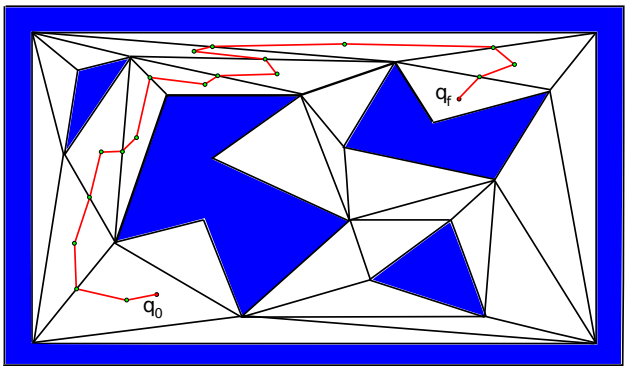


FIGURE 2.4 – Chemin déterminé entre deux configuration q_0 et q_f à partir d'une triangulation de l'espace de configurations **C**.

ou par triangulation [Lin82] peut être facilement calculée. Un graphe est alors construit comme suit : les noeuds sont définis aux barycentres des cellules obtenues et au milieu de leurs côtés. Une arête relie ensuite chaque noeud défini sur un côté au barycentre des deux cellules adjacentes. Comme précédemment, en reliant les configurations initiale q_0 et finale q_f aux noeuds du graphe les plus proches, une recherche heuristique dans le graphe résultant permet de trouver un chemin liant ces deux configurations.

Des méthodes de décomposition cellulaires plus complexes telles les "balles connectées" [BK01, VKA05] ont été conçues pour des environnements non structurés et des espaces de recherche (configurations ou états) de plus fortes dimensions. En couvrant l'espace de recherche libre par des sphères se chevauchant de la dimension de cet espace, il est possible comme précédemment de construire un graphe connexe dont une arête est définie entre chaque couple de sphères dont l'intersection est non vide.

Quelques extensions ont été proposées à la construction de cartes de routes complètes en environnements dynamiques : Erdmann et Al. [ELP87] ont proposé pour cela d'ajouter la dimension temporelle à l'espace de configuration. Une carte de route complète peut alors être définie dans ce nouvel espace. Fraichard a fait de même en définissant l'espace d'états-temps dans [Fra98] et en y construisant un graphe essayant de capturer la topologie de cet espace. L'ajout de la dimension temporelle à ces approches augmente néanmoins fortement la complexité de la recherche effectuée et le temps de parcours du graphe associé.

Bien qu'attirantes par leur simplicité, les méthodes combinatoires présentées ci-dessus sont en pratique très peu souvent utilisables : pour un

système robotique quelque peu complexe (forte dimensionalité de l'espace de configuration, systèmes redondants, topologie des obstacles quelconque), une représentation des obstacles $\mathbf{C}_{obs}$ dans l'espace de configuration est difficilement accessible, et leur calcul explicite bien trop complexe. De plus, la géométrie du graphe ne se soucie guère des contraintes sur le mouvement du système. Un robot disposant par exemple de contraintes non holonomes sera absolument incapable de suivre les arêtes du graphe sans une adaptation du plan. Des méthodes alternatives ont alors été apportées afin d'éviter la caractérisation de $\mathbf{C}_{obs}$.

Echantillonnage par grilles régulières

Lorsque la complexité du système robotique $\mathcal{A}$ est telle qu'il est difficile de déterminer la topologie de l'espace de configurations $\mathbf{C}$ (et l'espace $\mathbf{C}_{obs}$ des configurations du robot en collision avec un obstacle de l'environnement), une méthode alternative consiste à discrétiser $\mathbf{C}$ et à construire une approximation conservative de l'espace libre $\mathbf{C}_{libre} = \mathbf{C} \setminus \mathbf{C}_{obs}$. Pour ce faire, un échantillonnage de l'espace par une grille régulière de dimension n (où n est la dimension de $\mathbf{C}$) peut être effectué. Le graphe $\mathbf{G}$ en résultant est construit en définissant un noeud pour chaque échantillon et en le liant par une arête à chacun de ses 2^n voisins directs.

Une planification peut alors être effectuée entre deux configurations q_0 et q_f en assimilant chacune de ces configurations à un noeud du graphe $\mathbf{G}$, puis en explorant itérativement à partir de q_0 ses noeuds voisins jusqu'à atteindre q_f. La détermination de l'obstruction des noeuds et des arêtes du graphe par les obstacles peut ainsi être effectuée uniquement lors de leur exploration par une module de vérification de collision (souvent considéré comme une boîte noir pour ce genre d'approches).

L'espace libre atteignable par une telle grille est un sous-espace de $\mathbf{C}_{libre}$. La garantie de trouver une solution s'il en existe une est donc amoindrie et dépendante de la résolution de la grille. Ces approches sont alors dites complètes en résolution.

Def. 2 (Complétude en résolution) *Un algorithme est dit complet en résolution si quelque soit l'entrée, il rapporte correctement si une solution existe ou non à une résolution donnée. Lorsque l'algorithme ne trouve pas de solution, il peut en exister une à une résolution plus fine que celle choisie.*

Afin d'accélérer le processus de recherche sur le graphe, une représentation de l'environnement par grilles multirésolution fut proposé par la suite [VHKT00, KL05]. Une recherche de chemin peut alors être effectuée en premier lieu à basse résolution. Si aucune solution n'est trouvée les résolutions supérieures

sont examinées jusqu'à ce qu'une solution ait été trouvée. Cet échantillonnage par grilles a également été la source de représentation des espaces libre $\mathbf{C}_{libre}$ et obstrué $\mathbf{C}_{obs}$ par grilles d'occupation probabilistes [Elf90]. Une grille d'occupation probabiliste est une grille représentant une discrétisation régulière de l'espace dans laquelle chaque cellule est caractérisée par un probabilité d'occupation évaluée lors de la navigation par les capteurs du système robotique. La vérification de collision en chaque noeud du graphe induit est alors effectué en estimant la probabilité de collision de la cellule de la grille probabiliste associée. L'avantage de telles grilles est sa possibilité de mise à jour au cours du temps : lorsqu'un obstacle mobile se déplace ou lorsqu'un nouvel obstacle est détecté par les capteurs du système, les probabilités d'occupation des cellules de la grille associées sont réévaluées. Le graphe induit peut en conséquence être modifié, mais s'il désactive un noeud ou une arête choisis lors de la planification, un nouveau mouvement doit être replanifié (*cf.* Section 2.2.2).

Treillis de l'espace d'états

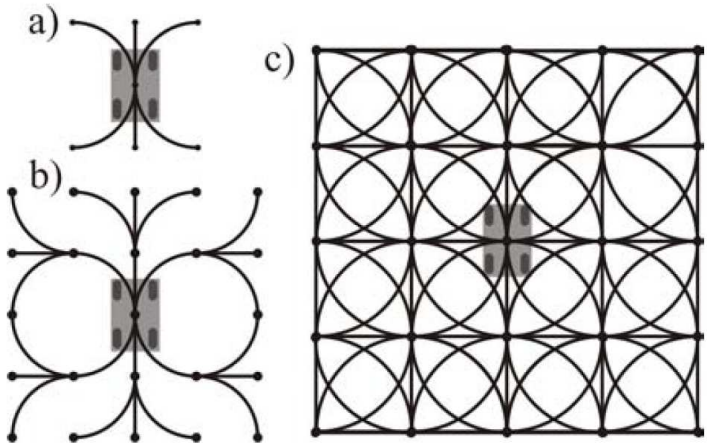


FIGURE 2.5 – Différentes étapes de construction d'un treillis sur l'espace d'état d'une voiture de Reeds & Shepp (source [PK05]).

Lorsque le système $\mathcal{A}$ dispose de contraintes sur son mouvement [Lat91] (vitesse, accélération, angle de braquage d'un robot de type voiture, etc), la connectivité de l'espace de configuration de $\mathcal{A}$ s'en trouve compromise : un chemin n'est pas nécessairement admissible pour $\mathcal{A}$ entre deux noeuds adjacents d'une grille régulière sur l'espace de configuration. Pivtoraiko et Kelly ont alors proposé la construction de treillis de l'espace d'état ([PK05]). Un

treillis de l'espace d'état est alors un graphe **G** dont chaque noeud représente un état du système $\mathcal{A}$ et chaque arête représente un chemin, admissible pour $\mathcal{A}$ (*cf.* Fig. 2.5). Cette approche est complète en résolution : le treillis de l'espace d'état caractérise l'ensemble de tous les chemins à une résolution donnée. Cette approche dispose néanmoins d'un fort inconvénient : elle est extrêmement coûteuse en terme de mémoire pour stocker l'ensemble des chemins admissibles pour le système. Une étude de la régularité de l'espace de configurations ainsi que la caractérisation de classes d'équivalence entre des chemins similaires sont alors effectués afin de limiter ce problème.

Cartes de routes probabilistes

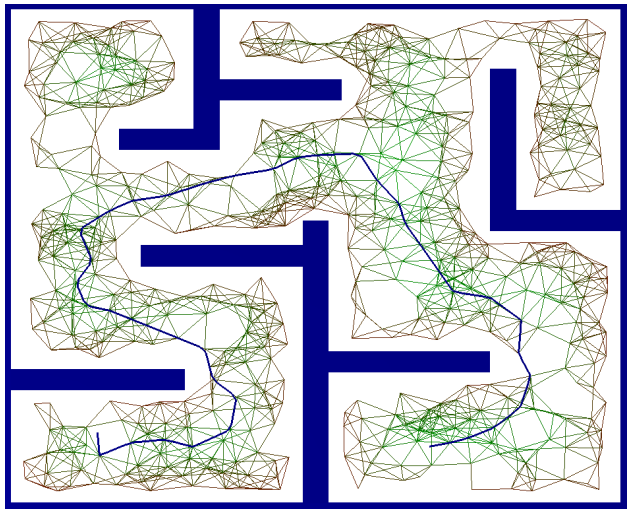


FIGURE 2.6 – Chemin planifié pour un robot de type différentiel à partir d'une roadmap calculée dans l'espace de configuration du robot

Dans le cas d'espace de recherche de forte dimension, une discrétisation régulière de l'espace peut s'avérer bien trop coûteuse. Afin de diminuer la taille du graphe sur cet espace, une solution consiste à construire une carte de route probabiliste (en anglais : probabilistic roadmap [KSLO96]) sur celui-ci. La construction d'une carte de route probabiliste s'effectue en sélectionnant des configurations aléatoires dans l'espace de recherche (*cf.* Fig. 2.6). Chaque nouvelle configuration est ajoutée au graphe si elle n'est pas en collision avec les obstacles de l'environnement. Dans ce cas un nouveau noeud est créé dans le graphe pour celle-ci. Des arêtes sont alors ajoutées à partir de cette configuration vers d'autres configurations appartenant déjà à la carte

de route dans le cas où ces deux configurations sont suffisamment proches, et qu'il existe un chemin libre de collisions entre celles-ci. La construction de la roadmap se termine généralement lorsqu'un nombre prédéterminé de noeuds constituent celle-ci.

Comme dans le cas des grilles régulières sur l'espace de configuration, l'espace libre $\mathbf{C}_{libre}$ est caractérisé par un sous-ensemble de configurations atteignables et de chemins admissibles. La représentation de l'espace libre étant dépendante d'un nombre aléatoire d'échantillons, on parle ici de complétude probabiliste.

Def. 3 (Complétude probabiliste) *Un algorithme est dit probabilistiquement complet si quelque soit l'entrée, la probabilité de rapporter correctement si une solution existe ou non tend vers un lorsque le nombre d'échantillons générés tend vers l'infini.*

La qualité de telles approches est alors évaluée par rapport à deux critères : sa *densité* et sa *dispersion*. La *dispersion* reflète la taille maximale de l'espace libre $\mathbf{C}_{libre}$ non couverte par les noeuds composant la carte de route probabiliste. La *densité* s'illustre par son aptitude à s'approcher aussi près que possible de toutes les configurations de l'espace libre $\mathbf{C}_{libre}$. De nombreuses extensions de ce type d'approches ont vu le jour afin d'optimiser ces deux paramètres tout en limitant le nombre de noeuds nécessaires.

Tout d'abord, [AW96] proposent un échantillonnage aux bornes de $\mathbf{C}_{libre}$ afin de pouvoir déterminer simplement un chemin vers une configuration proche des obstacles de l'environnement. Pour cela, lorsqu'une configuration aléatoirement choisie se trouve être en collision, elle est déplacée aléatoirement, puis une fois libre, elle est autant que possible connectée au reste du graphe. A l'opposé, [WAS99] essaient de maximiser l'espace libre autour de chaque noeud du graphe.

Dans [KSLO96], une probabilité de distribution est associée à chaque noeud du graphe. Lors de l'insertion d'un nouveau noeud, son positionnement est optimisé afin de maximiser la probabilité de distribution sur l'ensemble de l'espace libre $\mathbf{C}_{libre}$ et de pouvoir atteindre ainsi de nouvelle configurations non visibles jusqu'alors.

Une carte de route par visibilité (en anglais : visibility roadmap) est proposée en [SLN00, JS08]. Pour celle-ci, deux types de noeuds sont définis : les gardes et les connecteurs. Un noeud est un *garde* s'il n'est visible dans $\mathbf{C}_{libre}$ par aucun autre garde. Un *connecteur* est un noeud visible par au moins deux gardes. En construisant un graphe connecté tel que toute région de $\mathbf{C}_{libre}$ soit visible par un garde et que le nombre de noeuds total soit minimal, on obtient une carte de route de faible taille (limitant ainsi le temps

de recherche) et pouvant aisément connecter n'importe quel couple de noeuds de $\mathbf{C}_{libre}$.

Enfin dans le cas de navigation en environnement dynamique, [vdBO05] propose une mise à jour dynamique de la carte de route probabiliste. En présence d'obstacles mobiles, certains noeuds et arêtes sont ainsi désactivés ou réactivés afin d'adapter la planification aux mouvements des obstacles sans devoir reconstruire la carte intégralement.

Méthodes de replanification

En présence d'obstacles mobiles, certaines méthodes présentées ci-dessus permettent une adaptation du graphe sur lequel le chemin ou la trajectoire du robot est planifié, par exemple en déplaçant, activant ou désactivant des noeuds et des arêtes du graphe. Dans le cas où le plan initial passait par un noeud ou une arête n'étant plus valide, il est alors nécessaire de replanifier ([KL02]) tout ou partie du mouvement suivi. Pour cela, diverses méthodes ont été proposées : Stentz a initialement proposé une extension (nommée D* [Ste93]) de l'algorithme de recherche heuristique A* dans le cadre de la navigation de robots mobiles. Lors de l'invalidation d'un plan, celui-ci replanifie localement la partie du plan obstruée par les obstacles. Likhachev et al. ont proposé une extension "anytime" de l'algorithme A* ([LFG+05]) : à chaque instant, le robot effectue une recherche A* le plus loin possible vers le but en un temps de décision constant. D'autres méthodes de replanification ont vu le jour, telle [FS06b] modifiant une partie du plan obstrué en cas d'obstacles inattendus et améliorant le chemin suivi lorsque le temps le permet.

2.2.3 Méthodes par arbres

En parallèle de la planification classique par exploration d'un graphe de recherche sont apparues les méthodes par arbres. Celle-ci consistent, à partir de la configuration initiale du système, à construire un arbre se développant dans toutes les directions autour du robot dans l'espace de recherche. Elles sont donc bien adaptées dans le cas d'espaces de recherche à forte dimensionalité.

Les **Rapidly-exploring Random Trees (RRT)** initialement présentés par Lavalle [LaV98, LK01] représentent certainement l'une des approches de planification de mouvement les plus répandues à l'heure actuelle. A partir d'une configuration initiale q_0, l'espace de configuration du système est exploré en choisissant aléatoirement à chaque itération une nouvelle configuration q_{nv} non obstruée par les obstacles vers laquelle se diriger. La branche la plus proche de l'arbre déjà construit est alors déterminée puis étendue

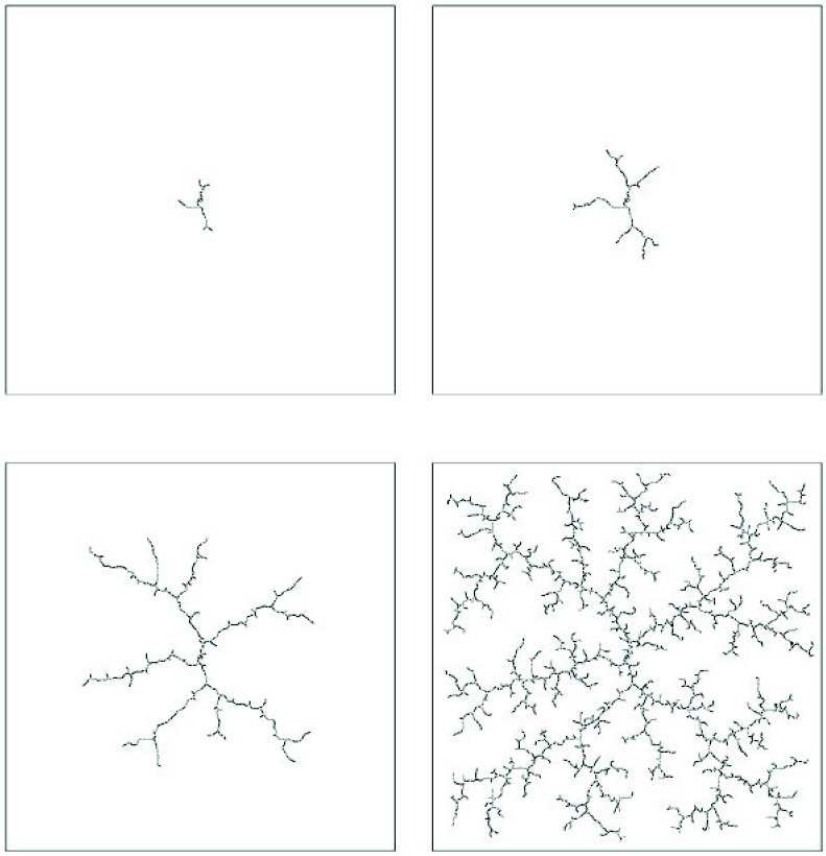


FIGURE 2.7 – Rapidly Exploring Random Trees : Etapes successives de construction de l'arbre de recherche dans l'espace des configurations du système robotique (source [LaV98]).

en direction de q_{nv}. En répétant le processus, l'espace de recherche est alors rapidement couvert, et un chemin vers toutes configurations de cet espace peut alors être facilement déterminée s'il en existe un (*cf.* Fig. 2.7). Le but de la planification étant néanmoins d'atteindre une configuration finale q_f, le processus essaie de déterminer un chemin liant la configuration la plus proche de l'arbre à q_f après un certain nombre d'itérations de l'expansion de l'arbre.

Dans le cas où le système évolue en environnement dynamique, une extension "anytime" des RRTs a également été proposée ([FS06a]) : l'arbre de recherche est mis à jour progressivement, et à chaque pas de temps le contrôle à appliquer est déterminé à partir de la racine de l'arbre menant vers la position se rapprochant le plus près du but.

FIGURE 2.8 – Le file d'Ariane : Algorithme de planification alternant deux étapes : une première étape d'exploration permettant d'étendre un arbre de recherche vers les régions inexplorées de l'espace de configuration, et une seconde étape de recherche permettant de vérifier s'il est possible de planifier un chemin jusqu'au but à partir des dernières branches de l'arbre construites.

Le **fil d'Ariane** (en anglais : Ariadne's Clew) présenté dans [BAEGM93, MAB98] explore l'espace de configuration du système robotique à partir de sa configuration initiale en construisant un arbre de recherche par l'alternance de deux méthodes (*cf.* Fig. 2.8) :

- Explore : Cette méthode a pour but d'explorer l'espace de configuration libre en y plaçant des balises aussi loin que possible des balises existantes. A l'initialisation, la seule balise disponible est la configuration initiale du système. A chaque nouvelle appel de la méthode "Explore", l'arbre est étendu à partir d'une des balises existantes choisie aléatoirement.
- Search : Cette méthode recherche autour d'une balise posée s'il est possible d'accéder directement à la configuration finale par un mouvement simple (chemin de Manhattan).

En répétant successivement ces deux étapes, l'arbre de recherche va s'étendre

rapidement sur tout l'espace de configuration accessible à partir de la configuration initiale, jusqu'à converger vers le but, ou s'arrêter s'il n'est plus possible de placer une balise à moins d'une distance minimale de celles déjà posées.

D'autres méthodes moins connues de planification par expansion d'un arbre peuvent être notées : Parmi celles-ci on trouve une **planification expansive sur l'espace de configuration** ([HLM99]) consistant à étendre itérativement un arbre de recherche en sélectionnant un noeud de l'arbre q_{ext} à étendre à partir d'une probabilité inversement proportionnelle au nombre de noeuds dans la région qui l'entoure. Une nouvelle configuration q_{nv} est alors choisie dans son voisinage proche et un chemin déterminé entre ces deux configurations.

Dans la même idée, [CP05] propose une **planification par marche aléatoire**. Le principe ici est très simple : une noeud de l'arbre déjà construit est choisi aléatoirement et étendu dans une direction aléatoire. La longueur du chemin à parcourir à chaque étape ainsi que la variation de la direction à prendre par rapport à l'étape précédente sont déterminées à partir des observations sur l'environnement, obtenues lors des extensions précédentes de l'arbre.

Ces deux dernières approches bien que simples et fonctionnelles disposent malheureusement de fortes difficultés à traverser de longs espaces libres.

2.2.4 Complexité des approches délibératives

Parmi toutes ces approches, il semble pertinent d'en étudier leur complexité. Celle ci peut être évaluée sous deux aspects : tout d'abord sa *borne maximale* peut être déterminée par l'existence d'un algorithme capable de déterminer une solution au problème de planification, celui-ci servant de preuve à la possibilité de résoudre le problème à une complexité donnée. Ensuite sa *borne minimale* donne une indication de la difficulté du problème. Reif fut en 1979 le premier à établir l'existence d'une borne minimale de la complexité PSPACE-dur [Rei79]. La *décidabilité* du problème de planification (existence d'une solution au problème donné) fut quant à elle établie par Schwarz et Sharir en 1983 dans le cas d'un système robotique simple sous contraint. Néanmoins, dès que des contraintes supplémentaires (cinématiques ou dynamiques) sont prises en compte la complexité du problème de planification de mouvement devient NP-dur [RS85, CR87].

Le temps de calcul de telles approches s'en trouve en conséquence bien trop élevé pour être exécutable en temps réel. Leur utilisation en environ-

nement dynamique s'en trouve donc fortement contrariée. Afin de s'assurer de la sécurité du mouvement d'un robot au milieu d'obstacles mobiles il est alors nécessaire de disposer d'un approche de détermination du mouvement réactive. Ces approches sont donc présentées dans la suite de ce chapitre.

2.3 Approches réactives

Les approches réactives consistent à calculer à chaque pas de temps (après récupération des informations sur l'environnement fournies par les capteurs du système) le contrôle instantané à appliquer sur les actionneurs du système.

2.3.1 Approches par champs de potentiel

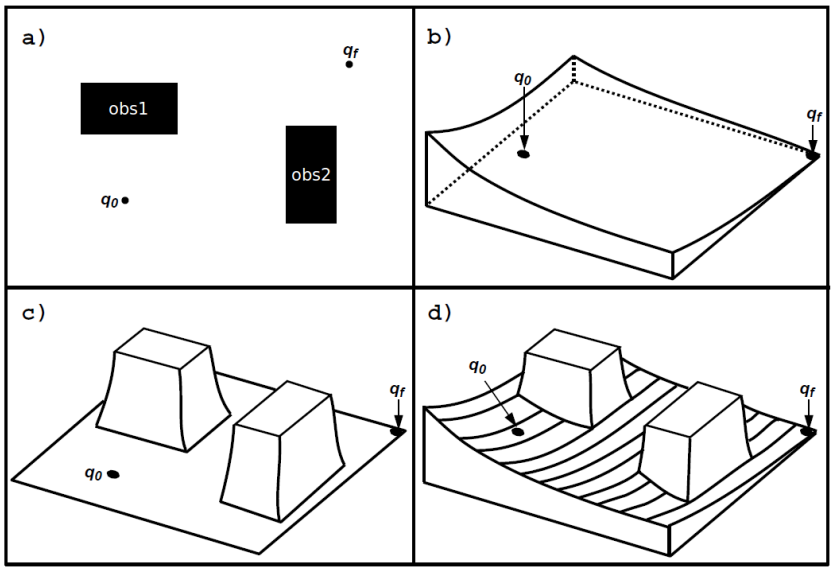


FIGURE 2.9 – Calcul d'un chemin entre deux configurations q_0 et q_f par méthode de champs de potentiels. (a) Espace de configuration du robot et représentation des obstacles $obs1$ et $obs2$ dans celui-ci. (b) Champ attractif généré par la position finale. (c) Champ répulsif exercé par les obstacles. (d) Combinaison des deux.

Les approches dîtes par champs de potentiel initialement proposées par Khatib [Kha86] consistent à considérer le robot mobile comme une particule soumise à divers champs électromagnétiques régissant son mouvement. Ses travaux considéraient les deux suivants (*cf.* Fig. 2.9) :

– Un champ de potentiel attractif provenant de la position finale p_f du système à atteindre.
– Un champ de potentiel répulsif provenant des obstacles statiques et mobiles de l'environnement.

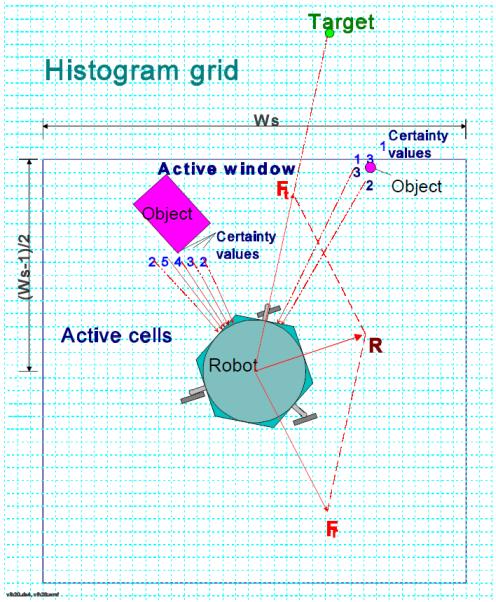
Initialement conçus pour le calcul du mouvement de bras manipulateurs, cette méthode dispose de l'avantage de calculer ces champs de potentiels dans l'espace de travail $\mathbf{W}$ (espace euclidien $\mathbb{R}^2$ ou $\mathbb{R}^3$ dans lequel une représentation des obstacles est disponible), définissant ainsi une direction privilégiée à suivre par l'élément terminal du bras dans cet espace. Une modification de la configuration du robot dans son espace articulaire est alors déduite de ce champ dans un second temps.

Bien que simple et élégante, cette méthode possède de nombreux inconvénients mis en évidence par Koren et Borenstein [KB91]. D'une part, cette approche est sujette à des minima locaux. Par conséquence, la convergence vers le but n'est pas assurée. D'autre part, ces potentiels peuvent donner lieu à de fortes oscillations du mouvement du robot en présence d'obstacles et principalement lorsque celui-ci navigue dans des passages étroits (couloirs, portes). Enfin, le vecteur de déplacement désigné par le champ de potentiel ne prend en aucun cas en compte la cinématique ou la dynamique du système robotique considéré. Un robot disposant de contraintes non-holonomes aura de sérieuses difficultés à suivre une telle direction.

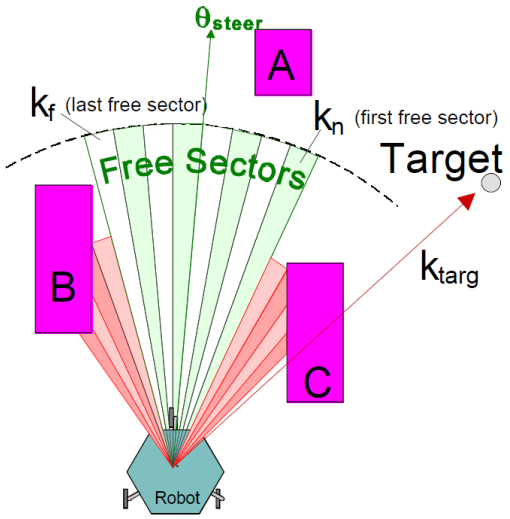
Malgré ces limitations, de nombreuses techniques de navigation ont découlé de ces champs de potentiels. Ils ont par exemple été adaptés à la navigation au milieu d'obstacles mobiles dans [GC02, Hua08] en prenant en compte non seulement la distance aux obstacles mais également la vitesse de ces derniers pour calculer les champs répulsifs.

2.3.2 Histogramme de champs de vecteurs

Dans la lignée des approches par champs de potentiels, sont apparus les histogrammes par champs de vecteurs (en anglais : Vector Field Histogram - VFH). Ceux-ci, introduits par Koren et Borenstein [BK91] sont nés de la combinaison des champs de potentiels et des grilles d'occupation : un histogramme basé sur une grille cartésienne de l'environnement est construit et mis à jour au fur et à mesure de la navigation afin de reporter la présence d'obstacles à proximité du robot (*cf.* Fig. 2.10(a)). Afin de choisir une direction à suivre, un histogramme polaire est construit à partir de la grille d'occupation : en discrétisant les différentes directions possibles autour du robot, l'histogramme polaire est construit en pondérant pour chaque secteur de la discrétisation polaire les cellules traversées de la grille d'occupation contenant des obstacles. Une fois cet histogramme polaire construit, des "vallées



(a) Fenêtre active autour du robot dans laquelle sont mise à jour les probabilités d'occupation par les obstacles.



(b) Histogramme polaire calculé à partir de la grille d'occupation. Les vallées libres d'obstacles sont déterminées afin de choisir une direction à suivre.

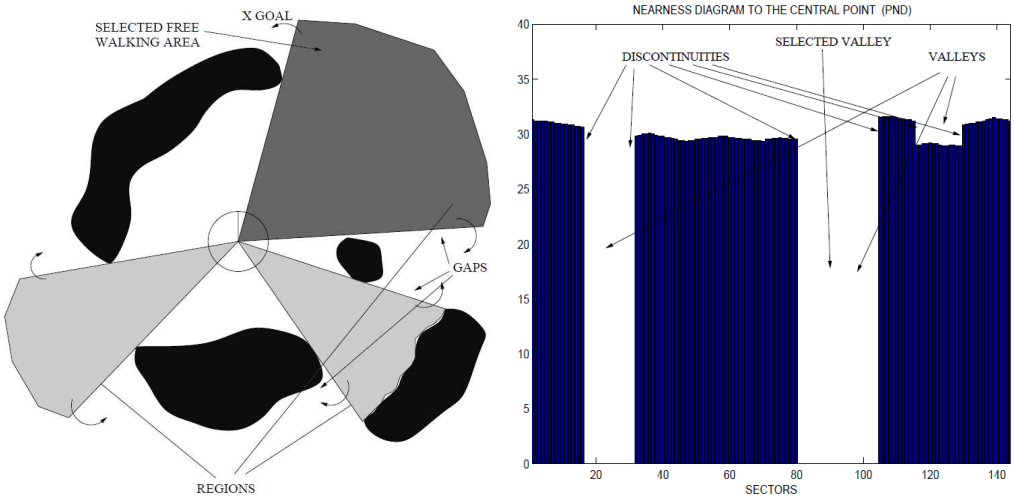
FIGURE 2.10 – Histogramme de champs de vecteurs : La présence d'obstacle est mise à jour sur une grille d'occupation au fur et à mesure du processus de navigation. Après avoir caractérisé les "vallées" libres d'obstacles sur un histogramme polaire autour du robot, il détermine la plus proche du but et calcule le contrôle nécessaire afin de se diriger vers celle-ci (source [BK91]).

candidates" sont déterminées comme les suites de secteurs contigus de l'histogramme polaire libres d'obstacles (*cf.* Fig. 2.10(b)). La direction à prendre par le système est alors déterminée par le milieu de la vallée menant le plus directement au but.

Initialement conçue pour la navigation de robots holonomes (pouvant naviguer dans toutes les directions), cette méthode a été étendue à plusieurs reprises afin de prendre en compte les dimensions du robot (par un espace de configuration implicite) et ses contraintes de vitesse [UB98]. Plus tard, les VFH ont été combinés à une recherche A* (VFH* [UB00]) afin de trouver un chemin menant vers le but et d'échapper ainsi aux minima locaux. Les méthodes VFH disposent néanmoins encore de fortes limitations : Elles ne prennent ni en compte la dynamique du système robotique, ni l'éventuelle présence d'obstacles mobiles ; le mouvement instantané du robot est calculé uniquement à partir des informations sur la position courante des obstacles,

leur vitesse n'est en aucun cas considérée.

2.3.3 Navigation par diagrammes de proximité



(a) Représentation de l'environnement autour du robot

(b) Caractérisation des vallées disponibles en repérant les discontinuités du graphe de proximité des obstacles

FIGURE 2.11 – Navigation par Diagrammes de Proximité (ND) : Méthode réactive utilisant une représentation de haut niveau de l'environnement en déterminant les vallées libres d'obstacles. Sélection dans un second temps d'une stratégie de navigation en fonction de la topologie de l'environnement et de la proximité des obstacles et de la position finale (source [MM00]).

L'approche de navigation par diagrammes de proximité (en anglais : Nearness Diagram Navigation - ND) proposée par Minguez et Montano [MM00] fortement inspirée des VFH se base sur la topologie de l'environnement pour choisir un contrôle à appliquer à chaque instant. Pour ce faire, deux diagrammes polaires sont construits : l'un représente la distance du centre du robot aux obstacles dans toutes les directions autour de celui-ci (*cf.*Fig. 2.11) ; le second calcule cette même distance à partir des contours du robot afin d'estimer la distance de sécurité conservée. L'étude des discontinuités de ces diagrammes de proximité permet de caractériser les "vallées" libres d'obstacles autour du robot. La différence majeure par rapport aux VFH consiste en la stratégie de navigation choisie. La méthode ND utilise une stratégie déterministe de choix du comportement à adopter en fonction de la distance

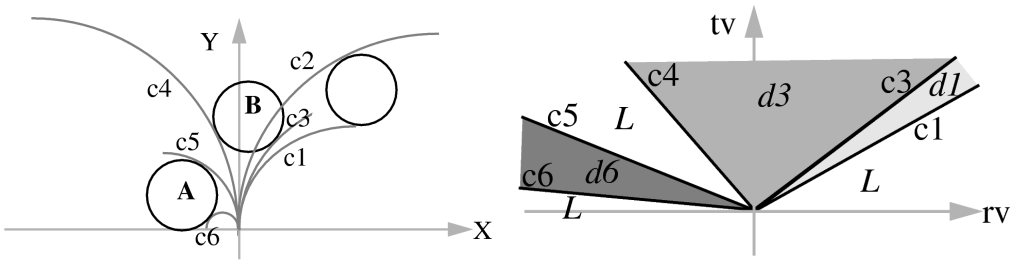
aux obstacles les plus proches, et en fonction de la topologie des obstacles qui l'entourent. Les options possibles se résument à

- contourner un obstacle
- passer entre deux obstacles
- aller tout droit vers le but

ainsi qu'à choisir une vitesse faible ou élevée suivant la proximité des obstacles. Un contrôle en vitesses linéaire et angulaire est alors calculé en fonction du comportement choisi.

De la même manière que les approches présentées précédemment, cette méthode n'est pas vraiment adaptée pour naviguer au milieu d'obstacles mobiles et ne prends pas en compte la dynamique du système.

2.3.4 Méthode de navigation courbure-vélocité



(a) Trajectoires candidates représentées dans l'espace de travail **W**.

(b) Contrôles correspondant aux trajectoires candidates dans l'espace des vitesses linéaire et angulaire.

FIGURE 2.12 – Méthode de navigation courbure-vélocité (CVM) : Caractérisation des mouvements possibles dans l'espace des vitesses linéaire et angulaire. Sélection du mouvement à suivre maximisant une fonction de coût (source [Sim96]).

A l'opposé des méthodes dîtes directionnelles présentées ci-dessus (essayant de se diriger vers le but en restant hors de portée des obstacles), se trouvent les méthodes dîtes de l'espace des vitesses. Celles-ci ont été introduites avec la méthode de navigation courbure-vélocité (en anglais Curvature-Velocity Method - CVM) proposée par Simmons [Sim96]. Son principe est le suivant : au lieu de choisir une direction à suivre en fonction de la position finale et de l'environnement puis de calculer une commande à appliquer dans l'espace des vitesses, la CVM évalue les différentes trajectoires possibles dans l'espace des vitesses puis sélectionne dans cet espace un mouvement à exécuter permettant d'éviter les obstacles et de se rapprocher du but. Afin de choisir quelle commande appliquer, à chaque couple vitesse

linéaire / vitesse angulaire (v, ω) est associée une fonction de coût **J** basée sur la distance aux obstacles, la modification de l'orientation du système par rapport au but et sur le temps nécessaire pour rejoindre le but (une vitesse linéaire élevée est donc préférée). La commande maximisant cette fonction de coût est alors sélectionnée et envoyée au robot lors du prochain pas de temps. Sélectionner directement la commande dans l'espace des vitesses permet ainsi prendre en compte la dynamique du robot : en effet, en considérant la durée $T = t_i - t_{i-1}$ de chaque pas de temps auquel est répété le processus de sélection de la nouvelle commande (v_i, ω_i) à appliquer au temps t_i, les contraintes d'accélérations linéaire a_{max} et angulaire η_{max} sont respectées si $(v_i, \omega_i) \in [v_{i-1} - T * a_{max}; v_{i-1} + T * a_{max}] \times [\omega_{i-1} - T * \eta_{max}; \omega_{i-1} + T * \eta_{max}]$.

D'après les propres auteurs de la CVM, celle-ci n'est pas toujours apte à garantir la sécurité du mouvement. La CVM peut en effet pousser le robot à passer relativement proche des obstacles et rentrer en collision avec ces derniers. Afin d'éviter ce genre de comportements, ces travaux ont été étendus dans [KS98]. Cette nouvelle méthode intitulée "Lane Curvature Method" (LCM) est un algorithme en deux étapes : Tout d'abord, des "lignes" permettant de passer entre les obstacles en maximisant la distance par rapport à ceux-ci sont déterminées. Ensuite une commande permettant soit de suivre une ligne soit de rejoindre l'une d'entre elles est calculée comme précédemment par CVM.

2.3.5 Fenêtre dynamique

Découlant des CVMs, l'approche de fenêtre dynamique (en anglais Dynamic Window - DW) présentée par [FWT97] conserve le principe de sélection d'un mouvement à suivre dans l'espace des vitesses. A l'instar de la plupart des approches réactives présentées précédemment, les DWs ont été développées pour des robots de type différentiel, contrôlés en vitesse linéaire v et vitesse angulaire ω. A chaque pas de temps, une nouvelle commande constante (v, ω) est sélectionnée parmi les vitesses respectant les contraintes suivantes :

- Contraintes cinématiques : $v \in [0; v_{max}]$ et $\omega \in [-\omega_{max}; \omega_{max}]$ où v_{max} et ω_{max} sont les vitesses maximales admissibles.
- Contraintes dynamiques : Les accélérations linéaire et angulaire a et η appliquées entre chaque pas de temps doivent être bornées. $a \in [-a_{max}; a_{max}]$ et $\eta \in [-\eta_{max}; \eta_{max}]$ où a_{max} et η_{max} sont les accélérations linéaires et angulaires maximales.
- Garantie de sécurité passive : Le système doit être certain de pouvoir s'arrêter avant d'entrer en collision avec un obstacle.

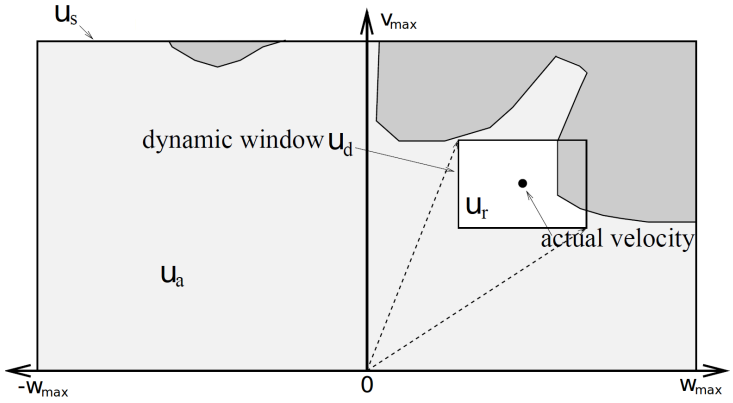


FIGURE 2.13 – Fenêtre Dynamique (DW) : Calcul du mouvement à appliquer à chaque pas de temps dans l'espace des vitesses $\tilde{u} = (v, \omega)$. Les commandes candidates $\tilde{u}_r$ sont situées à l'intersection des vitesses admissibles $\tilde{u}_s$, des vitesses respectant les contraintes dynamiques $\tilde{u}_d$ et des commandes $\tilde{u}_a$ permettant de s'arrêter avant de rentrer en collision avec un obstacle de l'environnement (source [FWT97]).

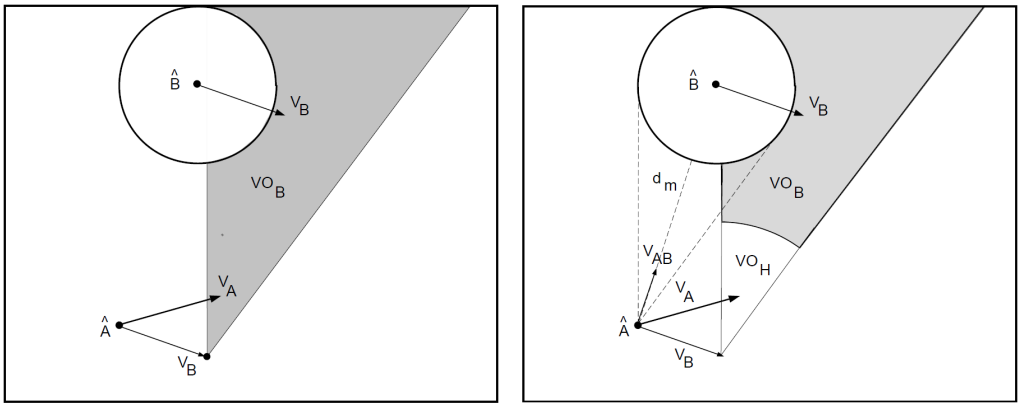
Notons $\tilde{u}_s$ l'ensemble des vitesses admissibles respectant les contraintes cinématiques, $\tilde{u}_d$ les vitesses respectant les contraintes dynamiques et $\tilde{u}_a$ les vitesses permettant de s'arrêter avant d'entrer en collision avec les obstacles de l'environnement (*cf.* Fig. 2.13). L'ensemble des commandes candidates $\tilde{u}_r$ est alors défini par :

$$\tilde{u}_r = \tilde{u}_s \cap \tilde{u}_d \cap \tilde{u}_a \tag{2.1}$$

Un fois cet ensemble défini, une fonction de coût similaire aux CVMs est définie. La commande choisie est de même la commande maximisant cette fonction de coût.

De par sa volonté d'assurer une sécurité passive, l'approche de DW est devenue l'une des approches de navigation réactives les plus populaires de nos jours. Cette approche initiale souffrait d'une limitation importante : seule la position courante des obstacles était prise en compte, mais pas leur mouvement. La navigation en environnement dynamique avec ce type d'approche en était donc fortement compromise. Afin de palier ce problème, Seder et Petrovic ont proposé une extension intitulée "Time Varying Dynamic Window" ([SP07]). Celle-ci calcule à chaque instant un ensemble de trajectoires probablement suivies par les obstacles dans le futur. Une vérification de collision à court terme peut donc être opérée.

2.3.6 Représentation des obstacles dans l'espace des vitesses



(a) Calcul du cône des vitesses interdites pour un horizon temporel infini

(b) Calcul du cône des vitesses interdites pour un horizon temporel limité

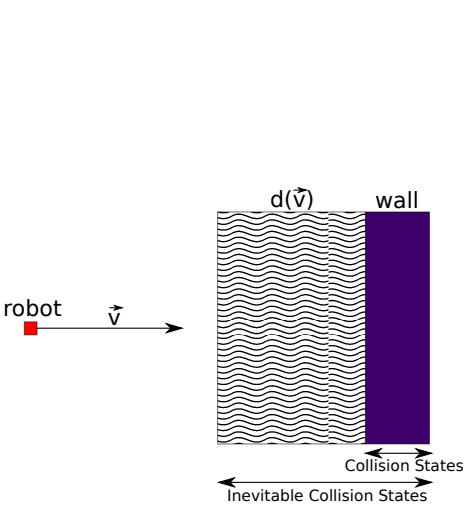
FIGURE 2.14 – Représentation des obstacles dans l'espace des vitesses (VO) : Etant donnés un robot $\mathcal{A}$ et un obstacle B, la région VO_B représente l'ensemble des vecteurs vitesse de $\mathcal{A}$ menant à une collision avec l'obstacle B si tous deux conservent leur mouvement. (source [FS98])

Dans le but de prendre en compte non seulement la dynamique du système robotique, mais également la dynamique de l'environnement dans lequel évolue ce système, Fiorini et Shiller ont introduit une approche intitulée "Velocity Obstacles" (VO [FS93]). Supposant une connaissance à priori du mouvement des obstacles mobiles, l'approche consiste à caractériser parmi les vitesses admissibles (respectant les contraintes cinématiques et dynamiques du système), celles menant à une éventuelle collision avec les obstacles dans le futur (jusqu'à un certain horizon temporel t_h). En supposant qu'un obstacle B va conserver une vitesse constante dans un futur proche (par exemple par approximation linéaire de sa vitesse courante), il est possible de déterminer les vitesses relatives du système $\mathcal{A}$ à cet obstacle menant à une collision dans le futur. L'ensemble de ces vitesses "interdites" s'illustre graphiquement comme présenté Fig. 2.14(a)) par un cône de vecteurs vitesse interdit. Certaines de ces vitesses ne conduiront bien sûr à une collision qu'après un temps relativement élevé. En limitant les VOs à un horizon temporel t_h (*cf.* Fig. 2.14(b)), on obtient ainsi une approximation raisonnable des vitesses interdites pour le système $\mathcal{A}$.

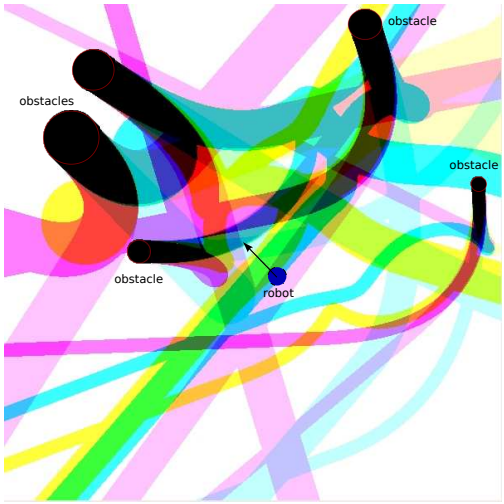
Plusieurs extensions de ces travaux ont vu le jour ces dernières années, telle la prise en compte d'approximation du modèle du futur des obstacles

plus complexes [LLS05] ou encore une tentative de définition du "bon horizon temporel" nécessaire à la garantie de la sécurité du mouvement [GSR09]. La détermination de cet horizon temporel est encore un sujet prêtant fortement à débattre à l'heure actuelle.

2.3.7 Navigation basée sur les états de collisions inévitables



(a) Principe des ICS : Si un robot mobile conserve sa vitesse v et entre dans la zone hachurée définie par sa distance de freinage $d(v)$, il n'aura plus le temps décélérer ou de tourner pour éviter la collision.



(b) Calcul des ICS dans un environnement peuplé d'obstacles mobiles : Les zones noires représentent les ICS pour l'ensemble des manoeuvres données.

FIGURE 2.15 – Etats de Collisions Inévitables (ICS) : Un ICS est un état tel que, quelque soit le contrôle appliqué en entrée d'un système robotique $\mathcal{A}$, il entrera en collision avec un obstacle. Ces états doivent donc être évités afin d'assurer la sécurité du système (source [MGF08]).

Visant désormais comme objectif l'insertion de robots autonomes en zones urbaines au milieu de piétons, de véhicules ou d'autres robots, la sécurité du mouvement des systèmes robotiques est devenue un axe prioritaire de recherche dans le domaine. Dans cette optique est née la notion d'état de collision inévitable fortement développée dans [FA03, PF07, MGF08, MGF09] et illustrée Fig. 2.15.

Def. 4 (Etat de Collision Inévitable (ICS)) *Un état s du système $\mathcal{A}$ est un état de collision inévitable si, quelque soit la séquence de contrôles*

appliquée en entrée du système $\mathcal{A}$, il existe un temps t auquel le système $\mathcal{A}$ est assuré de rentrer en état de collision.

De par cette définition, il devient clair qu'il est nécessaire pour assurer la sécurité d'un système robotique (et des autres agents évoluant dans son environnement) d'éviter non seulement les états de collisions mais également les ICS, *i.e.* les états menant inévitablement à une collision. La détermination des ICS nécessite, comme pour les VOs de prendre en considération la dynamique du système, mais également celle des obstacles mobiles qui l'entourent (et par conséquent, une prévision de leur mouvement dans le futur). Parthasarathi et Fraichard se sont efforcés de proposer une méthode basée sur une sous-approximation de l'espace des contrôles permettant d'évaluer si l'état d'un système robotique est un ICS ou non [PF07]. Dans la continuité de ces travaux, Martinez et Fraichard ([MGF09]) ont développé une méthode de navigation réactive permettant à chaque instant de passer d'un état non-ICS vers un autre état non-ICS.

Cette approche semble être à l'heure actuelle la meilleure option pour se rapprocher d'un risque de collision nul, néanmoins elle reste fortement coûteuse et dépendante de la fiabilité du modèle prévisionnel du mouvement des obstacles considéré.

2.3.8 Planification de mouvement partiel

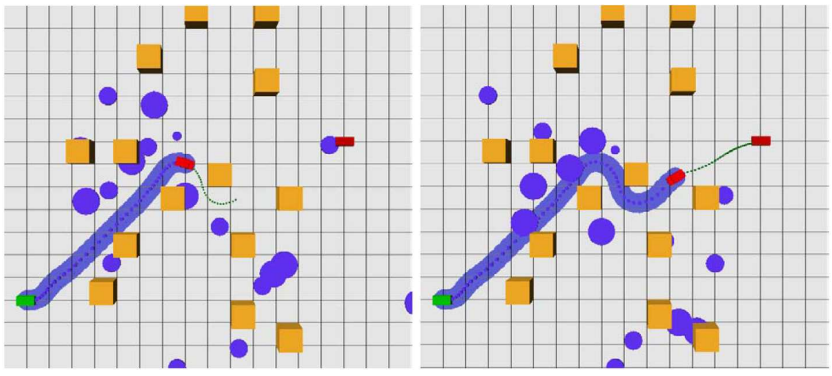


FIGURE 2.16 – Planification de mouvement partiel : Etant donné un temps de décision fixé limité, la méthode choisit de suivre la trajectoire planifiée se rapprochant le plus près du but si celui-ci n'a pas été atteint (source [PF05a]).

Une dernière approche réactive mérite d'être notée : il s'agit de la planification de mouvement partiel (en anglais : Partial Motion Planning - PMP)

utilisée par [FDF02, PF05b, PF05a]. Celle-ci consiste à calculer réactivement, en un temps de décision fixe, une trajectoire se rapprochant le plus possible du but (*cf.* Fig. 2.16). Cette méthode consiste en un algorithme à trois étapes répété à chaque pas de temps :

- Mise à jour du modèle de l'environnement à partir des entrées capteurs du robot.
- Recherche délibérative d'une trajectoire menant à l'état but. Si le but n'a pas été atteint après un temps de décision fixe, la trajectoire calculée s'en rapprochant le plus est choisie comme trajectoire à suivre.
- Enfin, le mouvement planifié au pas de temps précédent est exécuté.

Cette approche permet donc d'être réactive aux diverses évolutions de l'environnement tout en étant capable de sortir d'impasses non détectées à priori. Elle reste sujette à des minima locaux, mais y est néanmoins bien plus robuste que les approches citées précédemment.

2.3.9 Défaut de convergence vers le but

Toutes les approches réactives citées ci-dessus disposent d'une complexité suffisamment faible pour être employée en temps réel au cours de la navigation. Cette caractéristique n'assure cependant en aucun cas la sécurité du système robotique (il n'est pour la plupart de ces approches ni garanti qu'il sera capable de s'arrêter s'il rencontre un obstacle et encore moins qu'il sera capable d'éviter un obstacle hostile se dirigeant vers lui). La capacité à raisonner sur le futur de [SP07, LLS05, MGF09] tend à favoriser une navigation sûre, c'est donc l'une des caractéristiques que nous souhaiterions intégrer au développement d'une nouvelle technique de navigation.

Malgré ces avantages par rapport aux approches délibératives, aucune de ces méthodes n'est capable d'assurer la convergence vers le but. Nous présentons alors dans la section suivante une méthode de déformation de mouvement essayant à la fois de s'assurer que celui-ci sera bien atteint, tout en évitant les obstacles au cours de l'exécution du mouvement.

2.4 Déformation de mouvement

Entre approches délibératives calculant un mouvement complet jusqu'au but mais dont la complexité est trop élevée pour être utilisées en environnement dynamique, et approches réactives utilisables en temps réel mais dont la convergence vers le but est compromise, se trouve un large fossé.

Afin de palier les inconvénients de ces deux types d'approches, Quinlan et Khatib ont proposé en 1993 une approche basée sur la déformation de mou-

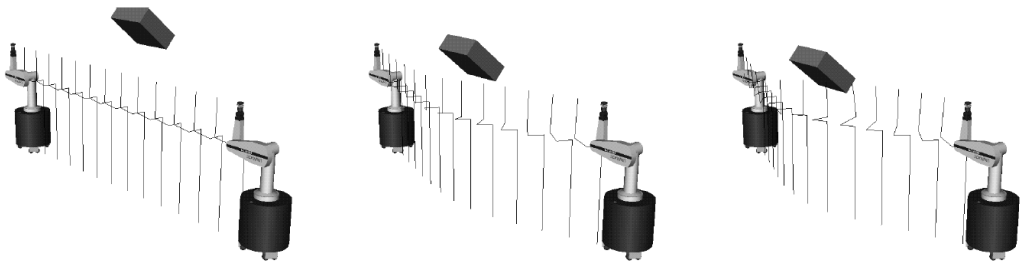


FIGURE 2.17 – Déformation de chemin : le chemin suivi par un robot mobile est modifié itérativement au cours du mouvement afin de s'écarter de la position d'un obstacle s'en approchant (source [BK97]).

vement [QK93]. Son principe est le suivant : Un chemin complet jusqu'au but est calculé à priori et fourni au système robotique. Au cours de l'exécution, la partie du mouvement restant à être exécutée est déformée continuellement en réponse aux informations sur l'environnement récupérées par les capteurs (*cf.* Fig. 2.17). Le système peut ainsi modifier son parcours en fonction du déplacement d'obstacles ou de l'imprécision de sa connaissance de l'environnement. La déformation résulte en général de deux types de contraintes : des contraintes externes dûes à la proximité des obstacles, et des contraintes internes destinées à maintenir la faisabilité et la connectivité du mouvement. Tant que la connectivité du chemin est maintenue, la convergence vers le but est assurée.

La déformation de chemin souffre néanmoins d'une forte contrainte : les déformations sont limitées à des homotopies (déformations continues) du chemin de départ, pouvant ainsi conduire à des comportements inappropriés. Imaginez par exemple un obstacle souhaitant couper le chemin suivi par le système. Si l'obstacle avance rapidement, il serait préférable de ralentir, et de le laisser passer. C'est entre autres à ce niveau que nous souhaitons apporter notre contribution.

Le manuscrit de thèse étant principalement focalisé autour de la déformation de trajectoire, un état de l'art plus exhaustif sur le sujet est présenté dans le chapitre suivant (3).

2.5 Conclusion

Nous avons présenté dans cet état de l'art sur la navigation autonome d'une part des approches dites délibératives *i.e.* destinées à planifier un mouvement complet entre deux configurations, d'autre part des approches dites

réactives *i.e.* calculant un nouveau mouvement à suivre à chaque pas de temps afin de pouvoir s'adapter au mouvement d'obstacles mobiles ou inattendus.

Cette thèse étant axée sur la navigation autonome en environnement dynamique, il va de soi qu'une simple planification de mouvement ne correspond pas à nos attentes tandis que le manque de convergence vers le but des approches réactives nous pousse à nous tourner vers une approche hybride palliant cet inconvénient. La déformation de mouvement, combinant intuitivement à la fois un évitement d'obstacles ainsi qu'une assurance de convergence vers le but a donc attiré notre attention. Celle-ci dispose néanmoins de deux inconvénients majeurs : d'une part la plupart des approches existantes limitent les déformations possibles à des homotopies du chemin de départ limitant ainsi fortement le comportement du système robotique contrôlé ; d'autre part, seule la position courante des obstacles est considérée à chaque instant pour effectuer la déformation. Il serait intéressant de considérer le mouvement futur des obstacles afin de pouvoir anticiper leur mouvement.

Nos travaux se sont donc concentrés sur la résolution de ces deux problèmes. Nous proposons dans le chapitre suivant une approche de déformation de trajectoire essayant d'y apporter une solution.

Chapitre 3

Déformation de Trajectoire

3.1 Introduction

3.1.1 Motivations

Les approches de détermination du mouvement pour un système robotique mobile se classifient généralement en approches réactives et délibératives (*cf.* chapitre 2). La déformation de mouvement fut initialement introduite par Quinlan et Khatib ([QK93]) afin de combler la brèche entre ces deux catégories d'approches. Celle-ci consiste à modifier itérativement au cours du temps un mouvement initialement planifié, afin de s'écarter des obstacles statiques et mobiles de l'environnement, tout en conservant un mouvement admissible pour le système robotique (respectant l'ensemble des contraintes sur son mouvement) et convergeant vers le but.

De nombreuses techniques de déformation de mouvement ont été proposées jusqu'ici (détaillées en Section 3.2), néanmoins la plupart d'entre elles accomplissent une *déformation de chemin i.e.* une déformation de la courbe géométrique suivie par le robot. Le problème intrinsèque à la déformation de chemin est qu'elle ne peut en aucun cas prendre en considération la dimension temporelle d'un environnement dynamique.

Pour ce faire, il est nécessaire de passer d'une déformation de chemin à une *déformation de trajectoire*, une trajectoire étant essentiellement un chemin géométrique paramétré par le temps. Il indique non seulement par où doit passer le système, mais également à quel moment et à quelle vitesse.

Au contraire de la déformation de chemin dans laquelle seules des déformations spatiales sont mises en oeuvre, la déformation de trajectoire procède à la fois à des déformations *spatiales et temporelles*. Le temps auquel chaque point de passage est atteint, ainsi que la vitesse utilisée peuvent en conséquence être modifiés.

3.1.2 Contribution

Nous présentons dans ce chapitre une méthode de déformation de trajectoire nommée `Teddy` (Trajectory Deformer [DF08b, FD09]) destinée à contrôler des systèmes robotiques complexes (robot différentiel, véhicule de type voiture [DF08a]). En intégrant la dimension temporelle au schéma de déformation initial modifiant un chemin dans l'espace, un modèle prévisionnel du comportement futur des obstacles peut être considéré. Alors que la déformation de chemin ne consistait qu'à écarter à chaque instant le chemin suivi par le robot de la position courante des obstacles, la *déformation de trajectoire* tend à anticiper leur mouvement. Ce comportement est obtenu similairement en écartant la trajectoire suivie par le robot du modèle prévisionnel du mouvement futur des obstacles. Considérant la dynamique du système robotique, la principale difficulté d'une telle approche repose sur le maintien de la connectivité de la trajectoire : une trajectoire définie entre deux configurations données est dîte connectée si l'ensemble des contraintes cinématiques et dynamiques du robot sont respectées tout au long de celle-ci, et si la convergence vers le but est assurée. Nous introduisons donc un générateur de trajectoires capable de déterminer une trajectoire admissible entre deux états-temps afin de surmonter cette difficulté.

A l'instar de la déformation de chemin, un module de planification est nécessaire afin de fournir la trajectoire nominale suivie par le robot. `Teddy` déforme périodiquement cette trajectoire au cours de l'exécution du mouvement à partir des informations sur l'environnement fournies par les capteurs du système robotique.

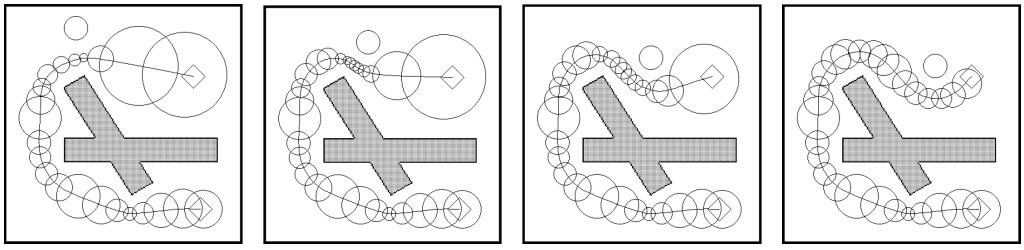
3.1.3 Plan du chapitre

Ce chapitre est organisé comme suit : les travaux précédents de déformation de mouvement ainsi que leur limitations sont décrits en Section 3.2. Le principe de notre approche de déformation de trajectoire `Teddy` est présenté en Section 3.3. Quelques résultats en simulation pour différents systèmes robotiques et dans différents scénarios sont présentés en Section 3.4. Son intégration dans une architecture complète ainsi que quelques résultats expérimentaux seront présentés dans un chapitre ultérieur (5). Enfin l'approche est discutée et une conclusion est apportée en Section 3.5.

3.2 Déformation de mouvement : approches existantes

Introduite par Quinlan et Khatib en 1993 [QK93], la déformation de mouvement a connu diverses variantes et extensions durant ces vingt dernières années. Nous présentons ci-dessous un bref aperçu des différentes méthodes proposées.

3.2.1 Bande élastique de Khatib



(a) (b) (c) (d)

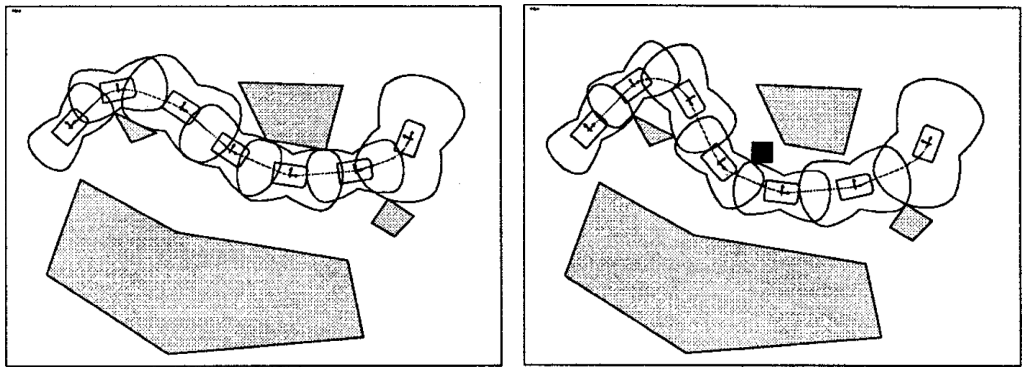
FIGURE 3.1 – Bande élastique : Un chemin initialement planifié est couvert par une séquence de disques caractérisant l'espace libre autour des configurations successives empruntées par le robot. La position et la tailles des disques sont modifiées au cours du déplacement des obstacles mobiles, modifiant ainsi le chemin emprunté par le robot (source [QK93]).

Les travaux précurseurs de déformation de mouvement [QK93] ("Elastic Band") ont été développés afin de combiner une planification de mouvement avec un évitement d'obstacles réactif. Le principe est le suivant : afin de déformer un chemin pour un robot holonome, celui-ci était représenté par une séquence de configurations successives. Autour de chaque configuration q était construite une "bulle" $\mathbf{B}(q)$ définie comme l'ensemble des configurations à une distance maximale d_{max} garantissant être sans collision avec les obstacles (*cf.* Fig. 3.1). Au cours de l'exécution du mouvement, deux types de forces étaient alors exercées sur chacune des bulles déterminant le chemin suivi :

- une force répulsive écartant chacune des bulles des obstacles.
- une force de contraction permettant de tendre le chemin lorsque les obstacles s'en écartent.

La position et la taille de chacune des bulles étaient alors modifiées suivant ces forces et un processus d'ajustement du nombre de bulles était mis en place afin de s'adapter à leurs variations.

Cette approche initiale ne prenant en compte que la cinématique du robot, et étant limitée à des systèmes holonomes, elle a donc ouvert la voie à de nombreuses améliorations et extensions possibles.



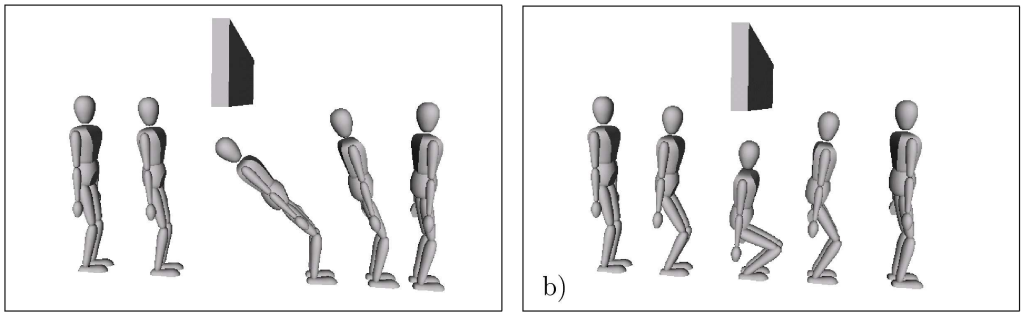
(a) (b)

FIGURE 3.2 – Bande élastique pour véhicule de type voiture : L'espace de configuration libre ("bulle") autour de chaque configuration successive du robot le long du chemin est adapté afin de prendre en compte les contraintes non-holonomes du véhicule (source [KJCL97]).

Suite à ces travaux, l'approche de "bande élastique" a été adapté dans [KJCL97] à des véhicules de type voiture. Les "bulles" définissant l'espace libre autour des configurations successives du chemin du robot sont adaptées afin de prendre en compte les contraintes non-holonomes de ces véhicules(*cf.* Fig. 3.2). Deux bulles non-holonomes connectées (*i.e.* dont l'intersection est non vide) garantissent qu'il existe, entre leurs centres respectifs, un chemin de Reeds & Shepp [RS90] totalement inclus dans l'espace de configuration décrit par l'union de ces deux bulles. Tant que la séquence de bulles définies entre la configuration initiale et la configuration but reste connectée, une succession de chemins de Reeds & Shepp non obstruée par les obstacles lie donc ces deux configurations. Après application des forces "externes" dues à une répulsion de la part des obstacles et "internes" contractant le chemin et essayant de garder la séquence de bulles connectées, un lissage par courbes de Bézier est opéré sur chaque chemin de Reeds & Shepp afin de garantir une courbure nulle en leurconfigurations initiale et finale et ainsi éviter toutes discontinuités.

Cette extension souffre malheureusement de nombreuses lacunes : Tout d'abord, comme précédemment les contraintes sur le mouvement du robot et la possible présence d'obstacles mobiles ne sont absolument pas prises en compte. Ensuite, le chemin résultant peut comporter de nombreuses

manoeuvres dues à l'utilisation de chemins de Reeds & Shepp. Enfin, le lissage peut aller à l'encontre du respect de la courbure maximale le long du chemin due aux contraintes non-holonomes du système. Même si ce problème est souligné, aucune solution n'y est apportée.



(a) Déformation de chemin sans prise en compte de la posture

(b) Déformation avec prise en compte de la posture

FIGURE 3.3 – Bande élastique pour robots humanoïdes (Elastic Strips) : Les forces appliquées par les obstacles sont calculées dans l'espace de travail puis retranscrites dans l'espace articulaire du robot. De plus, des contraintes de posture (fournissant un mouvement intuitif) et de tâche à accomplir sont prises en compte (source [BK02]).

Brock et Khatib ont également poursuivi les travaux de ce dernier afin d'adapter la déformation de mouvement à des robots disposant d'espace articulaires complexes tels les manipulateurs mobiles ([BK97]). Les "bulles" caractérisant l'espace libre autour de chaque configuration du chemin ont été abandonnées. Le chemin suivi est représenté dans l'espace de travail. En sélectionnant un nombre fini de points de contrôle sur le robot, les forces dues aux obstacles sont appliquées sur chacun de ces points de contrôle pour chaque configuration du chemin déformé, puis retranscrites dans l'espace articulaire du robot dans un second temps. Cette opération permet ainsi d'éviter les coûts de calcul élevés dûs à la représentation des obstacles dans l'espace de configuration du robot.

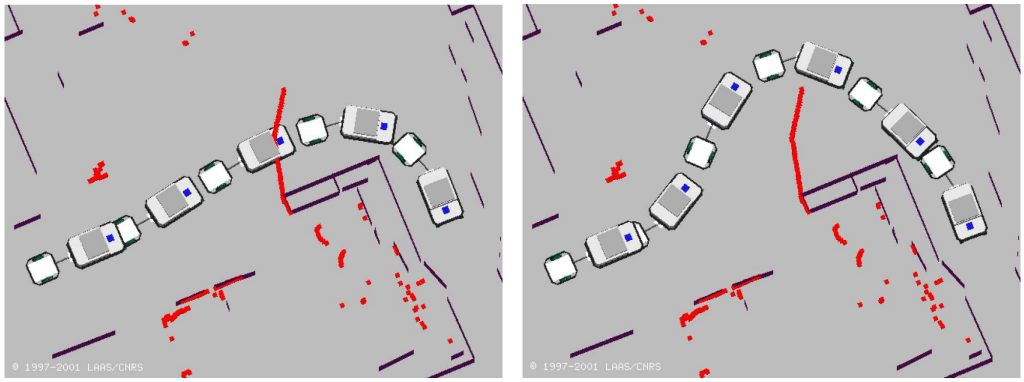
Dans le cas de système articulaires redondants, une amélioration possible de ces travaux consiste en la prise en compte de la tâche à accomplir et de la posture du robot dans le choix de la déformation à adopter [BK02]. Le calcul de la modification à apporter dans l'espace articulaire du robot en fonction des champs de potentiels répulsifs exercés par les obstacles en est ainsi modifié.

3.2.2 Carte de route élastique de Brock

Dans la lignée des bandes élastiques, Yang et Brock ont proposé une approche basée sur une carte de route élastique (elastic roadmap [YB06]). L'approche, destinée à la navigation de robots de type manipulateurs mobiles, construit initialement une roadmap dans l'espace de travail. Le choix de cet espace permet une mise à jour rapide de la roadmap en fonction du déplacement des obstacles mobiles. L'étude de l'existence d'un mouvement faisable entre deux noeuds est assurée par un contrôleur du système prenant en compte ses contraintes cinématiques et dynamiques, ainsi que ses contraintes de postures. Une fois la roadmap construite, un chemin entre une configuration initiale et une configuration finale est planifié par une recherche A*. L'heuristique utilisée prend en compte les contraintes de la tâche à assurer. Lors de la navigation, l'ensemble des noeuds de la roadmap à proximité d'obstacles mobiles sont déplacés et leur connectivité mise à jour. Dans le cas où l'un d'entre eux est néanmoins en collision avec un obstacle malgré la déformation ou qu'il ne soit plus consistant avec la tâche à accomplir, il est considéré comme invalide. Si le chemin suivi passait par l'un de ces derniers, un nouveau chemin est extrait de la roadmap, et le processus de navigation suit son cours sans interruption. Cette méthode dispose donc de l'avantage, par rapport aux méthodes précédentes, de répondre au problème d'homotopies du chemin (limitation à des déformations continues de ce dernier) en modifiant localement en temps réel une partie du plan initial grâce à la roadmap.

3.2.3 Déformation variationnelle de Lamiraux

La déformation de mouvement pour systèmes disposant de contraintes non-holonomes ayant été faiblement étudiée jusqu'alors, Lamiraux et Bonnafous proposèrent une méthode de déformation variationnelle pour un robot de type Hilare 2 (robot type voiture) tirant une remorque ([LBL04]) : A partir d'une représentation paramétrique par séries de Fourier du contrôle d'entrée, une trajectoire définie dans l'espace de configuration du robot au cours du temps peut être obtenue en injectant ce contrôle dans le modèle cinématique du système. Une fonction de potentiel définie par la proximité aux obstacles dans l'espace de configuration du robot est alors définie le long de cette trajectoire. En cas de collisions avec les obstacles lors de l'exécution du mouvement, le chemin suivi peut être écarté des obstacles en faisant décroître le champ de potentiel qui lui est associé. Pour cela une perturbation est appliquée sur le contrôle d'entrée. Le respect des contraintes non-holonomes est assuré en projetant les paramètres de Fourier définissant le contrôle dans l'es-



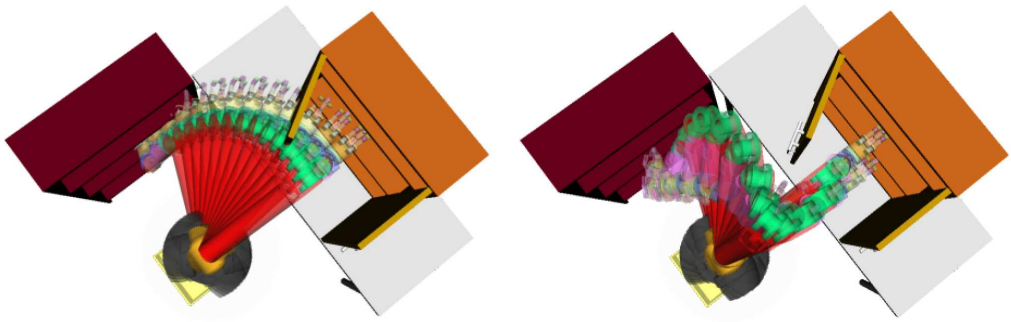
(a) Chemin initial (b) Chemin déformé

FIGURE 3.4 – Déformation de trajectoire variationnelle : En appliquant une perturbation sur les contrôles d'entrée d'un véhicule à remorque, le chemin qu'il suit est modifié afin de s'écarter des obstacles inattendus (source [LBL04]).

pace des paramètres satisfaisant ces contraintes (*cf.* Fig. 3.4). Cette méthode étant assez coûteuse, une extension de ces travaux a été proposée afin de filtrer les obstacles n'ayant aucune influence sur le chemin suivi [LLB05].

3.2.4 Optimisation de gradient de Ratliff & Zucker

Bien que non destinés à déformer une trajectoire au cours du temps de par leur temps de calcul élevé, des travaux plus récents intitulés CHOMP (Covariant Hamiltonian Optimization for Motion Planning) [RZBS09] se rapprochent fortement du principe de déformation : a partir d'une trajectoire Γ, pré-calculée entre une configuration initiale q_0 et une configuration finale q_f, dont la contrainte de non-collision est initialement relaxée, une trajectoire sans collision est planifiée en déformant continuellement la trajectoire initiale. Pour ce faire, une fonction de potentiel est définie le long de la trajectoire. Celle-ci prend à la fois en compte la distance des différents composants du robot aux obstacles (distance calculée dans l'espace de travail $\mathbf{W}$) et la variation de la vitesse et de l'accélération de chaque composant du robot le long de la trajectoire. La trajectoire est alors déformée par descente de gradient afin de minimiser cette fonction de potentiel, permettant ainsi de s'écarter des obstacles tout en assurant la continuité du mouvement. En pratique, la trajectoire est discrétisée afin de simplifier l'application des modifications à apporter sur celle-ci. Dans le cas d'environnements statiques, une représentation de l'espace de travail par voxels est opérée afin de pré-calculer pour chaque voxel libre sa distance à l'obstacle le plus proche. A l'instar



(a) Chemin initial en collision avec les obstacles

(b) Chemin déformé libre d'obstacles

FIGURE 3.5 – Planification par optimisation de gradient : A partir d'une trajectoire initialement en collision et reliant deux configurations buts, un potentiel représentant la distance aux obstacles et la variation du mouvement le long de cette trajectoire est calculé. Une déformation est alors appliquée sur cette trajectoire afin de s'écarter des obstacles tout en garantissant le respect des contraintes d'articulation du système (source [RZBS09]).

de la déformation variationnelle présentée précédemment (3.2.3), l'approche CHOMP destinée à planifier le mouvement de systèmes complexes (bras manipulateurs, robot humanoïdes) prend en compte les limites d'articulation du systèmes en projetant, après chaque itération de déformation, les paramètres articulaires du robot dans l'espace des paramètres articulaires admissibles dans le cas où ces contraintes auraient été dépassées.

Bien qu'adaptable à de nombreux systèmes robotiques, et garantissant en cas de succès à la fois la non-collision avec les obstacles statiques et la faisabilité du mouvement, cette méthode d'une part ne prend pas en compte la dynamique de l'environnement et d'autre part la trajectoire initialement planifiée pouvant être en collision, sa convergence vers un plan sans collision dépend de la nature des obstacles composant son espace de travail.

3.2.5 Planification multi-véhicules par déformation

Dans le même ordre d'idées que l'approche précédente, une planification de trajectoire par déformation pour multiples véhicules disposant de contraintes non-holonomes (voiture de Dubins), et évoluant au milieu d'obstacles mobiles a été proposée dans [AMJP09]. Afin de prendre en compte la dynamique de l'environnement, l'approche suppose disposer d'une connaissance à priori du mouvement des obstacles mobiles. Comme dans l'approche précédente 3.2.4, une trajectoire entre un état initial et un état final (prenant

donc en plus en compte les vitesses initiale et finale) est à priori déterminée indépendamment pour chaque véhicule. La durée, ainsi que la longueur et le profil de vitesse linéaire le long de ces trajectoires sont fixés. Chaque trajectoire est alors discrétisée en une séquence d'états donnés à des intervalles de temps fixés. Chaque état est considéré comme une particule dans un champ de force répulsif généré par le modèle à priori du mouvement des obstacles mobiles d'une part, et par les états intermédiaires des trajectoires des autres véhicules d'autre part. La position des robots définie en chacun de ces états intermédiaires composant leurs trajectoires est alors modifiée dans l'espace ($\mathbb{R}^2$) au temps initialement fixé. Les contraintes non-holonomes de chaque véhicule sont prise en compte de la façon suivante : Considérant trois états successifs d'une trajectoire, une distance minimale entre le premier et le troisième état doit être conservée afin de garantir une courbure maximale le long de cette trajectoire.

Bien que ne considérant qu'un modèle simplifié de véhicules de type voiture (pas de prise en compte de la dynamique du système), et étant fortement limitée par le temps final, la longueur et le profil de vitesse fixes de chaque véhicule, cette approche a le mérite de s'intéresser à la dynamique de l'environnement. Ce point est l'une des caractéristiques fondamentales de nos travaux comme nous le verrons au long de ce chapitre.

3.2.6 Limitations des approches précédentes

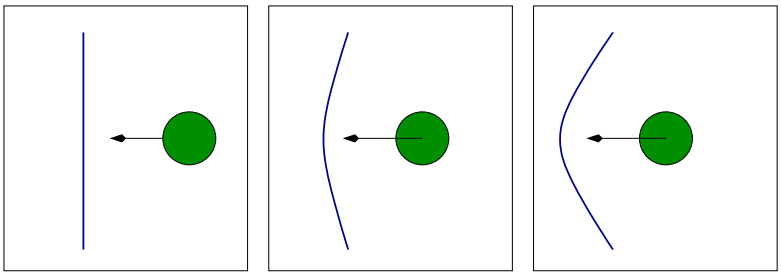


FIGURE 3.6 – Problème lié à la déformation de chemin : en réponse à l'approche d'un objet mobile qui coupe le chemin suivi par le robot, ce dernier est continuellement déformé afin de s'écarter de la position de l'obstacle. La faisabilité du chemin risque en conséquence d'être compromise, alors qu'il aurait été beaucoup plus simple de ralentir à l'approche de l'obstacle et de passer derrière lui.

Au travers des approches précédentes de déformation (3.2), une caractéristique commune à la plupart d'entre elles est à souligner : Seul le

chemin, *i.e.* la courbe géométrique de **W** suivie par le robot, est déformé. Seules la planification par optimisation de gradient (3.2.4) et la déformation variationnelle de Lamiraux (3.2.3) se différencient sur ce point, les profils de vitesse et d'accélération pouvant être modifiés. Néanmoins les travaux de Ratliff & Zucker ne prennent en compte que des obstacles statiques et sont trop coûteux pour être employés en environnement dynamique. Les travaux de Lamiraux sont quant à eux également fort coûteux et ne modifient la trajectoire suivie qu'à partir de "photographies" statiques de l'environnement prises à intervalles de temps régulier.

En dépit de propriétés intéressantes (évitement d'obstacles et convergence vers le but), la déformation de chemin souffre d'un inconvénient majeur illustré par l'exemple suivant : Supposez un scénario comme celui représenté par la figure 3.6 caractérisé par un obstacle s'apprêtant à couper le chemin suivi par le robot. Si l'obstacle continue son mouvement dans la même direction, la déformation de chemin continuera à étirer ce dernier dans ce sens, soulevant les deux problèmes suivants : d'une part, l'étirement du chemin risque de compromettre son suivi, un système robotique disposant de contraintes cinématiques non-holonomes (*e.g.* angle de braquage maximal d'une voiture) ne pourra peut-être pas pouvoir tourner suffisamment. D'autre part, si la vitesse de l'obstacle est importante, le système robotique risque de mettre du temps à le contourner voire de ne jamais y parvenir, alors qu'il aurait été beaucoup plus simple de le laisser passer.

Afin d'éviter ce genre de problème, il est nécessaire de pouvoir d'une part prendre en compte la dynamique de l'environnement dans lequel évolue le système, et d'autre part de pouvoir adapter la vitesse de ce dernier en raisonnant sur l'évolution future de l'environnement. Kurniawati & Fraichard ont alors proposé une approche de *déformation de trajectoire* réactive ([KF07]) qui, a l'instar de la planification par déformation de trajectoire multi-véhicules (3.2.5), prend en compte un modèle prévisionnel du comportement futur de l'environnement. En adaptant la trajectoire déformée à la fois dans l'espace et dans le temps par rapport à cette prévision, il est ainsi possible d'anticiper le mouvement des obstacles mobiles. L'approche proposée dans [KF07] n'était cependant pas capable de maintenir des contraintes non-holonomes repsectées au cours de la déformation, et les cas d'étude traités se limitèrent à un simple système de type masse ponctuelle.

Nous proposons alors dans la suite de ce chapitre la continuité de ces travaux de *déformation de trajectoire* et son extension à des systèmes robotiques disposant de contraintes cinématiques et dynamiques complexes (robot différentiel, véhicule type voiture).

3.3 Teddy : Déformation de trajectoire

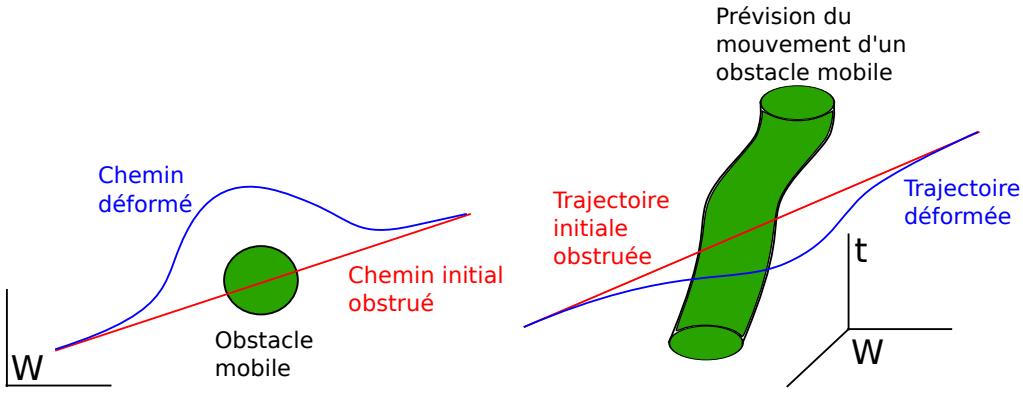
3.3.1 Notation & définitions

Soit un système robotique $\mathcal{A}$ évoluant dans un espace de travail $\mathbf{W}$ ($\mathbb{R}^2$or $\mathbb{R}^3$). La dynamique de $\mathcal{A}$ est décrite par un système d'équations différentielles de la forme :

$$\dot{s} = f(s, u)$$

où $s \in \mathbf{S}$ est l'état de $\mathcal{A}$, $\dot{s}$ sa dérivée par rapport au temps et $u \in \mathbf{U}$ un contrôle. $\mathbf{C}$, $\mathbf{S}$ et $\mathbf{U}$ représentent respectivement l'espace de configuration, l'espace d'état et l'espace de contrôle de $\mathcal{A}$. Soit $\tilde{u} : [0, t_f[\longrightarrow \mathbf{U}$ une trajectoire de contrôles, *i.e.* une séquence de contrôles au cours du temps. A partir d'un état initial s_0 (défini au temps 0) et sous l'action d'un contrôle d'entrée $\tilde{u}$, l'état du système $\mathcal{A}$ au temps t est dénoté par $s(s_0, \tilde{u}, t)$. Un couple $(s_0, \tilde{u})$ définit une *trajectoire d'états* de $\mathcal{A}$, *i.e.* une courbe de $\mathbf{S} \times \mathbf{T}$ où $\mathbf{T}$ dénote la dimension temporelle.

3.3.2 Principe de la déformation de trajectoire



(a) Déformation de chemin (b) Déformation de trajectoire

FIGURE 3.7 – Déformation de chemin vs. déformation de trajectoire : alors que la déformation de chemin se contente de déformer la courbe géométrique de l'espace suivie par le robot pour éviter un obstacle, la déformation de trajectoire utilise une prévision du mouvement des obstacles afin d'anticiper leur futurs mouvements.

A l'instar des approches de déformation de chemin qui l'ont précédées, la déformation de trajectoire combine une planification globale et un évitement réactif afin de permettre à un système robotique $\mathcal{A}$ de naviguer en toute

sécurité entre deux états s_0 et s_N du robot. Une trajectoire d'états $(s_0, \tilde{u})$ est initialement planifiée, et fournie en entrée du système $\mathcal{A}$.

Dans le cas de notre approche de déformation de trajectoire `Teddy`, une trajectoire d'états est discrétisée en une séquence de *noeuds*. Un noeud est un état-temps, il est dénoté par $n_i = (s_i, t_i)$. La trajectoire d'états discrète de $\mathcal{A}$ est donc $\Gamma = \{n_0, n_1 \cdots n_N\}$ où n_0 (resp. n_N) désigne le noeud initial (resp. final) de la trajectoire.

En exerçant sur les différents noeuds de cette trajectoire de $\mathbf{S} \times \mathbf{T}$ un ensemble de forces au cours de l'exécution du mouvement, la séquence de noeuds qui la compose se trouve modifiée à la fois dans l'espace d'états du système, mais aussi dans le temps. Pour ce faire, un modèle prévisionnel du comportement futur des obstacles $\mathbf{B}(t_c, \delta t_h) \subset \mathbf{W} \times \mathbf{T}$ (*cf.* Fig. 3.7), calculé à partir des entrées capteurs, est fourni au processus de déformation `Teddy` à chaque pas de temps t_c. La durée δt_h correspond à *l'horizon temporel i.e.* à la durée pendant laquelle le modèle prévisionnel est considéré comme étant suffisamment fiable. La trajectoire déformée n'est pas modifiée au delà de cette limite de temps.

A la différence de la déformation de chemin qui ne consistait qu'à écarter le chemin suivi par le système robotique de la position courante des obstacles, la déformation de trajectoire essaie donc d'anticiper le mouvement futur des obstacles mobiles en écartant la trajectoire suivi du modèle prévisionnel $\mathbf{B}(t_c, \delta t_h)$ mis à jour périodiquement. Nous présentons ci-dessous l'algorithme `Teddy` destiné à mettre en oeuvre cette déformation.

3.3.3 Algorithme de déformation `Teddy`

Les principales étapes de `Teddy` sont énumérées dans l'algorithme 1. `Teddy` opère périodiquement à une période de temps δt_p. Au cours de l'exécution du mouvement, il prend en entrée à l'instant courant t_c la partie restante de la trajectoire à exécuter $\Gamma^c = \{n_k, n_{k+1} \cdots n_N\} \mid \forall i \in [k; N], t_i > t_c$ (les noeuds de la trajectoire définis jusqu'à l'instant courant sont fixés et non modifiés), ainsi qu'un modèle de l'environnement comportant à la fois la position courante des obstacles mobiles dans $\mathbf{W}$ et le modèle prévisionnel de leur comportement futur $\mathbf{B}(t_c, \delta t_h) \subset \mathbf{W} \times \mathbf{T}$.

Chaque noeud de la trajectoire Γ^c est alors déplacé suite à l'application d'un ensemble de forces : tout d'abord des forces répulsives notées $\mathbf{F}_{rep}$ (connues dans la littérature comme forces externes) provenant des obstacles de l'environnement permettent d'écarter la trajectoire suivie du mouvement prévisionnel des obstacles afin de garantir la non-collision. Celles-ci sont détaillées en Section 3.3.4. Dans le cas où les obstacles ne seraient plus à proximité d'une partie de la trajectoire, une force élastique est appliquée

Algorithm 1: `Teddy`.

Input: $\Gamma^c = \{n_k, n_{k+1} \cdots n_N\}$, *modèle prévisionnel* $\mathbf{B}(t_c, \delta t_h)$
Output: $\Gamma^{\hat{c}} = \{\hat{n}_k, \hat{n}_{k+1} \cdots \hat{n}_{\hat{N}}\}$

`// Application des forces répulsives et élastiques`
$\Gamma^{c'} = \emptyset$;
foreach $n_i \in \Gamma^c$ **do**
    $n_i^{'} = n_i + \mathbf{F}_{rep}(n_i) + \mathbf{F}_{ela}(n_i)$;
    $\Gamma^{c'} = \Gamma^{c'} \cup n_i^{'}$;
end
`// Restauration de la cohérence temporelle`
$\Gamma^{c''} = RestaureCTemp\ (\Gamma^{c'})$;
`// Maintien de la connectivité de la trajectoire`
$\Gamma^{c'''} = \emptyset$;
foreach $n_i^{''} \in \Gamma^{c''}$ **do**
    $n_i^{'''} = n_i^{''} + \mathbf{F}_{conn}(n_i^{''})$;
    $\Gamma^{c'''} = \Gamma^{c'''} \cup n_i^{'''}$;
end
`// Rééchantillonnage de la trajectoire`
$\Gamma^{\hat{c}} = Rééchantillonne\ (\Gamma^{c'''})$;
`// Vérification de la validité de la trajectoire`
if *non valide* $(\Gamma^{\hat{c}})$ **then**
    Déterminer un nouveau plan;
end
return $\Gamma^{\hat{c}}$;

sur les noeuds qui la compose afin de "tendre" la trajectoire, limitant ainsi la longueur du chemin parcouru et les variations du mouvement. Ces forces élastiques sont détaillées en Section 3.3.5. Une fois ces deux forces appliquées, la faisabilité de la trajectoire n'est plus garantie : en effet, les noeuds étant déplacés librement dans $\mathbf{S} \times \mathbf{T}$ au gré des forces répulsives et élastiques, les contraintes cinématiques et dynamiques du système robotique peuvent ne plus être respectées le long de celles-ci. Une étape de maintenance de la connectivité de la trajectoire est donc nécessaire (*cf.* Section 3.3.6). Ensuite, dans le cadre des tests de collisions le long de la trajectoire déformée, un échantillonnage régulier des états-temps le long de celle-ci est nécessaire. Sa mise en place est décrite en Section 3.3.7.

A chaque temps $t_{\hat{c}} = t_c + \delta t_p$, après application de l'ensemble de ces

modifications sur la trajectoire d'états suivie par le robot, `Teddy` retourne la trajectoire déformée $\Gamma^{\hat{c}} = \{\hat{n}_k, \hat{n}_{k+1} \cdots \hat{n}_{\hat{N}}\}$ au contrôleur du système robotique avant de réitérer le processus de déformation. Il est important de noter que, comme dans le cas de la déformation de chemin, la déformation de trajectoire possède la limitation suivante : il n'est en aucun cas garanti que la trajectoire déformée $\Gamma^{\hat{c}}$ soit à la fois sans collision et connectée à chaque pas de temps. Cette approche reste en effet heuristique de nature. Un échec de la déformation se présente généralement lorsque la topologie de $\mathbf{S} \times \mathbf{T}$ change fortement, par exemple lorsqu'un passage initialement libre se trouve bloqué (porte qui se ferme). La non-collision et la connectivité de la trajectoire sont donc testées après chaque déformation. Ces tests sont décrits en Section 3.3.8. Dans le cas où l'une d'entre elles ne soit pas respectée, le processus de déformation et le mouvement du robot sont arrêtés, et un planificateur de trajectoire est alors ré-invoqué afin de déterminer un nouveau mouvement à suivre. Notez que le module de planification ne faisant pas partie intégrante de `Teddy`, il n'est pas discuté dans ce chapitre. Néanmoins, il sera décrit au chapitre 5 présentant l'intégration de `Teddy` à une architecture de navigation complète.

3.3.4 Forces répulsives

Les forces externes sont des forces répulsives exercées par les obstacles de l'environnement dans le but d'éviter toutes collisions. Elles sont dérivées d'un champ de potentiel V_{rep} calculé à partir du modèle prévisionnel du comportement futur des obstacles $\mathbf{B}(t_c, \delta t_h)$ fourni au temps t_c par les capteurs du système. Le modèle de l'environnement obtenu par les capteurs ainsi que le modèle prévisionnel du mouvement des obstacles $\mathbf{B}(t_c, \delta t_h)$ étant générés dans l'espace de travail $\mathbf{W} \times \mathbf{T}$, le champ de potentiel en résultant est alors calculé dans ce même espace. D'une manière similaire à [BK02], un ensemble de r points de contrôle p^j sont sélectionnés sur le corps du système $\mathcal{A}$. A chaque noeud $n_i = (s_i, t_i)$ de la trajectoire déformée Γ^c est donc associé un ensemble de points de contrôle $c_i^j = (p^j, t_i)$ dans $\mathbf{W} \times \mathbf{T}$. Pour chaque point de contrôle $c_i^j,\ j \in [1; r]$, V_{rep} est défini par :

$$V_{rep}(c_i^j) = \begin{cases} k_{rep} k_h(t_i)(d_0 - d_{wt}(c_i^j))^2 & \text{si } d_{wt}(c_i^j) < d_0 \\ 0 & \text{sinon} \end{cases} \tag{3.1}$$

où $d_{wt}(c_i^j)$ est la distance de c_i^j à l'obstacle le plus proche du modèle prévisionnel $\mathbf{B}(t_c, \delta t_h) \in \mathbf{W} \times \mathbf{T}$. d_0 est la distance maximale d'influence définie autour des obstacles, et k_{rep} est un gain répulsif. $k_h(t_i)$ est un gain relatif à la fiabilité du modèle prévisionnel $\mathbf{B}(t_c, \delta t_h)$. Il est généralement

défini par une fonction continue et décroissante telle que $k_h(t_c) = 1$ et $k_h(t_i) = 0,\ \forall t_i \geq t_c + \delta t_h$ (le modèle est supposé totalement fiable à l'instant courant et plus du tout fiable au delà de l'horizon temporel).

d_{wt} est une distance de $\mathbf{W} \times \mathbf{T}$. Elle est dérivée de la distance Euclidienne en définissant une métrique entre les dimensions spatiale et temporelle. Dans $\mathbb{R}^2$ par exemple, la distance d_{wt} entre $c_0 = (x_0, y_0, t_0)$ et $c_1 = (x_1, y_1, t_1)$ peut être calculée par :

$$\begin{aligned} {d_{wt}}^2(c_0, c_1) &= {w_s}^2(x_1 - x_0)^2 + {w_s}^2(y_1 - y_0)^2 \\ &\quad + {w_t}^2(t_1 - t_0)^2 \end{aligned} \tag{3.2}$$

où w_s et w_t représente respectivement les poids spatial et temporel associé à chacun de ces espaces. La force résultante de ce champ de potentiel agissant sur un point de contrôle c_i^j est alors définie par :

$$\mathbf{F}_{rep}^{wt}(c_i^j) = -\nabla V_{rep}(c_i^j) = k_{rep} k_h(t_i)(d_0 - d_{wt}(c_i^j))\frac{\mathbf{d}}{||\mathbf{d}||} \tag{3.3}$$

où $\mathbf{d}$ est le vecteur défini entre c_i^j et le point le plus proche des obstacles de l'environnement. Ces forces étant destinées à être appliquées sur la trajectoire d'état de $\mathbf{S} \times \mathbf{T}$ suivie par le système, elle doivent donc être retranscrites dans $\mathbf{S} \times \mathbf{T}$. La combinaison des forces définies dans $\mathbf{W} \times \mathbf{T}$ par chaque point de contrôle c_i^j produit premièrement une force dans $\mathbf{C} \times \mathbf{T}$ définie comme suit :

$$\mathbf{F}_{rep}^{ct}(q, t_i) = \sum_{j=1}^{r} J^T_{c_i^j}(q, t) \mathbf{F}_{rep}^{wt}(c_i^j) \tag{3.4}$$

où $J^T_{c_i^j}(q, t)$ représente la matrice Jacobienne de la configuration q du système robotique définie au point de contrôle c_i^j :

$$J^T_{c_i^j}(q, t_i) = \begin{pmatrix} \frac{\partial q^1}{\partial p_1^j} & \cdots & \frac{\partial q^1}{\partial p_m^j} & 0 \\ \vdots & \ddots & \vdots & \vdots \\ \frac{\partial q^n}{\partial p_1^j} & \cdots & \frac{\partial q^n}{\partial p_m^j} & 0 \\ 0 & \cdots & 0 & 1 \end{pmatrix} \tag{3.5}$$

où m est la dimension de $\mathbf{W}$, p_l^j la l^{eme} coordonnée de p^j, n la dimension de $\mathbf{C}$ et q^l la l^{eme} coordonnée de q. La retranscription finale de cette force dans $\mathbf{S} \times \mathbf{T}$ produisant la force $\mathbf{F}_{rep}(n) = \mathbf{F}_{rep}(s, t)$ est simplement obtenue en laissant les paramètres restant de l'état du système inchangés. Nous verrons en Section 3.3.6 comment ces paramètres sont ajustés au cours du processus de déformation.

3.3.5 Forces élastiques

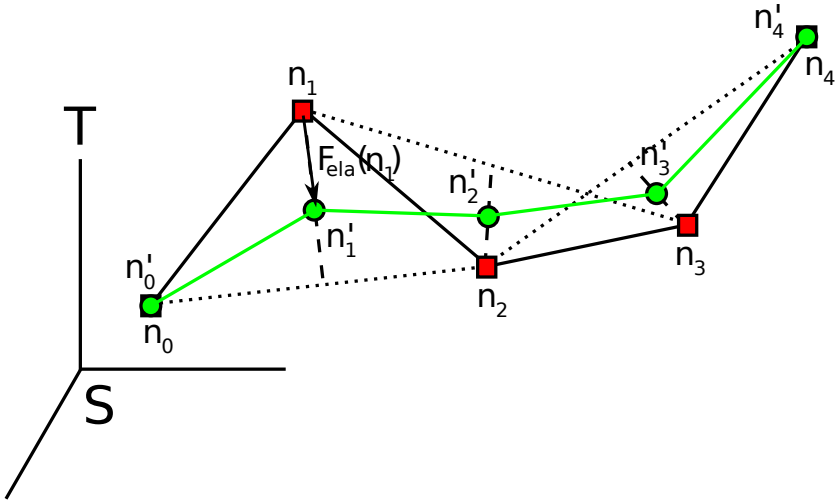


FIGURE 3.8 – Forces élastiques : Soit une trajectoire discrète initiale $\Gamma = \{n_0, \ldots, n_4\}$. Lorsque les noeuds de la trajectoire déformée sont hors de la région d'influence des obstacles, des forces élastiques $\mathbf{F}^{wt}_{ela}(n_i)$ sont appliquées sur chaque état-temps discret n_i de celle-ci afin de la "retendre". La trajectoire résultante $\Gamma' = \{n'_0, \ldots, n'_4\}$ limite ainsi la distance parcourue et les variations du mouvement (dans cet exemple, le gain élastique est $k_{ela} = \frac{1}{2}$).

Lorsque les obstacles de l'environnement ne sont plus à proximité de la trajectoire suivie par le système robotique, il peut être intéressant de "retendre" la trajectoire. Cette procédure permet d'une part de minimiser la longueur du chemin parcouru, laissant ainsi davantage de manoeuvres quant au profil de vitesse utilisé le long de celui-ci. D'autre part les variations du mouvement peuvent ainsi être limitées.

Considérons donc trois noeuds successifs n_-, n et n_+ de la trajectoire déformée Γ^c tels que, les points de contrôle associés c^j $(j \in [1;r])$ définis sur le robot à l'état-temps (noeud) n soient hors de portée du champ répulsif des obstacles. Une force élastique est alors appliquée sur le noeud intermédiaire n :

$$\mathbf{F}^{wt}_{ela}(n) = \begin{cases} k_{ela} * \left(\frac{1}{2}n_- + \frac{1}{2}n_+ - n\right) & \text{si } d_{wt}(c^j) \geq d_0,\ \forall j \in [1;r] \\ 0 & \text{sinon} \end{cases} \tag{3.6}$$

où $k_{ela} \in [0;1]$ est le gain associé à cette force élastique.

Notez que cette force ne suffit en aucun cas à remplacer le processus de maintenance de connectivité de la trajectoire décrit en Section 3.3.6, le

respect des contraintes cinématiques et dynamiques du systèmes n'étant en aucun cas assuré. De la même manière, le processus de maintien de la connectivité ne permet en général pas de réduire la longueur du chemin parcouru. Ces deux étapes sont donc bien complémentaires.

3.3.6 Maintien de la connectivité de la trajectoire

Chaque noeud de la trajectoire déformée $\Gamma^{c'}$ étant déplacé librement dans $\mathbf{S} \times \mathbf{T}$ au gré des forces répulsives et élastiques, la connectivité et la faisabilité de la trajectoire peuvent être perdues. Dans le cas de fortes déformations, il peut même s'avérer que la cohérence temporelle de la trajectoire (garantie que le temps soit strictement croissant le long de la trajectoire) soit compromise ! Il est donc nécessaire de s'assurer de la validité de la trajectoire, et que les contraintes cinématiques et dynamiques du système robotique soient toujours respectées le long de celle-ci. De par la discrétisation de $\Gamma^{c'}$, la trajectoire est considérée être connectée s'il existe un mouvement faisable entre chaque couple de noeuds successifs de $\Gamma^{c'}$. La connectivité de la trajectoire est donc liée à l'atteignabilité d'un état-temps par un second état-temps. Ce concept est décrit ci-dessous.

Caractérisation des espaces atteignables

Def. 5 (Atteignabilité) *Une état s d'un système robotique $\mathcal{A}$ est dit atteignable par un second état s_0 de $\mathcal{A}$ en un temps δt s'il existe une trajectoire respectant les contraintes cinématiques et dynamiques de $\mathcal{A}$ partant de s_0 à un temps t_0 et arrivant à l'état s au temps $t_0 + \delta t$.*

L'ensemble des états atteignables à partir d'un état s_0 à un temps t est défini par :

$$\mathcal{R}(s_0, t) = \{s \in \mathbf{S} \mid \exists \tilde{u}, s(s_0, \tilde{u}, t_f) = s\} \tag{3.7}$$

Cette notion peut être trivialement étendue à un intervalle de temps $[t_{min}; t_{max}]$:

$$\mathcal{R}(s_0, t_{min}, t_{max}) = \{s \in \mathbf{S} \mid \exists t \in [t_{min}; t_{max}], \exists \tilde{u}, s(s_0, \tilde{u}, t) = s\} \tag{3.8}$$

De la même manière, l'ensemble des états pouvant atteindre un état s_f durant un intervalle de temps $[t_{min}; t_{max}]$ sont définis par :

$$\mathcal{R}^{-1}(s_f, t_{min}, t_{max}) = \{s \in \mathbf{S} \mid \exists t \in [t_{min}; t_{max}], \exists \tilde{u}, s(s, \tilde{u}, t) = s_f\} \tag{3.9}$$

Soient trois états-temps consécutifs $n_- = (s_-, t_-)$, $n = (s, t)$ et $n_+ = (s_+, t_+)$ de la trajectoire déformée $\Gamma^{c'}$. La connectivité de $\Gamma^{c'}$ est assurée à

l'état-temps $n = (s,t)$ ssi $(s,t) \in \mathcal{R}(s_-, t_-, \infty)$ et $(s_+, t_+) \in \mathcal{R}(s,t,\infty)$. En d'autres termes, l'état-temps intermédiaire (s,t) doit appartenir à l'intersection $\mathcal{R}(s_-, t_-, \infty) \cap \mathcal{R}^{-1}(s_+, -\infty, t_+)$. Une solution permettant de restaurer la connectivité de la trajectoire déformée $\Gamma^{c'}$ serait alors de calculer une force $\mathbf{F}_{conn}(n)$ à appliquer sur le noeud intermédiaire n de chaque triplé d'états-temps consécutifs de $\Gamma^{c'}$, tel que $\mathbf{F}_{conn}(n)$ ramène n dans l'espace atteignable $\mathcal{R}(s_-, t_-, \infty) \cap \mathcal{R}^{-1}(s_+, -\infty, t_+)$ défini entre son prédécesseur et son successeur. Une solution satisfaisante serait par exemple de calculer $\mathbf{F}_{conn}(n)$ à partir d'un champ de potentiel attractif entre n et le barycentre H de l'intersection $\mathcal{R}(s_-, t_-, \infty) \cap \mathcal{R}^{-1}(s_+, -\infty, t_+)$.

La caractérisation de tels espaces atteignables pour un système robotique arbitraire est cependant un processus dépendant de la dimension de l'espace d'état du système et de la linéarité ou non linéarité de sa dynamique (*cf.* [ADF+06, Mit07]), et peut donc s'avérer fort coûteuse pour des systèmes complexes. `Teddy` disposant d'un temps limité δt_p pour déformer la trajectoire, il est nécessaire qu'il soit capable de procéder au maintien de la connectivité de la trajectoire aussi efficacement que possible. Pour ce faire, une solution alternative au calcul des espaces atteignables est proposée : Nous souhaiterions essayer de calculer une seule et unique trajectoire $\Gamma(n_-, n_+)$ respectant l'ensemble des contraintes cinématiques et dynamiques du système robotique entre chaque couple de noeuds (n_-, n_+) et nous assurer que l'état-temps intermédiaire n appartienne à $\Gamma(n_-, n_+)$. Un générateur de trajectoire a donc été utilisé dans ce but. Néanmoins, si la cohérence temporelle entre deux états-temps n_- et n_+ a été perdue (si $t_- \geq t_+$), il est impossible de générer une trajectoire entre ceux-ci. Le maintient de la connectivité de la trajectoire déformée est alors effectué en deux étapes : tout d'abord, la cohérence temporelle est rétablie si nécessaire ; une génération de trajectoire faisable entre chaque triplé d'états-temps consécutifs de $\Gamma^{c'}$ est ensuite opérée. Ces deux étapes sont détaillées ci-dessous.

Rétablissement de la cohérence temporelle

Notons $\Gamma^{c'}$ la trajectoire déformée au temps t_c après application des forces externes et élastiques. Comme précisé précédemment, la connectivité de $\Gamma^{c'}$ peut avoir été perdue lors de l'application de ces deux forces. Mais avant même de pouvoir essayer de rétablir cette connectivité, il est nécessaire de s'assurer que la cohérence temporelle de la trajectoire n'a pas été perdue. Grâce à la discrétisation de $\Gamma^{c'} = \{n'_k, n'_{k+1} \cdots n'_N\} \mid \forall i \in [k; N], t'_i > t'_c$ elle peut être testée simplement : la cohérence temporelle est considérée perdue si

$$\exists n'_i, n'_j \in \Gamma^{c'} \mid j > i \;\&\; t'_j < t'_i \tag{3.10}$$

Algorithm 2: *RestaureCTemp* : Restauration de la cohérence temporelle de la trajectoire.

Input: $\Gamma^{c'} = \{n_k^{'}, n_{k+1}^{'} \cdots n_N^{'}\}$
Output: $\Gamma^{c''} = \{n_k^{''}, n_{k+1}^{''} \cdots n_N^{''}\}$

$\Gamma^{c''} = \emptyset$;
$\Gamma^{c''} = \Gamma^{c''} \cup n_k^{'}$;
// Restauration de la cohérence temporelle de chaque noeud par rapport à son prédécesseur
foreach $i \in [k; N-1]$ **do**
  $n_{i+1}^{''} = n_{i+1}^{'}$
  // Test de dépassement de la vitesse maximale du robot
  foreach $l \in [1; r]$ **do**
    if $t_{i+1}{''} < t_i{''} + \frac{p_{i+1}^l - p_i^l}{\dot{p}_{max}^l}$ **then**
      $t_{i+1}{''} = t_i{''} + \frac{p_{i+1}^l - p_i^l}{\dot{p}_{max}^l}$
    end
  end
  $\Gamma^{c''} = \Gamma^{c''} \cup n_{i+1}^{''}$;
end
return $\Gamma^{c''}$;

Dans un tel cas, il sera impossible de déterminer un quelconque mouvement faisable entre les noeuds $n_i^{'}$ et $n_j^{'}$.

Supposons un système robotique $\mathcal{A}$ constitué d'un seul et unique corps, défini par sa position p appartenant à l'espace de travail $\mathbf{W} = \mathbb{R}^2$ ou $\mathbb{R}^3$. A partir des contraintes cinématiques et dynamiques de $\mathcal{A}$, il est possible de déterminer la vitesse maximale $\dot{p}_{max}$ de $\mathcal{A}$ dans $\mathbf{W}$. La cohérence temporelle de la trajectoire suivie par $\mathcal{A}$ entre $n_i^{'}$ et $n_j^{'}$ peut alors s'étendre à la condition suivante :

$$\exists n_i^{'}, n_j^{'} \in \Gamma^{c'}, \mid j > i \;\&\; t_j^{'} < t_i^{'} + \frac{p_j - p_i}{\dot{p}_{max}} \tag{3.11}$$

Dans le cas d'un système robotique $\mathcal{A}$ plus complexe (bras manipulateur, manipulateur mobile, etc), sur lequel sont définis plusieurs points de contrôle caractérisés par leurs positions respectives $p^k,\ k \in [1; r]$, cette condition devient :

$$\exists n_i^{'}, n_j^{'} \in \Gamma^{c'} \;\&\; \exists k \in [1; r] \mid j > i \;\&\; t_j^{'} < t_i^{'} + \frac{p_j^k - p_i^k}{\dot{p}_{max}^k} \tag{3.12}$$

L'algorithme 2 est alors exécuté après application des forces externes et élastiques. Même s'il n'assure en aucun cas qu'après son exécution l'ensemble des contraintes sur le mouvement de $\mathcal{A}$ soient respectées le long de la trajectoire, il permet au moins de rétablir la cohérence temporelle nécessaire à la faisabilité du mouvement. De plus, c'est une manière heuristique de faciliter le processus de restauration de la connectivité de la trajectoire lui succédant, en réduisant au préalable les dépassements des vitesses maximales du système.

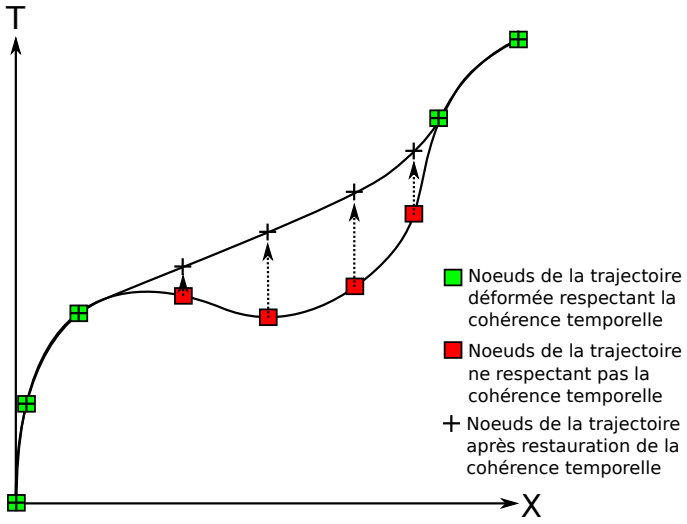


FIGURE 3.9 – Restauration de la cohérence temporelle de long de la trajectoire d'un système se déplaçant le long d'un axe unique X au cours du temps T. La pente de la partie modifiée de cette courbe correspond à la vitesse maximale du système le long de X.

Notez qu'un tel procédé peut entraîner un décalage en chaîne de plusieurs noeuds de la trajectoire dû au temps minimal conservé entre chacun des noeuds (relatif à la distance qui les sépare). Cependant en pratique, en s'assurant que la trajectoire initiale n'ait pas été planifiée à vitesse maximale, un intervalle réduit d'états-temps successifs se trouve concerné (*cf.* Fig. 3.9).

Restauration de la connectivité par génération de trajectoires

Une fois la cohérence temporelle assurée, le processus de restauration de la connectivité peut être mis en place. Nous rappelons que son but est de vérifier si les contraintes cinématiques et dynamiques du système robotique $\mathcal{A}$ sont vérifiées tout au long de la trajectoire déformée $\Gamma^{c''}$. Cette trajectoire

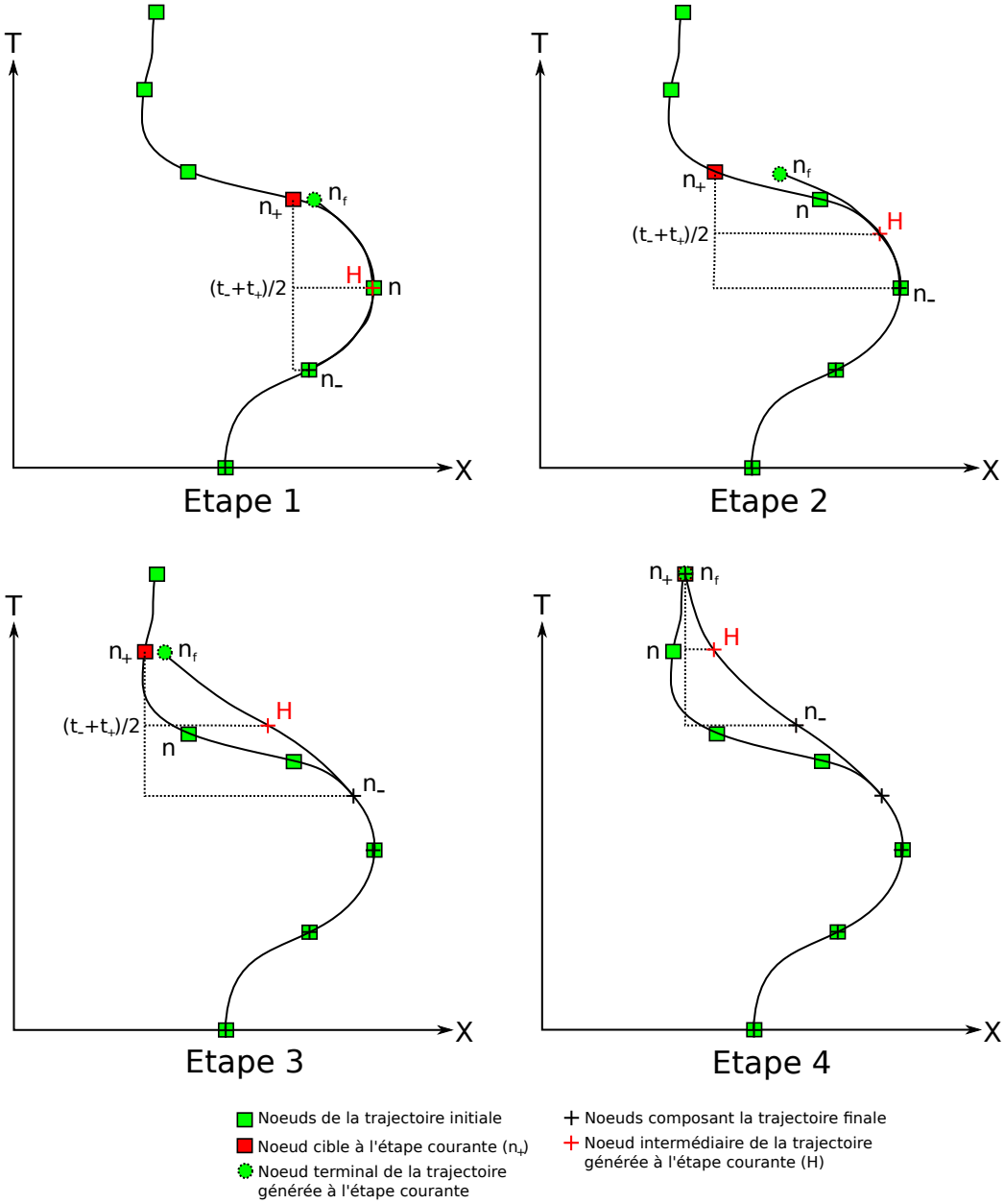


FIGURE 3.10 – Restauration de la connectivité de la trajectoire déformée : un générateur de trajectoires nommé `Tiji` est utilisé afin de déterminer l'existence d'un mouvement admissible entre chaque triplet d'états-temps consécutifs n_-, n, n_+ de la trajectoire déformée Γ^c. Dans le cas où n_+ n'est pas atteignable par n_-, une trajectoire $\Gamma(n_-, n_+)$ admissible s'arrêtant en un état s_f proche de s_+ au temps t_+ est déterminé. Son état-temps intermédiaire H (défini au temps $\frac{t_-+t_+}{2}$) remplace alors l'état-temps n de Γ^c. En appliquant ce procédé à partir de l'état-temps courant jusqu'à l'état-temps terminal de la trajectoire déformée, on peut ainsi rétablir sa connectivité.

Algorithm 3: *RestaureConnec* : Restauration de la connectivité de la trajectoire déformée.

Input: $\Gamma^{c''} = \{n''_k, n''_{k+1} \cdots n''_N\}$
Output: $\Gamma^{c'''} = \{n'''_k, n'''_{k+1} \cdots n'''_N\}$

$H = n''_k$;
$\Gamma^{c'''} = H$;
// Restauration de la connectivité entre chaque triplé de noeuds de la trajectoire déformée
foreach $n''_i \in \{n''_{k+1}, n''_{k+2} \cdots n''_{N-1}\}$ **do**
    // Récup. des premier et dernier noeuds du triplet
    $n_- = H$;
    $n_+ = n''_{i+1}$;
    // Génération de trajectoire entre n_- et n_+
    $\Gamma(n_-, n_+) = \texttt{Tiji}(n_-, n_+)$;
    // Récup. du noeud intermédiaire de la traj. générée
    $H = \Gamma(n_-, n_+, \frac{t_- + t_+}{2})$;
    $\Gamma^{c'''} = \Gamma^{c'''} \cup H$;
end
$\Gamma^{c'''} = \Gamma^{c'''} \cup n'''_N$;
return $\Gamma^{c'''}$;

étant discrétisée en une séquence de noeuds $\Gamma^{c''} = \{n''_k, n''_{k+1} \cdots n''_N\} \mid \forall i \in [k; N], t''_i > t''_m$, nous cherchons donc à maintenir l'existence d'une trajectoire admissible entre chaque triplet n_-, n, n_+ de noeuds successifs de $\Gamma^{c''}$.

Notre processus de restauration de la connectivité (résumé par l'algorithme 3) part donc du principe suivant : Au lieu d'essayer de calculer l'espace atteignable $\mathcal{R}(s_-, t_-, \infty) \cap \mathcal{R}^{-1}(s_+, -\infty, t_+)$ caractérisant l'ensemble de tous les mouvements admissibles partant de l'état s_- au temps t_- et atteignant l'état s_+ au temps t_+, nous proposons d'essayer de calculer une seule et unique trajectoire $\Gamma(n_-, n_+)$ connectant ces deux états-temps. Si une telle trajectoire existe, le noeud n peut être ramené vers l'état intermédiaire de $\Gamma(n_-, n_+)$ (état défini au temps $t = \frac{t_- + t_+}{2}$). En appliquant ce procédé sur chaque triplet n_-, n, n_+ de $\Gamma^{c''}$, sa connectivité est garantie.

Déterminer une trajectoire $\Gamma(n_-, n_+)$ entre deux états-temps $n_- = (s_-, t_-)$ et $n_+ = (s_+, t_+)$ satisfaisant les contraintes cinématiques et dynamiques d'un système $\mathcal{A}$ relève d'un processus de génération de trajectoire. Un problème se pose néanmoins dans le cadre de la déformation : les noeuds n_- et n_+ étant déplacés au gré de forces extérieures, il n'y a aucune garantie

qu'une telle trajectoire existe ! Néanmoins, en déterminant une trajectoire à partir de n_- et dont l'état final s_f défini au temps t_+ ($n_f = (s_f, t_+)$) se trouve "aussi près que possible" de l'état s_+ (via une métrique sur l'espace d'état $\mathbf{S}$), il est possible en appliquant ce procédé sur les noeuds successifs de $\Gamma^{c''}$ de "rattraper" la trajectoire déformée tout en garantissant sa faisabilité (*cf.* Fig. 3.10).

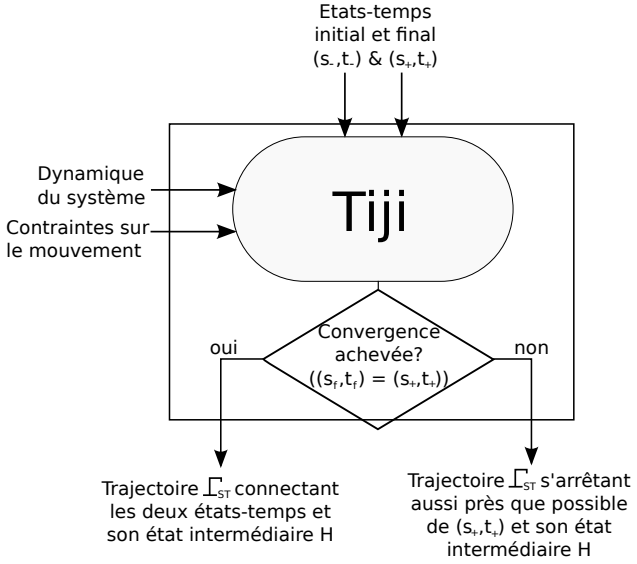


FIGURE 3.11 – Utilisation du générateur de trajectoires `Tiji` dans le cadre du processus de déformation de trajectoire `Teddy`.

Malheureusement, un tel générateur de trajectoire n'existait pas à notre connaissance dans la littérature. Nous avons alors développé un algorithme de génération de trajectoire nommé `Tiji` ([DFMG09]) correspondant à ces attentes. La génération de trajectoire étant un sujet de discussion à part entière, `Tiji` est décrit plus en détails dans le chapitre 4. Au sein de la déformation, `Tiji` est utilisé comme une boîte noire (*cf.* Fig. 3.11) prenant en entrée les états-temps $n_- = (s_-, t_-)$ et $n_+ = (s_+, t_+)$ et retournant une trajectoire $\Gamma(n_-, n_+)$, son état-temps intermédiaire H (état de $\Gamma(n_-, n_+)$ défini au temps $t = \frac{t_- + t_+}{2}$), et un booléen statuant si l'état final de $\Gamma(n_-, n_+)$ correspond à n_+ ou non.

Que le noeud but ait été atteint ou non, la force $\mathbf{F}_{conn}(n)$ appliquée sur un noeud n de la trajectoire déformée $\Gamma^{c''} = \{n''_k, n''_{k+1} \cdots n''_N\}$ est définie par :

$$\mathbf{F}_{conn}(n) = H - n \tag{3.13}$$

Chaque noeud n de $\Gamma^{c''}$ est alors repositionné sur l'état-temps intermédiaire H d'une trajectoire provenant de son prédécesseur n_- et s'arrêtant sur ou à proximité de son successeur n_+. En appliquant ce processus par propagation à partir du premier noeud n''_k de $\Gamma^{c''}$ jusqu'à son dernier noeud n''_N, la connectivité de la trajectoire est assurée de proche en proche. Le seul noeud vers lequel la connectivité n'est pas garantie est donc n''_N. En effet, si la trajectoire générée par `Tiji` à partir du noeud n''_{N-2} vers le noeud n''_N n'a pas atteint son but, alors il n'est pas possible comme pour les précédents de choisir un noeud final différent du but. C'est le seul et unique cas dans lequel la trajectoire peut se retrouver déconnectée. Ce cas est discuté en Section 3.3.8.

3.3.7 Rééchantillonnage de la trajectoire

Au cours de la déformation, l'écart entre les noeuds de la trajectoire peut soit augmenter progressivement ou au contraire réduire considérablement (aussi bien dans la dimension spatiale que temporelle) suite aux différentes forces appliquées. Afin de mener à bien les étapes successives de déformation, il est désirable de conserver un échantillonnage régulier de la trajectoire : La dicrétisation de la trajectoire définit d'une part le nombre de noeuds sur lesquels les différentes forces (externes, élastiques, maintient de la connectivité) sont appliquées, mais également les différents états du robot auxquels la vérification de la non-collision va être effectuée. La distance entre deux noeuds consécutifs de la trajectoire ne doit donc pas être trop importante afin d'éviter que la portion de trajectoire définie entre ceux-ci ne soit en collision. D'autre part, utiliser une distance trop faible entre les noeuds de la trajectoire déformée augmenterait fortement les oscillations le long de cette dernière et en conséquence le risque de perte de connectivité.

Comme nous le verrons dans le chapitre suivant (4), les trajectoires $\Gamma(n_-, n_+)$ générées par `Tiji` lors de l'étape précédente de restauration de la connectivité de la trajectoire déformée disposent d'une expression analytique (forme close ou quadratique). La trajectoire déformée $\Gamma^{c'''}$ étant obtenue par la concaténation de telles trajectoires, il est facile de l'évaluer à n'importe quel temps donné. Le rééchantillonnage de la trajectoire déformée s'effectue donc simplement en réévaluant $\Gamma^{c'''}$ à un pas de temps constant à partir de l'expression analytique des trajectoires intermédiaires qui la composent. La trajectoire rééchantillonnée est donc notée $\Gamma^{\hat{c}}$. Cette dernière constitue la nouvelle trajectoire passée en entrée de l'étape suivante de déformation.

Algorithm 4: Vérification de la validité de la trajectoire.

Input: $\Gamma^{\hat{c}} = \{\hat{n}_k, \hat{n}_{k+1} \cdots \hat{n}_N\}$,
Modèle prévisionnel $\mathbf{B}(t_c, \delta t_h)$, δt_{seuil}
Output: *Validité de la trajectoire*

```
// La trajectoire est supposée initialement valide
valide = vrai
// Teste les éventuelles collisions le long de la
   trajectoire
foreach n̂_i ∈ Γ^ĉ do
    if t̂_i < t_c + δt_seuil then
        // Teste si le noeud courant est en collision
        if estEnCollision(n_i) then
            valide = faux
        end
    end
end
// Teste si le dernier noeud de la trajectoire est
   déconnecté
if t̂_N̂ < t_c + δt_seuil & n̂_N̂ ∉ R(n̂_{N̂-2}) then
    valide = faux
end
return valide;
```

3.3.8 Vérification de la validité de la trajectoire

L'approche de déformation de trajectoire `Teddy` étant heuristique par principe, il est impossible de garantir après chaque déformation qu'elle soit sans collision et connectée à l'état but. Il est alors nécessaire de vérifier ces deux conditions à chaque pas de temps. Si la trajectoire est en collision ou déconnectée dans un futur suffisamment lointain, il est possible d'essayer de corriger le problème au cours des itérations suivantes de `Teddy`. Nous définissons alors un intervalle de temps δt_{seuil} à partir de l'instant présent pendant lequel il est nécessaire de garantir la non-collision et la connectivité du mouvement. Si ces deux conditions ne sont pas respectées, une manoeuvre de freinage doit alors être calculée afin d'arrêter le système et de pouvoir planifier une nouvelle trajectoire pour rejoindre l'état but. L'algorithme 4 résume le processus de vérification de la validité de la trajectoire.

Le test de collision est effectué en vérifiant à partir du modèle prévisionnel du comportement futur des obstacles $\mathbf{B}(t_c, \delta t_h)$ si la position du robot cor-

respondant à chaque état de la trajectoire définie jusqu'au temps $t_c + \delta t_{seuil}$ est en collision avec celui-ci. Un module de test de collision externe est utilisé pour cela. Quant à la connectivité, il n'est nécessaire comme expliqué en Section 3.3.6 que de vérifier si l'antépénutième et le dernier noeud de la trajectoire $\hat{n}_{\hat{N}-2}$ et $\hat{n}_{\hat{N}}$ sont connectés, et encore sous condition que $t_{\hat{N}} < t_c + \delta t_{seuil}$.

Le seuil de vérification de la validité de la trajectoire δt_{seuil} est déterminé de telle sorte qu'il soit possible de garantir la *sécurité passive* du système *i.e.* que le système robotique soit capable de freiner et d'atteindre une vitesse nulle avant de rentrer en collision. Il importe peu au cours du processus de déformation que la trajectoire soit en collision ou déconnectée après ce temps, ces problèmes pouvant être résolus durant les itérations suivantes de déformation. Le seuil δt_{seuil} peut par exemple être fixé à la valeur suivante :

$$\delta t_{seuil} = \frac{v_{max}}{a_{max}} + \delta t_p \tag{3.14}$$

où v_{max} et a_{max} sont les vitesse et décélération maximales du système, et δt_p est la période de temps à laquelle est exécuté `Teddy`. Même si ce choix est destiné à assurer la sécurité passive du système, celle-ci ne sera cependant assurée que si le modèle du comportement futur des obstacles considéré est conservatif. Dans le cas où une collision ou la perte de la connectivité est détectée avant le seuil δt_{seuil}, une manoeuvre de freinage est simplement calculée en décélérant au maximum jusqu'à atteindre une vitesse nulle. Une fois le système arrêté, une nouvelle trajectoire vers le but peut alors être replanifiée.

3.4 Résultats en simulation

Les différents systèmes étudiés dans le cadre de nos travaux de déformation de trajectoire sont présentés ici. Tout d'abord nous nous attardons en Section 3.4.2 sur le cas d'une simple masse ponctuelle afin d'illustrer les différents comportements possible proposés par l'approche de déformation de trajectoire `Teddy`. Ensuite, nous montrons en Sections 3.4.3 et 3.4.4 que `Teddy` gère aisément des systèmes plus complexes tels un robot différentiel et un véhicule de type voiture.

Des résultats expérimentaux sur une chaise roulante automatisée (robot différentiel) sont présentés chapitre 5 afin de compléter ces résultats.

3.4.1 Spécifications de l'implémentation

Avant de présenter les différents cas d'étude, voici quelques spécifications de l'implémentation : `Teddy` a été développé en C++ et testé sur une sta-

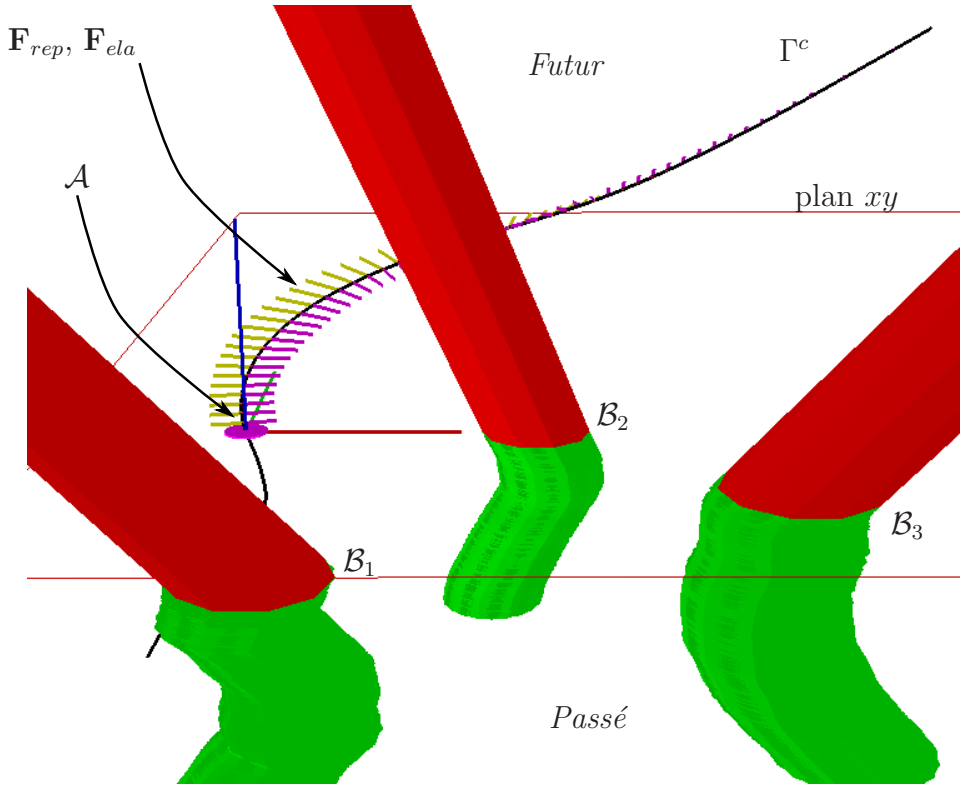


FIGURE 3.12 – Illustration du principe général de déformation de trajectoire sur un scénario dans lequel le système $\mathcal{A}$ évolue dans un environnement obstrué de trois obstacles mobiles circulaires $\mathcal{B}_i$, i=1-3. La dimension temporelle est représentée sur l'axe vertical, le passé se trouvant en dessous du plan xy (correspondant au présent), et le futur au dessus. Les obstacles se déplacent ici de manière aléatoire mais le modèle prévisionnel de leur comportement suppose qu'ils conserveront leur direction et leur vitesse dans le futur. Les forces externes et élastiques sont représentées par des vecteurs sur chaque noeud de la trajectoire.

tion de travail équipée d'un Core i7 (3.4 GHz, 6 GB RAM, SE : Linux). Ces premiers résultats de simulation ont été effectués sur un environnement simulé fournissant au système à chaque étape de déformation la position du robot contrôlé et les positions et vitesses exactes des obstacles mobiles circulaires qui l'entourent. Un modèle complet des obstacles statiques est également fourni au préalable afin de permettre au système de planifier sa trajectoire initiale. A partir des positions et vitesses de chaque obstacle mobile, un modèle prévisionnel du mouvement des obstacles est construit en supposant que chacun d'entre eux continuera son mouvement dans la même

direction et à la même vitesse. La Figure 3.12 illustre le simulateur utilisé et la manière dont `Teddy` opère pour déformer sa trajectoire.

3.4.2 Cas d'étude 1 : masse ponctuelle

Le système

Pour commencer, `Teddy` a été appliqué au cas simple d'un robot $\mathcal{A}$ naviguant sur une surface plane 2D et caractérisé par une dynamique de type double intégrateur (masse ponctuelle). Un état de $\mathcal{A}$ est défini par (p, v) désignant respectivement sa position et sa vitesse 2D : $p = (x, y)$ et $v = (v_x, v_y)$. La dynamique de $\mathcal{A}$ est donnée par le système suivant :

$$\begin{pmatrix} \dot{p} \\ \dot{v} \end{pmatrix} = \begin{pmatrix} v \\ a \end{pmatrix} \tag{3.15}$$

où a représente le contrôle en accélération appliqué en entrée du système $\mathcal{A}$:

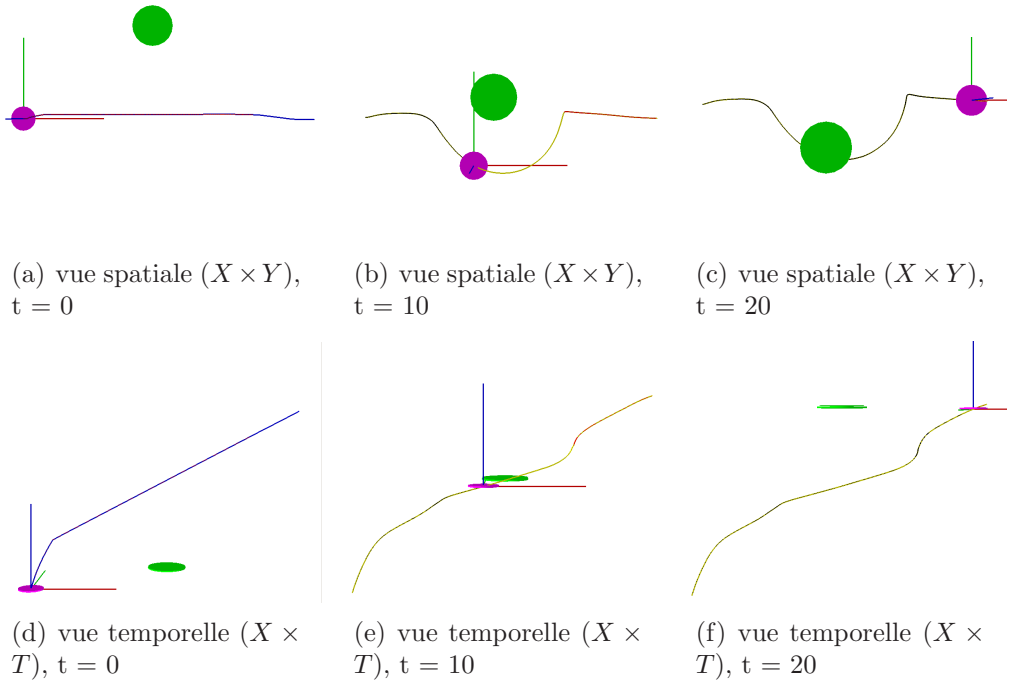
$$|a| \leq a_{\max} \text{ et } |v| \leq v_{\max} \tag{3.16}$$

Scénario 1 : cisaillement

Afin de souligner l'intérêt de la déformation de trajectoire par rapport à la déformation de chemin, un scénario de "cisaillement" a été choisi en premier lieu, celui-ci étant problématique pour la déformation de chemin (*cf.* Section 3.2.6).

`Teddy` repose sur un certain nombre de paramètres pour fonctionner correctement : les gains des forces externes k_{rep} et élastiques k_{ela}, mais également la fonction de distance d_{wt} nécessaire à la détermination de l'obstacle le plus proche dans $\mathbf{W} \times \mathbf{T}$. Les deux exemples ci-dessous ont été sélectionnés afin de montrer l'impact de cette fonction de distance sur les différents comportements de `Teddy` qu'elle peut engendrer. Dans chaque exemple, la trajectoire initiale a une durée de 20s et est échantillonnée par 320 noeuds.

Pour ce même scénario, `Teddy` réagit de manières complètement différentes en fonction du choix des poids spatial w_s et temporel w_t réglant la distance d_{wt} : Dans le premier exemple (Fig. 3.13), davantage de poids est donné à la dimension spatiale. Le système choisit donc de passer devant l'obstacle qui va croiser son chemin. Mais pour cela son chemin est déformé de manière à garder une distance de sécurité suffisante par rapport à l'obstacle. Dans le second exemple (Fig. 3.14), davantage de poids est donné à la déformation temporelle. Le chemin suivi est alors à peine modifié alors que le temps de passage à chaque point l'est fortement, et par conséquent le



(a) vue spatiale ($X \times Y$), t = 0

(b) vue spatiale ($X \times Y$), t = 10

(c) vue spatiale ($X \times Y$), t = 20

(d) vue temporelle ($X \times T$), t = 0

(e) vue temporelle ($X \times T$), t = 10

(f) vue temporelle ($X \times T$), t = 20

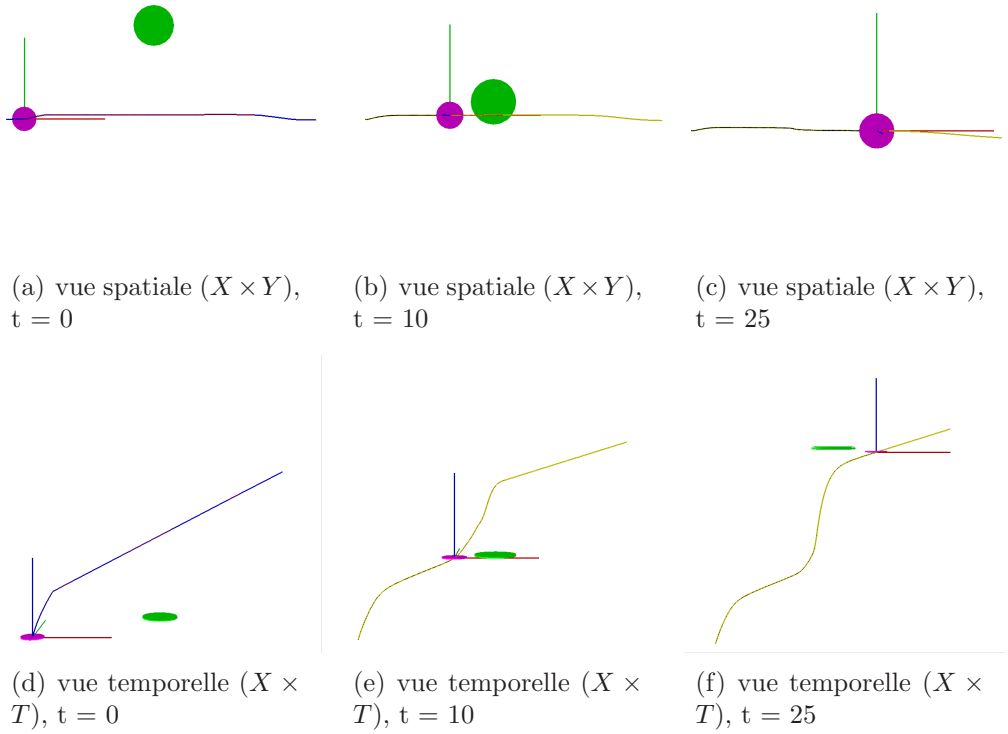
FIGURE 3.13 – Système double intégrateur, scénario de cisaillement (déformation spatiale) : $\mathcal{A}$ se déplace de gauche à droite, l'obstacle se déplace vers le bas. Les figures du haut représentent le chemin déformé à différents instants (vue $x \times y$) tandis que ceux du bas représentent le profil de vitesse de $\mathcal{A}$ à ces mêmes moments (vue $x \times t$).

profil de vitesse du robot l'est également. Le système choisit alors de laisser passer l'obstacle en ralentissant en premier lieu, puis en accélérant en fin de course afin de rattraper le retard perdu.

Ces deux exemples montrent la forte influence du choix des paramètres régissant le comportement de `Teddy`. Ils illustrent également le principal avantage de la déformation de trajectoire par rapport à la déformation de chemin : Le système est capable ici de modifier non seulement la courbe géométrique qu'il va suivre, mais également son profil de vitesse permettant ainsi d'accélérer pour passer rapidement devant un obstacle ou au contraire de décélérer pour le laisser passer.

Scénario 2 : multiples obstacles statiques et mobiles

Dans ce second scénario illustré en Fig. 3.15, le système double intégrateur évolue au milieu d'une dizaine d'obstacles statiques rectangulaires et d'une quarantaine d'obstacles mobiles circulaires (la vidéo complète est disponible



(a) vue spatiale ($X \times Y$), t = 0

(b) vue spatiale ($X \times Y$), t = 10

(c) vue spatiale ($X \times Y$), t = 25

(d) vue temporelle ($X \times T$), t = 0

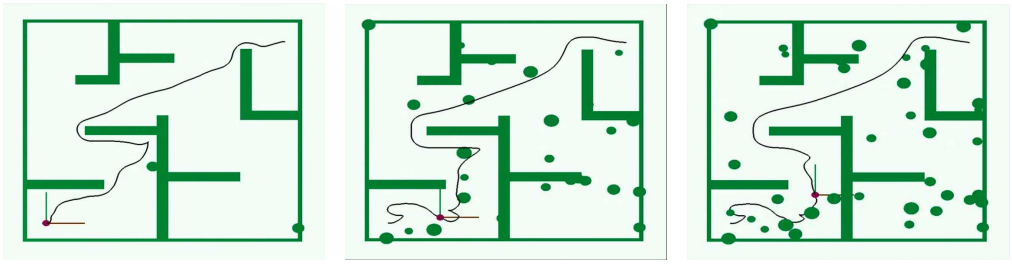
(e) vue temporelle ($X \times T$), t = 10

(f) vue temporelle ($X \times T$), t = 25

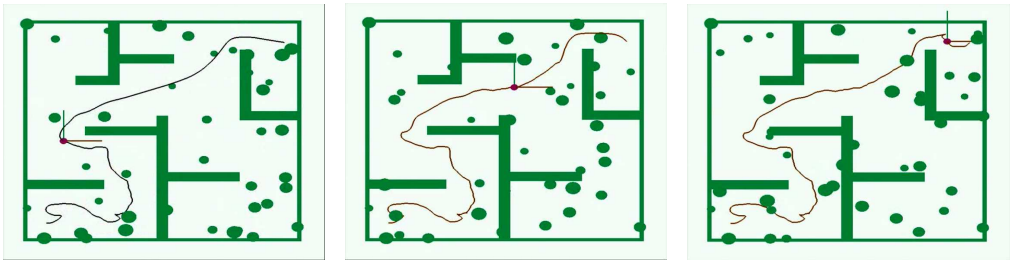
FIGURE 3.14 – Système double intégrateur, scénario de cisaillement (déformation temporelle) : $\mathcal{A}$ se déplace de gauche à droite, l'obstacle se déplace vers le bas. Les figures du haut représentent le chemin déformé à différents instants (vue $x \times y$) tandis que ceux du bas représentent le profil de vitesse de $\mathcal{A}$ à ces mêmes moments (vue $x \times t$).

à l'adresse suivante : http ://emotion.inrialpes.fr/fraichard/films/teddy-pointmass.mpg). Les tests ont été procédés comme suit : A partir d'un état initial fixé, une trajectoire vers un état but libre choisi aléatoirement est calculée. Comme précédemment, à chaque étape de déformation, l'algorithme récupère l'état courant du robot, la trajectoire suivie, ainsi que la position et la vitesse instantanée courante des obstacles mobiles afin d'en déduire un modèle prévisionnel en ligne droite. Néanmoins ces derniers peuvent à chaque instant modifier à la fois leur direction et leur vitesse. Une fois l'état but de la trajectoire atteint, un nouvel état but est choisi aléatoirement et le processus de déformation est répété sur la nouvelle trajectoire calculée vers ce dernier.

En terme de performances, les deux parties les plus coûteuses de l'algorithme sont d'une part la génération de trajectoire par `Tiji` utilisée pour maintenir la connectivité de la trajectoire, et d'autre part la détermination de



(a) $t = 0$ (b) $t = 5$ (c) $t = 10$



(d) $t = 20$ (e) $t = 25$ (f) $t = 30$

FIGURE 3.15 – Système double intégrateur évoluant au milieu d'obstacles statiques et mobiles : Les différentes figures représentent la trajectoire déformée à différents pas de temps (vue $x \times y$).

Echantillonnage de la traj. (en sec.)	Temps de déformation moyen (en ms)	Nombre de collisions	Nombre de pertes de connectivité
0.15	40.76	0	0
0.15	42.56	1	0
0.30	22.32	0	0
0.30	24.18	0	0
0.50	15.26	1	0
0.50	16.99	1	0

TABLE 3.1 – Résumé des performances du module de déformation de trajectoire `Teddy` pour le système de type masse ponctuelle.

l'obstacle le plus proche à partir de chaque noeud de la partie de la trajectoire à déformer. La vitesse d'exécution de `Teddy` est donc fortement dépendante de l'échantillonnage de la trajectoire. La table 3.1 résume les performances de `Teddy` en fonction de cet échantillonnage. Chaque jeu de test résumé dans celle-ci dure cinq minutes.

La trajectoire déformée est échantillonnée à intervalles de temps réguliers.

Le temps caractérisant l'échantillonnage est donc le temps approximatif séparant deux noeuds consécutifs de la trajectoire. Pour un système de type masse ponctuelle, les temps de calcul de la déformation sont très largement suffisant pour être exécutés en temps réel. Une collision peut néanmoins arriver de temps à autre : les obstacles pouvant être hostiles et la méthode de déformation étant heuristique, quelques cas restent problématiques. Par exemple, si un obstacle change de direction au dernier moment alors que le robot est à proximité, ou si le robot a entrepris de passer entre deux obstacles et que ceux-ci se rapprochent plus rapidement que prévu, il ne sera pas toujours possible de les éviter. Néanmoins en pratique, ces cas arrivent assez rarement, d'où les bonnes performances de l'algorithme. D'autre part, en cas de collisions prévues dans un futur proche (*cf.* Section 3.3.8), une manoeuvre de décélération est calculée afin d'assurer la sécurité passive du système. Cependant, l'approche de déformation étant dépendante du modèle prévisionnel du comportement futur des obstacles, même cette sécurité passive n'est jamais véritablement assurée.

3.4.3 Cas d'étude 2 : robot différentiel

Le système

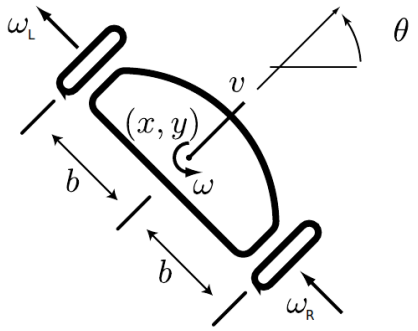


FIGURE 3.16 – Le robot différentiel $\mathcal{A}_d$.

Un état d'un système différentiel planaire $\mathcal{A}_d$ (Fig. 3.16) est défini par un 5-ulpet $s = (x, y, \theta, \omega_L, \omega_R)$ où (x, y) représentent les coordonnées du centre des roues, θ est la direction principale de $\mathcal{A}_d$ et ω_L (resp. ω_R) est la vitesse angulaire de la roue gauche (resp. droite). Un contrôle de $\mathcal{A}_d$ est défini par le couple $u = (\eta_L, \eta_R)$ où η_L (resp. η_R) est l'accélération angulaire de la roue gauche (resp. droite).

Connaissant les vitesses des roues, les vitesses linéaire et angulaire principales de $\mathcal{A}_d$ sont données par :

$$v(t) = \frac{\mathbf{r} * (\omega_R(t) + \omega_L(t))}{2} \tag{3.17}$$

$$\omega(t) = \frac{\mathbf{r} * (\omega_R(t) - \omega_L(t))}{2 * \mathbf{b}} \tag{3.18}$$

où $\mathbf{b}$ est la distance entre le centre du robot différentiel et les roues, et $\mathbf{r}$ est le rayon des roues (*cf.* Fig. 3.16).

Le mouvement de $\mathcal{A}_d$ est alors régi par les équations différentielles suivantes :

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{\omega}_L \\ \dot{\omega}_R \end{pmatrix} = \begin{pmatrix} v\cos(\theta) \\ v\sin(\theta) \\ \omega \\ \eta_L \\ \eta_R \end{pmatrix} \tag{3.19}$$

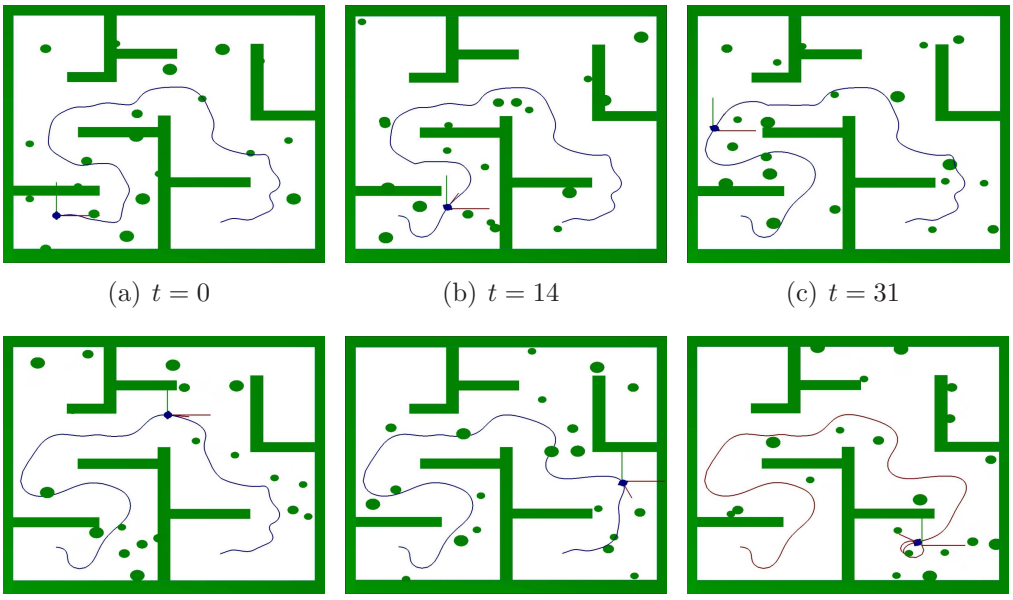
Nous considérons les contraintes suivantes sur le mouvement du robot différentiel :

$$(\omega_L, \omega_R) \in [-\omega_{max}; \omega_{max}]^2, (\eta_L, \eta_R) \in [-\eta_{max}; \eta_{max}]^2 \tag{3.20}$$

Scénario : multiples obstacles statiques et mobiles

Le robot différentiel $\mathcal{A}_d$ a été évalué dans un environnement obstrué de nombreux obstacles statiques et mobiles (*cf.* Fig. 3.17) similaire à celui présenté en Section 3.4.2. Une vidéo complète de ce scénario est disponible à l'adresse suivante : http ://emotion.inrialpes.fr/fraichard/films/teddy-diffdrive.mpg. Les performances pour le robot différentiel sont résumées en table 3.2. Comme dans le cas précédent, chaque jeu de test a une durée de cinq minutes.

Malgré des contraintes cinématiques et dynamiques bien plus complexes que le système double intégrateur, $\mathcal{A}_d$ évolue aisément dans cet environnement grâce au modèle prévisionnel du mouvement des obstacles qu'il construit. Toutefois, lorsque l'échantillonnage de la trajectoire est un peu trop important (0.15 secondes entre deux états-temps successifs), l'algorithme de déformation est un peu trop lent (moins de dix déformations par secondes) et le nombre de collisions augmente donc légèrement. Lorsque l'échantillonnage est un peu trop faible (0.80 secondes), le nombre de collision risque également d'augmenter. En pratique un échantillonnage tous les 0.3-0.5 secondes s'est avéré convainquant. Nous pouvons également observer parmi ces tests une



(a) $t = 0$ (b) $t = 14$ (c) $t = 31$

(d) $t = 53$ (e) $t = 72$ (f) $t = 88$

FIGURE 3.17 – Robot différentiel évoluant au milieu d'obstacles statiques et mobiles : Les différentes figures représentent la trajectoire déformée à différents pas de temps (vue $x \times y$).

Echantillonnage de la traj. (en sec.)	Temps de déformation moyen (en ms)	Nombre de collisions	Nombre de pertes de connectivité
0.15	143.73	2	0
0.15	133.08	1	0
0.30	58.25	0	0
0.30	55.94	0	1
0.50	32.21	0	0
0.50	36.44	1	0
0.80	28.80	1	0
0.80	26.52	1	0

TABLE 3.2 – Résumé des performances du module de déformation de trajectoire `Teddy` pour le système de type robot différentiel.

perte de la connectivité de la trajectoire lors d'un test. Celui-ci s'est produit alors que le robot avait quasiment atteint son but ; un obstacle s'étant approché au dernier moment, le robot a été contraint d'accélérer et n'a pas pu trouver de trajectoire sur le moment permettant de rejoindre son but.

Néanmoins en pratique, le robot différentiel pouvant tourner sur lui-même, il est vraiment rare que la connectivité de la trajectoire soit perdue.

3.4.4 Cas d'étude 3 : robot de type voiture

Le système

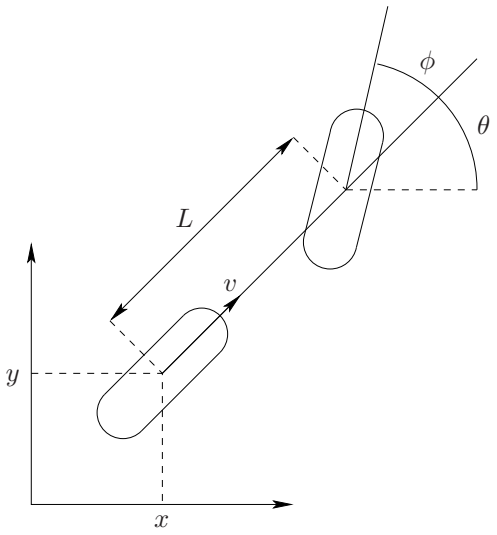


FIGURE 3.18 – Le véhicule de type voiture $\mathcal{A}_c$.

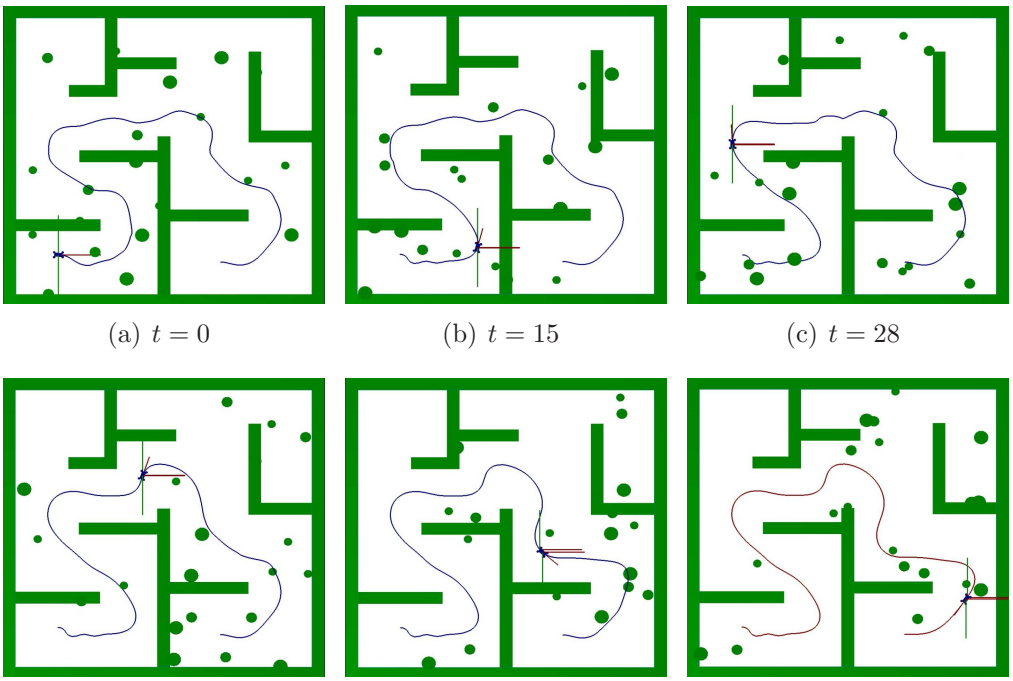
Un état d'un système de type voiture $\mathcal{A}_c$ est défini par un 5-uplet $s = (x, y, \theta, \phi, v)$ où (x, y) sont les coordonnés de la roue arrière du véhicule (ou du centre des roues arrières), θ est son orientation principale, ϕ est l'orientation de ses roues avant (angle de braquage), et v est la vitesse linéaire de sa roue arrière. Un contrôle de $\mathcal{A}_c$ est défini par le couple $u = (a, \zeta)$ où a est l'accélération linéaire de sa roue arrière, et ζ la vitesse de braquage. Le mouvement de $\mathcal{A}_c$ est régi par les équations différentielles suivantes :

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{\phi} \\ \dot{v} \end{pmatrix} = \begin{pmatrix} v\cos(\theta) \\ v\sin(\theta) \\ v\tan(\phi)/L \\ \zeta \\ a \end{pmatrix} \tag{3.21}$$

où L est la distance entre la roue avant et la roue arrière du système $\mathcal{A}_c$. Les contraintes additionnelles suivantes sont considérées :

$$v \in [0, v_{\max}], |\phi| \leq \phi_{\max}, |a| \leq a_{\max} \text{ et } |\zeta| \leq \zeta_{\max} \tag{3.22}$$

Scénario : multiples obstacles statiques et mobiles



(a) $t = 0$ (b) $t = 15$ (c) $t = 28$

(d) $t = 41$ (e) $t = 53$ (f) $t = 61$

FIGURE 3.19 – Robot de type voiture évoluant au milieu d'obstacles statiques et mobiles : Les différentes figures représentent la trajectoire déformée à différents pas de temps (vue $x \times y$).

Le robot de type voiture $\mathcal{A}_c$ a été testé dans le même environnement que précédemment (Fig. 3.19). Une vidéo complète de ce scénario est disponible à l'adresse suivante : http ://emotion.inrialpes.fr/fraichard/films/teddy-carlike.mpg. Les performances pour ce système sont résumées en table 3.3.

Malgré les contraintes non-holonomes qui caractérisent un tel système, il est également capable de se frayer un chemin sans collisions jusqu'à son but. Le générateur de trajectoire `Tiji` permet une nouvelle fois de conserver les contraintes du système satisfaites tout au long de la trajectoire et malgré les multiples déformations. Cependant, la connectivité de la trajectoire est souvent perdue : en effet, lorsque le système arrive proche de son but, il arrive fréquemment que ses contraintes non-holonomes l'empêchent d'atteindre la position finale avec l'orientation but initialement prévue, suite aux déformations successives. Si l'orientation finale du robot est libre, on arrive à éviter la plupart de ces problèmes de perte de connectivité. Mais ce cas montre néanmoins l'une des limitations de l'approche de déformation : pour

Echantillonnage de la traj. (en sec.)	Temps de déformation moyen (en ms)	Nombre de collisions	Nombre de pertes de connectivité
0.15	162.34	2	4
0.15	153.77	3	3
0.30	64.91	2	4
0.30	60.01	0	5
0.50	35.56	1	6
0.50	40.65	1	3
0.80	29.43	1	2
0.80	32.40	3	4

TABLE 3.3 – Résumé des performances du module de déformation de trajectoire `Teddy` pour le système de type voiture.

des systèmes disposant de contraintes non-holonomes, `Teddy` peut mener le robot à proximité de son but en évitant les obstacles statiques et mobiles qu'il croisera, mais une manoeuvre supplémentaire sera souvent nécessaire pour aider le robot à s'arrêter à un emplacement prédéterminé.

3.5 Conclusion et perspectives

La *déformation de chemin* est une méthode de navigation modifiant itérativement au cours du temps un chemin planifié à priori en fonction de l'évolution de son environnement. Cette méthode a l'avantage d'être capable d'éviter les obstacles au milieu desquels le robot navigue, tout en s'assurant de la convergence vers le but tant que le chemin est connecté. Les méthodes de déformation de mouvement proposées jusqu'alors disposaient néanmoins de deux inconvénients majeurs : tout d'abord, les premières méthodes de déformations étaient limitées à des homotopies du chemin initial. Il était donc impossible pour le système robotique de laisser un obstacle couper le chemin qu'il suivait, ou d'adapter sa vitesse par rapport à l'évolution de l'environnement. Ensuite, quelques méthodes de déformations permettant de modifier une trajectoire (*i.e.* un chemin paramétré par le temps) sont apparues. Néanmoins ces dernières ne raisonnaient que sur la position des obstacles observés à chaque instant sans prendre en compte leur mouvement.

Nous avons alors développé une nouvelle approche de *déformation de trajectoire* destinée à palier ces inconvénients pour divers systèmes disposant de dynamiques complexes. Celle-ci consiste à augmenter le schéma de déformation par la dimension temporelle et de déformer la trajectoire dans

l'espace d'états-temps $\mathbf{S} \times \mathbf{T}$ en *raisonnant sur le futur* de l'environnement. Pour ce faire, un modèle prévisionnel du mouvement futur des obstacles est pris en compte. Des forces répulsives sont alors exercées sur la trajectoire suivie par le système à partir de ce modèle prévisionnel résultant ainsi en des modifications de la trajectoire à la fois *spatiales* et *temporelles*. Des forces élastiques permettent de limiter les oscillations de la trajectoire et de réduire la longueur du chemin parcouru lorsque plus aucun obstacle n'est à proximité. Enfin un processus de maintien de la connectivité de la trajectoire est appliqué afin de garantir sa faisabilité ainsi que sa convergence vers le but.

Notre approche est alors capable, à la différence des méthodes précédentes, de déformer librement dans $\mathbf{S} \times \mathbf{T}$ la trajectoire d'états de systèmes robotiques complexes tout en garantissant sa faisabilité. Tout en conservant les atouts de la déformation de chemin, l'approche permet ainsi d'anticiper le mouvement des obstacles dynamiques qui entourent le robot, diminuant par conséquence les risques de collisions.

Chapitre 4

Génération de Trajectoires avec Contrainte sur le Temps Final

4.1 Introduction

4.1.1 Motivations

La génération de trajectoires pour un système robotique donné consiste en la détermination d'une trajectoire faisable (*i.e.* qui respecte la dynamique du système) entre un état initial et un état final. Les méthodes de génération de trajectoires existantes diffèrent quant au type de trajectoires calculées, à la manière dont elles sont déterminées et enfin aux critères optimisés le long de ces trajectoires. Il est cependant intéressant de noter que la génération de trajectoire *entre deux états-temps*, *i.e.* permettant d'atteindre l'état final à un temps fixe donné, a très peu été étudiée (*cf.* Section 4.2). Néanmoins lorsqu'un système robotique est placé dans un environnement dynamique (comportant des obstacles mobiles), il devient important de calculer une trajectoire permettant non seulement d'atteindre un état but mais également d'atteindre cet état à un temps fixe donné ou durant un intervalle de temps garantissant que le système ne sera pas en collision avec les obstacles mobiles à cet instant. Cette contrainte, que nous nommerons *contrainte sur le temps final* a rarement été considérée jusqu'alors. Pourtant, elle semble inhérente à tous problèmes de navigation en environnement dynamique.

La navigation en environnement dynamique soulève également une seconde contrainte : La prise de décision du mouvement à suivre à chaque instant doit s'effectuer suffisamment rapidement pour que le système ait le temps de s'adapter à l'évolution de l'environnement dans lequel il navigue (le système robotique peut être en danger par le seul fait de rester passif). Cette contrainte est nommée *contrainte sur le temps de décision*. Elle est déterminée

par la nature, la vitesse et la proximité des objets mobiles : plus le temps nécessaire à un obstacle pour entrer en collision avec le système robotique est réduit, plus le robot doit être capable de calculer une nouvelle trajectoire à suivre rapidement afin d'anticiper son mouvement.

Alors que la génération de trajectoire a été déjà fortement étudiée dans la littérature, ce problème reste complexe à résoudre spécialement pour des systèmes sujets à des contraintes différentielles. La considération de la *contrainte sur le temps final* accroît la complexité de la génération de trajectoire, le temps devenant une dimension supplémentaire de ce problème. Une difficulté majeure se pose alors : étant données les contraintes cinématiques et dynamiques du système, il n'y a aucune garantie qu'une trajectoire existe entre deux états donnés à des temps fixes ou contraints. Cette question relève de la *décidabilité* du problème. Afin d'y répondre, il est nécessaire de déterminer l'espace atteignable (dans l'espace d'états-temps) à un temps donné à partir de l'état initial, et de vérifier si l'état but appartient à cet espace. Pour des systèmes réalistes, une telle analyse reste malheureusement trop coûteuse pour des applications en temps réel. Dans cette situation, une alternative possible consiste à essayer de déterminer une trajectoire faisable sans répondre au préalable à la décidabilité du problème. Dans le cas où une trajectoire valide est trouvée, son existence est prouvée ; dans le cas contraire, on ne peut que supposer qu'il n'en existe pas sans en apporter la preuve. A moins de ne disposer d'une solution alternative, le processus de navigation risque donc d'être interrompu.

4.1.2 Contributions

Nous présentons dans ce chapitre un générateur de trajectoire nommé `Tiji` intégrant la *contrainte sur le temps final*. `Tiji` est conçu pour déterminer des trajectoires pour des systèmes disposant de dynamiques complexes sujets à des contraintes différentielles (*e.g.* véhicules de type voiture). Son efficacité lui permet d'être utilisé en temps réel (respectant ainsi la *contrainte sur le temps de décision*). L'approche est similaire sur le principe à des travaux tels [GG00] ou [KN03] : une représentation paramétrique est supposée afin de réduire la dimension de l'espace de recherche. Un ensemble de paramètres initiaux déterminant une trajectoire n'atteignant pas forcément le but est calculé. Une recherche sur l'espace des paramètres est alors effectuée par optimisation numérique : les paramètres initiaux sont modifiés itérativement dans le but de faire converger l'état-temps final de la trajectoire paramétrée vers l'état-temps but. Dans le cas où l'état but n'est pas atteignable dans l'intervalle de temps fixé (si le temps final est mal choisi), `Tiji` retourne une trajectoire faisable s'arrêtant "aussi près que

possible" de celui-ci, évitant ainsi l'échec du processus de navigation. `Tiji` diffère donc des approches précédentes par sa prise en compte de la *contrainte sur le temps final* et par le calcul de ces *trajectoires approchées* garantissant dans tous les cas que les contraintes cinématiques et dynamiques du système seront respectées. L'approche de déformation de trajectoire présentée au Chapitre 3 requiert une telle méthode de génération de trajectoires. C'est pour combler ce manque que `Tiji` a été conçu, néanmoins une telle approche peut s'avérer fortement utile en étant intégrée dans d'autres schémas de navigation en environnement dynamique. Nous rappelons enfin que `Tiji` n'est qu'un générateur de trajectoires et ne choisit pas lui-même la contrainte sur le temps final. Celle-ci doit être déterminée par la méthode de navigation qui l'intègre.

4.1.3 Plan du chapitre

Ce chapitre est organisé comme suit : tout d'abord un état de l'art des méthodes de génération de mouvement est présenté en Section 4.2. Le problème de génération de trajectoires et sa résolution générale par méthode variationnelle sont respectivement décrites en Sections 4.3 et 4.4. L'approche de génération de trajectoires `Tiji` est alors présentée en Section 4.5 et illustrée pour un robot différentiel en Section 4.6. Des résultats en simulation ainsi que l'adaptation à d'autres systèmes sont présentés en Sections 4.7 et 4.8. Une discussion de l'approche et une conclusion sur les travaux réalisés sont enfin fournies en Section 4.9.

4.2 Génération de mouvement : Approches existantes

Des travaux de Dubins ([Dub57]) jusqu'aux travaux récemment développés par l'université de Carnegie Mellon lors du *Darpa Urban Challenge* ([HK07]), de nombreuses approches de génération de mouvement ont vu le jour durant ces cinquante dernières années. Celles-ci peuvent être globalement classées en trois types d'approches : la combinaison de primitives simples, la résolution d'un problème aux limites et enfin la recherche d'un contrôle optimal. Nous décrivons ci-dessous les principaux travaux réalisés pour chacune d'entre elles.

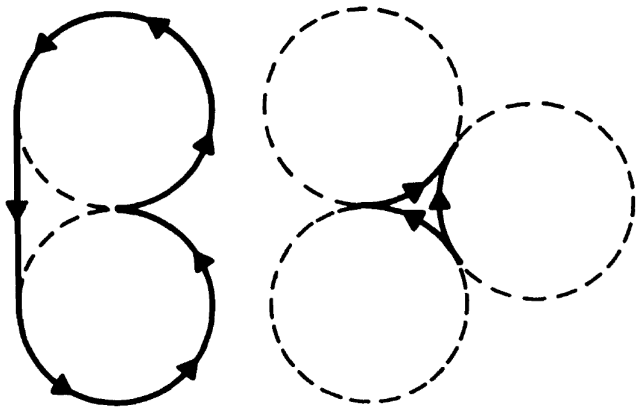


FIGURE 4.1 – Deux exemples de chemins optimaux pour un véhicule de type voiture pouvant se déplacer en avant et en arrière composés uniquement de lignes droites et d'arcs de cercle (source [RS90]).

4.2.1 Combinaison de primitives géométriques

La génération de mouvement par combinaison de primitives consiste à déterminer un chemin par la concaténation de primitives géométriques simples respectant les contraintes cinématiques du système considérées. Les tous premiers travaux de ce type ont été développés en 1957 par Dubins [Dub57] qui cherchait à trouver des chemins optimaux entre deux positions et tangentes fixées lorsque la courbure et bornée. Pour ce faire, il montra que ces chemins optimaux sont composés par la concaténation d'au plus trois primitives de type lignes droites et arcs de cercle. La communauté roboticienne s'étant depuis fortement intéressée à la navigation de véhicules de type voiture disposant de telles contraintes de courbure (contraintes non holonomes dues à l'angle de braquage maximal caractérisant ce type de véhicules), les travaux de Dubins ont été réutilisés ou étendus à maintes reprises. Reeds & Shepp ([RS90]) ont entre autres proposé la caractérisation de chemins optimaux pour une voiture pouvant se déplacer en avant et en arrière. Ils ont donc montré qu'il existait 48 combinaisons possibles de lignes droites et d'arcs de cercle pour trouver les chemins optimaux dans ce cas.

Les deux approches citées ci-dessus possèdent néanmoins un inconvénient majeur : des chemins composés uniquement de lignes droites et d'arcs de cercle disposent de discontinuités de la courbure entre les différentes primitives obligeant le robot à s'arrêter entre chacune d'elles. L'utilisation d'arcs de clothoïdes a donc été proposée afin de palier ce problème ([KM85]). Les clothoïdes sont des courbes de l'espace affine de dimension 2 dont la courbure

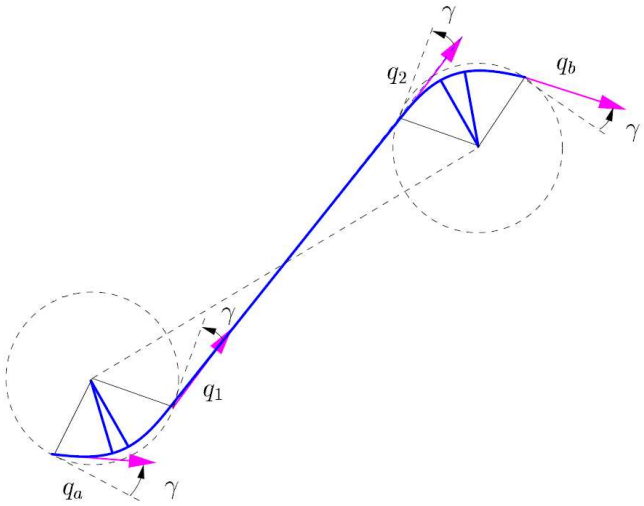


FIGURE 4.2 – Chemin à courbure continue calculé entre deux configurations q_a et q_b. Le chemin résultant est constitué d'une ligne droite, quatre arcs de clothoïde et deux arcs de cercle (source [Sch98]).

varie linéairement en fonction de l'abscisse curviligne.

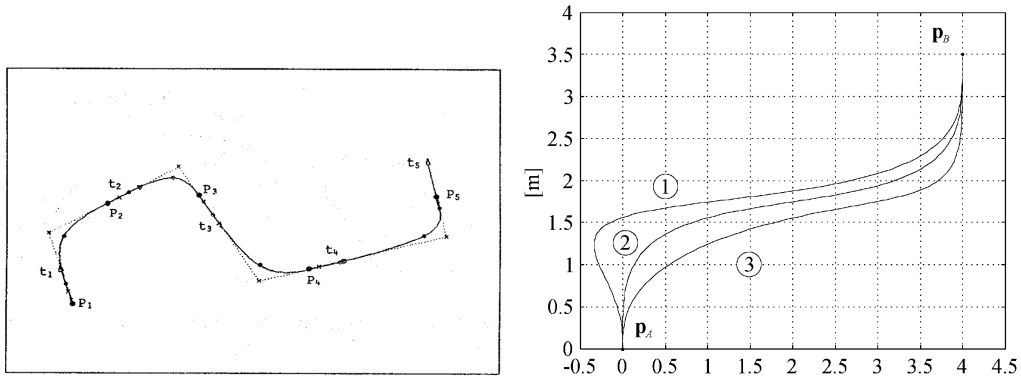
Ces travaux ont été étendus par [FS04] en constituant des chemins sous-optimaux à courbure continue : En insérant des arcs de clothoïdes entre chaque couples de ligne droite et d'arc de cercle ou de deux arcs de cercle consécutifs, il est ainsi possible d'obtenir des chemins dont la courbure est continue entre n'importe quel couple de configurations données (*cf.* Fig. 4.2). L'un des reproches souvent fait à l'encontre de l'utilisation de clothoïdes est son manque d'expression de forme close. En effet les coordonnées du chemin défini par de telles primitives sont données par les intégrales de Fresnel ([Sch98]) qui ne sont pas calculable analytiquement.

Une approche alternative pour générer des trajectoires à courbure continue a été proposée par Nelson [Nel89]. Celui-ci utilise des courbes polaires polynomiales pour décrire des virages symétriques entre deux configurations à courbure nulle et des polynômes cartésiens pour décrire des manoeuvres de changement de ligne *i.e.* des chemins reliant deux lignes parallèles dont l'orientation est identique. Mais si la concaténation de ces deux types de chemins dispose d'une expression de forme close, aucunes bornes sur la courbure ou la dérivée de courbure ne sont ici prises en compte.

Balkom & Mason ont enfin cherché de leur côté à déterminer les trajectoires optimales pour un robot différentiel dont la vitesse angulaire des roues (mais pas son accélération) est bornée [BM00b, BM00a]. Ils ont alors montré grâce au principe du maximum de Pontryagin [Kir70] que celles-ci sont com-

posées d'au plus cinq manoeuvres de types translation en ligne droite ou rotation autour de l'axe du robot. En fonction de la position relative entre les configurations initiale et finale, il est possible de déduire la séquence de manoeuvres à adopter.

4.2.2 Résolution d'un problème aux limites



(a) B-spline (source [KT89]) (b) η^3-splines (source [PBR07])

FIGURE 4.3 – Différents types de splines utilisés pour déterminer un chemin entre deux configurations.

La seconde catégorie d'approches consiste en la résolution d'un problème aux limites. Ces méthodes consistent à essayer de déterminer une seule et unique courbe paramétrique de haut degré liant deux configurations données. La difficulté de ces approches est alors de garantir le respect des contraintes sur le mouvement le long du chemin géométrique défini par la courbe.

Tout d'abord, différents types de splines (fonctions définies par morceaux par des polynômes) ont été proposés (*cf.* Fig. 4.3). [KT89] utilisent des B-splines d'ordre trois, courbes approximantes générées à partir d'un nombre fini de points de contrôle. Une B-Spline d'ordre 3 est comprise dans l'enveloppe convexe définie par ses points de contrôle, sa courbure est continue, de plus en ajustant les points de contrôles successifs, il est possible de définir des points de passages de cette courbe et de fixer leurs tangentes. Leur courbure n'est cependant pas bornée. Pour résoudre ce problème, Bianco et al. définirent des η^3-splines ([PBR07, BG09]), pour planifier une trajectoire entre deux états-temps dont les vitesses et accélérations sont bornées. Ces splines, dépendant de sept paramètres possèdent une courbure et une dérivée de courbure continues. Une détermination analytique de ces paramètres est effectuée afin de joindre les états-temps initial et final puis un processus d'optimisation

de ces paramètres fut proposé afin de respecter les contraintes de courbure et de vitesse le long de la trajectoire, néanmoins sans garantie de convergence. Delingette et al. ([DHI91]) proposèrent une représentation du chemin par spline intrinsèque d'ordre n (dont le profil de courbure est un polynôme d'ordre n) pour générer un chemin admissible entre deux configurations (positions - tangentes) données. Une méthode variationnelle est ici également utilisée afin de satisfaire les contraintes du mouvement : A partir d'un chemin initial en ligne droite liant les positions initiale et finale, des points de contrôle sont définis afin de construire la spline intrinsèque induite sur ce segment. Les points de contrôle définissant cette spline sont alors itérativement modifiés dans le but de satisfaire les tangentes requises aux points initiaux et finaux tout en minimisant la courbure tout au long de cette trajectoire. La méthode est illustrée dans le cas de splines cubiques et requiert un minimum de douze points de contrôle le long du chemin dans le but de converger vers une solution admissible.

En parallèle, [KH89] a proposé une approche basée à partir de spirales cubiques. Une spirale cubique est définie par une courbe dont la tangente est définie par une fonction cubique de l'abscisse curviligne. Ces spirales sont utilisées pour déterminer un chemin entre deux configurations (positions - tangentes) symétriques, néanmoins lorsque les configurations initiale et finale ne respectent pas cette condition, il est montré qu'il est possible de décomposer le chemin en deux parties dont l'une est une spirale cubique et la seconde une ligne droite ou un arc de cercle.

Takahashi et al. [THN89] ont proposé de leur côté une approche basée sur une courbe polynomiale quintique définissant le profil d'une trajectoire entre deux configurations données. Les profils de courbure et de vitesse de cette trajectoire sont déterminés en calculant analytiquement les paramètres du polynôme quintique minimisant une fonction de coût caractérisant la variation de l'accélération (jerk) le long de la trajectoire. Sur le même principe, [GT08] définit les profils $x(t)$ et $y(t)$ d'une trajectoire planifiée par des polynômes d'ordre six. Les quatorze paramètres correspondant sont déterminés analytiquement et minimisent une fonction de coût basée sur la longueur du chemin. En présence d'obstacles mobiles, cette fonction de coût est modifiée de manière à maximiser la distance aux obstacles au cours du temps à partir d'une équation de leur mouvement (*cf.* Fig. 4.4). Cette méthode dispose de l'avantage de déterminer une trajectoire entre deux états-temps fixés, néanmoins elle ne prend pas en compte la moindre contrainte, que ce soit sur la courbure ou sur la vitesse du système le long la trajectoire.

Une génération de chemin par interpolation d'arcs de cercles est proposée en [LL01]. Soient deux configurations $q_1 = (x_1, y_1, \theta_1, \kappa_1)$ et $q_2 =$

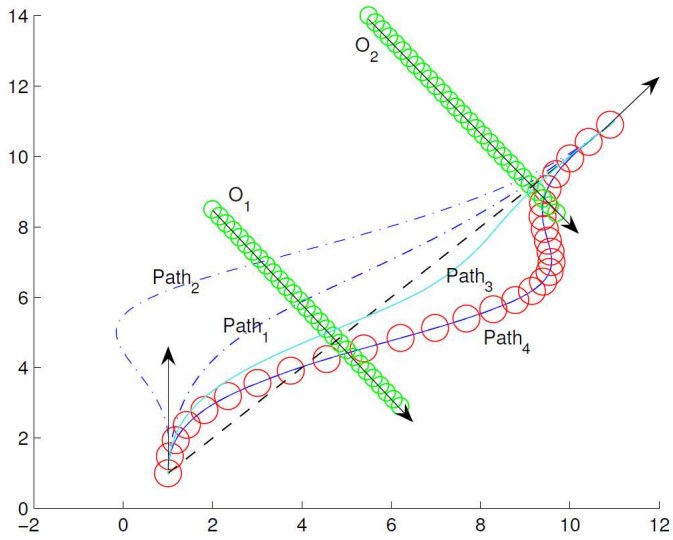


FIGURE 4.4 – Paramétrisation des profils de position $x(t)$ et $y(t)$ d'une trajectoire par des polynômes d'ordre six. Une trajectoire minimisant la longueur du chemin et maximisant la distance aux obstacles le long de celle-ci est alors déterminée en calculant analytiquement les paramètres de ces polynômes. *Path1*, *Path2*, *Path3* et *Path4* représentent quatre chemins candidats déterminés par cette approche en ajustant ses paramètres heuristiques (source [GT08]).

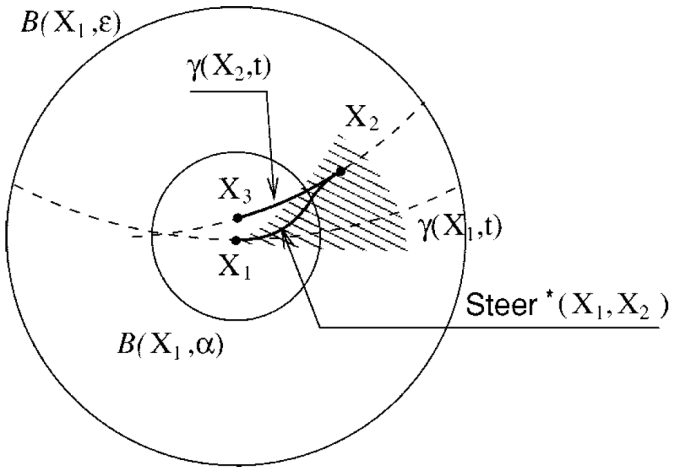


FIGURE 4.5 – Génération de chemin par interpolation des arcs de cercles définis par l'inverse de la courbure aux configurations initiale et finale (source [LL01]).

$(x_2, y_2, \theta_2, \kappa_2)$ définies par la position du système, son orientation et sa courbure. Les arcs de cercles de courbures respectives κ_1 et κ_2 et passant par (x_1, y_1) et (x_2, y_2) sont déterminés, et un chemin est construit en interpolant ces arcs de cercle entre les deux configurations (*cf.* Fig. 4.5). Un tel chemin dispose d'une courbure continue mais pas nécessairement bornée. Dans le cas où les contraintes de courbures soient excédées, une méthode permettant de définir un point de rebroussement intermédiaire est proposée.

Enfin Gallina et Gasparetto ont proposé à leur tour une méthode variationnelle utilisant des sommes d'harmoniques pour représenter la trajectoire générée. Une somme d'harmonique est une série finie dont les différents termes sont des fonctions trigonométriques paramétrées. Ces fonctions sont intéressantes de par leur continuité C^∞. Une fonction de coût caractérisant la longueur du chemin et la satisfaction des contraintes sur les bornes de la trajectoires est alors minimisée. Pour ce faire, une simple méthode de descente de gradient de la fonction de coût est utilisée. Bien que garantissant la convergence vers le but et la continuité du mouvement, ces travaux ne prennent pas en compte les éventuelles contraintes sur la courbure ou la vitesse utilisée le long de ce chemin.

Bien qu'une nouvelle fois assez rapides à calculer grâce à des méthodes analytiques ou variationnelles de détermination du chemin ou de la trajectoire, les méthodes de recherche d'une courbe de haut degré entre deux états ou configurations ont en général de fortes difficultés à respecter les contraintes internes sur le mouvement du système considéré (telle un borne sur la courbure ou la vitesse).

4.2.3 Recherche d'un contrôle optimal

Une dernière catégorie d'approches de génération de mouvement est fondée sur la recherche d'un contrôle optimal à appliquer en entrée d'un système robotique donné pour lier deux configurations ([BH75]). Bien que très difficile à déterminer en général pour un système complexe, une approximation de ce contrôle optimal peut être calculée en tirant partie de propriétés intéressantes du système de contrôle du robot.

Tout d'abord, les systèmes de contrôle *nilpotents* ont été étudiés ([LS93]) : A partir de la cinématique du robot considéré, un système cinématique étendu est défini de manière à ce qu'un chemin entre deux configurations données soit aisément déterminé dans ce dernier. Dans le cas d'un système nilpotent, les composantes supplémentaires du contrôle du système cinématique étendu peuvent s'exprimer par combinaison linéaire des composantes du contrôle initial. Une fois le contrôle trouvé dans le système étendu, un contrôle du robot considéré définissant un chemin admissible

entre deux configurations données peut alors être déduit. Les systèmes robotiques disposant de telles caractéristiques sont cependant fort limités. Bellaïche et al. ont tenté dans [BLC93] d'approximer le système de contrôle d'un robot non-holonome par un système nilpotent. Il en résulte malheureusement un contrôle constant par morceaux ne joignant pas exactement la configuration but.

Ensuite, Rouchon et al. se sont intéressés aux systèmes de contrôle *différentiellement plat* ([RFLM93]). Un système est dit différentiellement plat s'il existe un ensemble fini de paramètres différentiellement indépendant, mais lié aux paramètres d'état et de contrôle du système robotique par des fonctions différentielles. Le problème de détermination d'un mouvement entre deux configurations est alors ramené à une recherche dans cet espace de paramètres. Il en résulte un contrôle du système robotique continu liant les deux configurations d'entrée, mais ne respectant pas nécessairement les contraintes sur le mouvement du robot (*e.g.* bornes sur le contrôle). Cette méthode a été illustrée dans [SLL+97] pour un robot différentiel et dans [LL00] pour une voiture.

Enfin, Murray et Sastry ont proposé la représentation du système de contrôle sous *forme chaînée* ([MS93]) transformant le système de contrôle initial en un système de contrôle de plus fort degré de controllabilité. En choisissant itérativement sur chaque paramètre du système de contrôle chaîné une modification à appliquer afin de mener une configuration initiale vers une configuration but et en déduisant la modification résultante à appliquer sur le contrôle du système robotique, la méthode permet de converger vers un contrôle du robot liant les deux configurations données. La difficulté de cette approche consiste néanmoins à transformer un système de contrôle quelconque sous forme chaînée.

Lorsque le système de contrôle du robot ne dispose pas des propriétés citées ci-dessus, une solution alternative permettant de déterminer une approximation du contrôle optimal entre deux configurations données consiste à utiliser une méthode variationnelle. A partir d'une représentation paramétrique du mouvement, celle-ci consiste à modifier itérativement ces paramètres afin d'assurer d'une part que les configurations initiale et but soient connectées, et d'autre part que les contraintes internes du système soient respectées.

Howard and Kelly ont présenté une telle méthode variationnelle de génération de trajectoire dans [KN03, HK07] : Une représentation paramétrique du contrôle d'entrée du système robotique par fonctions polynomiales ou trapézoïdales est proposée. A partir d'une trajectoire admissible quelconque partant de l'état initial du système, la méthode modifie

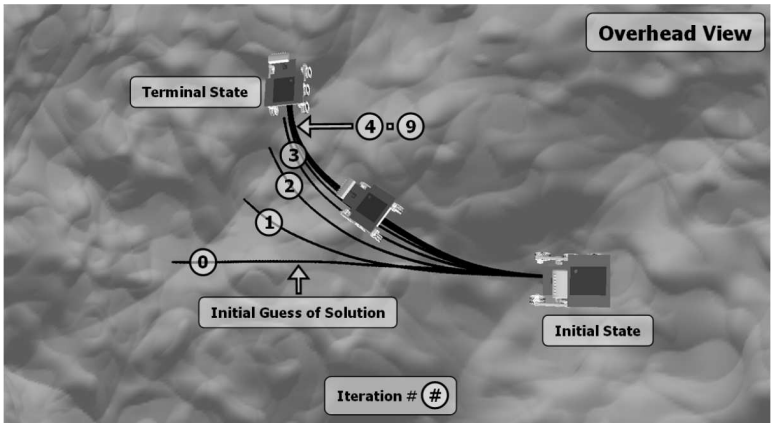


FIGURE 4.6 – Méthode variationnelle de recherche d'un contrôle optimal : une représentation paramétrique du contrôle d'entrée d'un véhicule est itérativement modifiée afin de faire converger l'état final de la trajectoire vers l'état but (source [HK07]).

itérativement le contrôle d'entrée afin de faire converger l'état final de la trajectoire vers l'état but. Le problème d'optimisation sous contraintes posé ici consiste alors en la minimisation d'une fonction de coût représentant divers critères comportant tout aussi bien la longueur du chemin parcourue que la courbure totale ou encore l'énergie consommée. Les contraintes sur les bornes de la trajectoire sont intégrées dans la fonction de coût par l'utilisation de multiplicateurs de Lagrange ([Ber96]). Une méthode de Newton est enfin utilisée afin de minimiser cette fonction de coût. Afin d'améliorer les performances et la convergence de la méthode, une base de connaissance de paramètres initiaux permettant de converger vers divers états-temps tout autour du système est utilisée. L'efficacité de la méthode est telle qu'elle est capable de calculer une trajectoire en quelques millisecondes seulement (*cf.* Fig. 4.6).

4.2.4 Contrainte sur le temps final dans les approches existantes

Parmi les approches de génération de trajectoire existantes, seules [PBR07, GT08] se sont intéressées à la détermination d'une trajectoire entre deux états-temps, c'est à dire entre deux états définis à des temps fixes donnés. Cette contrainte semble pourtant inhérente à toutes méthodes de navigation en environnement dynamique : imaginez par exemple un véhicule robotisé souhaitant traverser une intersection. Si des voitures circulent sur la route

perpendiculaire, notre robot devra s'assurer que l'intersection est libre au moment où il souhaite la franchir. S'il est capable d'évaluer la position et la vitesse des autres véhicules, et par conséquent les intervalles de temps pendant lesquels ils occuperont l'intersection, il peut alors en déduire les plages de temps disponibles pour s'y engager. En considérant sa propre dynamique et ses propres contraintes, notre robot doit alors être capable de générer une trajectoire passant par l'intersection à un temps fixé ou contraint par les plages de temps dont il dispose pour pouvoir la traverser.

L'approche de détermination d'une trajectoire par polynômes d'ordre six citée ci-dessus et proposée en [GT08] ne permet pas de répondre à ce problème : en effet, elle ne prend pas en compte les contraintes cinématiques et dynamique du système robotique. Les travaux de Bianco ([PBR07]) sur les η^3-splines seraient presque à même d'y apporter une solution, néanmoins il ne se soucient guère de la *décidabilité* du problème (aptitude le l'algorithme à spécifier après un nombre d'itérations fini, s'il existe une solution ou non au problème posé), et supposent qu'il existe toujours une solution. Il serait pourtant regrettable de se rendre compte uniquement de l'échec de la méthode une fois en collision avec un autre véhicule.

Même en supposant qu'ils soient capables d'identifier cette situation, que faire dans le cas où la méthode échoue ? Arrêter le processus de navigation ? Générer une trajectoire s'arrêtant avant l'intersection ? Le mouvement des autres véhicules risquant d'évoluer au cours du temps, il serait préférable pendant l'approche de l'intersection d'essayer de prévoir un mouvement permettant de s'engager au mieux au milieu des autres usagers. Même s'il semble impossible à un instant donné de pouvoir s'engager, nous pouvons essayer de déterminer le mouvement nous permettant de nous approcher *"aussi près que possible"* du but pendant la plage de temps disponible en espérant que les autres véhicules adapterons leur vitesse à notre comportement. Il va de soi qu'une fois à faible distance de l'intersection, il est quand même nécessaire de garantir d'être capable de s'arrêter au cas où l'intersection reste bloquée.

Nous avons présenté lors du chapitre précédent (3) une approche de déformation de trajectoire nécessitant également une telle approche de génération de trajectoire permettant de trouver une trajectoire entre deux états-temps tout en respectant les contraintes sur le mouvement du système robotique, et capable de s'arrêter *"aussi près que possible"* de l'état final lorsque ce dernier n'est pas atteignable. Nous proposons alors dans la suite de ce chapitre une nouvelle approche de génération de trajectoire intitulée `Tiji` répondant à cette attente.

4.3 Problème de la génération de trajectoires

4.3.1 Génération de trajectoires "classique"

Soit $\mathcal{A}$ un système robotique évoluant dans un espace de travail $\mathbf{W}$ ($\mathbb{R}^2$ou $\mathbb{R}^3$). La dynamique de $\mathcal{A}$ est décrite par :

$$\dot{s} = f(s, u) \tag{4.1}$$

$$\textit{contraint par} \begin{cases} \mathbf{h_s}(s) \leq 0 \\ \mathbf{h_u}(u) \leq 0 \end{cases} \tag{4.2}$$

où $s = \{s^1, \ldots, s^n\} \in \mathbf{S}$ est l"état de $\mathcal{A}$, $\dot{s}$ sa dérivée par rapport au temps et $u = \{u^1, \ldots, u^m\} \in \mathbf{U}$ un contrôle. $\mathbf{S}$ et $\mathbf{U}$ dénotent respectivement l'espace d'état et l'espace de contrôle de $\mathcal{A}$. $\mathbf{h_s}$ et $\mathbf{h_u}$ représentent respectivement les *bornes sur l'état et le contrôle* contraignant le mouvement du système $\mathcal{A}$. En d'autres termes, il existe $s^i, i \in [1; n]$ (resp. $u^j, j \in [1; m]$) tel que $s^i \in [s^i_{min}; s^i_{max}]$ (resp. $u^j \in [u^j_{min}; u^j_{max}]$). L'ensemble de ces bornes représente les *contraintes sur le mouvement* de $\mathcal{A}$.

Une *trajectoire de contrôles* $\tilde{u} : ([0, t_f] \longrightarrow \mathbf{U}, t \longmapsto u(t))$ est définie par une séquence temporelle de contrôles. A partir d'un état initial s_0 (défini au temps 0) et sous l'action d'une *trajectoire de contrôles* $\tilde{u}$, une *trajectoire d'états* $\tilde{s}$ de $\mathcal{A}$ est définie par $\tilde{s} : ([0, t_f] \longrightarrow \mathbf{S}, t \longmapsto s(t) = \tilde{s}(s_0, \tilde{u}, t))$ *i.e.* une courbe de $\mathbf{S} \times \mathbf{T}$ où $\mathbf{T}$ désigne la dimension temporelle. Chaque état $s(t)$ de cette trajectoire est obtenu par intégration de la dynamique du système :

$$s(t) = \tilde{s}(s_0, \tilde{u}, t) = s_0 + \int_0^t f(s, u)dt \tag{4.3}$$

Etant données deux états s_0 et s_g, la génération de trajectoire pour un système robotique $\mathcal{A}$ régi par (4.1) consiste à trouver une *trajectoire de contrôle faisable* $\tilde{u}$ partant de s_0 et permettant d'atteindre l'état but s_g. La *faisabilité* d'une trajectoire peut être définie formellement comme suit :

Def. 6 (Trajectoire faisable) *Une trajectoire de contrôles $\tilde{u}$ (resp. une trajectoire d'états $\tilde{s}$) définie sur un intervalle de temps $[t_0; t_f]$ et régie par (4.1) est dite faisable si elle respecte les contraintes sur le mouvement définies en (4.2) pour tous $t \in [t_0; t_f]$.*

Soit s_f l'état final atteint par une trajectoire candidate $\tilde{u}$ appliquée à partir de s_0 durant un temps t_f.

$$s_f = s(t_f) = \tilde{s}(s_0, \tilde{u}, t_f) = s_0 + \int_0^{t_f} f(s, u)dt \tag{4.4}$$

Une trajectoire de contrôles faisable $\tilde{u}$ est une solution du problème de génération de trajectoire si son état final s_f respecte l'équation suivante :

$$\mathbf{h_f}(s_f, s_g) = s_g - s_f = 0 \tag{4.5}$$

Cette contrainte est nommée dans la littérature la *contrainte sur l'état final*. La fonction $\mathbf{h_f}(s_f, s_g)$ représente l'erreur entre l'état final s_f de la trajectoire candidate et l'état but s_g que nous souhaitons atteindre.

4.3.2 Ajout de la contrainte sur le temps final

Le point de départ de notre contribution consiste en l'intégration d'une contrainte additionnelle au problème de génération de trajectoire énoncé en Section 4.3.1 : Nous souhaiterions prendre en compte une *contrainte sur le temps final i.e.* un intervalle de temps $[t_{min}; t_{max}]$ durant lequel la trajectoire se termine. Le temps final t_f de la trajectoire de contrôles $\tilde{u}$ doit alors impérativement appartenir à cet intervalle de temps. Formellement, il est nécessaire de prendre en compte les contraintes supplémentaires sur $\mathbf{h_t} = \{\mathbf{h_t^-}; \mathbf{h_t^+}\}$ définies par :

$$\begin{cases} \mathbf{h_t^-}(t_f) = -t_f + t_{min} \leq 0 \\ \mathbf{h_t^+}(t_f) = t_f - t_{max} \leq 0 \end{cases} \tag{4.6}$$

En prenant en compte la contrainte supplémentaire sur le temps final (4.6), notre nouveau problème de génération de trajectoire peut s'exprimer formellement par :

$$\begin{array}{ll} \text{Trouver :} & \tilde{u} : ([0; t_f] \longrightarrow \mathbf{U}, t \longmapsto u(t)) \\ \text{sujet à :} & \mathbf{h_f}(s_f, s_g) = 0 \\ & \mathbf{h_s}(s) \leq 0 \\ & \mathbf{h_u}(u) \leq 0 \\ & \mathbf{h_t}(t_f) \leq 0 \end{array} \tag{4.7}$$

4.3.3 Contrainte sur le temps final et atteignabilité

Notre problème de génération de trajectoire ne diffère du problème standard que par la contrainte additionnelle sur le temps final. Ce changement change cependant fortement la complexité de celui-ci : Soit un système robotique $\mathcal{A}$. Dans un espace d'état ouvert et libre, le système $\mathcal{A}$ est dit *contrôlable* si tous $s \in \mathbf{S}$ de $\mathcal{A}$ garantit la propriété suivante :

$$\forall (s_0, s) \in \mathbf{S}^2, \exists \tilde{u}, \exists t_f, s = \tilde{s}(s_0, \tilde{u}, t_f) \tag{4.8}$$

Ce critère statue pour un système s'il est possible de déterminer des trajectoires admissibles pour celui-ci entre chaque couple d'états de $\mathbf{S}$. La controllabilité n'est cependant plus valide dans le cas où t_f est contraint. L'existence d'une trajectoire connectant les états-temps initial et final dans un tel cas est liée à la notion d'*atteignabilité* déjà introduite en Section 3.3.6. Nous rappelons que l'espace atteignable $\mathcal{R}(s_0, t_f)$ à partir d'un état initial s_0 à un temps donné t_f est défini par :

$$\mathcal{R}(s_0, t_f) = \{s \in \mathbf{S} \mid \exists \tilde{u}, \tilde{s}(s_0, \tilde{u}, t_f) = s\} \tag{4.9}$$

Cette notion peut être trivialement étendue à un intervalle de temps :

$$\mathcal{R}(s_0, t_{min}, t_{max}) = \{s \in \mathbf{S} \mid \exists t_f \in [t_{min}; t_{max}], \exists \tilde{u}, \tilde{s}(s_0, \tilde{u}, t_f) = s\} \tag{4.10}$$

Comme l'illustre l'Eq. (4.10), l'ensemble des états-temps atteignables durant un intervalle de temps n'est qu'un sous-ensemble de l'espace d'états-temps $\mathbf{S} \times \mathbf{T}$ du système $\mathcal{A}$.

Nous souhaiterions proposer une nouvelle méthode de génération de trajectoire permettant de déterminer des trajectoires faisables quelque soit la paire d'états-temps initial et but. Cette méthode doit alors être capable de converger vers le but lorsqu'une solution existe et, lorsque l'état-temps but n'est pas atteignable, nous voudrions trouver une trajectoire faisable dont l'état-temps final soit "aussi proche que possible" du but. En d'autres termes, nous souhaitons trouver un état final s_f de la trajectoire calculée appartenant à l'espace atteignable $\mathcal{R}(s_0, t_{min}, t_{max})$, et tel que la distance (calculée via une métrique donnée sur l'espace d'état $\mathbf{S}$) entre cet état et l'état but s_g soit minimale. Notez que dans le cas de systèmes dynamiques complexes, la détermination de l'espace atteignable $\mathcal{R}(s_0, t_{min}, t_{max})$ est trop coûteuse pour être intégrée à un générateur de trajectoire nécessitant un calcul de ces dernières en temps réel.

Afin de pouvoir calculer ces trajectoires alternatives dont l'état final s_f respecte les contraintes énoncées ci-dessus, nous modifions le problème de génération de trajectoire comme suit :

$$\begin{array}{ll} \text{Trouver :} & \tilde{u} : ([0; t_f] \longrightarrow \mathbf{U}, t \longmapsto u(t)) \\ \text{minimisant :} & \mathbf{h_f}(s_f, s_g) \\ \text{sujet à :} & \mathbf{h_s}(s) \leq 0 \\ & \mathbf{h_u}(u) \leq 0 \\ & \mathbf{h_t}(t_f) \leq 0 \end{array} \tag{4.11}$$

Des méthodes variationnelles constituent une bonne option pour trouver une solution à un tel problème d'optimisation sous contraintes. Nous présentons alors dans la section suivante comment résoudre un problème de génération de trajectoires avec de telles méthodes.

4.4 Génération de trajectoire par méthode variationnelle

Considérons le problème de génération de trajectoire décrit en (4.11) destiné à déterminer une trajectoire de contrôles $\tilde{u}$ entre deux états-temps $(s_0, 0)$ et $(s_g, t_g), t_g \in [t_{min}; t_{max}]$ et supposons une trajectoire initiale $\tilde{u}_{init}$, partant de s_0 au temps 0 et atteignant s_f au temps t_f. Résoudre un problème de génération de trajectoire par méthode variationnelle consiste à modifier itérativement la trajectoire $\tilde{u}_{init}$ afin de faire correspondre son état final s_f avec l'état but s_g tout en conservant la satisfaction des contraintes données en (4.2) et (4.6). Les principales étapes de la méthode variationnelle sont présentées ci-dessous.

4.4.1 Représentation paramétrique du contrôle

Tout d'abord, une paramétrisation est utilisée dans le but de réduire la dimension de l'espace de recherche en le limitant à l'ensemble des trajectoires définies par l'espace de paramètres $\mathbf{P}$ considéré. Bien qu'il ne permette de représenter qu'un sous-ensemble de toutes les trajectoires admissibles, une paramétrisation appropriée suffit généralement à trouver une solution à notre problème de génération de trajectoires dans le cas où l'état but est atteignable.

Le profil de la trajectoire de contrôles $\tilde{u}$ est alors redéfini comme suit :

$$\tilde{u}(t) = \tilde{u}(p, t), \forall t \in [0; t_f] \tag{4.12}$$

où $p = (p_1, \ldots, p_k) \in \mathbf{P} = \mathbb{R}^k$ est un vecteur de paramètres.

Une paramétrisation de la trajectoire d'états $\tilde{s}$ est également considérée :

$$\tilde{s}(t) = \tilde{s}(s_0, \tilde{u}(p), t) = \tilde{s}(s_0, p, t), \forall t \in [0; t_f] \tag{4.13}$$

Et la dynamique de $\mathcal{A}$ définie en (4.4) est modifiée en conséquence :

$$\dot{s}(t) = f(s(t), u(t)) = f(\tilde{s}(s_0, p, t), \tilde{u}(p, t)) = f(p, t) \tag{4.14}$$

Notez que pour un ensemble de valeurs des paramètres quelconque (définissant ainsi un contrôle et un temps final quelconque), les trajectoires d'états et de contrôles paramétrées résultantes ne respectent pas nécessairement les contraintes sur le mouvement (4.2) et sur le temps final (4.6). Une représentation paramétrique de la trajectoire de contrôles $\tilde{u}$ (resp. trajectoire d'états $\tilde{s}$) satisfaisant ces contraintes est notée $\tilde{u}^*$ (resp. $\tilde{s}^*$).

4.4.2 Modification itérative de la trajectoire

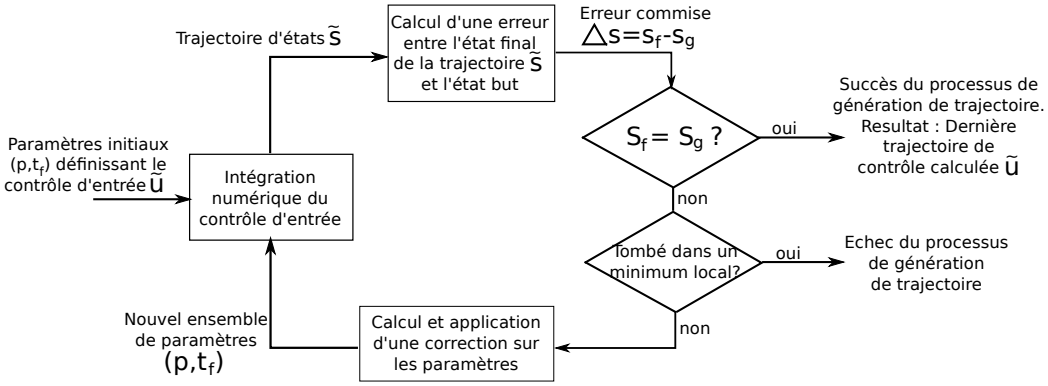


FIGURE 4.7 – Principe général d'une méthode variationnelle de génération de trajectoires.

Etant donné un ensemble de paramètres initiaux p définissant la trajectoire de contrôle $\tilde{u}$ et le temps final t_f déterminant sa durée, l'ensemble de paramètres (p,t_f) doit être amélioré afin d'obtenir une trajectoire de contrôle satisfaisant la contrainte sur l'état final (4.5), les contraintes sur le mouvement (4.2) et la contrainte sur le temps final (4.6).

Pour ce faire, des corrections itératives sont appliquées sur ces paramètres jusqu'à ce que l'état final s_f de la trajectoire modifiée converge vers l'état but s_g ou échoue lors de sa tentative. Ce processus itératif est illustré par la Fig. 4.7.

A partir des paramètres initiaux p, une expression analytique de la trajectoire de contrôles $\tilde{u}$ est déterminée, et la trajectoire d'états $\tilde{s}$ est calculée par intégration de ce contrôle d'entrée. Etant donné l'état final s_f de la trajectoire d'états, une erreur entre celui-ci et l'état but s_g est calculée puis une correction à appliquer sur les paramètres p en est déduite afin de réduire cette erreur.

La considération d'un système dynamique complexe soumis à des contraintes sur son mouvement ainsi que la prise en compte de la contrainte sur le temps final de la trajectoire soulèvent néanmoins quelques difficultés au cours du processus d'optimisation :

– Premièrement, comment calculer la correction à appliquer sur l'ensemble de paramètres afin de converger vers l'état-temps but si une solution existe ou vers un état-temps s'en rapprochant dans le cas contraire ? Dans l'incapacité de déterminer à priori l'existence d'une solution, ces deux cas doivent être considérés de la même manière.

– Deuxièmement, comment garder les contraintes sur le mouvement satisfaites après application d'une ou plusieurs corrections sur l'ensemble des paramètres définissant le contrôle ? Même dans le cas de paramètres initiaux définissant une trajectoire admissible, l'application de corrections consécutives sur ces paramètres peut résulter en une trajectoire ne respectant plus les contraintes sur le mouvement. De plus, si l'état-temps but n'est pas atteignable, nous aimerions pouvoir garantir que la trajectoire finale produite respecte également ces contraintes même si elle n'atteint pas le but.
– Enfin, toute méthode variationnelle est sujette à des problèmes de minima locaux. Comment les éviter ?

Nous proposons alors dans la section suivante un nouvel algorithme permettant de résoudre ce problème de génération de trajectoire intégrant la *contrainte sur le temps final* et apportant une solution aux difficultés énoncées ci-dessus.

4.5 Tiji : Générateur de trajectoire avec contrainte sur le temps final

Le coeur de notre contribution présentée dans ce chapitre réside dans l'intégration de la *contrainte sur le temps final* à une méthode de génération de trajectoires variationnelle, et à l'apport d'une solution au problème d'atteignabilité inhérent à cette contrainte. L'idée principale de la solution proposée ici consiste à calculer une trajectoire de contrôles *faisable* (respectant les contraintes sur le mouvement et sur le temps final) quelque soit la paramétrisation d'entrée, et à assurer la convergence de la méthode en dépit des modifications apportées à la trajectoire résultante. Grâce aux paramétrisations des trajectoires de contrôles et d'états proposées en (4.12) et (4.13), notre problème de génération de trajectoires initialement formalisé en (4.11) est reformulé comme suit :

$$\begin{array}{ll} \text{Trouver :} & (p, t_f)\ definissant\ \tilde{u}^* : ([0; t_f] \longrightarrow \mathbf{U}, t \longmapsto u(t) = \tilde{u}^*(p, t)) \\ \text{minimisant :} & \mathbf{J}(s_0, p, t_f, s_g) = \|\mathbf{h_f}(s_f, s_g)\|^2 \\ \text{sujet à :} & \mathbf{h_s}(p) \leq 0 \\ & \mathbf{h_u}(p) \leq 0 \\ & \mathbf{h_t}(t_f) \leq 0 \end{array} \tag{4.15}$$

Dans le but d'appliquer les corrections itératives sur l'ensemble des paramètres définissant la trajectoire de contrôles, une fonction de coût $\mathbf{J}(s_0, p, t_f, s_g)$ est considérée. Celle-ci représente simplement une distance de

Mahalanobis (norme pondérée) entre l'état final s_f de la trajectoire courante et l'état but s_g. Lorsque cette distance est nulle, s_f et s_g sont confondus.

Un nouvel algorithme de génération de trajectoires nommé `Tiji`, consistant à résoudre ce problème d'optimisation, est décrit ci-dessous.

4.5.1 `Tiji` : Présentation de l'algorithme

Algorithm 5: `Tiji`

Input: $s_0, s_g, [t_{min}; t_{max}]$
Output: $\tilde{u}^*$, `succès de la convergence?`

1 $i = 0$;
2 $(p, t_f) = ParametresInitiaux(s_0, s_g, [t_{min}; t_{max}])$;
3 **repeat**
4 *stage 1 :* Calcul d'une trajectoire de contrôles faisable $\tilde{u}^*$;
5 *stage 2 :* Mise à jour des paramètres (p, t_f) afin de converger vers l'état but;
6 $i = i + 1$;
7 **until** $\mathbf{J}(s_0, p, t_f, s_g) \leq \varepsilon$ **or** $i = i_{\max}$;
8 **return** $(\tilde{u}^*, s_f = s_g?)$;

L'algorithme 5 décrit notre adaptation d'un processus d'optimisation standard permettant de générer une trajectoire entre deux états-temps. Etant donné un état initial s_0, un état but s_g et un intervalle de temps $[t_{min}\,; t_{max}]$ contraint pour rejoindre ce dernier, un ensemble de paramètres initiaux (p, t_f) est déterminé. Ces paramètres définissent la trajectoire qui sera itérativement modifiée afin d'atteindre l'état but s_g. Le calcul des paramètres initiaux est discuté en Section 4.5.2.

Une fois ces paramètres initiaux déterminés, l'algorithme cycle jusqu'à convergence ou échec. Un cycle est constitué de deux étapes principales :

– ***Etape 1 :*** **Calcul d'une trajectoire de contrôles faisable** $\tilde{u}^*$
A partir de la paramétrisation donnée en (4.12), une trajectoire de contrôles $\tilde{u}$ est calculée. Cependant, au cours du processus d'optimisation, les modifications successives des paramètres p définissant le contrôle peuvent résulter en une trajectoire ne satisfaisant plus les *contraintes sur le mouvement* (4.2) du système robotique ou la *contrainte sur le temps final* (4.6). Un processus de modification de la trajectoire est alors appliqué afin de la rendre faisable. Les contraintes sur le mouvement et sur l'état final correspondant à des bornes sur les différents

paramètres d'état et de contrôle du robot définis le long de sa trajectoire (*cf.* Section 4.3.1). La faisabilité du mouvement est alors assurée en saturant les différents profils de contrôle et d'état ne respectant pas ces contraintes. Ce procédé retourne alors une trajectoire de contrôle $\tilde{u}^*$ (et la trajectoire d'état $\tilde{s}^*$ associée) garantissant que toutes les contraintes considérées soient respectées. Le processus de saturation est décrit en Section 4.5.3.

– ***Etape 2 :*** **Mise à jours des paramètres p et t_f afin de converger vers l'état but**
Une fois qu'une trajectoire faisable $\tilde{u}^*$ est obtenue, il est possible de vérifier si son état final s_f (défini au temps t_f) correspond à l'état but s_g. La fonction de coût **J** définie lors du problème d'optimisation (4.15) est conçue pour être minimale lorsque cette condition est satisfaite. Si l'état but n'a pas encore été atteint, les variations de **J** par rapport aux paramètres (p, t_f) sont étudiées afin de la faire décroître. Une correction $cor_{(p,t_f)}$ est alors calculée et appliquée sur ces paramètres (*cf.* Section 4.5.4).
La convergence vers le but est atteinte lorsque la valeur de la fonction de coût **J** est suffisamment faible. Si ce critère n'est pas assuré après un nombre fixé d'itérations de l'algorithme (dans le cas où la méthode serait tombée dans un minimum local), le processus d'optimisation est stoppé.

4.5.2 Détermination des paramètres initiaux de la trajectoire

Au cours de la première étape de l'algorithme 5, un ensemble de paramètres permettant de calculer la trajectoire initiale doit être fourni. Le succès de la méthode d'optimisation (dû aux minima locaux) et le nombre d'étapes requis pour converger vers le but dépendent fortement du choix de ces paramètres initiaux, il ne doit donc pas être négligé.

Une solution simple consiste à déterminer ces paramètres analytiquement à partir d'une intuition sur le contrôle à appliquer : Par exemple, en calculant la distance approximative à parcourir par le robot entre les états initiaux et finaux, il est possible d'évaluer la vitesse et le temps requis pour rejoindre l'état but et d'en déduire le contrôle à appliquer. Cette méthode a l'avantage d'être rapide à calculer et de nécessiter peu de mémoire, néanmoins il peut en règle générale s'avérer très difficile de trouver une bonne heuristique permettant de déterminer ces paramètres.

Une seconde solution consiste à précalculer une base de paramètres

initiaux contenant un échantillonnage de l'espace d'états-temps ainsi que les paramètres requis pour atteindre chaque état-temps de cette base. On détermine alors les états-temps de cette base (et les paramètres associés) correspondant au mieux aux états-temps initial et final entre lesquels une trajectoire est générée. Une telle méthode permet en général de réduire considérablement le nombre d'étapes de convergence nécessaires, néanmoins sa taille en mémoire peut être considérable si la dimension de l'espace d'état du système est importante.

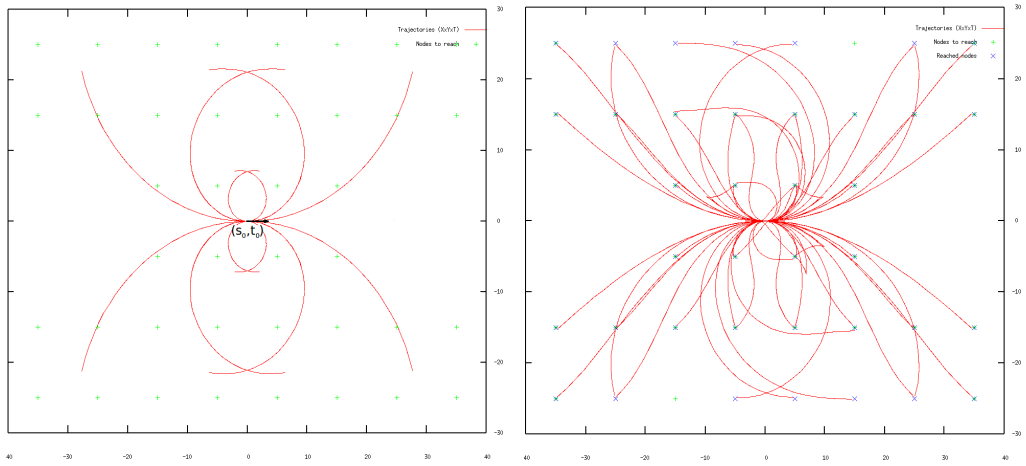
Cette seconde solution a été utilisée. La Fig. 4.8 illustre la méthode de construction d'une base de paramètres initiaux pour notre générateur de trajectoires `Tiji` : Soit l'état-temps initial $(s_0, t_0) = \{{s_0}^1, \ldots, {s_0}^n, t_0\}$ et l'état-temps final $(s_g, t_g) = \{{s_g}^1, \ldots, {s_g}^n, t_g\}$. Nous échantillonnons alors chaque caractéristique $s^i, i \in [1; n]$ de ces deux états afin d'obtenir un échantillonage régulier de l'espace des paramètres d'entrée de l'algorithme de génération de trajectoire 5. Afin d'essayer de déterminer une trajectoire (et les paramètres permettant de la générer) pour chacune de ces valeurs d'entrées, nous utilisons un premier ensemble de trajectoire dont nous connaissons déjà les paramètres associés. Celles-ci peuvent par exemple être calculées à partir d'un échantillonnage sur l'espace de contrôle. Les noeuds (états-temps) de la base les plus proches des états terminaux des trajectoires déjà calculées sont déterminés (Fig. 4.8(a)). `Tiji` est alors utilisé pour calculer les paramètres requis pour atteindre ces noeuds en partant des paramètres des trajectoires les plus proches (Fig. 4.8(b)). Nous essayons alors d'atteindre de proche en proche les noeuds de l'échantillonnage sur l'espace d'état-temps de cette manière (Fig. 4.8(c)). Le processus s'arrête lorsque `Tiji` ne trouve plus aucun noeud atteignable sur l'échantillonnage de l'espace d'états-temps.

4.5.3 Calcul d'une paramétrisation admissible de la trajectoire

A partir des paramètres d'entrée (p, t_f) les différents profils de la trajectoire de contrôle $\tilde{u}$ peuvent être calculés (*cf.* (4.12)). Etant donné un état-temps initial s_0 et l'expression analytique de $\tilde{u}$ ainsi obtenue, chaque état $s(t)$ de la trajectoire d'états s suivie par le système robotique est alors déterminé par l'intégration de la dynamique du système :

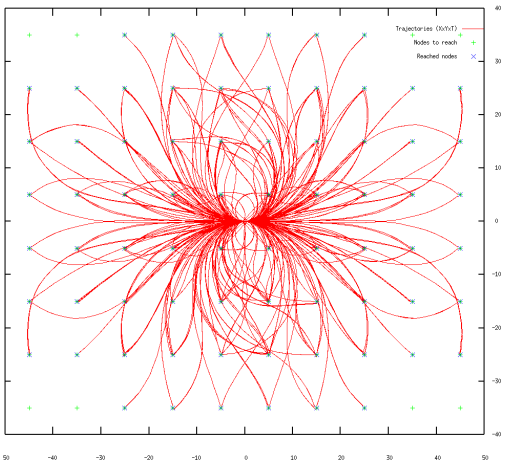
$$s(t) = \tilde{s}(p, t) = s_0 + \int_t f(p, t)dt, \forall t \in [0; t_f] \tag{4.16}$$

Même si l'ensemble de paramètres initiaux fournit une trajectoire initiale faisable, les corrections consécutives appliquées sur ces paramètres peuvent



(a) Initialisation (vue $x \times y$)

(b) Première itération (vue $x \times y$)



(c) Seconde itération (vue $x \times y$)

FIGURE 4.8 – Premières étapes de la construction d'une base de paramètres permettant de déterminer les paramètres initiaux à fournir en entrée de `Tiji` afin de générer une trajectoire entre deux états-temps : Lors de l'initialisation, un ensemble de trajectoires initiales (basé sur un échantillonage du contrôle d'entrée) est fourni. On essaie alors d'atteindre l'ensemble des noeuds de l'échantillonnage sur l'espace d'états-temps proches des états-temps terminaux des trajectoires précédemment calculées. La base de paramètres est alors construite de proche en proche au cours des itérations suivantes à partir des noeuds de l'échantillonnage sur l'espace d'états-temps déjà atteints.

la rendre infaisable (les contraintes sur le mouvement ou sur le temps final peuvent être violées). Afin d'éviter ce problème, nous proposons une solution permettant d'obtenir une trajectoire faisable quelque soit l'ensemble de paramètres d'entrée : Pour ce faire, nous procédons à la saturation des différents profils de contrôle et d'état ne satisfaisant pas les contraintes du système. Les différentes étapes de ce processus de saturation sont décrites ci-dessous.

Application de la *contrainte sur le temps final*

La *contrainte sur le temps final* est considérée en premier lieu. Elle est très simplement prise en compte en empêchant le paramètre d'entrée t_f correspondant à la durée de la trajectoire de prendre une valeur en dehors de l'intervalle de temps $[t_{min}; t_{max}]$. Un temps final approprié est donc obtenu comme suit :

$$t_f^* = \begin{cases} t_{min} & \text{si } \mathbf{h_t}^-(t_f) = -t_f + t_{min} > 0 \\ t_{max} & \text{si } \mathbf{h_t}^+(t_f) = t_f - t_{max} > 0 \\ t_f & \text{sinon} \end{cases} \tag{4.17}$$

Cette nouvelle valeur du temps final t_f^* devient alors, durant la suite de cette étape d'optimisation, le temps auquel sera évalué l'état final s_f de la trajectoire d'états. Il reste donc à prendre en compte les *contraintes sur le mouvement* du système robotique. La section suivante décrit alors leur intégration.

Saturation des profils de contrôle et d'état non admissibles

Etant donné les paramètres d'entrée (p, t_f^*) les profils des trajectoires de contrôles et d'états peuvent être calculés à partir des Eqs. (4.12) et (4.13). Ces fonctions peuvent cependant dépasser les *contraintes sur le mouvement i.e.* les *bornes sur le contrôle* $\mathbf{h_u}$ et les *bornes sur l'état* du système $\mathbf{h_s}$ sur un ou plusieurs intervalles de temps. Nous notons alors $\mathbf{I}^{\overline{U}}$ (resp. $\mathbf{I}^{\overline{S}}$) l'ensemble des intervalles des temps sur lesquels les bornes sur le contrôle (resp. bornes sur l'état) n'ont pas été respectées. A partir de la formalisation des bornes sur le contrôle et sur l'état données en Section 4.3.1, nous pouvons exprimer ces intervalles comme suit :

$$\begin{aligned} \Big\{\mathbf{I}^{\overline{U}} = \Big\{\mathbf{I}_1^{\overline{U}}, \ldots, \mathbf{I}_k^{\overline{U}}\Big\} \,|\forall t \in \mathbf{I}_i^{\overline{U}}, i \in [1;k], \\ \exists u^j(t), j \in [1;m], u^j(t) < u^j_{min} \text{ or } u^j(t) > u^j_{max}\Big\} \end{aligned} \tag{4.18}$$

$$\begin{aligned} \Big\{\mathbf{I}^{\overline{S}} = \Big\{\mathbf{I}_1^{\overline{S}}, \ldots, \mathbf{I}_l^{\overline{S}}\Big\} \,|\forall t \in \mathbf{I}_i^{\overline{S}}, i \in [1;l], \\ \exists s^j(t), j \in [1;n], s^j(t) < s^j_{min} \text{ or } s^j(t) > s^j_{max}\Big\} \end{aligned} \tag{4.19}$$

L'idée générale du processus de saturation consiste alors à déterminer ces intervalles ainsi que les nouveaux profils de contrôle ou d'état devant y être définis. Les bornes sur le contrôle et sur l'état sont appliquées successivement, néanmoins les trajectoires de contrôles et d'états étant liées, un processus de saturation en quatre étapes a été établi :

– Etape 1 : Application des bornes du contrôle sur la trajectoire de contrôles
A partir de la représentation paramétrique du contrôle donnée par l'Eq. (4.12), une simple étude des variations de ces fonctions suffit à déterminer l'ensemble des intervalles $\mathbf{I}^{\overline{U}}$ sur lesquels les *bornes sur le contrôle* sont dépassées.
Cet ensemble d'intervalles représente simplement les intervalles de temps au cours desquels un ou plusieurs contrôles d'entrée ont excédé leurs bornes minimales ou maximales. Une trajectoire de contrôle $\tilde{u}_{\{\overline{u}\}}$ respectant ces limites est donnée par :

$$\tilde{u}_{\{\overline{u}\}}(p,t) = \begin{cases} \mathbf{u}_{extl}(t) & \text{si } t \in \mathbf{I}^{\overline{U}} \\ \tilde{u}(p,t) & \text{sinon} \end{cases} \tag{4.20}$$

où $\mathbf{u}_{extl}(t)$ représente le contrôle extrême (minimal ou maximal) applicable au temps t respectant les bornes sur le contrôle. Des fonctions paramétriques par morceaux sont donc nécessaires pour représenter ce nouveau profil du contrôle ainsi que ceux des fonctions d'états proposées ci-dessous.

– Etape 2 : Calcul de la trajectoire d'états à partir de la trajectoire de contrôles bornée
Une trajectoire d'états $\tilde{s}_{\{\overline{u}\}}$ prenant en compte les *bornes sur le contrôle* peut être déduite en intégrant $\tilde{u}_{\{\overline{u}\}}$ à partir de l'état initial s_0 du système :

$$\tilde{s}_{\{\overline{u}\}}(p,t) = s_0 + \int_t f(s_0, \tilde{u}_{\{\overline{u}\}}(p,t), t)dt \tag{4.21}$$

– Etape 3 : Application des bornes de l'état sur la trajectoire d'états
A partir de cet instant, les seules contraintes n'ayant pas été prises en compte sont les *bornes sur l'état du système*. Comme pour les bornes sur le contrôle, l'ensemble des intervalles $\mathbf{I}^{\overline{S}}$ dépassant les bornes sur l'état est déterminé en étudiant les variations de l'Eq. (4.13). La trajectoire d'état $\tilde{s}_{\{\overline{u}\}}$ doit alors être saturée sur ces intervalles afin de prendre en compte l'ensemble des contraintes souhaitées. La trajectoire d'états $\tilde{s}^*$ résultante est donc exprimée par :

$$\tilde{s}^*(p,t) = \begin{cases} \mathbf{s}_{extl}(t) & \text{si } t \in \mathbf{I}^{\overline{S}} \\ \tilde{s}_{\{\overline{u}\}}(p,t) & \text{sinon} \end{cases} \tag{4.22}$$

où $\mathbf{s}_{extl}(t)$ est la valeur extrême (minimale ou maximale) de l'état au temps t respectant les bornes d'état. La trajectoire d'états $\tilde{s}^*$ garantie donc de respecter l'ensemble des toutes les *contraintes sur le mouvement* du système $\mathcal{A}$ et par conséquence d'être faisable.

– Etape 4 : Dérivation d'une trajectoire de contrôles faisable
Enfin, une trajectoire de contrôles faisable $\tilde{u}^*$ est dérivée de la trajectoire d'états $\tilde{s}^*$:

$$\tilde{u}^*(p,t) = \begin{cases} \mathbf{u}_{extl}(t) & \text{si } t \in \mathbf{I}^{\overline{U}} - \mathbf{I}^{\overline{S}} \\ 0 & \text{si } t \in \mathbf{I}^{\overline{S}} \\ \tilde{u}(p,t) & \text{sinon} \end{cases} \tag{4.23}$$

De la même manière que pour $\tilde{s}^*$, le mouvement décrit par la trajectoire de contrôle $\tilde{u}^*$ appliquée à partir de l'état initial s_0, a la garantie d'être faisable pour le système $\mathcal{A}$.

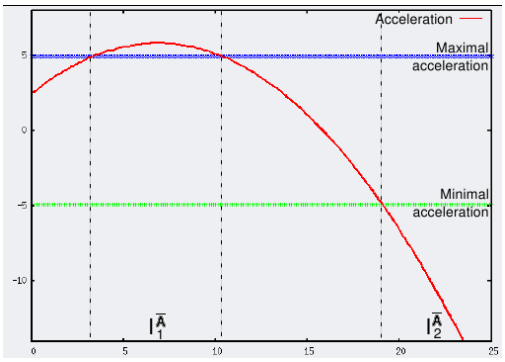
Exemple

La Fig. 4.9 illustre le processus de saturation des trajectoires d'états et de contrôles présenté au cours de la section précédente. Supposez un système 1D de type double intégrateur disposant de bornes sur sa vitesse et son accélération. Soit le profil paramétrique initial de l'accélération du système donné en Fig. 4.9(a). Les bornes d'accélération étant dépassées sur les intervalles $\mathbf{I}_1^{\overline{A}}$ et $\mathbf{I}_2^{\overline{A}}$, nous saturons l'accélération sur ces intervalles afin d'obtenir un nouveau profil respectant ces contraintes (Fig. 4.9(c)). En intégrant ce dernier à partir de l'état initial s_0 du système, nous obtenons alors un profil de vitesse du système satisfaisant les bornes sur l'accélération, mais pas celles sur la vitesse (Fig. 4.9(d)). Nous saturons alors celle-ci sur l'intervalle $\mathbf{I}_1^{\overline{V}}$ où les bornes de vitesse sont dépassées. Le nouveau profil obtenu (Fig. 4.9(f)) satisfait donc l'ensemble des contraintes sur le mouvement. Afin d'obtenir le contrôle d'entrée admissible correspondant à ce profil de vitesse, nous redérivons celui-ci. La Fig. 4.9(e) présente alors le profil d'accélération admissible pouvant être passé en entrée du système robotique.

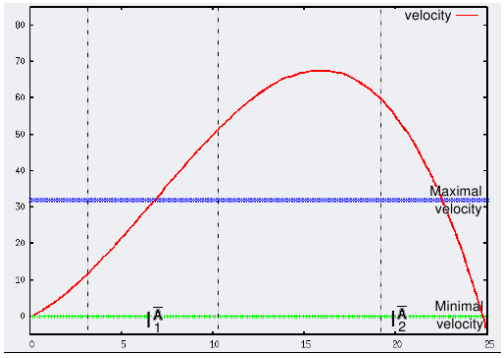
4.5.4 Mise à jour de la paramétrisation de la trajectoire

Une fois qu'une trajectoire faisable a été déterminée, son état final s_f est facilement obtenu comme suit :

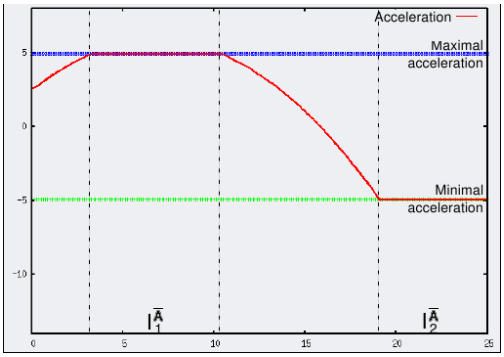
$$s_f = \tilde{s}^*(p, {t_f}^*) \tag{4.24}$$



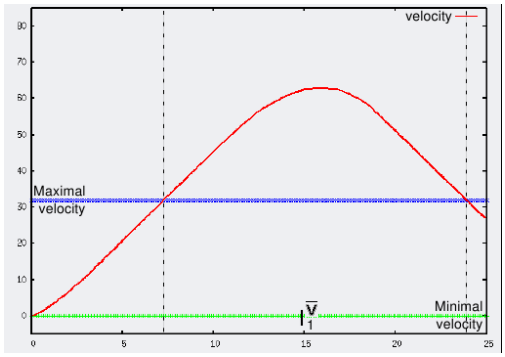
(a) Etape 1 : Profil d'accélération linéaire $a(t)$ avant saturation.



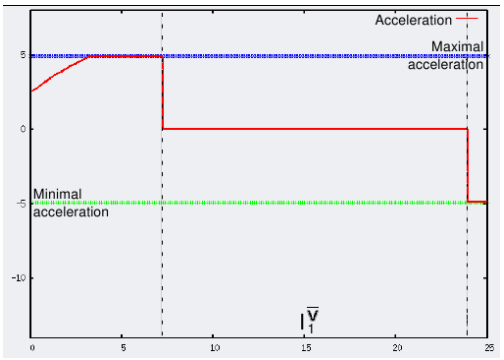
(b) Etape 1 : Profil de vitesse linéaire $v(t)$ avant saturation.



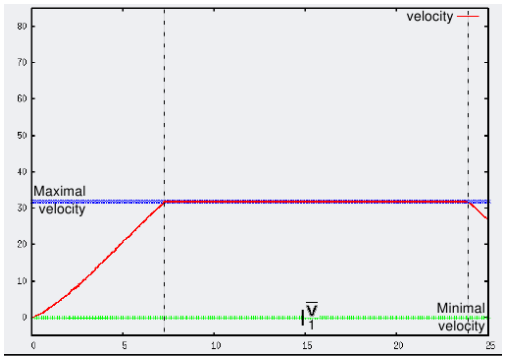
(c) Etape 2 : Profil d'accélération linéaire $a(t)$ après saturation des intervalles sur lesquelles les bornes d'accélération minimales et maximales sont dépassées.



(d) Etape 2 : Profil de vitesse linéaire $v(t)$ prenant en compte les bornes d'accélération. Il est obtenu par intégration du profil d'accélération donné en Fig. 4.9(c).



(e) Etape 3 : Profil d'accélération linéaire $a(t)$ prenant en compte toutes les *contraintes sur le mouvement*. Obtenu par intégration du profil de vitesse donné en Fig. 4.9(f).



(f) Etape 3 : Profil de vitesse linéaire $v(t)$ après saturation des intervalles sur lesquels les bornes de vitesse sont dépassées.

FIGURE 4.9 – Différentes étapes de calcul des profils de vitesse et accélération linéaire. Les croix représentent les bornes minimales et maximales sur ces différents profils.

Si s_f est suffisamment proche de l'état but s_g, *i.e.* si $\mathbf{J}(s_0, p, t_f, s_g) \leq \varepsilon$, où ε est un seuil fixé, alors l'approche variationnelle a convergé comme nous le souhaitions et une solution a été trouvée. Dans le cas contraire, il est nécessaire d'appliquer une correction sur les paramètres d'entrée $(p, {t_f}^*)$ afin d'y parvenir.

Afin de procéder au calcul d'une correction sur ces paramètres permettant de minimiser la fonction de coût $\mathbf{J}(s_0, p, t_f, s_g)$, de nombreuses méthodes d'optimisation existent (voir [NW99, Ber96] pour un résumé de ces approches). Néanmoins, elle peuvent généralement être classifiées en quatre catégories : les méthodes de relaxation (optimisation séquentielle de chaque paramètre), les méthodes de descente de gradient (recherche d'une direction descendante donnée par le gradient de la fonction de coût), les méthodes de Newton ou assimilées (approximation locale au second ordre de la fonction de coût) et enfin la méthode stochastique de recuit simulé (en anglais : simulated annealing [KGV83, GKR94]). Les principales différences entre ces différentes méthodes tiennent au choix de la direction et du pas de descente, et par conséquence, à leur coût de calcul. Une méthode de Newton standard a été utilisée dans notre cas.

Un minimum local de la fonction de coût est trouvé lorsque ses dérivées partielles $\left[\frac{\partial \mathbf{J}}{\partial(p,t_f)}\right]$ par rapport aux paramètres (p, t_f) sont égales à zéro. Une racine de $\left[\frac{\partial \mathbf{J}}{\partial(p,t_f)}\right]$ peut alors être déterminée en linéarisant son expression comme suit :

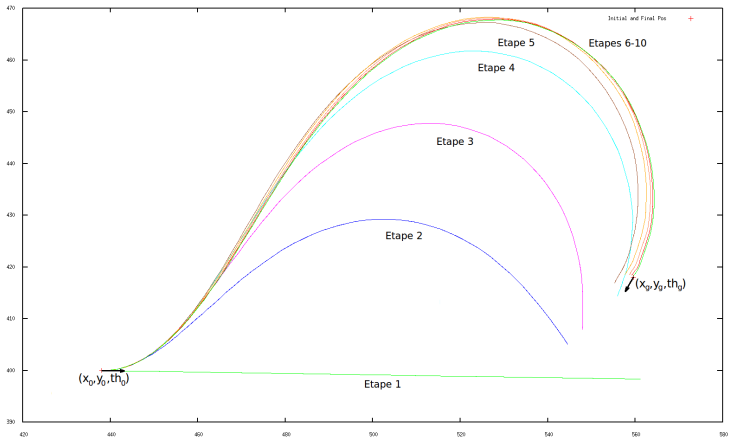
$$\left[\frac{\partial \mathbf{J}}{\partial(p,t_f)}\right] \simeq \left[\frac{\partial^2 \mathbf{J}}{\partial(p,t_f)^2}\right] \Delta(p,t_f) \tag{4.25}$$

où $\Delta(p, t_f)$ est l'erreur supposée commise sur les paramètres , et où $\left[\frac{\partial^2 \mathbf{J}}{\partial(p,t_f)^2}\right]$ sont les dérivées partielles de second ordre de la fonction de coût par rapport à ces paramètres (matrice de dimensions $(dim\ \mathbf{P}+1) \times (dim\ \mathbf{P}+1)$). Afin de minimiser la fonction de coût $\mathbf{J}$, une correction à appliquer sur les paramètres (p, t_f) est alors calculée comme suit :

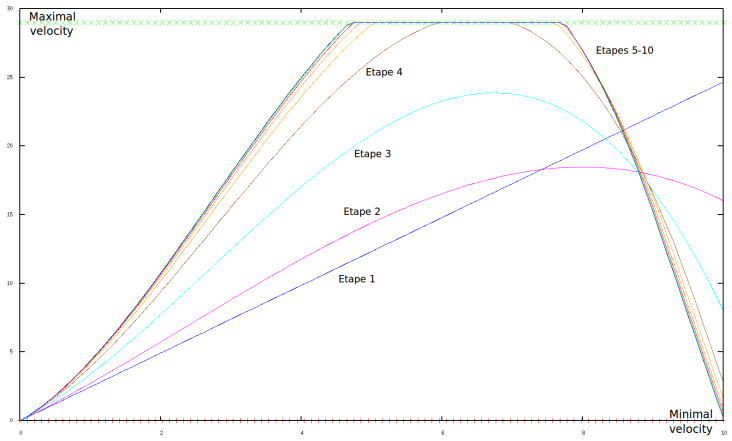
$$cor_{(p,t_f)} = -\tau \left[\frac{\partial^2 \mathbf{J}}{\partial(p,t_f)^2}\right]^{-1} \left[\frac{\partial \mathbf{J}}{\partial(p,t_f)}\right] \tag{4.26}$$

où $\tau \in [0;1]$ est le gain de la correction. La matrice inverse des dérivées partielles de second ordre représente alors la direction de la correction, et $\tau \left[\frac{\partial \mathbf{J}}{\partial(p,t_f)}\right]$ le pas de descente.

La Fig. 4.10 illustre la convergence de la méthode après application de plusieurs corrections sur l'ensemble des paramètres définissant le contrôle d'un véhicule.



(a) Profil du chemin (vue $x \times y$)



(b) Profil de la vitesse (v(t))

FIGURE 4.10 – Trajectoires successives d'un véhicule (définies ici par le chemin suivi et le profil de vitesse utilisé) obtenues après application des corrections successives appliquées sur les paramètres définissant son contrôle.

Un exemple complet de l'application du module de génération de trajectoire `Tiji` à un véhicule différentiel est proposé dans la section suivante.

4.6 Cas d'étude : Robot différentiel

Nous présentons dans cette section une application complète du générateur de trajectoire `Tiji` à un véhicule différentiel.

4.6.1 Modèle du système

Le modèle d'un robot différentiel a déjà été présenté dans le chapitre précédent en Section 3.4.3. Nous rappelons néanmoins ici ses principales caractéristiques.

Un état d'un système différentiel planaire $\mathcal{A}_d$ est défini par un 5-ulpet $s = (x, y, \theta, \omega_L, \omega_R)$ où (x, y) représentent les coordonnées du centre des roues, θ est la direction principale de $\mathcal{A}_d$ et ω_L (resp. ω_R) est la vitesse angulaire de la roue gauche (resp. droite). Un contrôle de $\mathcal{A}_d$ est défini par le couple $u = (\eta_L, \eta_R)$ où η_L (resp. η_R) est l'accélération angulaire de la roue gauche (resp. droite).

Connaissant les vitesses des roues, les vitesses linéaire et angulaire principales de $\mathcal{A}_d$ sont données par :

$$v(t) = \frac{\mathbf{r} * (\omega_R(t) + \omega_L(t))}{2} \tag{4.27}$$

$$\omega(t) = \frac{\mathbf{r} * (\omega_R(t) - \omega_L(t))}{2 * \mathbf{b}} \tag{4.28}$$

où $\mathbf{b}$ est la distance entre le centre du robot différentiel et les roues, et $\mathbf{r}$ est le rayon de ses roues.

Le mouvement de $\mathcal{A}_d$ est régi par les équations différentielles suivantes :

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{\omega}_L \\ \dot{\omega}_R \end{pmatrix} = \begin{pmatrix} v\cos(\theta) \\ v\sin(\theta) \\ \omega \\ \eta_L \\ \eta_R \end{pmatrix} \tag{4.29}$$

Les contraintes sur le robot différentiel suivantes sont considérées :

$$(\omega_L, \omega_R) \in [-\omega_{max}; \omega_{max}]^2, (\eta_L, \eta_R) \in [-\eta_{max}; \eta_{max}]^2 \tag{4.30}$$

4.6.2 Représentation paramétrique du contrôle

Afin de générer une trajectoire pour le système différentiel $\mathcal{A}_d$, nous utilisons une représentation paramétrique polynomiale du contrôle (η_L, η_R) définie comme suit :

$$\eta_L(t) = 2\alpha_1 + 4\beta_1 t + 6\gamma_1 t^2, \forall t \in [0; t_f] \tag{4.31}$$

$$\eta_R(t) = 2\alpha_2 + 4\beta_2 t + 6\gamma_2 t^2, \forall t \in [0; t_f] \tag{4.32}$$

A partir des paramètres $p = (\alpha_1, \beta_1, \gamma_1, \alpha_2, \beta_2, \gamma_2)$ définissant le contrôle, l'état final s_f atteint par le système en provenance de s_0 est obtenu en intégrant l'Eq. (4.29).

4.6.3 Choix des paramètres initiaux

Une base de paramètres initiaux a été calculée afin de déterminer les paramètres (p, t_f) utilisés en entrée de la méthode variationnelle. Rappelons qu'une telle base sert à stocker les paramètres (p, t_f) requis afin de générer les trajectoires liant chaque couple d'états-temps $\{(s_0, 0); (s_g, t_g)\}$ sélectionnés sur un échantillonnage grossier de l'espace d'états-temps (application $\mathbf{S}^2 \times \mathbf{T}^2 \longrightarrow \mathbf{P}$). En considérant les invariances par translations et rotations sur l'espace et le temps entre les états-temps initial et final, la dimension de l'espace d'entrée de cette base de paramètres a été réduite à 8 (Δx, Δy, $\Delta\theta$, ω_L^0, ω_R^0, ω_L^g, ω_R^g, Δy).

L'échantillonnage courant de cette base nous a permis de stocker environ 150000 échantillons de paires d'états-temps initial/but ainsi que les paramètres associés permettant de générer une trajectoire entre eux. Une telle base de données prend une dizaine de méga-octets en mémoire, mais permet d'améliorer considérablement le nombre d'étapes requises pour converger vers une solution (voir Section 4.7.1). Un exemple simple d'une telle base de paramètres est illustré par la Fig. 4.11.

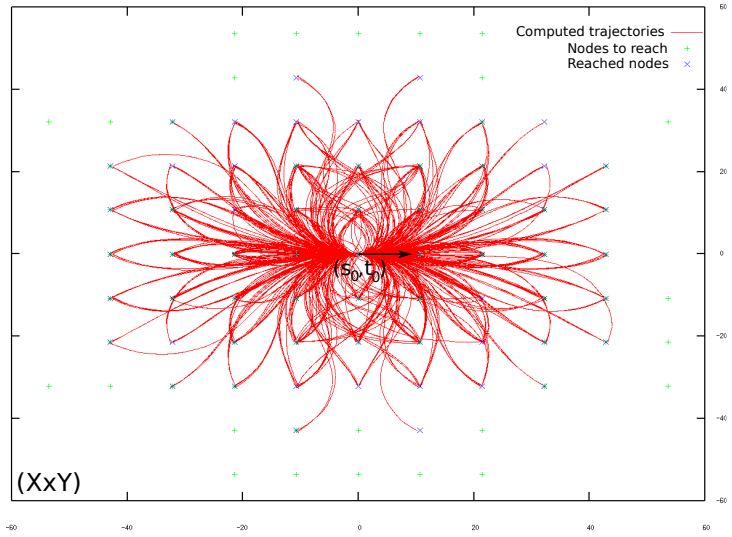
Etant donnés ces paramètres initiaux définissant une trajectoire partant d'un état-temps $(s_0, 0)$, nous présentons dans la section suivante la méthode utilisée pour s'assurer de l'admissibilité de cette trajectoire.

4.6.4 Détermination d'une trajectoire faisable

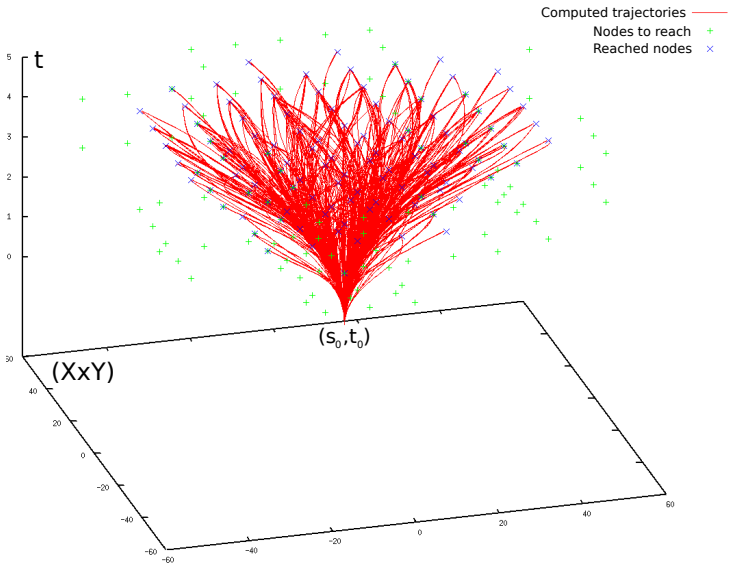
A partir des paramètres d'entrée p, une trajectoire de contrôles faisable $\tilde{u}^*(p, t)$ et la trajectoire d'états $\tilde{s}^*(p, t)$ qui en résulte, sont déterminées. Dans le cas d'un robot différentiel, elles sont définies comme suit :

$$\tilde{u}^*(p,t) = \begin{pmatrix} \eta_L^* \\ \eta_R^* \end{pmatrix}(p,t), \quad \tilde{s}^*(p,t) = \begin{pmatrix} x^* \\ y^* \\ \theta^* \\ \omega_L^* \\ \omega_R^* \end{pmatrix}(p,t) \tag{4.33}$$

Pour calculer ces différentes fonctions, nous intégrons les équations (4.29) à partir des paramètres d'entrée p, puis dans le cas où celles-ci ne respecteraient pas les contraintes sur le mouvement, elles sont modifiées afin d'obtenir une trajectoire faisable. Les différentes étapes requises pour calculer une trajectoire faisable dans le cas du robot différentiel sont décrites ci-dessous. Nous illustrons en Fig. 4.12 un exemple des différentes fonctions d'état calculées afin d'aider le lecteur à suivre les différentes étapes de ce processus.

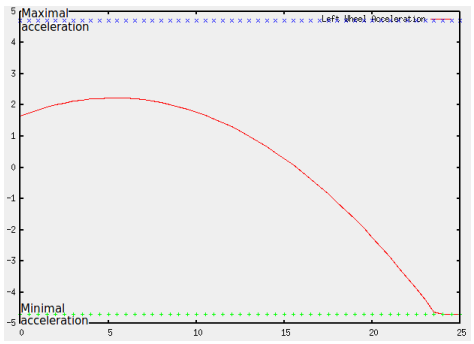


(a) $vue(x \times y)$

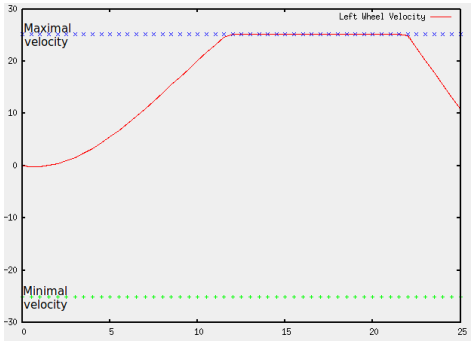


(b) vue $(x \times y \times t)$

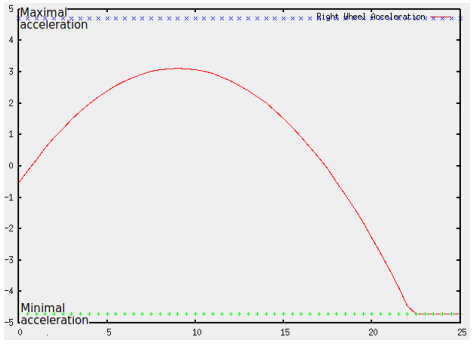
FIGURE 4.11 – Exemple de trajectoires définissant une base de paramètres d'entrée du générateur de trajectoire `Tiji` pour un système différentiel. Cette base permet de sélectionner facilement de bons paramètres permettant, à partir d'un état-temps initial $(s_0, 0)$, de converger rapidement vers un état-temps but (s_g, t_f).



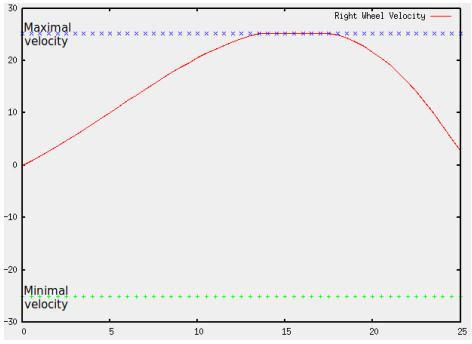
(a) Accélération angulaire de la roue gauche $\eta_L(t)$



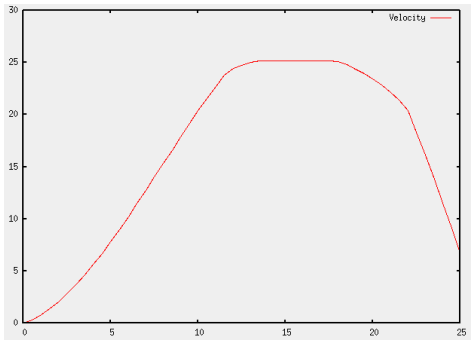
(b) Vitesse angulaire de la roue gauche $\omega_L(t)$



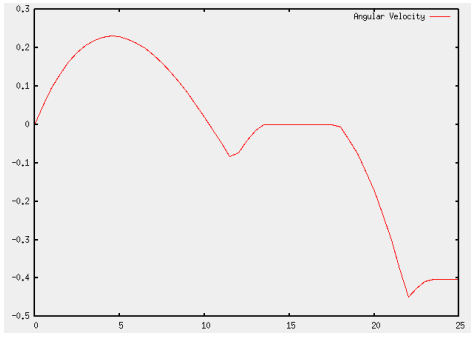
(c) Accélération angulaire de la roue droite $\eta_R(t)$



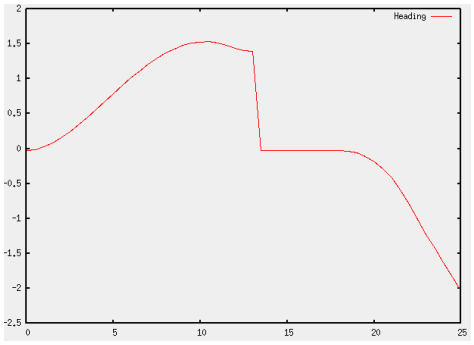
(d) Vitesse angulaire de la roue droite $\omega_R(t)$



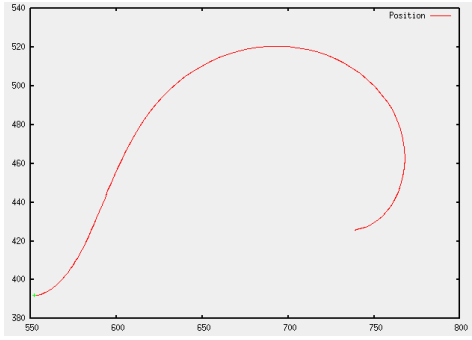
(e) Vitesse linéaire du robot $v(t)$



(f) Vitesse angulaire du robot $\omega(t)$



(g) Orientation du robot $\theta(t)$



(h) Chemin suivi $(x \times y)$

FIGURE 4.12 – Profils des différentes fonctions d'état calculées par `Tiji` à partir d'un ensemble arbitraire de paramètres d'entrée définissant le contrôle du robot différentiel.

Calcul du profil d'accélération de la roue gauche

Nous calculons en premier lieu le profil d'accélération de la roue gauche du robot différentiel grâce à son expression paramétrique donnée en (4.31). Son profil polynomial initial peut néanmoins dépasser les bornes d'accélération du système. Les racines des équations suivantes sont alors calculées :

$$2\alpha_1 + 4\beta_1 t + 6\gamma_1 t^2 - \eta_{max} = 0 \tag{4.34}$$
$$2\alpha_1 + 4\beta_1 t + 6\gamma_1 t^2 + \eta_{max} = 0 \tag{4.35}$$

Résoudre ces équations (par calcul de leur racines) permet alors de trouver les intervalles sur lesquels les bornes d'accélération sont dépassées. Notons $\mathbf{I}_L^{\overline{\eta}^-}$ (resp. $\mathbf{I}_L^{\overline{\eta}^+}$) l'ensemble des intervalles de temps sur lesquels les bornes minimales (resp. maximales) d'accélération sont violées et notons $\mathbf{I}_L^{\eta} = [0; t_f] \backslash \{\mathbf{I}_L^{\overline{\eta}^-}, \mathbf{I}_L^{\overline{\eta}^+}\}$ l'ensemble des intervalles sur lesquels elles sont respectées. Le profil d'accélération polynomial est alors saturé comme suit :

$$\eta_L(p,t) = \begin{cases} -\eta_{max} & \text{si } t \in \mathbf{I}_L^{\overline{\eta}^-} \\ \eta_{max} & \text{si } t \in \mathbf{I}_L^{\overline{\eta}^+} \\ 2\alpha_1 + 4\beta_1 t + 6\gamma_1 t^2 & \text{si } t \in \mathbf{I}_L^{\eta} \end{cases} \tag{4.36}$$

Cette fonction paramétrique par morceaux respecte désormais les contraintes d'accélération des roues du système.

Calcul du profil de vitesse de la roue gauche

Le profil de vitesse de la roue gauche est calculé par intégration du profil d'accélération associé. Il est alors défini comme suit :

$$\begin{aligned} \omega_L(p,t) = \omega_L^0 + \sum_{i \in \mathbf{I}_L^{\eta}} (2\alpha_1 dt_i + 2\beta_1 {dt_i}^2 + 2\gamma_1 {dt_i}^3) \\ -\eta_{max} \sum_{i \in \mathbf{I}_L^{\overline{\eta}^-}} dt_i + \eta_{max} \sum_{i \in \mathbf{I}_L^{\overline{\eta}^+}} dt_i \end{aligned} \tag{4.37}$$

avec dt_I la longueur (durée) de l'intervalle de temps I.

Comme dans le cas de l'accélération, cette fonction peut dépasser les bornes de vitesse maximale du système robotique. L'analyse des variations de cette fonction est alors réalisée afin de trouver les intervalles de temps $\mathbf{I}_L^{\overline{\omega}^-}$ (resp. $\mathbf{I}_L^{\overline{\omega}^+}$) où la borne minimale (resp. maximale) sur la vitesse des roues est dépassée, et $\mathbf{I}_L^{\omega} = [0; t_f] \setminus \{\mathbf{I}_L^{\overline{\omega}^-}, \mathbf{I}_L^{\overline{\omega}^+}\}$ l'ensemble des intervalles de temps sur

lesquels elles sont respectées. Le profil final de la vitesse de la roue gauche ω_L^* (cf fig. 4.12(b)) est alors défini par :

$$\omega_L^*(p,t) = \begin{cases} \omega_L^0 + \sum_{i \in \mathbf{I}_L^\eta} (2\alpha_1 dt_i + 2\beta_1 {dt_i}^2 + 2\gamma_1 {dt_i}^3) & \\ -\eta_{max} \sum_{i \in \mathbf{I}_L^{\overline{\eta}^-}} dt_i & \\ +\eta_{max} \sum_{i \in \mathbf{I}_L^{\overline{\eta}^+}} dt_i & \text{si } t \in \mathbf{I}_L^\omega \\ -v_{max} & \text{si } t \in \mathbf{I}_L^{\overline{\omega}^-} \\ v_{max} & \text{si } t \in \mathbf{I}_L^{\overline{\omega}^+} \end{cases} \tag{4.38}$$

De même, le profil d'accélération de la roue gauche η_L^* garantissant le respect de l'ensemble des contraintes du mouvement du système est calculé par dérivation de ω_L^* par rapport au temps. Son expression est donnée par :

$$\eta_L^*(p,t) = \begin{cases} 0 & \text{si } t \in \mathbf{I}_L^{\overline{\omega}^-} \cup \mathbf{I}_L^{\overline{\omega}^+} \\ -\eta_{max} & \text{si } t \in \mathbf{I}_L^{\overline{\eta}^-} \setminus \{\mathbf{I}_L^{\overline{\omega}^-} \cup \mathbf{I}_L^{\overline{\omega}^+}\} \\ \eta_{max} & \text{si } t \in \mathbf{I}_L^{\overline{\eta}^+} \setminus \{\mathbf{I}_L^{\overline{\omega}^-} \cup \mathbf{I}_L^{\overline{\omega}^+}\} \\ 2\alpha_1 + 4\beta_1 t + 6\gamma_1 t^2 & \text{si } t \in \mathbf{I}_L^{\eta\omega} \end{cases} \tag{4.39}$$

où $\mathbf{I}_L^{\eta\omega}$ est l'ensemble des intervalles de temps où ni les contraintes sur la vitesse ni les contraintes sur l'accélération n'ont été dépassées. L'Eq. (4.39) détermine alors le profil du contrôle en accélération final de l'étape courante de convergence de la méthode variationnelle(cf Fig. (4.12(a))). Notez que cette fonction doit correspondre à l'Eq. (4.36) dans le cas où ni les bornes de vitesse ni les bornes d'accélération n'ont été dépassées.

Le calcul de l'accélération η_R^* et de la vitesse ω_R^* de la roue droite sont complètement similaires. Nous ne les développerons donc pas ici, néanmoins les Fig. 4.12(c) et 4.12(d) donnent un aperçu de leurs profils.

Calcul des vitesses linéaire et angulaire du robot, de son orientation et de sa position

Les fonctions d'état restantes à calculer, à savoir la vitesse angulaire du robot ω^*, sa vitesse linéaire v^*, son orientation θ^*, et enfin sa position x^* en X et y^* en Y, ne nécessitent plus aucun processus additionnel de saturation. Elles sont donc calculées à partir des profils de vitesse et accélération saturés des roues grâce aux équations standard définies en (4.29).

Les figures 4.12(e), 4.12(f), 4.12(g) et 4.12(h) illustrent respectivement ces différents profils pour un ensemble de paramètres d'entrée arbitraires. Dans tous les cas, les trajectoires de contrôle $\tilde{u}^*(p,t)$ et d'état $\tilde{s}^*(p,t)$ sont faisables grâce à leur saturation.

4.6.5 Calcul d'une correction sur les paramètres

Une fois toutes les fonctions d'état calculées, il est possible d'évaluer l'état final s_f de la trajectoire. S'il est suffisamment proche de l'état but, la génération de trajectoire est un succès. Dans le cas contraire, une nouvelle correction sera appliquée sur les paramètres d'entrée p. Celle-ci est calculée directement à partir de l'Eq. 4.26. Après application de cette correction, tout le processus de détermination d'une trajectoire admissible par saturation des profils de contrôle et d'état doit être réitéré jusqu'à ce que l'état final s_f de la trajectoire ait enfin convergé.

Etant désormais dotés de tous les outils nécessaires pour générer une trajectoire pour un système différentiel avec `Tiji`, nous présentons dans la section suivante quelques résultats en simulation obtenus pour ce système.

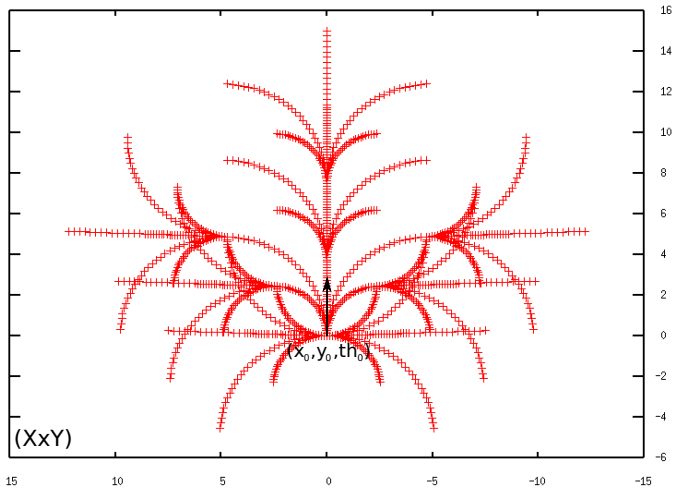
4.7 Résultats en simulation

Le générateur de trajectoires `Tiji` est destiné à calculer une trajectoire entre deux états données à des temps fixés ou contraints. Il doit alors converger vers une solution lorsque l'état but est atteignable et trouver une trajectoire admissible dont l'état terminal s'arrête "aussi près que possible" de l'état but dans le cas contraire.

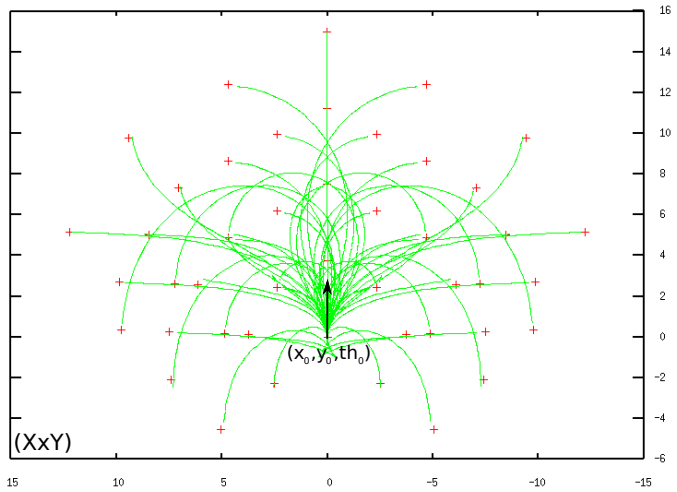
`Tiji` a été implémenté en C++ et testé sur un ordinateur de bureau (Core-i7@3.4GHz, 6GB RAM, SE : Linux). Nous présentons ici différents scénarios pour le système différentiel présenté en Section 4.6 caractérisant des états-temps but atteignables et non atteignables afin de valider les différentes propriétés de `Tiji`. Le nombre maximum d'itérations de la méthode variationnelle a été heuristiquement fixé à vingt. Après vingt étapes de convergence, `Tiji` considère que l'état but n'est pas atteignable sous la contrainte de temps final, et retourne donc la trajectoire, calculée au cours des différentes itérations de l'algorithme, dont l'état terminal est le plus proche de l'état but.

4.7.1 Cas 1 : Etats-temps buts atteignables

Afin d'évaluer les performances du générateur de trajectoires `Tiji`, des arbres d'échantillonnage (*cf.* Fig. 4.13) ont été implémentés. Les arbres d'échantillonnage ont pour but de déterminer la topologie de l'espace atteignable pour un système donné : A partir d'un état initial s_0, d'un temps fixe t_f, d'un nombre h d'intervalles de temps (hauteur de l'arbre) et d'un nombre b de contrôles possibles (branchement de l'arbre), chaque trajectoire



(a) Arbre d'échantillonnage (vue $x \times y$)



(b) Trajectoires générées (vue $x \times y$)

FIGURE 4.13 – Cas des états-temps finaux atteignables : Un arbre d'échantillonnage décrivant un sous-ensemble des trajectoires faisables pour un robot différentiel est calculé. `Tiji` est évalué en essayant d'atteindre les états-temps terminaux de chacune des branches de l'arbre (représentés par les croix rouges sur la Fig. 4.13(b)).)

de l'arbre d'échantillonnage $\mathbf{stree}(s_0, t_f, b, h)$ est calculée en appliquant h contrôle admissibles constant successifs de durée $\frac{t_f}{h}$ en entrée du robot à partir de l'état s_0. Tous les états terminaux (feuilles) de l'arbre d'échantillonnage sont donc atteignables au temps t_f. Chaque contrôle constant est sélectionné parmi un échantillonnage régulier des contrôles disponibles à chaque pas de

Système Robotique Différentiel	Sans base de Paramètres initiaux	Avec base de Paramètres initiaux
nombre d'états-temps évalués	327680	327680
taux de convergence (%)	94.58	99.80
nombre moyen d'étapes requises (en cas de succès)	6.38	3.23
temps moyen requis par trajectoire (ms)	4.78	2.84

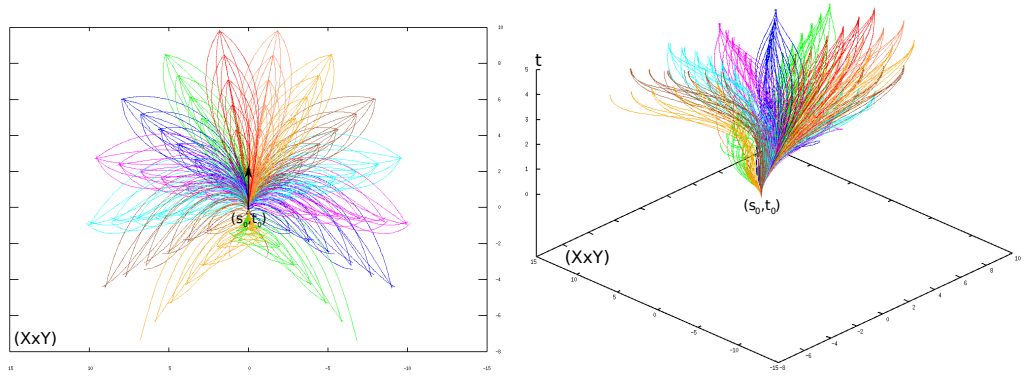
TABLE 4.1 – Performances du générateur de trajectoire `Tiji` pour un système différentiel dans le cas où tous les états-temps buts sont atteignables.

temps. Le nombre de trajectoires représentées par l'arbre d'échantillonnage est donné par b^h. Une hauteur et un branchement raisonnables (entre 6 et 10) suffisent à caractériser globalement tout l'espace atteignable à partir de s_0 au temps t_f.

Un arbre d'échantillonnage a été calculé pour le robot différentiel décrit en Section 4.6 en considérant diverses vitesses initiales de chaque roue. Le générateur de trajectoire `Tiji` a donc été utilisé afin d'essayer d'atteindre chaque état-temps terminal de l'arbre. La contrainte sur le temps final a été ici réduite à la valeur unique t_f. La Fig. 4.13 illustre un arbre d'échantillonnage simple (hauteur=3, branchement=3) et les trajectoires générées correspondantes pour le système différentiel.

Du point de vue de ses performances, l'algorithme de génération de trajectoires `Tiji` dépend principalement du nombre d'étapes (applications de corrections successives) requises pour converger ver le but. La table 4.1 résume le nombre moyen d'étapes de convergences requises et le temps de calcul nécessaire par trajectoire pour le système robotique différentiel. Deux cas ont été considérés suivant la disponibilité d'une base de paramètres initiaux ou non. Une telle base permet d'améliorer le taux de convergence d'une part, et réduit le nombre moyen d'étapes de convergence nécessaires et donc en conséquence le temps de calcul correspondant.

La Fig. 4.14 illustre un exemple de trajectoires générées pour le robot différentiel essayant d'atteindre diverses positions autour de lui. La contrainte sur le temps final ayant été fortement relâchée ici, le système rejoint facilement chaque état but en choisissant le temps le plus adapté dans l'intervalle de temps donné.



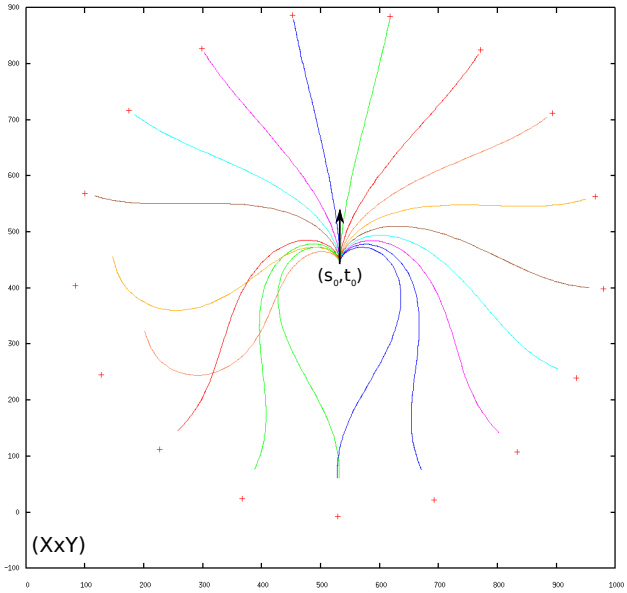
(a) vue $x \times y$ (b) vue $x \times y \times t$

FIGURE 4.14 – Trajectoires générées pour un système différentiel essayant d'atteindre diverses positions autour de lui avec une contrainte sur le temps relâchée ($t_f \in [2; 5]$).

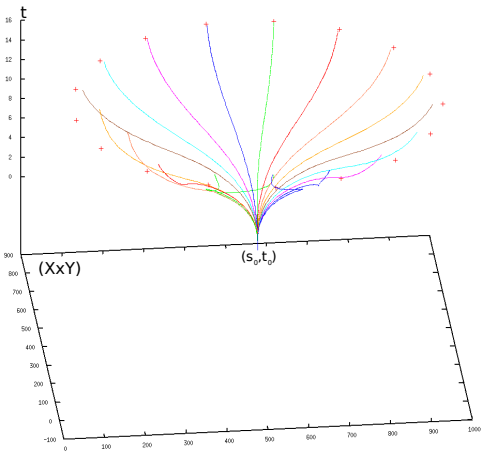
4.7.2 Cas 2 : Etats-temps buts non atteignables

`Tiji` étant destiné à être capable de générer des trajectoires admissibles s'arrêtant "aussi près que possible" de l'état-temps but lorsque ce dernier n'est pas atteignable, nous proposons ici quelques résultats permettant de mettre en valeur la pertinence des trajectoires alternatives calculées. La Fig. 4.15 présente un ensemble de trajectoires calculées pour atteindre divers états du robot différentiel à un temps fixe t_f donné. Le système partant d'un état-temps initial $(s_0, 0)$, les différents états-temps but ont été choisi sur un arbre d'échantillonnage **stree**(s_0, t_{tree}, b, h) partant de cet état, mais tel que t_{tree} soit nettement supérieur à t_f afin de garantir que les états but ne soient pas atteignables au temps t_f.

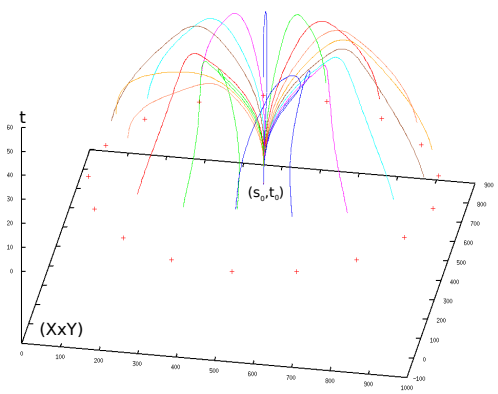
Afin de conclure sur la pertinence des trajectoires alternatives calculées, nous calculons un coefficient d'évaluation μ pour chaque trajectoire partant de $(s_0, 0)$ et n'atteignant pas son but (s_g, t_f). Ce coefficient est destiné à comparer la distance $d(s_g, \mathcal{R}(s_0, t_f))$ entre l'état but s_g et l'état le plus proche de l'espace atteignable $\mathcal{R}(s_0, t_f)$ à la distance $\|s_g - s_f\|$. Le calcul d'espace atteignable étant une tâche difficile pour des systèmes dynamiques complexes. Cependant, si les états-temps terminaux de l'arbre d'échantillonnage **stree**(s_0, t_f, b, h) ne représentent qu'un sous-ensemble de l'espace atteignable $\mathcal{R}(s_0, t_f)$, nous pouvons néanmoins approximer l'état-temps le plus proche de l'espace atteignable par l'état-temps le plus proche de l'arbre **stree**(s_0, t_f, b, h) dans le cas où celui-ci dispose d'une hauteur et d'un branchement suffisamment élevés.



(a) vue $x \times y$



(b) vue $x \times y \times t$



(c) vue $x \times y \times v$

FIGURE 4.15 – Trajectoires calculées pour atteindre divers états autour du robot différentiel à un temps fixé. Ses bornes d'accélération et de vitesse l'empêche d'atteindre tous les couples de position-vitesse buts, néanmoins des trajectoires alternatives garantissant l'ensemble des contraintes sur le mouvement et s'arrêtant "aussi près que possible" des états buts résultent de la méthode variationnelle de génération de trajectoire proposée.

	Système Robotique Différentiel
nombre d'états-temps évalués	10000
temps moyen requis par trajectoire (ms)	13.56
coefficient d'évaluation μ moyen	1.14

TABLE 4.2 – Performances du générateur de trajectoire `Tiji` pour un robot différentiel dans le cas où les états-temps but ne sont pas atteignables.

Nous calculons alors le coefficient d'évaluation μ comme suit :

$$\mu = \frac{\|s_g - s_f\|}{\|s_g - \mathbf{stree}(s_0, t_f, b, h)\|} \tag{4.40}$$

où $\|s_g - \mathbf{stree}(s_0, t_f, b, h)\|$ est la distance entre s_g et l'état terminal (défini au temps t_f) le plus proche de l'arbre d'échantillonnage **stree**(s_0,t_f,b,h). Notez qu'une métrique sur l'espace d'état est nécessaire afin de calculer la distance entre ces états. Le coefficient μ tend alors vers 1 lorsque l'état terminal s_f de la trajectoire générée correspond exactement à l'état de l'arbre le plus proche de l'état but (via la métrique utilisée).

Les performances obtenues lorsque les états-temps but ne sont pas atteignables sont résumées en table 4.2. Les temps de calcul pour les trajectoires alternatives restent suffisamment faibles, tandis que l'erreur moyenne entre les états terminaux des trajectoires calculées et les états les plus proches du but qu'il aurait été possible d'atteindre s'élèvent en moyenne à 14% de la distance minimale entre ces états.

4.8 Autres cas d'étude

Afin de montrer la diversité des systèmes auxquels notre générateur de trajectoires `Tiji` peut s'appliquer, nous en présentons deux supplémentaires ci-dessous. Nos choix se sont attardé sur un véhicule de type voiture, intéressant pour les contraintes non-holonomes qui le caractérisent et sur un système de type vaisseau spatial simplifié, évoluant dans le plan 2D. Nous décrivons rapidement ceux deux systèmes en Sections 4.8.1 et 4.8.2 avant de présenter, comme pour le système différentiel des résultats dans les cas où

les états-temps finaux sont atteignables et non atteignables (Sections 4.8.3 et 4.8.4).

4.8.1 Véhicule de type voiture

Le système de type voiture ayant également déjà été présenté lors du chapitre en Section 3.4.4, nous en rappelons ici brièvement les principales caractéristiques.

Un état d'un système de type voiture $\mathcal{A}_c$ est défini par un 5-uplet $s = (x, y, \theta, \phi, v)$ où (x, y) sont les coordonnés de la roue arrière du véhicule (ou du centre des roues arrières), θ est son orientation principale, ϕ est l'orientation de ses roues avant (angle de braquage), et v est la vitesse linéaire de sa roue arrières. Un contrôle de $\mathcal{A}_c$ est défini par le couple $u = (a, \zeta)$ où a est l'accélération linéaire de sa roue arrière, et ζ la vitesse de braquage. Le mouvement de $\mathcal{A}_c$ est régi par les équations différentielles suivantes :

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{\phi} \\ \dot{v} \end{pmatrix} = \begin{pmatrix} v\cos(\theta) \\ v\sin(\theta) \\ v\tan(\phi)/L \\ \zeta \\ a \end{pmatrix} \tag{4.41}$$

où L est la distance entre la roue avant et la roue arrière du système $\mathcal{A}_c$. Les contraintes additionnelles suivantes sont considérées :

$$v \in [0, v_{\max}], |\phi| \leq \phi_{\max}, |a| \leq a_{\max} \text{ and } |\zeta| \leq \zeta_{\max} \tag{4.42}$$

Notez qu'une étude complète de l'application de `Tiji` à un véhicule de type voiture a été proposée lors de l'une de nos précédentes publications ([DFMG09]). Le lecteur peut s'y référer pour plus de détails à ce sujet.

4.8.2 Vaisseau spatial

Le dernier système présenté est un modèle de vaisseau spatial simplifié $\mathcal{A}_s$, évoluant dans le plan 2D (x, y). L'état de $\mathcal{A}_s$ est défini par un 6-uplet $s = (x, y, \theta, v_x, v_y, \omega)$ où (x, y) sont les coordonnées du centre de rotation du vaisseau spatial, θ correspond à son orientation principale, v_x (resp. v_y) est sa vitesse linéaire le long de l'axe X (resp. axe Y) du plan 2D, et enfin ω est sa vitesse angulaire. Un contrôle de $\mathcal{A}_s$ est défini par le couple $u = (a, \eta)$, où a est l'accélération linéaire appliqué le long de l'orientation principale de $\mathcal{A}_s$, et η est son accélération angulaire. Le mouvement de $\mathcal{A}_s$ est alors régi par le système différentiel suivant :

Système robotique	Véhicule de type voiture	Vaisseau spatial 2D
nombre d'états-temps évalués	393216	65536
taux de convergence (%)	96.52	98.88
nombre moyen d'étapes requises (en cas de succès)	7.08	8.92
temps moyen par trajectoire (ms)	5.12	37.50

TABLE 4.3 – Performances du générateur de trajectoire `Tiji` pour un véhicule de type voiture et un vaisseau spatial, dans le cas où tous les états-temps buts sont atteignables.

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{v_x} \\ \dot{v_y} \\ \dot{\omega} \end{pmatrix} = \begin{pmatrix} v_x \\ v_y \\ \omega \\ a\cos(\theta) \\ a\sin(\theta) \\ \eta \end{pmatrix} \tag{4.43}$$

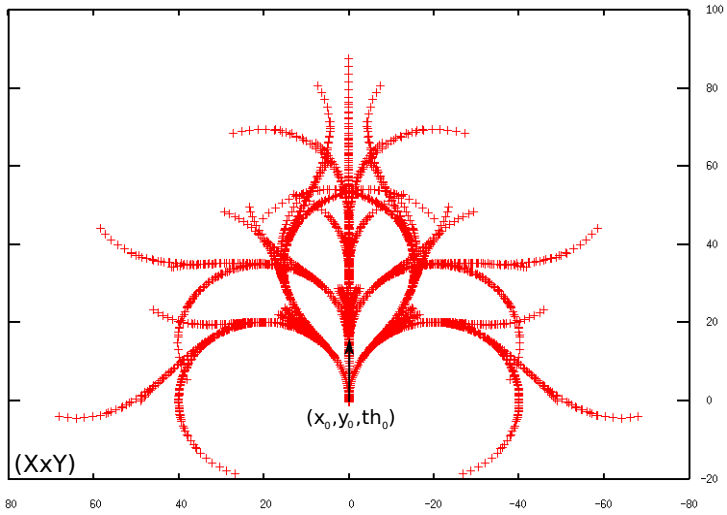
Et les contraintes sur son mouvement suivantes sont considérées :

$$\begin{array}{l} (v_x, v_y) \in [-v_{max}; v_{max}]^2,\ \omega \in [-\omega_{max}; \omega_{max}], \\ a \in [-a_{max}; a_{max}],\ \eta \in [-\eta_{max}; \eta_{max}] \end{array} \tag{4.44}$$

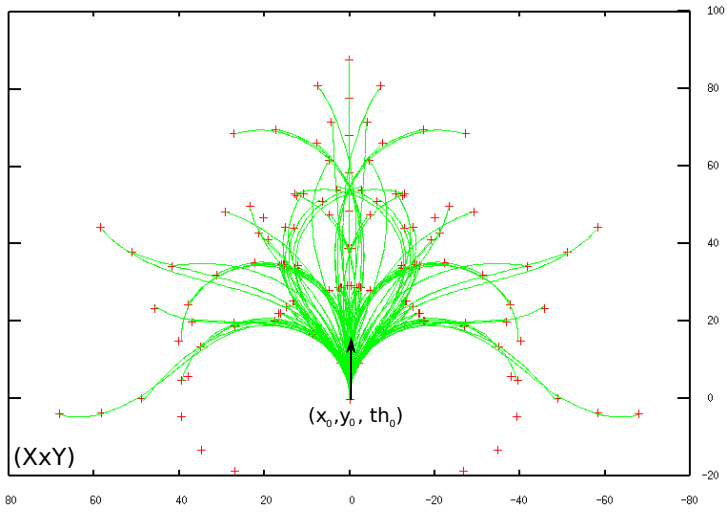
4.8.3 Cas d'étude 1 : Etats-temps but atteignables

Comme dans le cas du robot différentiel, des arbres d'échantillonage ont été implémentés afin de pouvoir évaluer le taux de convergence du générateur de trajectoires `Tiji` dans le cas où les états-temps but sont atteignables. Les trajectoires générées pour la voiture et le vaisseau spatial sont respectivement illustrées par les Figs. 4.16 et 4.17.

La table 4.3 présente les performances associées à ces résultats. Les taux de convergence du générateur de trajectoire pour ces deux systèmes semblent également satisfaisant. Quant aux temps de calculs, ils sont suffisants pour utiliser cette méthode en temps réel, même si la génération de trajectoire pour le vaisseau spatial est un peu plus lente de par la complexité de son système.



(a) Arbre d'échantillonnage (vue $x \times y$)

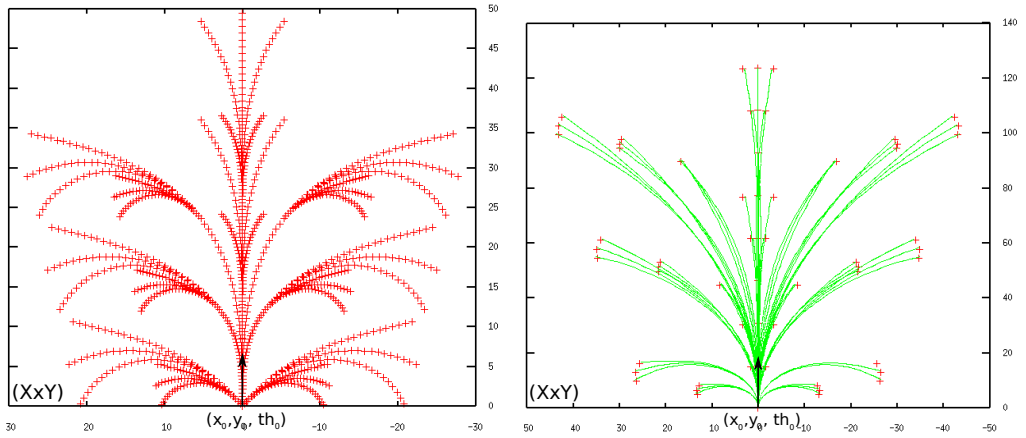


(b) Trajectoires générées (vue $x \times y$)

FIGURE 4.16 – Arbre d'échantillonnage décrivant un sous-ensemble des trajectoires admissibles d'un véhicule de type voiture et trajectoires calculées par `Tiji` pour atteindre chacun des états terminaux de l'arbre (représentés par les croix rouges en Fig. 4.16(b)).

4.8.4 Cas d'étude 2 : Etats-temps buts non atteignables

Les Fig. 4.18 et 4.19 présentent des résultats additionnels des trajectoires générées pour une voiture et un vaisseau spatial dans le cas où les états-



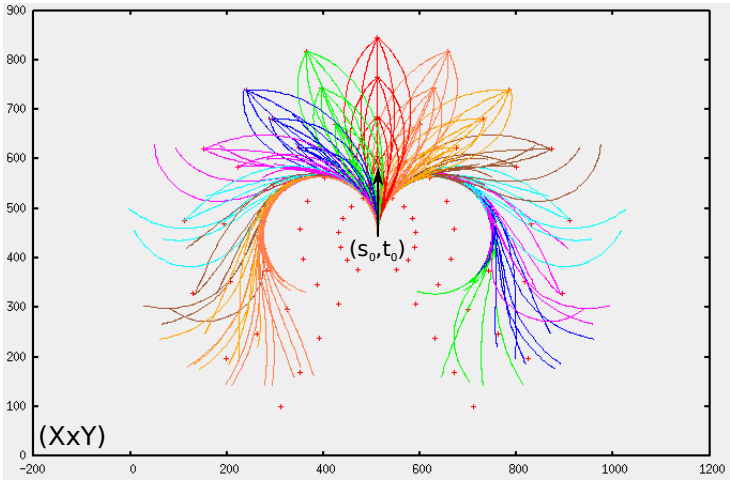
(a) Arbre d'échantillonnage (vue $x \times y$) (b) Trajectoires générées (vue $x \times y$)

FIGURE 4.17 – Arbre d'échantillonnage décrivant un sous-ensemble des trajectoires admissibles pour le modèle simplifié d'un vaisseau spatial évoluant dans le plan 2D, et trajectoires générées par `Tiji` pour atteindre chacun des états terminaux de l'arbre (représentés par les croix rouges en Fig. 4.17(b)).

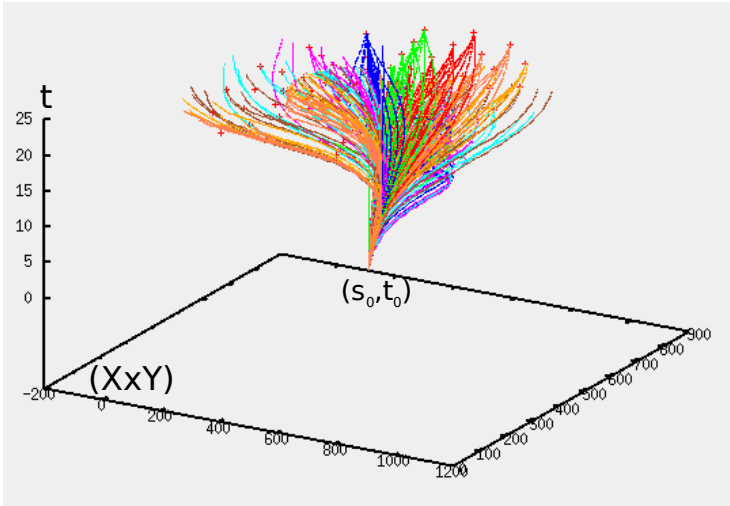
Système robotique	Véhicule de type voiture	Vaisseau spatial 2D
nombre d'états-temps évalués	10000	10000
temps moyen par trajectoire (ms)	10.41	43.94
coefficient d'évaluation μ moyen	1.15	1.51

TABLE 4.4 – Performances du générateur de trajectoire `Tiji` pour des systèmes de types voiture et vaisseau spatial dans le cas où les états-temps but ne sont pas atteignables.

temps but ne sont pas atteignables. Même si les contraintes sur le mouvement combinées aux contraintes sur le temps final empêchent les différents systèmes d'atteindre leur but, les trajectoires alternatives calculées permettent de s'arrêter à proximité de ces états buts. La table 4.4 résume les performances de `Tiji` pour ces deux systèmes dans le cas d'états-temps but non atteignables.



(a) vue $x \times y$

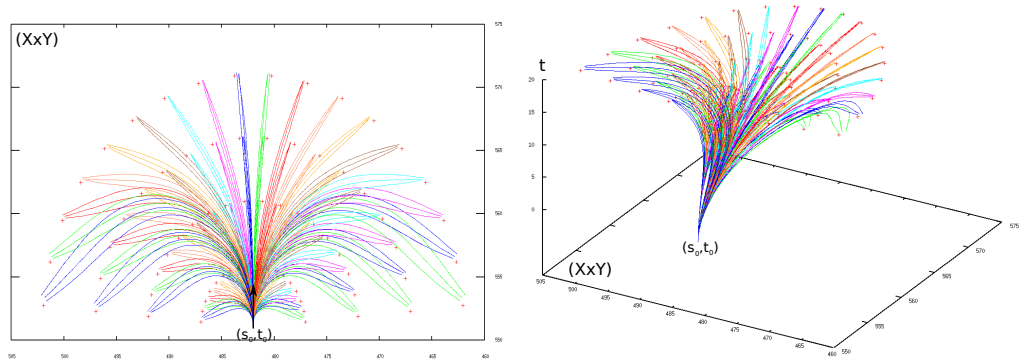


(b) vue $x \times y \times t$

FIGURE 4.18 – Trajectoire calculées pour un véhicule de type voiture dans le but d'atteindre divers états du système (représentés par des croix) à un temps final contraint ($t_f \in [15; 25]$). Certains états ne sont pas atteignables à cause des contraintes de vitesse, accélération, angle de braquage et vitesse de braquage définies par le système.

4.9 Conclusion et Perspectives

Ce chapitre a donc présenté le générateur de trajectoire `Tiji` utilisé pour déterminer des trajectoires entre deux états-temps pour des systèmes dynamiques complexes. La principale qualité de `Tiji` consiste à intégrer une



(a) vue $x \times y$ (b) vue $x \times y \times t$

FIGURE 4.19 – Trajectoires calculées dans le but d'atteindre divers états (croix) à un temps fixé autour d'un robot de type vaisseau spatial 2D.

contrainte sur le temps final i.e. des bornes sur le temps auquel doit être atteint l'état but. La prise en compte de cette contrainte soulève néanmoins la question de *décidabilité* du problème : lorsque le temps est contraint, il n'est en aucun cas certain que l'état but soit atteignable sous cette contrainte de temps. Après avoir mis en évidence cette difficulté, nous avons proposé le calcul de trajectoires approchées permettant de respecter toutes les *contraintes sur le mouvement* du système robotique et de s'arrêter "aussi près que possible" de l'état but lorsque celui-ci n'est pas atteignable.

`Tiji` est basé sur une méthode variationnelle de génération de trajectoire : une représentation paramétrique du contrôle d'entrée du système est proposée. A partir de paramètres initiaux, une première trajectoire n'atteignant pas forcément le but est alors calculée. En modifiant itérativement ces paramètres la trajectoire résultante peut converger vers le but. Afin de s'assurer de la validité de la trajectoire, ses différents profils sont saturés lorsqu'ils dépassent les contraintes sur le mouvement du robot. Lorsque l'état final n'est pas atteignable, l'état terminal de la trajectoire converge alors vers un état s'en rapprochant, tout en garantissant la satisfaction de l'ensemble des contraintes du système.

`Tiji` a donc été intégré à la méthode de déformation de trajectoire `Teddy` afin de de maintenir la faisabilité et la convergence vers le but de la trajectoire déformée (*cf.* Section 3.3.6). `Teddy` n'est cependant qu'un exemple d'application de notre générateur de trajectoire, il peut néanmoins être intégré à bien d'autres méthodes de navigation en environnement dynamique nécessitant à un robot d'arriver à une position donnée à un temps fixé ou borné. Il a par exemple été intégré lors de l'un de nos travaux ([DF10]) à un module de suivi de trajectoire afin de déterminer le contrôle à appliquer en entrée d'une chaise

roulante automatisée (*cf.* chapitre 5). De nombreuses méthodes actuelles de navigation essayent de prendre en compte la dynamique de l'environnement, et par conséquence de planifier des trajectoires à partir de prévisions sur le mouvement des obstacles mobiles. Afin de planifier une trajectoire au milieu d'obstacles obstruant le passage du robot sur des plages de temps bornés, une telle méthode de génération de trajectoires en temps contraint peut s'avérer nécessaire. Nous espérons alors qu'elle pourrait inspirer les futurs travaux de navigation en environnement dynamique.

Chapitre 5

Application à une chaise roulante automatisée

5.1 Introduction

Les deux chapitres précédents (3 et 4) ont présenté d'une part une approche de navigation basée sur une déformation de trajectoire, ainsi qu'un générateur de trajectoire local nécessaire à sa mise en oeuvre. Bien que précédemment illustrée par quelques résultats en simulation, nous souhaitions intégrer l'approche de déformation de trajectoire `Teddy` sur un robot réel. Nos travaux ont alors été implémentés sur une chaise roulante automatisée. Ce chapitre présente tout d'abord le système robotique considéré en Section 5.2. Les différentes tâches nécessaires à l'élaboration d'un système autonome sont détaillées en Section 5.3. L'approche de déformation de trajectoire `Teddy` est enfin évaluée dans trois scénarios distincts présentés en Section 5.4.

5.2 Chaise roulante automatisée : le système

La chaise roulante automatisée (*cf.* Fig. 5.1) sur laquelle ont été réalisées nos expérimentations est une plate-forme robotique développée par Blue-Botics lors du projet MOVEMENT [1]. Ce projet était destiné à améliorer la mobilité des personnes âgées ou disposant d'incapacités physiques.

L'architecture matérielle disponible sur la chaise roulante ainsi que l'architecture logicielle mise en place afin de lui permettre de naviguer en toute autonomie sont présentées ci-dessous.

1. Modular Versatile Mobility Enhancement Technology, Projet de recherche n^o 511670

FIGURE 5.1 – Chaise roulante automatisée utilisée lors des expérimentations de l'approche de déformation de trajectoire `Teddy`.

5.2.1 Architecture matérielle

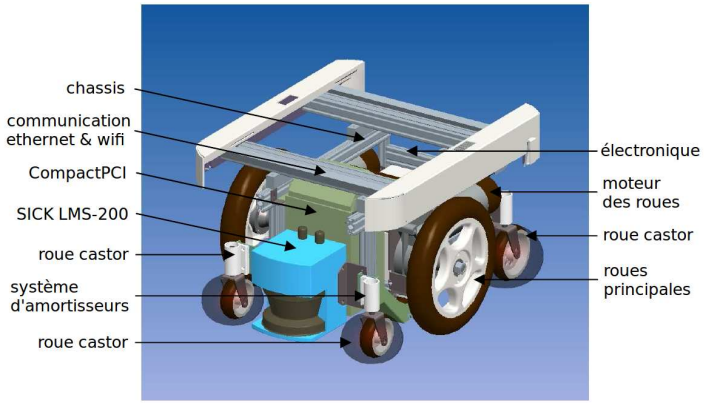


FIGURE 5.2 – Composants principaux du châssis de la chaise roulante permettant au système de percevoir son environnement et de se déplacer.

Les dimensions de la chaise roulante automatisée sont de 0.67m de long sur 0.56m de large et 1.30m de hauteur pour une charge maximale de 150kg. Elle peut se déplacer à une vitesse linéaire de $1.39m/s$, dispose d'une accélération de $1.35m/s^2$, et peut tourner sur elle-même à une vitesse de $1.5rad/s$. Les

roues étant motorisées, elle peut soit être commandée manuellement grâce à un joystick, soit être contrôlée par un PC embarqué de type CompactPCI.

La chaise est équipée d'un capteur LIDAR (LIght Detection And Ranging) de type SICK LMS-200, lui permettant de percevoir son environnement. Ce dernier dispose d'un champ de perception de 180 degrés, d'une résolution de 0.5 degrés (361 rayons sur l'ensemble du champ de perception), et d'une portée de 16m.

Afin de commander la chaise, le PC embarqué est relié à un switch Ethernet, auquel un second ordinateur peut être branché. Il dispose également d'une connexion Wifi 802.11b/g. Un ordinateur portable branché en Ethernet et équipé d'un Core i7 (2.2 GHz, 4 GB RAM, SE : Linux) a donc été utilisé dans le but de planifier un mouvement, d'identifier les obstacles mobiles et de calculer la commande à envoyer à la chaise afin de les éviter. L'architecture logicielle mise en place en conséquence sur cette dernière machine est décrite ci-dessous.

5.2.2 Architecture logicielle

L'architecture logicielle utilisée pour la navigation autonome de la chaise roulante est illustrée par la Fig. 5.3. Elle comprend quatre composants principaux :

- Au plus près du système robotique se trouve un **serveur des données capteurs et actionneurs** nommé *WuweiProxy* destiné à collecter les informations fournies par les capteurs de la chaise roulante et à envoyer les commandes à ses actionneurs. Ce processus, exécuté en boucle, utilise des appels de procédure à distance via le protocole d'échange TCP afin de récupérer à chaque instant les scans lasers fournis par le capteur SICK, ainsi que l'odométrie du robot. De même, la commande est envoyée directement par WuweiProxy au moteur contrôlant les roues de la chaise. Si la chaise ne reçoit aucune commande pendant une durée de une seconde, la connexion est supposée perdue, et le système est directement arrêté afin d'éviter toutes collisions.

- Afin de partager les informations collectées par le robot, un **middleware d'échange d'informations** nommé *Hugr* est directement connecté au serveur d'entrées/sorties. Cet outil a été développé au sein de l'INRIA Rhône-Alpes. Il consiste en une mémoire partagée stockant et distribuant les données utilisées par les différents processus nécessaires à la navigation (planificateur, contrôleur, cartographie, suivi des obstacles mobiles, etc). Chacun de ces processus est alors

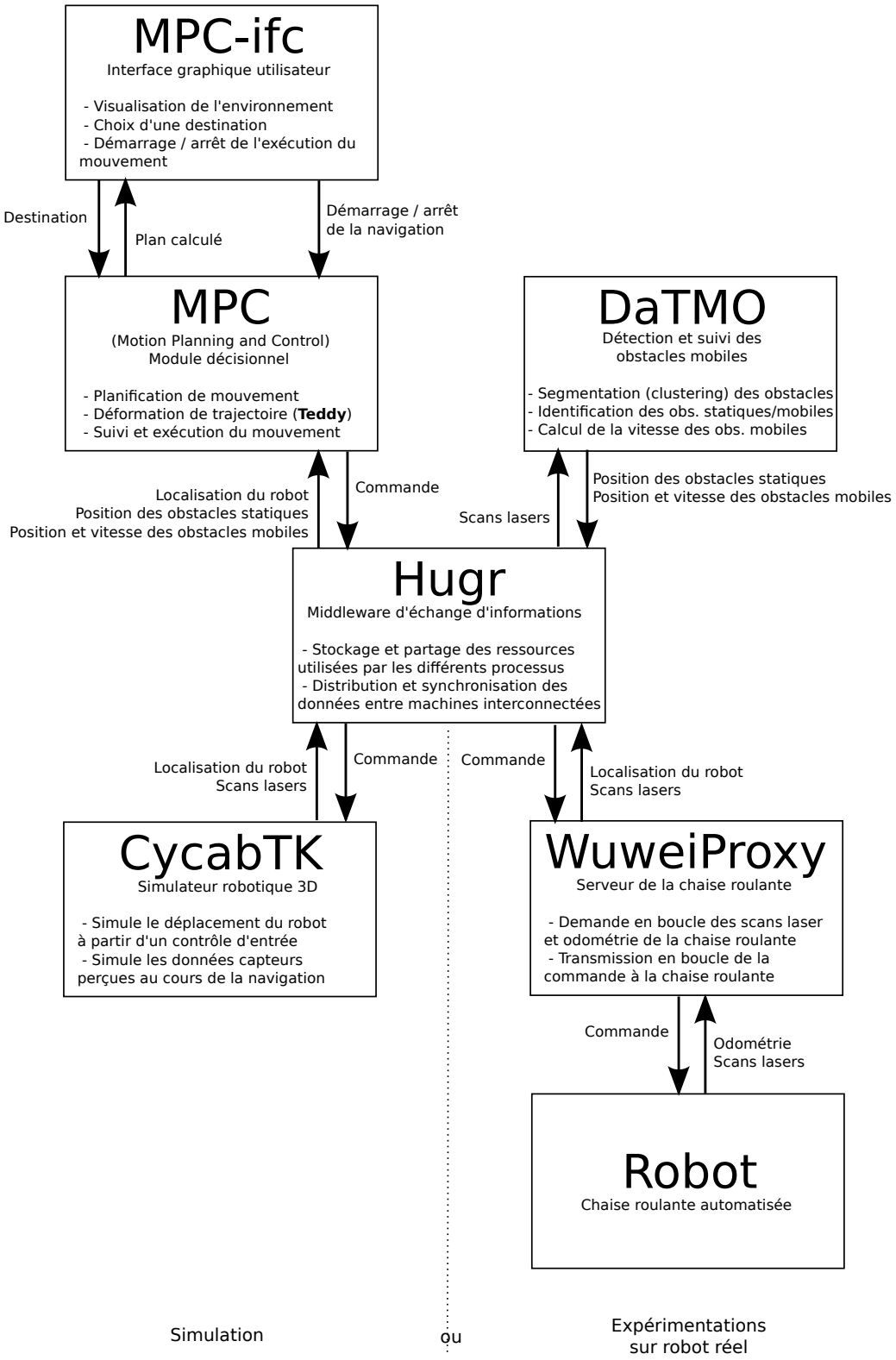


FIGURE 5.3 – Architecture logicielle utilisée pour la navigation autonome de la chaise roulante automatisée.

libre de s'abonner aux données disponibles dont il a besoin, et peut interroger la nouvelle valeur ou la modifier de manière asynchrone.

– La chaise roulante fournissant des scans lasers bruts, il est nécessaire de construire une représentation de l'environnement de plus haut niveau pour faciliter la décision du mouvement à entreprendre pour rejoindre un but. Une module de **détection et suivi des obstacles statiques et mobiles**, développé au sein de l'équipe de recherche ([VA09]) a été utilisé. Celui-ci consiste à segmenter l'environnement en une liste d'obstacles statiques polygonaux caractérisés par leur position, et une liste d'obstacles dynamiques caractérisés par leur position et leur vitesse instantanée, à partir des scans lasers. Ces travaux sont détaillés en Section 5.3.2.

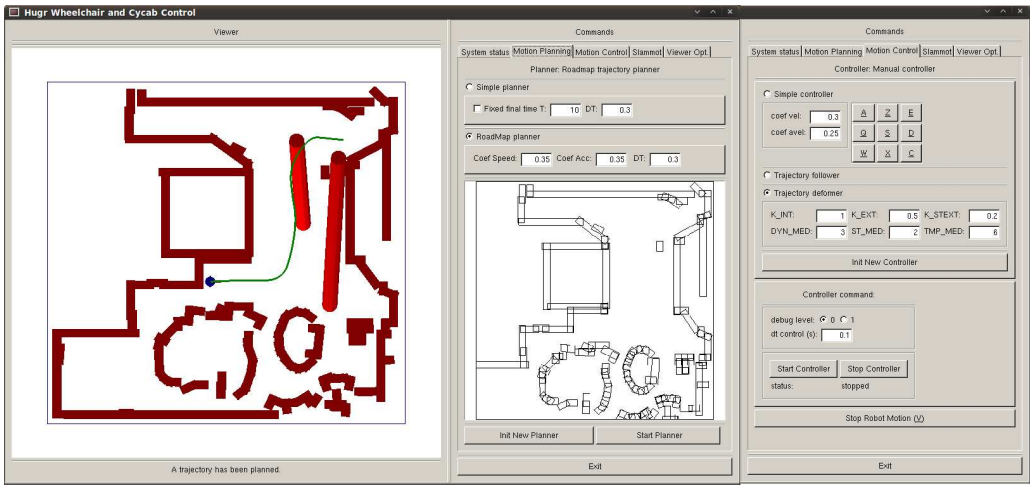
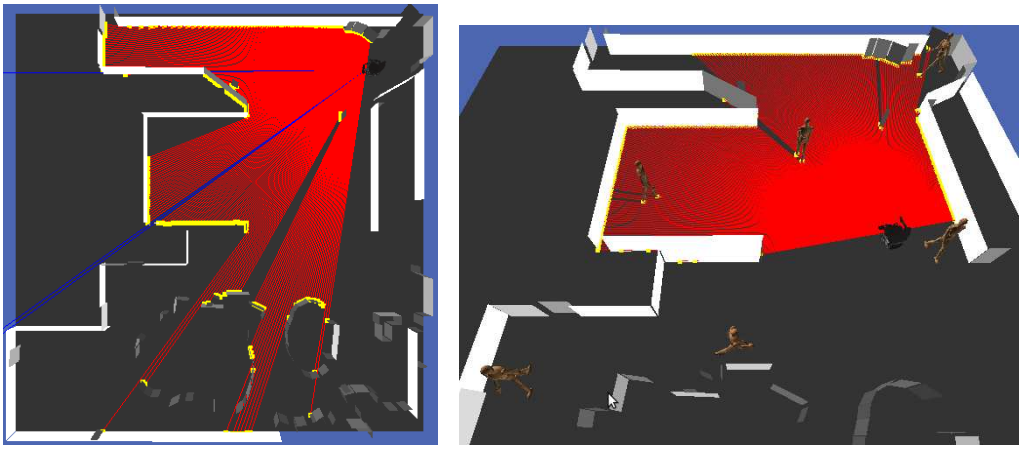


FIGURE 5.4 – Interface graphique utilisée afin de planifier un mouvement et de lancer les processus de déformation et d'exécution d'une trajectoire. Une fenêtre OpenGL permet de superviser la trajectoire planifiée (courbe verte), les obstacles statiques détectés (rectangles rouges) ainsi que le modèle prévisionnel du comportement futur des obstacles mobiles (cylindres rouges). L'utilisateur peut par ailleurs choisir différents modes de planification et d'exécution du mouvement, et ajuster les paramètres pour chacune de ces méthodes via cette interface.

– A partir de la localisation du robot et d'une représentation de l'environnement qui l'entoure, il est possible de **décider d'un mouvement à suivre** afin de rejoindre un but sans entrer en collision avec les obstacles statiques et mobiles de l'environnement. Toute cette partie décisionnelle comportant aussi bien la planification et le suivi de mou-

vement que l'évitement d'obstacles (par déformation de trajectoire) a été développée par nos soins au sein d'un module nommé *MPC* (Motion Planning and Control). Une interface graphique associée (*cf.* Fig. 5.4) a été développée afin d'aider l'utilisateur à choisir une destination, lancer l'exécution du mouvement, et superviser le processus de déformation `Teddy`.



(a) (b)

FIGURE 5.5 – Simulateur CycabTK utilisé afin de tester l'architecture complète de navigation proposée. Le hall d'entrée de l'INRIA Rhône-Alpes dans lequel ont été effectuées nos expérimentations sur la chaise roulante a été reproduit afin d'effectuer des tests préliminaires de nos travaux de déformation dans cet environnement.

Notez enfin qu'un simulateur nommé *CycabTK* (*cf.* Fig. 5.5) a été utilisé au préalable afin de reproduire l'environnement dans lequel les expérimentations sur la chaise roulante ont été effectuées. Ce dernier permet de simuler le déplacement du robot à partir du contrôle d'entrée qui lui est fourni, et retourne des données capteurs simulées perçues au cours de la navigation. Il nous a donc permis d'effectuer des tests préliminaires avant d'effectuer les expérimentations sur le robot réel.

5.2.3 Modélisation du système robotique

La chaise roulante automatisée est un robot différentiel. Néanmoins, le système étant contrôlé en vitesse linéaire et vitesse angulaire, la dynamique considérée a été quelque peu modifiée par rapport à celle du robot différentiel considéré en Section 3.4.3.

Un état de la chaise roulante automatisée est donc défini par un 5-ulpet $s = (x, y, \theta, v, \omega)$ où (x, y) représentent les coordonnées du centre des roues, θ est la direction principale de la chaise, v sa vitesse linéaire et ω sa vitesse angulaire. Un contrôle de la chaise est défini par le couple $u = (v, \omega)$. Notons a l'accélération linéaire et η l'accélération angulaire sur lesquelles des contraintes sont considérées.

Le mouvement de la chaise est alors régi par les équations différentielles suivantes :

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{v} \\ \dot{\omega} \end{pmatrix} = \begin{pmatrix} v\cos\theta \\ v\sin\theta \\ \omega \\ a \\ \eta \end{pmatrix} \tag{5.1}$$

Les contraintes suivantes sont considérées sur le mouvement de la chaise roulante :

$$\begin{array}{l} v \in [-v_{max}; v_{max}],\ v_{max} = 1.35\ m/s \\ \omega \in [-\omega_{max}; \omega_{max}],\ \omega_{max} = 1.5\ rad/s \\ a \in [-a_{max}; a_{max}],\ a_{max} = 1.39\ m/s^2 \\ \eta \in [-\eta_{max}; \eta_{max}]\ \eta_{max} = 1.5\ rad/s^2 \end{array} \tag{5.2}$$

Notez qu'aucune borne sur l'accélération angulaire maximale n'était initialement définie. Néanmoins elle a été volontairement ajoutée afin d'éviter d'obtenir un mouvement saccadé lors de la planification ou du suivi du mouvement.

5.3 Mise en place des différentes tâches de navigation

De la localisation du système robotique dans son environnement jusqu'à l'exécution du mouvement calculé pour atteindre un but prédéfini, l'élaboration d'une architecture de navigation complète nécessite la mise en place de nombreuses tâches. Nous présentons dans cette section les moyens disponibles et mis en oeuvre afin de pouvoir évaluer nos travaux de déformation de trajectoire sur la chaise roulante automatisée présentée précédemment.

5.3.1 Localisation et cartographie

La localisation du robot est initialement effectuée à partir de l'odométrie mesurée au cours de la navigation. Néanmoins, il est possible de fournir à la

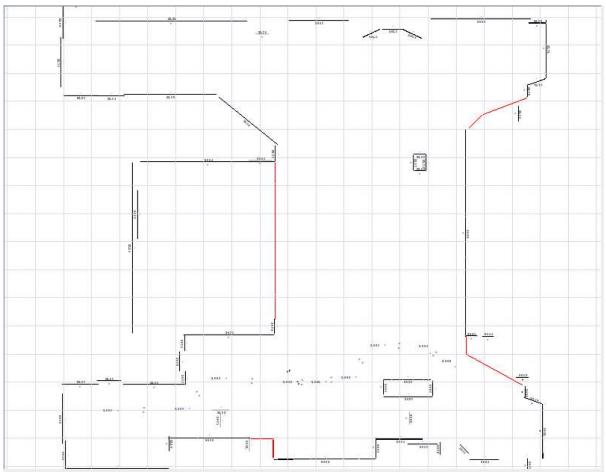


FIGURE 5.6 – Carte interne précalculée, utilisée afin d'améliorer la localisation de la chaise. Les segments noirs représentent les murs réels repérés par le LIDAR. Les murs rouges sont des murs virtuels ajoutés manuellement afin de prendre en compte des obstacles qui n'auraient pas été détectés.

chaise une carte précalculée afin de l'aider à l'améliorer. La déformation de trajectoire nécessitant une planification a priori d'une trajectoire à suivre, nous avons alors décidé de construire une telle carte. Pour cela, la chaise a été déplacée une première fois manuellement dans son environnement : en s'arrêtant en divers endroits et en collectant les données fournies par le LIDAR, la position des obstacles statiques a été enregistrée. Un logiciel fourni par BlueBotics a été utilisé dans ce but. Ce dernier donne la position des obstacles sous forme de segments (*cf.* Fig. 5.6). Lorsque le LIDAR est incapable de repérer un obstacle (escaliers, miroirs, etc.), il est possible d'ajouter manuellement des murs virtuels afin d'éviter au système d'emprunter de tels chemins lors de ses passages ultérieurs à proximité.

5.3.2 Détection et suivi des obstacles mobiles

Disposant d'une carte de l'environnement statique, il reste désormais à identifier les obstacles mobiles au cours de la navigation. Pour cela, un algorithme de détection et de suivi d'obstacles mobiles, développé au sein de notre équipe de recherche ([VA09]), a été utilisé. Celui-ci consiste à extraire les obstacles mobiles de l'environnement statique, en comparant au cours du temps les différents scans lasers fournis par le LIDAR, et en évaluant les zones de l'environnement ayant bougé. Une segmentation (clustering) des zones mouvantes est effectuée afin d'identifier un nombre fini d'obstacles mo-

biles. En identifiant au cours du temps les différents obstacles mobiles, et en mesurant leur déplacement, il est alors possible de déduire, en plus de leur position, leur vitesse instantanée.

5.3.3 Construction d'un modèle prévisionnel de l'environnement

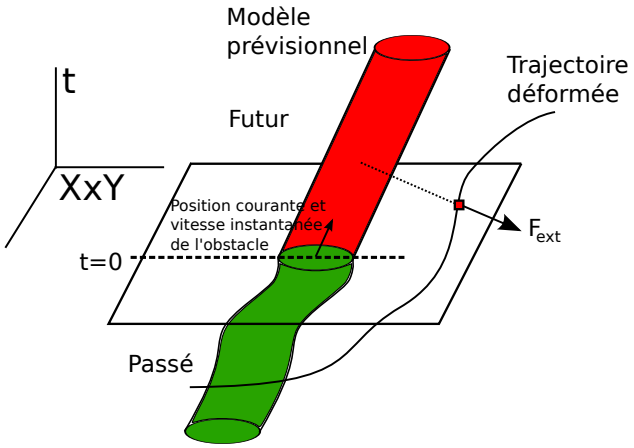
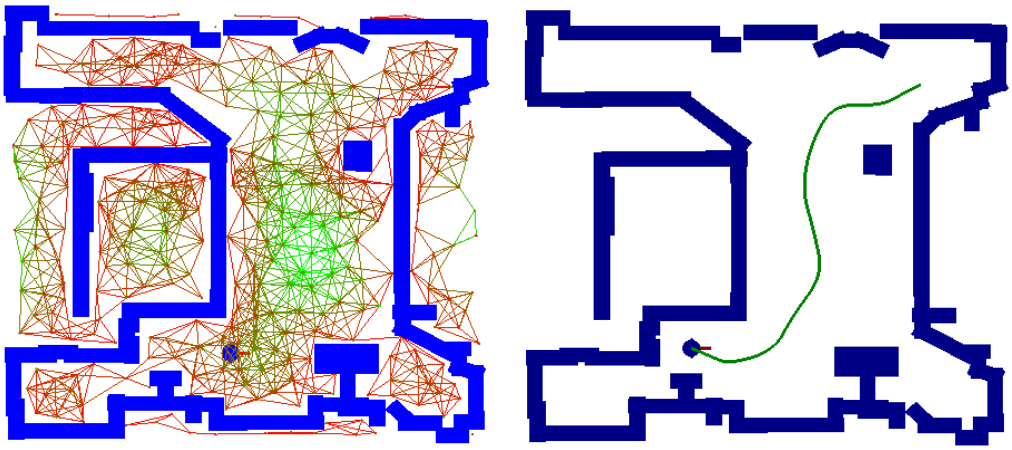


FIGURE 5.7 – Modèle prévisionnel du mouvement des obstacles mobiles déterministe et linéaire, utilisé pour la déformation de trajectoire. A partir de la position courante et de la vitesse de chaque obstacle, le modèle prévisionnel est calculé en supposant que les obstacles mobiles détectés continueront à bouger en ligne droite dans la même direction et à la même vitesse.

L'approche de déformation de trajectoire `Teddy` que nous souhaitons évaluer dans ce chapitre nécessite, en plus de détecter les obstacles mobiles, de déterminer un modèle prévisionnel de leur comportement futur. Etant donné le large choix de possibilités pour déterminer une telle prédiction (un aperçu des différentes méthodes disponibles est donné en annexe 1), un modèle approprié a été choisi lors de nos expérimentations à partir des données disponibles fournies par les capteurs du système. L'approche de détection et suivi des obstacles mobiles utilisée fournissant des informations sur leur position et leur vitesse instantanée, notre choix s'est porté sur un modèle prévisionnel déterministe linéaire (*cf.* Fig. 5.7). Celui-ci suppose que chaque obstacle mobile détecté continuera dans un futur proche à bouger en ligne droite, dans la même direction et à la même vitesse. Malgré la simplicité de ce modèle, les expérimentations décrites dans la suite de ce chapitre ont montré

que, s'il est mis à jour assez rapidement, il peut s'avérer suffisamment fiable en pratique.

5.3.4 Planification de mouvement



(a) Carte de route probabiliste (b) Trajectoire admissible calculée

FIGURE 5.8 – Création d'une carte statique constituée d'obstacles polygonaux afin de considérer la chaise roulante comme une masse ponctuelle. Les dimensions des obstacles sont augmentées du rayon de la chaise roulante. Une carte de route (roadmap) est enfin calculée afin de déterminer un chemin rapidement dans l'espace libre.

Afin de déformer une trajectoire, un plan initial doit être préalablement calculé. Pour cela, la carte interne de la chaise roulante représentant les obstacles statiques de l'environnement est utilisée. Elle est tout de même quelque peu modifiée : tout d'abord, un unique point de contrôle est défini sur la chaise roulante en son centre de rotation. Les dimensions de chaque obstacle sont augmentées du rayon de la chaise afin de ramener le problème de planification à celui d'une masse ponctuelle. Une carte statique constituée d'obstacles polygonaux est ainsi obtenue. Une carte de route probabiliste (probabilistic roadmap) est ensuite calculée dans l'espace de travail $\mathbf{W} = X \times Y$ du robot (*cf.* Fig. 5.8(a)). En associant à chacun de ses noeuds un poids dépendant de la distance à l'obstacle le plus proche, un algorithme A^* permet alors de déterminer très rapidement un chemin s'écartant des obstacles vers un but donné (*cf.* Fig. 5.8(b)).

A partir de ce chemin, une trajectoire admissible pour la chaise roulante est calculée en remplaçant des parties successives du chemin par des trajectoires générées par `Tiji` pour ce système. Pour ce faire, la contrainte sur

le temps final est relâchée afin de laisser libre le temps de parcours de la trajectoire. Même si l'intérêt premier de notre générateur de trajectoire est ici négligé, cela n'empêche de pouvoir l'utiliser pour des tâches plus simples comme une planification locale délibérative.

5.3.5 Evitement d'obstacles réactif

Une fois la trajectoire initiale planifiée, l'évitement réactif d'obstacles est totalement géré par l'approche de déformation de trajectoire `Teddy`. Celle-ci prend en entrée à chaque étape de déformation :

- la carte de l'environnement statique et le modèle prévisionnel du comportement des obstacles mobiles.
- la localisation du robot sur cette carte de l'environnement.
- la trajectoire courante suivie.

A la fin de chaque étape de déformation, la non-collision et la connectivité de la trajectoire sont vérifiées dans un futur proche (*cf.* Section 3.3.8). Si l'une de ces deux conditions n'est pas assurée, une manoeuvre de décélération est calculée afin de stopper le système. Dans le cas contraire, la trajectoire déformée est transmise au système de contrôle du robot (suivi de trajectoire énoncé ci-dessous), afin de calculer la commande à envoyer.

5.3.6 Exécution du mouvement

Enfin, un processus de suivi de trajectoire est utilisé afin de déterminer le contrôle à appliquer à chaque instant. Le contrôle est alors déterminé de la façon suivante : au temps courant t_c, le processus de suivi évalue tout d'abord l'état s_g de la trajectoire qui devrait être atteint dans un futur proche, au temps $t_c + \delta t$ (en pratique, $\delta t = 0.5\ sec$). Le générateur de trajectoire `Tiji` est alors utilisé afin d'essayer de déterminer une trajectoire de contrôle permettant de rejoindre l'état-temps but $(s_g, t_c + \delta t)$ à partir de la localisation courante. Les propriétés de `Tiji` sont une nouvelle fois exploitées ici : en effet, si aucun mouvement admissible n'est trouvé pour rejoindre le but, la trajectoire s'arrête "aussi près que possible" de l'état-temps final. Dans le cas où le robot aurait dévié de la trajectoire (incertitude sur la localisation ou le contrôle), il est alors possible de "rattraper" la trajectoire dans le futur. `Tiji` fournissant directement un profil de contrôle de forme close, ce profil est évalué durant le prochain pas de temps afin de déterminer la commande à envoyer aux actionneurs (moteurs des roues).

5.3.7 Gestion des ressources et ordonnancement des tâches

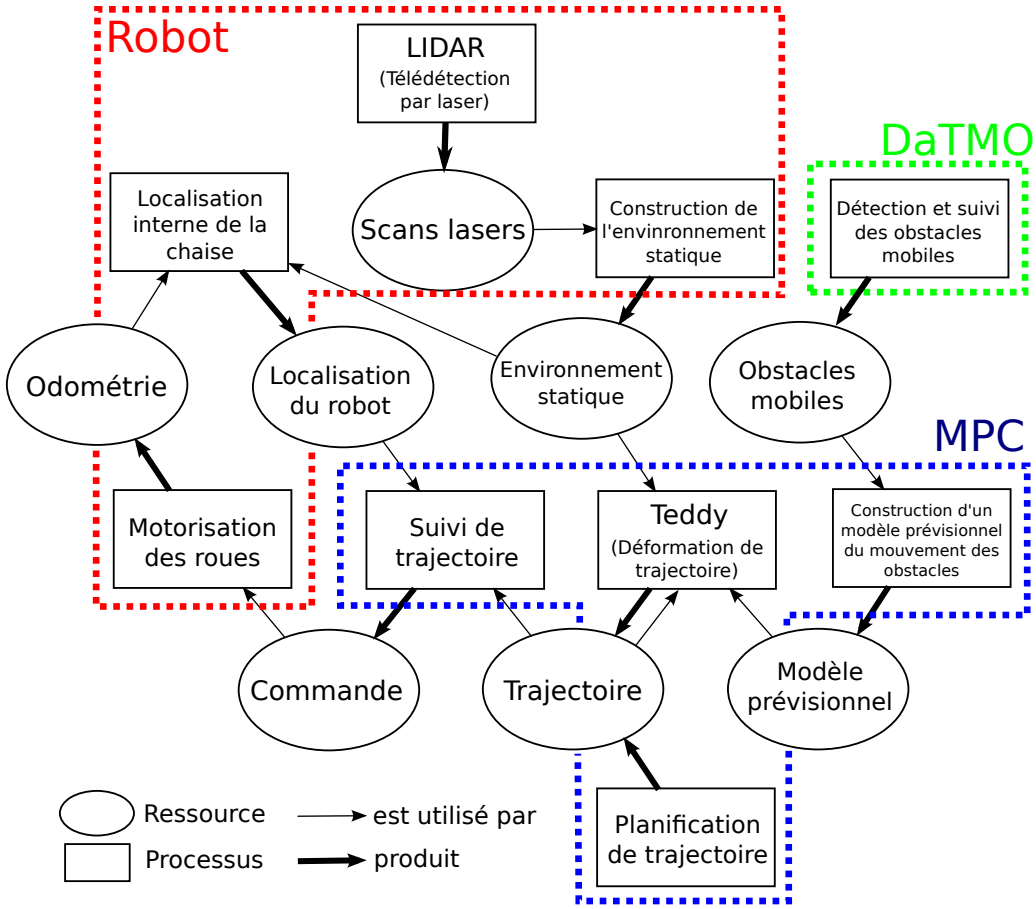


FIGURE 5.9 – Partage des ressources et ordonnancement des tâches au cours de la navigation : chacun des processus de localisation, télédétection laser, suivi des obstacles mobiles, déformation de trajectoire, suivi de trajectoire, et motorisation des roues sont exécutés en parallèle et se partagent les différentes ressources. Seule la construction d'un modèle prévisionnel du comportement futur des obstacles mobiles est exécutée en série avec le processus de déformation.

Parmi toutes les tâches citées ci-dessus, seule la planification et la cartographie n'ont en général à être exécutées qu'une seule et unique fois pour naviguer entre deux états donnés. Toutes les autres tâches doivent être exécutées continuellement au cours du temps jusqu'à ce que le robot ait atteint son but. Leur temps de calcul pouvant fortement varier, les tâches

de localisation, télédétection laser, suivi des obstacles mobiles, déformation de trajectoire, suivi de trajectoire, et motorisation des roues sont donc exécutées en parallèle. Seule la construction d'un modèle prévisionnel du comportement futur des obstacles mobiles, qui n'est utilisée que par le processus de déformation, est exécutée en série avec ce dernier. Toutes ces différentes tâches partagent un ensemble de ressources comme illustré en Fig. 5.9. Chaque processus effectue donc une copie locale des ressources dont il a besoin afin de ne les bloquer que sur une courte période de temps. Toutes ces données sont pour cela publiées dans la mémoire partagée du middleware Hugr permettant de les synchroniser et de les distribuer entre plusieurs stations de travail.

5.4 Evaluation de l'approche de déformation de trajectoire

Toute l'architecture de navigation ayant été présentée, cette section propose alors des résultats expérimentaux permettant de valider nos travaux de déformations de trajectoire sur un robot réel. Pour cela, la déformation de trajectoire a été illustrée dans divers scénarios présentés ci-dessous.

5.4.1 Conditions expérimentales

(a) (b)

FIGURE 5.10 – Hall de l'INRIA Rhône-Alpes dans lequel ont été effectuées nos expérimentations.

Nous présentons tout d'abord les conditions dans lesquelles nos expérimentations ont été réalisées. Afin de permettre au système de déformer

Processus	Fréquence d'exécution
Odométrie	100 Hz
Télédétection laser	37 Hz
Calcul de la localisation	100 Hz
Détection et suivi des obstacles mobiles	20 Hz
Suivi de trajectoire (Calcul de la commande)	20 Hz
Motorisation des roues (Envoi de la commande)	100 Hz

TABLE 5.1 – Fréquence des différents processus exécutés en parallèle de la déformation de trajectoire au cours de la navigation.

convenablement la trajectoire suivie par le robot, un environnement suffisamment ouvert et parsemé d'obstacles statiques était nécessaire. Nos expérimentations ont pris place dans le hall d'accueil de l'INRIA Rhône-Alpes (*cf.* Fig. 5.10), laboratoire de recherche dans lequel ont été développés ces travaux. Les dimensions de ce hall sont de 15 mètres sur 12. L'environnement étant fixe, une carte statique a pu être construite. En plus des obstacles statiques déjà présents dans l'environnement, des piétons ont fait office d'obstacles mobiles afin de tester l'algorithme de déformation. Le capteur sick de la chaise roulante permettant de repérer les obstacles étant situé à une dizaine de centimètres du sol, seuls les pieds des piétons pouvaient être observés. La position et la vitesse des obstacles observés s'en trouvaient donc fortement bruitées. Afin de remédier à ce problème, chaque piéton se déplaçant dans notre environnement poussait un objet volumineux (corbeille à papiers) de manière à être bien détecté. Ce chapitre étant destiné à évaluer l'approche de déformation de trajectoire et non la méthode de détection des obstacles mobiles utilisée, cette astuce a été tolérée.

Pour chaque scénario présenté, une carte statique de cet environnement est donc fournie a priori. Une destination est choisie par l'utilisateur, et une trajectoire jusqu'à ce but est planifiée à partir de la position initiale de la chaise. Au cours de la navigation, les processus parallèles à la déformation sont exécutés à des fréquences fixées décrites en table 5.1. La déformation est quant à elle calculée aussi rapidement que possible sur l'un des processeurs du Core i7 dont est équipé l'ordinateur portable utilisé pour contrôler le robot. Nous rappelons que ses performances dépendent de la résolution de

	Temps moyen d'exécution (en ms)	Fréquence moyenne d'exécution (en Hz)	Nombre de collisions	Perte de connectivité ?
Test 1	77.82	12.85	0	Non
Test 2	84.96	11.77	0	Non
Test 3	94.46	10.26	0	Non

TABLE 5.2 – Performances du processus de déformation de trajectoire `Teddy` en présence d'obstacles statiques uniquement pour différents tests.

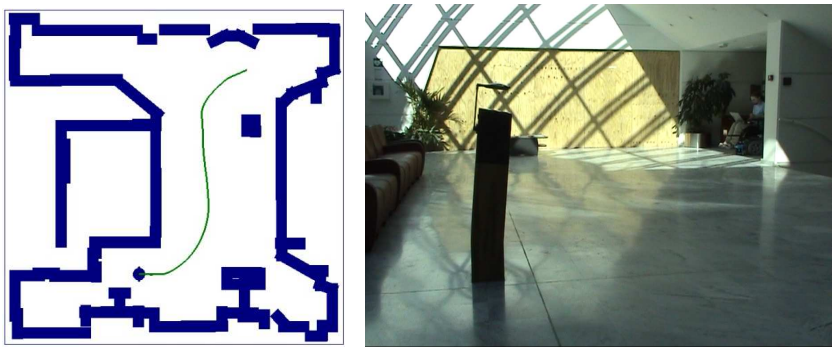
la trajectoire déformée (la trajectoire est discrétisée en une séquence d'états-temps espacés régulièrement dans le temps toutes les 0.15 sec.), de l'horizon temporel considéré (des forces sont appliquées sur les états-temps de la trajectoire définis dans les 12 prochaines secondes à partir de l'instant courant), et bien entendu de la complexité de la dynamique du système. Les temps de calcul du processus de déformation `Teddy` sont donc donnés dans les sections suivantes pour chaque scénario considéré.

5.4.2 Scénario 1 : déformation au milieu d'obstacles statiques

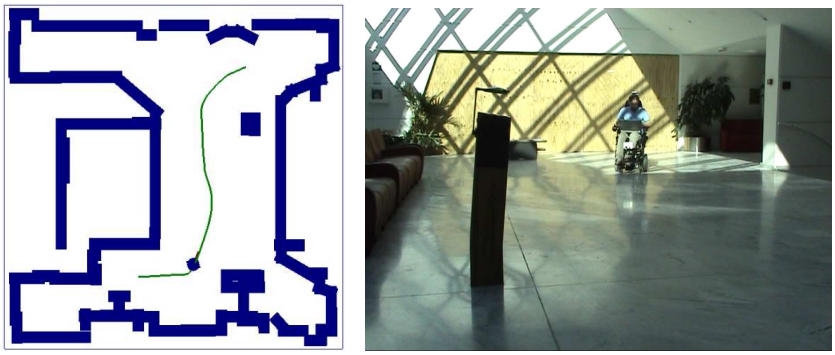
Le premier scénario présenté est une déformation de trajectoire au milieu d'obstacles statiques uniquement. Un plan initial sans collisions avec ceux-ci est calculé. Au cours de l'exécution, la trajectoire est alors simplement déformée de manière à s'écarter un peu plus des obstacles statiques tout en gardant la la faisabilité de la trajectoire et la convergence vers le but assurées. La Fig. 5.11 illustre à la fois l'enregistrement de la trajectoire déformée au cours du temps lors de l'un des tests de ce scénario, ainsi que des captures de la vidéo du mouvement exécuté par la chaise correspondante (la vidéo complète est disponible à l'adresse suivante : http ://emotion.inrialpes.fr/fraichard/films/teddy-sc1-withoutObs.avi).

Bien que très simple, ce scénario préliminaire nous a permis de tester le bon fonctionnement de l'architecture complète de navigation. Les performances de `Teddy` obtenues lors de trois tests de ce scénario sont résumées en table 5.2. Le temps d'exécution de `Teddy` est suffisamment faible pour que le système ait le temps d'adapter la trajectoire suivie à l'environnement. Aucune difficulté particulière n'étant à souligner ici, la navigation s'effectue sans collisions ni perte de connectivité.

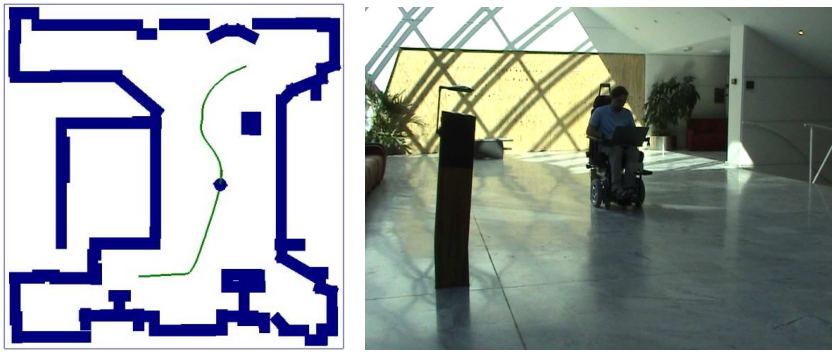
Néanmoins l'un des regrets du choix de l'architecture utilisée concerne la conservation de la localisation interne de la chaise calculée par rapport à



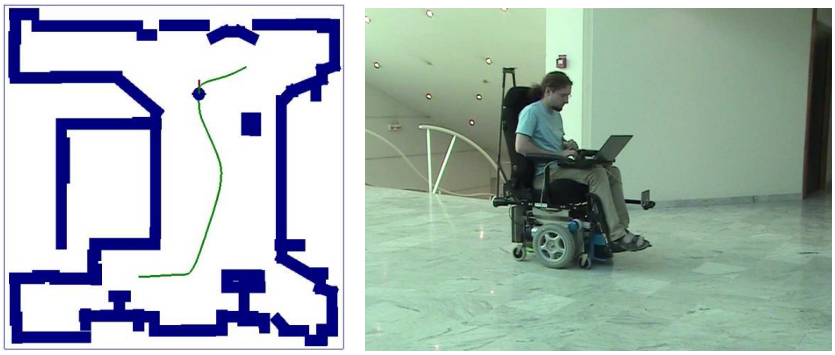
(a) t = 0 (b) t = 0



(c) t = 13 (d) t = 13



(e) t = 19 (f) t = 19



(g) t = 31 (h) t = 31

FIGURE 5.11 – Scénario 1 : Déformation au milieu d'obstacles statiques.

l'odométrie et à la carte statique de référence. Celle-ci était en effet parfois assez erronée. Sans disposer de mesure précise, l'erreur sur la localisation a été évaluée à plusieurs dizaines de centimètres pouvant atteindre 50 ou 60cm dans les pires des cas pour des trajectoires d'à peine une vingtaine de mètres. Si dans ce scénario, aucun soucis n'a été signalé, cette erreur peut devenir vite problématique lors de la navigation dans des passages étroits ou au milieu d'obstacles pouvant s'approcher fortement du robot. Le processus de localisation sera donc certainement remplacé pour de futures expérimentations.

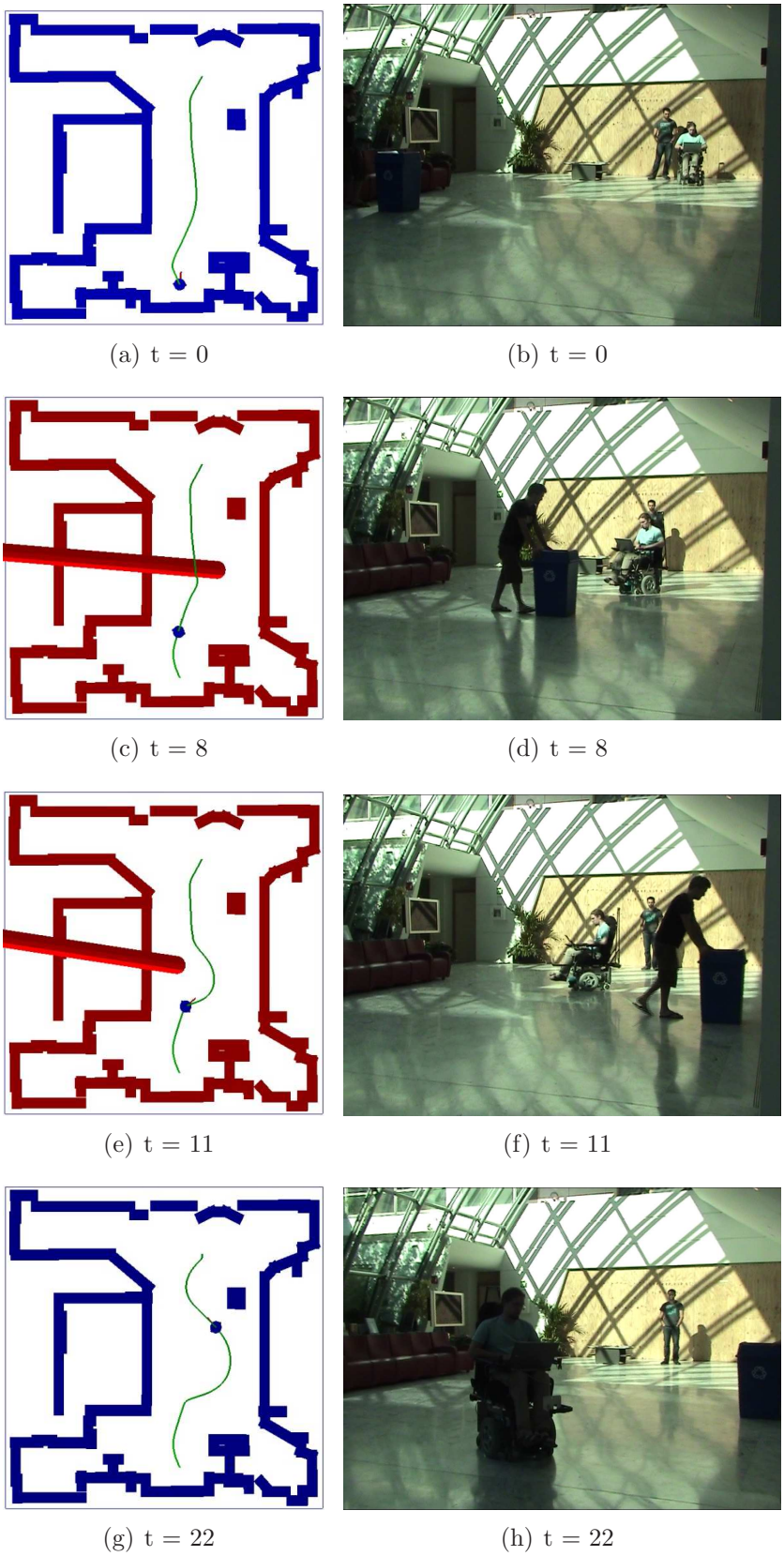
Les données fournies par l'algorithme de détection et de suivi des obstacles mobiles se sont avérées également encore assez bruitées malgré les précautions prises pour bien détecter les piétons. De nombreux faux positifs ont été signalés (apparitions d'obstacles mobiles n'existant pas). Néanmoins, leur apparitions étaient en général suffisamment brèves pour ne pas trop influer sur la déformation de la trajectoire.

Ce premier scénario a également permis de valider les modules de planification et de suivi de trajectoire utilisant le générateur de trajectoire `Tiji`. Ce dernier est certainement perfectible, néanmoins des résultats satisfaisants ont été observés : lorsque la chaise roulante essaie de suivre une trajectoire planifiée à vitesse maximale grâce à ce processus, l'erreur entre la trajectoire de référence suivie et la position de la chaise supposée (l'erreur sur la localisation n'est ici pas prise en compte) au cours du temps n'excède jamais une quinzaine de centimètres.

5.4.3 Scénario 2 : cisaillement

Le second scénario illustré est le cisaillement : ce dernier est caractérisé par un unique obstacle coupant le chemin du robot. Il est intéressant car problématique pour une simple déformation de chemin, qui se contenterait d'étirer le chemin dans la même direction que la marche de l'obstacle au fur et à mesure que ce dernier avance (*cf.* Section 3.2.6). La Fig. 5.12 illustre comme précédemment à la fois la trajectoire déformée ainsi que le mouvement exécuté par la chaise roulante pour suivre celle-ci (la vidéo complète est disponible à l'adresse suivante : http ://emotion.inrialpes.fr/fraichard/films/teddy-sc2-pathCutting.avi).

Comme il était souhaité, l'utilisation d'un modèle prévisionnel afin de déformer la trajectoire a permis, contrairement à la déformation de chemin, d'adopter divers comportements : si l'obstacle avance assez vite, `Teddy` déforme la trajectoire de manière à laisser passer l'obstacle et à garder une marge de sécurité supplémentaire après son passage (*cf.* Fig. 5.12(e), 5.12(f)). Si l'obstacle avance plus lentement, le robot accélère légèrement et contourne l'obstacle par l'avant tout en gardant également une distance de sécurité



(a) t = 0 (b) t = 0

(c) t = 8 (d) t = 8

(e) t = 11 (f) t = 11

(g) t = 22 (h) t = 22

FIGURE 5.12 – Scénario 2 : Cisaillement. Un obstacle coupe le chemin suivi par le robot.

	Temps moyen d'exécution (en ms)	Fréquence moyenne d'exécution (en Hz)	Nombre de collisions	Perte de connectivité ?
Test 1	88.18	11.34	0	Non
Test 2	105.93	9.44	0	Non
Test 3	95.97	10.42	0	Non

TABLE 5.3 – Performances du processus de déformation de trajectoire `Teddy` en présence d'un obstacle coupant le chemin du robot pour différents tests.

suffisante devant ce dernier.

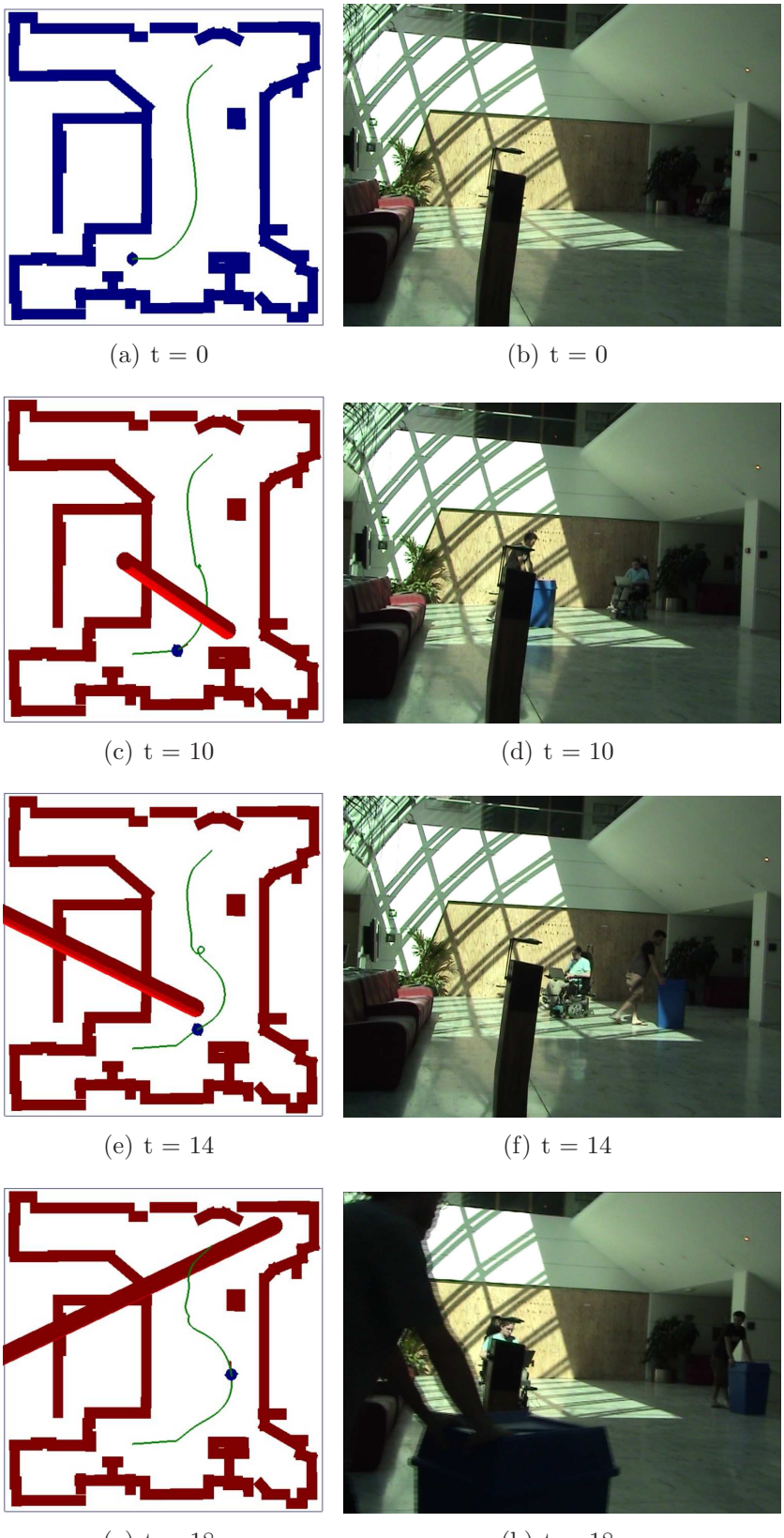
Les performances de ce second scénario sont résumées en table 5.3. La vitesse d'exécution de `Teddy` a à peine été altérée par rapport au scénario précédent, et permet au système d'être suffisamment réactif pour anticiper le mouvement des obstacles. De nouveau, aucune collision ou perte de connectivité n'ont été signalées.

Ce scénario demeure assez simple. De plus la méthode se basant sur une prédiction du mouvement qui peut s'avérer inexacte, rien n'assure complètement la sécurité du système. Si l'obstacle s'était mis à changer constamment de vitesse et de direction, il n'est pas certain que le robot aurait été capable de l'éviter. Enfin, le télémètre laser ne disposant que d'un champ de perception de 180 degrés, le robot est complètement incapable de percevoir l'obstacle durant l'intervalle de temps où il se trouve dans son angle mort afin de la contourner. Un second télémètre laser devrait être installé prochainement à l'arrière de la chaise roulante afin de pouvoir continuer à observer le comportement des obstacles dans ces conditions.

5.4.4 Scénario 3 : multiples obstacles

Le dernier scénario présente une situation plus générale et complexe où la chaise roulante navigue au milieu de multiples obstacles. Les Fig. 5.13 et 5.14 illustrent un tel scénario où deux obstacles circulent (la vidéo complète est disponible à l'adresse suivante : http ://emotion.inrialpes.fr/fraichard/films/teddy-sc3-multiObs.avi). En pratique, des expérimentations caractérisant jusque trois obstacles coupant le chemin du robot ont été effectuées.

Les performances obtenues lors de tels tests sont résumées en table 5.4. Lorsque le nombre d'obstacles augmente, la trajectoire suivie s'en trouve plus fortement modifiée. Quelques cas problématiques sont alors apparus : Lors du premier test effectué, les nombreuses modifications de la trajectoire auraient pu conduire à une collision (évitée à temps grâce à l'arrêt d'urgence). Celle-ci n'était pas directement liée à la déformation, mais à l'architecture de



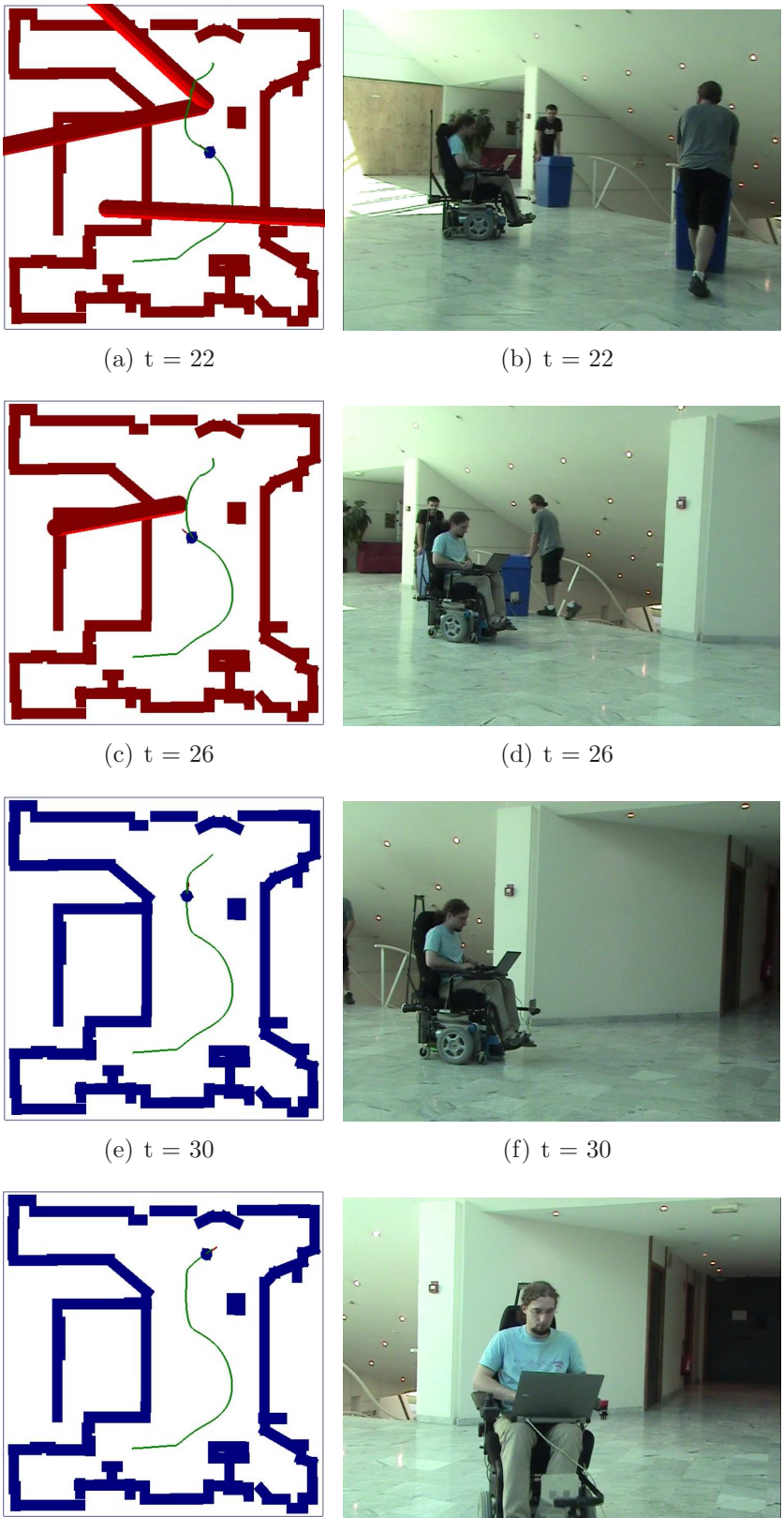
(a) t = 0 (b) t = 0

(c) t = 10 (d) t = 10

(e) t = 14 (f) t = 14

(g) t = 18 (h) t = 18

FIGURE 5.13 – Scénario 3 : Déformation au milieu de multiples obstacles dynamiques (partie 1).



(a) t = 22 (b) t = 22

(c) t = 26 (d) t = 26

(e) t = 30 (f) t = 30

(g) t = 43 (h) t = 43

FIGURE 5.14 – Scénario 3 : Déformation au milieu de multiples obstacles dynamiques (partie 2).

	Nombre d'obstacles	Temps moyen d'exécution (en ms)	Fréquence moyenne (en Hz)	Nombre de collisions	Perte de connectivité ?
Test 1	2	104.93	9.53	1	Non
Test 2	2	85.40	11.71	0	Non
Test 3	2	98.23	10.16	0	Non
Test 4	3	82.92	12.06	0	Oui
Test 5	3	91.58	10.92	0	Non

TABLE 5.4 – Performances du processus de déformation de trajectoire `Teddy` en présence de multiples obstacles pour différents tests.

navigation utilisée : le processus de localisation utilisé étant particulièrement bruité, et la carte statique n'étant pas mise à jour au cours du temps, le robot s'est retrouvé en fin de trajectoire à environ 60 centimètres de la position à laquelle il était supposé se trouver. Cette position se trouvant dans un passage étroit, le robot s'est retrouvé un peu trop près du mur pour que nous nous permettions de poursuivre l'expérimentation.

Lors du quatrième test effectué, un second problème est apparu : alors que le robot avait quasiment rejoint sa position finale, un obstacle s'est approché de cette dernière. Le système a alors dû modifier une fois de plus sa trajectoire, cependant étant trop près du but, il a cette fois été incapable de la reconnecter. Cette situation avait déjà été observée en simulation. Lorsque ce cas arrive, il est alors nécessaire de replanifier un nouveau mouvement.

Malgré ces deux cas problématiques, `Teddy` a réussi dans les autres cas à assurer la sécurité et la convergence vers le but lors de la navigation, tout en gardant des temps de calcul raisonnables permettant au robot d'être réactif.

5.5 Analyse et discussion des résultats

Les expérimentations présentées ci-dessus ont permis de valider l'approche de déformation sur un robot réel. Grâce à une information sur la position et la vitesse des obstacles mobiles, une prédiction simple de leur mouvement futur a été calculée (on a supposé à chaque instant que les obstacles allaient continuer leur mouvement dans la même direction et à la même vitesse). A partir de cette prédiction, le processus de déformation de trajectoire `Teddy` a réussi à s'adapter au comportement des obstacles mobiles et à anticiper leur mouvement. En fonction de la vitesse et de la position de chaque obstacle, la déformation de trajectoire permet au robot d'adopter divers comporte-

ments : soit d'accélérer pour passer devant un obstacle, de ralentir pour passer derrière lui, ou encore tout simplement de le contourner. L'approche est certes très dépendante de la fiabilité du modèle prévisionnel du comportement futur des obstacles ; néanmoins nous avons montré dans les scénarios ci-dessus qu'avec un processus de déformation exécuté suffisamment rapidement (environ 10Hz) et un modèle prévisionnel mis à jour à cette même vitesse, la non-collision peut être assurée.

Ces expérimentations ont également permis d'illustrer l'application du générateur de trajectoire `Tiji` à une planification de mouvement délibérative, et à un suivi de trajectoire réactif. Dans ce dernier cas particulièrement, la prise en compte de la *contrainte sur le temps final* (utilisée pour atteindre un état de la trajectoire à suivre à un temps fixé), et la possibilité de calculer des trajectoires approchées s'arrêtant "aussi près que possible" de l'état final lorsque ce dernier n'est pas atteignable ont pris tout leur sens. Une loi de commande plus appropriée aurait pu être choisie, cependant nous avons essayé d'utiliser la totalité des capacités de l'outil de génération de trajectoire `Tiji` afin de valoriser la contribution qu'il apporte.

Dans un futur proche, il serait néanmoins très intéressant d'améliorer certaines briques de l'architecture de navigation utilisées, notamment le calcul de la localisation du robot qui s'est avéré assez peu fiable. Le remplacement imminent du middleware HUGR par un système plus ouvert tel ROS [2], nous permettra certainement de profiter d'outil développés par la communauté roboticienne afin d'améliorer la fiabilité de notre approche sur robot réel.

2. http ://www.ros.org/wiki/

Chapitre 6

Conclusions et Perspectives Générales

6.1 Contributions

Cette thèse aborde le sujet de la navigation autonome en environnement dynamique et propose une approche décisionnelle de détermination de mouvement pour un robot mobile basée sur une *déformation de trajectoire* (nommée `Teddy`) combinant à la fois une planification délibérative et un évitement d'obstacles réactif. Celle-ci fonctionne de la manière suivante : une trajectoire initiale est planifiée vers le but, et une discrétisation de celle-ci en une séquence d'états-temps consécutifs est considérée. Cette trajectoire est alors transmise au système robotique afin d'être exécutée. Au cours de la navigation plusieurs forces sont appliquées sur chaque état-temps de la trajectoire restant à être parcourue afin d'assurer à la fois la non-collision avec les obstacles statiques et mobiles de l'environnement, la faisabilité du mouvement déformé, et la convergence vers le but.

Notre première contribution consiste à considérer un *modèle prévisionnel du comportement futur des obstacles mobiles*. En appliquant des forces répulsives sur les états-temps de la trajectoire suivie à partir de ce modèle prévisionnel, il est alors possible de déformer la trajectoire à la fois dans l'*espace* et dans le *temps* et ainsi d'*anticiper* le mouvement des obstacles mobiles de l'environnement. Lorsque la trajectoire se trouve hors de la région d'influence des obstacles, des forces élastiques sont appliquées sur celle-ci afin de la "retendre" et de limiter ainsi la longueur du chemin parcouru.

L'une des difficultés majeures de la déformation de trajectoire consiste à maintenir la faisabilité du mouvement déformé ainsi que la convergence vers le but. Pour s'assurer que ces deux conditions sont respectées, il est nécessaire

de vérifier que chaque état-temps de la trajectoire discrétisée soit à la fois atteignable par son prédécesseur et qu'il puisse atteindre son successeur. Afin d'éviter le calcul d'espaces atteignables qui peut s'avérer fort coûteux pour des systèmes dynamiques complexes, un générateur de trajectoire nommé `Tiji` constituant notre seconde contribution a été développé. Ce dernier permet de considérer une contrainte sur le temps final *i.e.* un temps fixé ou un intervalle de temps durant lequel l'état final doit être atteint. Basé sur une méthode variationnelle, `Tiji` est donc destiné à permettre au robot d'atteindre un état-temps final lorsque cela est possible (lorsqu'un mouvement admissible pour le système robotique considéré existe), et de s'arrêter "aussi près que possible" du but dans le cas contraire, tout en respectant l'ensemble de toutes les contraintes définies par le robot.

Cette approche de génération de trajectoire a été utilisée au sein du module de déformation `Teddy` afin de s'assurer que chaque triplé d'états-temps successifs de la trajectoire déformée soit connecté (qu'il existe un mouvement faisable passant par ces trois états-temps), et à rétablir cette connectivité dans le cas contraire.

Notre troisième et dernière contribution consiste en la validation des travaux de génération et de déformation de trajectoire sur un robot réel : une chaise roulante automatisée. Une architecture de navigation complète a été mise en place dans ce but. Des travaux de localisation, de cartographie, de détection et de suivi des obstacles mobiles existants ont été utilisés. Quant aux travaux de planification, d'évitement d'obstacles (par déformation de trajectoire), et de suivi de trajectoire (calcul de la commande), ils ont été complètement réalisés par nos soins. La méthode de déformation de trajectoire `Teddy` a alors été illustrée dans divers scénarios réalisés dans un environnement statique connu comportant un ou plusieurs obstacles mobiles (piétons). Ces exemples ont montré qu'il est possible d'assurer la non-collision grâce à la déformation de trajectoire même avec un modèle prévisionnel du comportement des obstacles mobiles simpliste, en mettant à jour ce modèle et en appliquant le processus de déformation à une fréquence satisfaisante (environ 10 Hz).

En prenant un peu de recul par rapport aux travaux de déformation réalisés, si l'on compare ces travaux avec des méthodes de navigations existantes, on peut reprocher à `Teddy` d'être une méthode assez coûteuse par rapport à d'autre méthodes réactives, ou encore d'être heuristique et par conséquent de n'apporter que très peu de garanties par rapport à la sécurité du système. Néanmoins, elle est capable à la fois, de *garantir la convergence vers le but* tant que la trajectoire est connectée et d'assurer un *évitement d'obstacles réactif* au cours du temps. `Teddy` constitue donc une méthode de

détermination de mouvement complète, adaptable à de multiples systèmes robotiques évoluant dans divers environnements. De plus, son aptitude à *anticiper leur mouvement* grâce au modèle prévisionnel du comportement des obstacles me semble, même s'il ne suffit pas, être un composant essentiel à la garantie de la sécurité du système robotique et des autres agents qui l'entourent.

6.2 Perspectives et travaux futurs

De nombreuses extensions à l'approche de déformation de trajectoire `Teddy` présentée dans ce manuscrit sont envisageables. Tout d'abord, la méthode étant heuristique, il serait souhaitable d'étendre ces travaux afin d'en améliorer la sécurité. Pour cela deux options peuvent être considérées :

- La première consisterait à améliorer le modèle de prédiction du mouvement futur des obstacles mobiles. Le modèle actuellement utilisé est très simpliste et se base uniquement sur les informations perçues par les capteurs du système robotique au cours du temps. Dans le cas d'environnements partiellement connus, il peut être possible d'identifier des schémas de mouvements typiques des agents qui y circulent (*cf.* annexe 1). Un apprentissage de ces derniers permettrait donc d'améliorer la fiabilité de la prédiction du mouvement des obstacles mobiles.
- Une seconde option consisterait à calculer au cours de la navigation les états de collisions inévitables du système robotique (état d'un système robotique menant inévitablement à une collision quelque soit la séquence de contrôles appliquée en entrée de ses actionneurs [FA03, PF07, MGF09]). En déformant la trajectoire en s'assurant d'une part que les états-temps qui la composent ne soient pas en collision avec les obstacles mais également qu'ils ne soient pas des états de collisions inévitables, la sécurité du système robotique serait fortement améliorée.

Une autre extension possible serait l'application de notre approche de déformation de trajectoire à la coordination de multiples véhicules ([RBA07, OLK09]). En planifiant pour chaque agent une trajectoire initiale, puis en appliquant entre chacune d'elles des forces répulsives, il serait possible de de coordonner les mouvements des différents agents afin de s'assurer que deux d'entre eux n'empruntent pas un passage au même moment. Ahmadzadeh et al. ([AMJP09]) ont déjà proposé une approche basée sur cette idée. Néanmoins, leurs travaux ne déformaient ces trajectoires que dans l'espace. En utilisant `Teddy` dans de mêmes circonstances, les mouvements des différents agents pourraient être modifiés à la fois dans l'espace et dans

le temps. Cette modification améliorerait la coordination dans des zones critiques telles les passages étroits ou les intersections, dans lesquels seul un agent peut s'engager.

Une dernière perspective envisageable serait la prise en compte de l'incertitude sur la localisation, la perception, et le contrôle du robot. Prendre en compte directement ces données au sein de l'architecture de navigation permettrait d'améliorer la fiabilité et la sécurité de l'approche. Cette question n'a pour l'instant absolument pas été abordée dans notre processus de déformation de trajectoire et reste donc largement ouverte.

Annexe 1

Construction d'un modèle prévisionnel de l'environnement

La fiabilité de l'approche de déformation de trajectoire présentée tout au long de ce manuscrit est très fortement dépendante de la prédiction effectuée sur le mouvement des obstacles mobiles. Différents modèles de leur mouvement peuvent être considérés, dépendants de la connaissance à priori sur l'environnement : connaissance des acteurs, de leurs intentions et de leur comportement dans la situation donnée, ou suppositions sur leurs mouvements futurs probables inférées à partir des informations perçues sur leur mouvement actuel. La plupart des approches actuelles de prédiction de mouvement se basent sur une estimation des propriétés physiques des objets mobiles et sur les lois de la physique régissant leur mouvement. L'utilisation d'un modèle cinématique ou dynamique du mouvement des différents agents est ainsi requise pour obtenir de telles prédictions. Basées sur cette hypothèse, les méthodes courantes de prédiction de mouvement peuvent être classifiées en deux catégories : méthodes de prédiction à court terme, et méthodes de prédiction à moyen et long termes. Ces deux catégories sont présentées ci-dessous.

Prédiction à court terme

Lorsqu'aucune information sur les intentions des agents mobiles environnants n'est disponible, une prédiction de leur mouvement peut être effectuée à partir des informations obtenues par les capteurs embarqués sur le système robotique. En récupérant par exemple la position et la vitesse courante des différents agents mobiles, une estimation de leur mouvement à court terme peut être calculée. Différents modèles du mouvement futur des obstacles peuvent être envisagés (*cf.* Fig. 6.1) :

– Modèle *déterministe* : un unique mouvement possible est considéré. Cette option peut être envisagée soit lorsque le mouvement de l'ob-

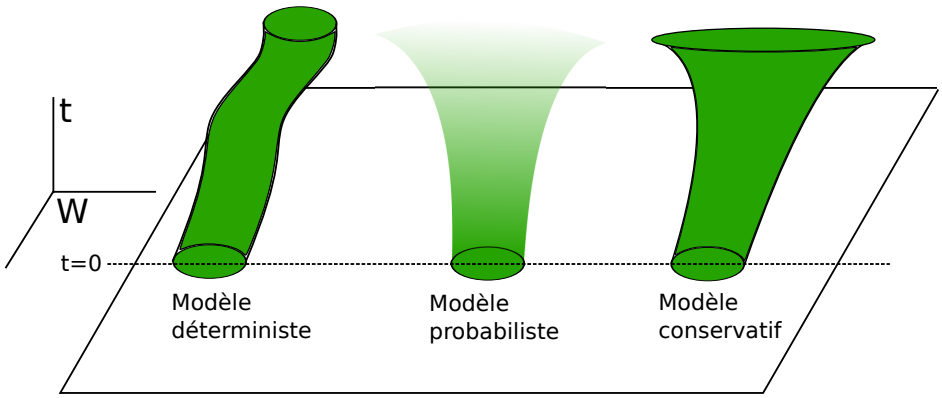


FIGURE 6.1 – Différentes modélisations possible du mouvement futur d'obstacles mobiles : Le choix de la représentation à adopter peut dépendre de la connaissance à priori sur l'environnement ainsi que du type de capteurs disponibles pour percevoir l'environnement.

stacle est entièrement connu à priori, ou en règle générale, lorsque le mouvement *le plus probable* est estimé.
- Modèle *probabiliste* : un ensemble de différents mouvements possibles est considéré et à chacun d'entre eux est associée la probabilité que ce mouvement soit suivi.
- Modèle *conservatif* : connaissant la dynamique de l'obstacle mobile ainsi que son état courant, l'ensemble de tous les mouvements possible est considéré

Le choix de la représentation à adopter dépend du type d'informations mises à disposition par les capteurs ainsi que de l'application dans laquelle le modèle est utilisé.

Prédiction à moyen et long terme

Dans le cas où le système robotique évolue dans un environnement partiellement ou complètement connu, et où les agents mobiles sont identifiables, un modèle plus élaboré de leur mouvement peut être construit. En observant et étudiant le mouvement des obstacles mobiles, il est possible d'apprendre leur comportement habituel dans un environnement donné, ainsi qu'un schéma des différents chemins fréquemment empruntés. Grâce à ces données, une prédiction de leur mouvement futur à moyen ou long terme peut être déduite.

Deux classes principales d'approches de prédiction de mouvement à long terme peuvent être soulignées, se démarquant par leur représentation et par

le moyen d'apprendre et de prédire les modèles de mouvement possibles. La première classe est basée sur des *prototypes de trajectoires* ([ME01, BBT02, VF04, JJS04]). Ces approches consistent à déterminer un ensemble des trajectoires les plus couramment suivies. Les étapes de prédiction et d'apprentissage sont complètement indépendantes.

Un algorithme de partitionnement ("clustering") utilisant une mesure de dissimilarités entre paires de trajectoires est généralement utilisé pour apprendre l'échantillon de trajectoires type sélectionné. Une fois ce modèle obtenu, la même mesure de dissimilarité peut être réutilisée pour prédire la trajectoire suivie par un agent à partir de son mouvement initial observé. Bien que la prédiction de mouvement à partir de prototypes de trajectoires semble être le type d'approches le plus couramment utilisé en pratique, elle souffre de trois inconvénients majeurs : Premièrement, le nombre d'échantillons doit être déterminé à priori (une méthode heuristique peut être utilisée en pratique). Deuxièmement, seules les trajectoires ayant été observées durant l'étape d'apprentissage peuvent être prédîtes. Enfin, les profils de vitesse de ces trajectoires sont généralement fixés. Pourtant en pratique ceux-ci peuvent fortement varier suivant l'agent qui parcourt la trajectoire.

Une seconde classe d'approche basée sur une *discrétisation de l'espace d'état* ([THM+95, RBS03, VFAL08]) a donc émergé afin de pallier ces limitations. Utilisant une structure de données nommée "modèle de Markov caché" (en anglais : Hidden Markov Model - HMM), chaque état discret de l'espace d'état par lesquels les agents mobiles passent est représenté par un état du HMM. A partir des observations du mouvement des objets mobiles à un instant donné, les probabilités de transition entre chaque paire d'états du HMM sont apprises. Pour ce faire, ces méthodes partent sur l'hypothèse que la probabilité de transition vers un état dépend uniquement de l'état précédent et non de ces antécédents. Une fois le modèle de Markov caché construit, et étant donné une observation de l'état d'un agent mobile, son mouvement (la suite d'états discrets par lesquels il va passer) peut être prédit par inférence Bayésienne. Contrairement aux méthodes basées sur prototypes de trajectoires, l'apprentissage de la structure du modèle, des probabilités de transition et la prédiction peuvent ici se faire simultanément.

Au delà de ces deux grandes classes, quelques autres méthodes peuvent être notées : Des *réseaux de neurones à plusieurs niveaux* ont été utilisés dans [JH95, SB00]. Considérant un premier niveau associant à chaque neurone un état discret de l'espace d'état des objets mobiles, et une second niveau gardant une "trace" de la transition entre chaque couple d'état, des modèles de trajectoires peuvent être appris et prédits. Enfin, une *approche orientée vers le but à atteindre* présentée dans [DH04] essaie de prendre en compte les intentions des agents observés en évaluant le but qu'ils souhaitent

atteindre afin de prédire leur mouvement. Les croyances des agents sont ainsi prises en compte en modélisant le monde à partir de leur propre perception.

Bibliographie

[ADF+06] E. Asarin, T. Dang, G. Frehse, A. Girard, C. Le Guernic, and O. Maler. Recent progress in continuous and hybrid reachability analysis. In *Proc. of the IEEE Int. Conf. on Computer Aided Control Systems Design*, Munich (DE), October 2006.

[AMJP09] Ali Ahmadzadeh, Nader Motee, Ali Jadbabaie, and George Pappas. Multi-vehicle path planning in dynamically changing environments. In *IEEE Int. Conf. on Robotics and Automation*, 2009.

[Aur91] F. Aurenhammer. Voronoi diagrams : A survey of a fundamental geometric data structure. *ACM Comput. Surv.*, 23, 1991.

[AW96] N. M. Amato and Y. Wu. A randomized roadmap method for path and manipulation planning. In *Proc. IEEE Int. Conf. on Robotics & Automation*, pages 113–120, 1996.

[BAEGM93] P. Bessiere, J. Ahuactzin, T. El-Ghazali, and E. Mazer. The ariane's clew algorithm : Global planning with local methods. In *IEEE-RSJ Int. Conf. on Intelligent Robots and Systems*, pages 1373–1380, 1993.

[BBT02] M. Bennewitz, W. Burgard, and S. Thrun. Learning motion patterns of persons for mobile service robots. In *IEEE Int. Conf. on Robotics and Automation*, pages 3601–3606, Washington (USA), 2002.

[Ber96] D.P. Bertsekas. *Constrained Optimization and Lagrange Multiplier Methods*. Academic Press, 1996.

[BG09] Lo Bianco and Oscar Gerelli. Generation of paths with minimum curvature derivative with η^3-splines. *IEEE Transactions on Automation Science and Engineering*, 2009.

[BH75] A. E. Bryson and Y. C. Ho. *Applied optimal control*. Hemisphere Publishing, 1975.

[BK91] J. Borenstein and Y. Koren. The vector field histogram - fast obstacle avoidance for mobile robots. *IEEE Journal of Robotics and Automation*, 7(3) :278–288, 1991.

[BK97] Oliver Brock and Oussama Khatib. Elastic strips : Real-time path modification for mobile manipulation. In *The Eighth International Symposium of Robotics Reasearch*, Hayama, Japan, October 1997.

[BK01] 0 Brock and L.E. Kravaki. Decomposition-based motion planning : a framework for real-time motion planning in high dimensional configuration spaces. In *IEEE Int. Conf. on Robotics and Automation*, pages 1469–1474, 2001.

[BK02] O. Brock and O. Khatib. Elastic strips : a framework for motion generation in human environments. *Int. Journal of Robotics Research*, 21(12), December 2002.

[BLC93] A. Bellaïche, J.-P. Laumond, and M. Chyba. Canonical nilpotent approximation of control systems : application to nonholonomic motion planning. In *Proc. of the IEEE Int. Conf. on Decision and Control*, San Antonio, TX (US), 1993.

[BM00a] D.J. Balkcom and M.T. Mason. Extremal trajectories for bounded velocity differential drive robots. In *IEEE Int. Conf. on Robotics and Automation*, 2000.

[BM00b] D.J. Balkcom and M.T. Mason. Time optimal trajectories for bounded velocity differential drive robots. In *IEEE Int. Conf. on Robotics and Automation*, 2000.

[Cha87] B. Chazelle. Voronoi diagrams : A survey of a fundamental geometric data structure. *Algorithmic and Geometric Aspects of Robotics*, pages 145–185, 1987.

[CP05] S. Carpin and G. Pillonetto. Robot motion planning using adaptive random walks. *IEEE Trans. on Robotics & Automation*, 21(1) :129–136, 2005.

[CR87] J. Canny and J.H. Reif. New lower bound techniques for robot motion planning problems. In *IEEE Symposium on the Foundations of Computer Science*, pages 49–60, 1987.

[DF08a] V. Delsart and T. Fraichard. Navigating dynamic environments using trajectory deformation. In *IEEE-RSJ Int. Conf. on Intelligent Robots and Systems*, 2008.

[DF08b] Vivien Delsart and Thierry Fraichard. Reactive trajectory deformation to navigate dynamic environments. In *European Robotics Symposium*, Prague Tchèque, République, 2008.

[DF10] V. Delsart and T. Fraichard. Tiji, a generic trajectory generation tool for motion planning and control. In *IEEE-RSJ Int. Conf. on Intelligent Robots and Systems*, 2010.

[DFMG09] V. Delsart, T. Fraichard, and L. Martinez-Gomez. Real-time trajectory generation for car-like vehicle navigating dynamic environments. In *IEEE Int. Conf. on Robotics and Automation*, 2009.

[DH04] H. Dee and D. Hogg. Detecting inexplicable behaviour. In *Proc. of the British Machine Vision Conference*, pages 49–55, 2004.

[DHI91] H. Delingette, M. Hébert, and K. Ikeuchi. Trajectory generation with curvature constraint based on energy minimization. In *IEEE-RSJ Int. Conf. on Intelligent Robots and Systems*, 1991.

[Dub57] L.E. Dubins. On curves of minimal length with a constraint on average curvature and with prescribed initial and terminal positions and tangents. *American Journal of Mathematics*, 79, 1957.

[Elf90] A. Elfes. Occupancy grids : A stochastic spatial representation for active robot perception. In *Proc. 6th Conference on Uncertainty in AI*, pages 60–70, july 1990.

[ELP87] M. Erdmann and T. Lozano-Pérez. On multiple moving objects. *Algorithmica*, 2(4) :477–521, 1987.

[FA03] Thierry Fraichard and Hajime Asama. Inevitable collision states a step towards safer robots. In *IEEE-RSJ Int. Conf. on Intelligent Robots and Systems*, Las Vegas, NV(US), 2003.

[FD09] Thierry Fraichard and Vivien Delsart. Navigating dynamic environments with trajectory deformation. *Journal of Computing and Information Technology*, 17, 2009.

[FDF02] E. Frazzoli, M. A. Dahleh, and E. Feron. Real-time motion planning for agile autonomous vehicles. *AIAA Journal of Guidance, Control and Dynamics*, 25(1) :116–129, 2002.

[Fra98] T. Fraichard. Trajectory planning in a dynamic workspace : a state-time space approach. *Advanced Robotics*, 13(1) :75–94, 1998.

[FS93] P. Fiorini and Z. Shiller. Motion planning in dynamic environments using the relative velocity paradigm. In *IEEE Int. Conf. on Automation & Robotics*, volume 1, pages 560–566, 1993.

[FS98] P. Fiorini and Z. Shiller. Motion planning in dynamic environments using velocity obstacles. *International Journal on Robotics Research*, 17(7) :760–772, July 1998.

[FS04] T. Fraichard and A. Scheuer. From reeds and shepp to continuous curvature paths. *Transactions on Robotics*, 20, 2004.

[FS06a] Dave Ferguson and Anthony Stentz. Anytime rrts. In *IEEE-RSJ Int. Conf. on Intelligent Robots and Systems*, Beijing, China, Oct. 2006.

[FS06b] David Ferguson and Anthony Stentz. Using interpolation to improve path planning : The field d* algorithm. *Journal of Field Robotics*, 23(2) :79–101, 2006.

[FWT97] D. Fox, Burgard W., and S. Thrun. The dynamic window approach to collision avoidance. *IEEE Journal on Robotics and Automation*, 4(1), 1997.

[GC02] S.S. Ge and Y.J. Cui. Dynamic motion planning for mobile robots using potential field method. *Autonomous Robots*, 13(3) :207–222, 2002.

[GG00] P. Gallina and A. Gasparetto. A technique to analytically formulate and to solve the 2-dimensional constrained trajectory planning problem for a mobile robot. *Journal of Intelligent and Robotic Systems*, 27, 2000.

[GKR94] V. Granville, M. Krivanek, and J.-P. Rasson. Simulated annealing : A proof of convergence. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(6) :652–656, 1994.

[GM06] S. Garrido and L. Moreno. Path planning for mobile robot navigation using voronoi diagram and fast marching. In *IEEE-RSJ Int. Conf. on Intelligent Robots and Systems*, pages 2376–2381, 2006.

[GSR09] Oren Gal, Zvi Shiller, and Elon Rimon. Efficient and safe online motion planning in dynamic environments. In *Proc. of the Intl Conf. on Robotics & Automation*, Kobe, Japan, May 2009.

[GT08] Yi Guo and Tang Tang. Optimal trajectory generation for nonholonomic robots in dynamic environments. 2008.

[HK07] T. Howard and A. Kelly. Optimal rough terrain trajectory generation for wheeled mobile robots. *Int. Journal of Robotics Research*, 2007.

[HKLR02] D. Hsu, R. Kindel, J.-C. Latombe, and S. Rock. Randomized kinodynamic motion planning with moving obstacles. *Int. Journal of Robotics Research*, 21(3), 2002.

[HLM99] D. Hsu, J.-C. Latombe, and R. Motwani. Path planning in expansive configuration spaces. *Int. Journal Computational Geometry & Applications*, 9 :495–512, 1999.

[HNR68] P.E. Hart, N.J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. on Systems Science and Cybernetics*, 4(2) :100–107, 1968.

[Hua08] L. Huang. Velocity planning for a mobile robot to track a moving target - a potential field approach. *Robotics and Autonomous Systems*, 57(1) :55–63, 2008.

[JH95] N. Johnson and D. Hogg. Learning the distribution of object trajectories for event recognition. In *Proc. of the British Machine Vision Conference*, volume 2, pages 583–592, sept 1995.

[JJS04] I. Junejo, O. Javed, and M. Shah. Multi feature path modeling for video surveillance. In *In Proc. of the 17th conference of the International Conference on Pattern Recognition*, pages 716–719, 2004.

[JS08] L. Jaillet and T. Simeon. Path deformation roadmaps : Compact graphs with useful cycles for motion planning. *Int. Journal of Robotics Research*, 27(11-12), 2008.

[KB91] Y. Koren and J. Borenstein. Potential field methods and their inherent limitations for mobile robot navigation. In *IEEE Int. Conf. on Robotics and Automation*, April 1991.

[KF07] H. Kurniawati and Th. Fraichard. From path to trajectory deformation. In *Proc. of the IEEE-RSJ Int. Conf. on Intelligent Robots and Systems*, San Diego, CA (US), October 2007.

[KGV83] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598) :671–680, 1983.

[KH89] Y. Kanayama and B. I. Hartman. Smooth local path planning for autonomous vehicles. In *IEEE Int. Conf. on Robotics and Automation*, 1989.

[Kha86] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *Int. Journal of Robotics Research*, 5(1), 1986.

[Kir70] D.E. Kirk. Optimal control theory, an introduction. 1970.

[KJCL97] M. Khatib, H. Jaouni, R. Chatila, and J.-P. Laumond. Dynamic path modification for car-like nonholonomic mobile robots. In *IEEE Int. Conf. on Robotics and Automation*, 1997.

[KL02] S. Koenig and M. Likhachev. Improved fast replanning for robot navigation in unknown terrain. In *IEEE Int. Conf. on Robotics and Automation*, 2002.

[KL05] C.-T. Kim and J.-J. Lee. Mobile robot navigation using multiresolution electrostatic potential field. 2005.

[KM85] Y. Kanayama and N. Miyake. Trajectory generation for mobile robots. In *In Proc. of the Int. Symp. on Robotics Research*, 1985.

[KN03] Alonzo Kelly and Bryan Nagy. Reactive nonholonomic trajectory generation via parametric optimal control. In *The International Journal of Robotic Research*, volume 22, July-August 2003.

[KS98] N.Y. Ko and R.G. Simmons. The lane-curvature method for local obstacle avoidance. In *IEEE-RSJ Int. Conf. on Intelligent Robots and Systems*, 1998.

[KSLO96] L. Kravaki, P. Svestka, J-C. Latombe, and M. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration space. *IEEE Trans. Robotic and Automation*, 12(4) :566–580, 1996.

[KT89] K. Komoriya and K. Tanie. Trajectory design and control of a wheel-type mobile robot using b-spline curve. In *Proc. of the IEEE-RSJ Int. Conf. on Intelligent Robots ans Systems*, 1989.

[Lat91] Jean-Claude Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, 1991.

[LaV98] S. M. LaValle. Rapidly-exploring random trees : A new tool for path planning. *TR 98-11*, October 1998.

[Lav06] S. M. Lavalle. *Planning Algorithms*. Cambridge University Press, 2006.

[LBL04] F. Lamiraux, D. Bonnafous, and O. Lefebvre. Reactive path deformation for nonholonomic mobile robots. *IEEE Trans. on Robotics and Automation*, 20(6), December 2004.

[LFG+05] M. Likhachev, D. Ferguson, G. Gordon, A. Stentz, and S. Thrun. Anytime dynamic a* : An anytime replanning algorithm. 2005.

[Lin82] A. Lingas. The power of non-rectilinear holes. In *In Proceedings 9th International Colloquium on Automata*, pages 369–383, 1982.

[LK01] S. M. LaValle and J. J. Kuffner. Randomized kinodynamic planning. *International Journal of Robotics Research*, 20(5) :378–400, 2001.

[LL00] F. Lamiraux and J.-P. Laumond. Smooth motion planning for car-like vehicles. In *Proc. of the Int. Conf. on Intelligent Autonomous Systems*, pages 1005–1012, Venezia (IT), July 2000.

[LL01] F. Lamiraux and J.-P. Laumond. Smooth motion planning for car-like vehicle. *IEEE Transactions on Robotics and Automation*, 17, 2001.

[LLB05] O. Lefebvre, F. Lamiraux, and D. Bonnafous. Fast computation of robot-obstacle interactions in nonholonomic trajectory deformation. In *IEEE Int. Conf. on Robotics and Automation*, april 2005.

[LLS05] F. Large, C. Laugier, and Z. Shiller. Navigation among moving obstacles using the nlvo : Principles and applications to intelligent vehicles. *Autonomous Robots Journal*, 19(2), September 2005.

[LS93] G. Lafferière and H. Sussmann. A differential geometric approach to motion planning. In Z. Li and J. Canny, editors, *Nonholonomic Motion Planning*, pages 235–270. Kluwer, 1993.

[MAB98] E. Mazer, J. Ahuactzin, and P. Bessiere. The ariane's clew algorithm. *Journal of Artificial Intelligence Research*, 9, Nov. 1998.

[ME01] D. Makris and T. Ellis. Finding paths in video sequences. In *In Proc. of the British Machine Vision Conf.*, pages 263–172, 2001.

[MGF08] Luis Martinez-Gomez and Thierry Fraichard. An efficient and generic 2d inevitable collision state-checker. In *IEEE-RSJ Int. Conf. on Intelligent Robots and Systems*, France Nice, 2008.

[MGF09] Luis Martinez-Gomez and Thierry Fraichard. Collision avoidance in dynamic environments : an ics-based solution and its comparative evaluation. In *Proc. of the Intl Conf. on Robotics & Automation*, Kobe, Japan, May 2009.

[Mit07] I.M. Mitchell. Comparing forward and backward reachability as tools for safety analysis. In *Hybrid Systems : Computation and Control*, number 4416 in Lecture Notes in Computer Science. Springer, 2007.

[MM00] J. Minguez and L. Montano. Nearness diagram navigation (nd) : A new real time collision avoidance approach. In *IEEE-RSJ Int. Conf. on Intelligent Robots and Systems*, 2000.

[MS93] R. M. Murray and S. S. Sastry. Nonholonomic motion planning : steering using sinusoids. *IEEE Trans. on Automatic Control*, 38 :700–716, 1993.

[Nel89] W. L. Nelson. Continuous curvature paths for autonomous vehicles. In *IEEE Int. Conf. on Robotics and Automation*, volume 3, pages 1260–1264, Scottsdale, AZ (US), May 1989.

[Nil69] N. J. Nilsson. A mobile automaton : An application of artificial intelligence techniques. In *Proc of the 1st International Conference on Artificial Intelligence*, pages 509–520, 1969.

[NW99] J. Nocedal and N.J. Wright. *Numerical Optimization*. Springer Verlag, 1999.

[OLK09] Apollon S. Oikonomopoulos, Savvas G. Loizou, and Kostas J. Kyriakopoulos. Coordination of multiple non-holonomic agents with input constraints. In *IEEE Int. Conf. on Robotics and Automation*, Kobe, Japan, may 2009.

[OM06] E. Owen and L. Montano. A robocentric motion planner for dynamic environments using the velocity space. In *IEEE-RSJ Int. Conf. on Intelligent Robots and Systems*, 2006.

[PBR07] Aurelio Piazzi, Lo Bianco, and Massimo Romano. η^3-splines for the smooth path generation of wheeled mobile robots. *IEEE Transactions on Robotics*, 23(5), 2007.

[PF05a] Stephane Petti and Thierry Fraichard. Partial motion planning framework for reactive planning within dynamic environments. In *IEEE Int. Conf. on Robotics and Automation*, Barcelona, Spain, 2005.

[PF05b] Stephane Petti and Thierry Fraichard. Safe Navigation of a Car-Like Robot in a Dynamic Environment. In *Proc. of the European Conf. on Mobile Robots*, Ancona (IT), 09 2005.

[PF07] R. Parthasarathi and T. Fraichard. An inevitable collision state checker for a car-like vehicle. In *Proc. of the Intl Conf. on Robotics & Automation*, Roma, Italy, April 2007.

[PK05] M. Pivtoraiko and A. Kelly. Generating near minimal spanning control sets for constrained motion planning in discrete state spaces. In *IEEE-RSJ Int. Conf. on Intelligent Robots and Systems*, 2005.

[QK93] S. Quinlan and O. Khatib. Elastic bands : Connecting path planning and control. In *IEEE Int. Conf. on Robotics and Automation*, Atlanta, GA (US), May 1993.

[RBA07] W. Ren, R. W. Beard, and E. M. Atkins. Information consensus in multivehicle cooperative control : Collective group behavior through local interaction. *IEEE Control Systems Magazine*, 27(2), 2007.

[RBS03] A. H. J. Rittscher, A. Blake, and G. Stein. Mathematical modelling of animate and intentional motion. In *Philosophical transactions - Royal Society of London. Biological sciences*, pages 475–490, 2003.

[Rei79] J.H. Reif. Complexity of the mover's problem and generalizations. In *In Proc. of the 20th IEEE Symposium on Foundations of Computer Science*, pages 421–427, 1979.

[RFLM93] P. Rouchon, M. Fliess, M. Lévine, and Ph. Martin. Flatness, motion planning and trailer system. In *IEEE Proc. of the IEEE Conf. on Decision and Control*, pages 2700–2705, San Antonio, TX (US), December 1993.

[RS85] J.H. Reif. and M. Sharir. Motion planning in the presence of moving obstacles. In *IEEE Symposium on the Foundations of Computer Science*, pages 144–154, 1985.

[RS90] J.A. Reeds and L.A. Shepp. Optimal paths for a car that goes forward and backward. *Pacific J. Math.*, 145, 1990.

[RZBS09] Nathan Ratliff, Matt Zucker, J. Andrew Bagnell, and Siddhartha Srinivasa. Chomp : Gradient optimization techniques for efficient motion planning. In *IEEE Int. Conf. on Robotics and Automation*, 2009.

[SB00] N. Sumpter and A. Bulpitt. Learning spatio-temporal patterns for predicting object behaviour. *Image and Vision Computing*, 18(9) :697–704, 2000.

[Sch98] Alexis Scheuer. *Planification de chemins à courbure continue pour robot mobile non holonome.* Thèse de doctorat d'état, Institut National Polytechnique de Grenoble, Grenoble, France, 1998.

[Sim96] R.G. Simmons. The curvature-velocity method for local obstacle avoidance. In *IEEE Int. Conf. on Robotics and Automation*, pages 3375–3382, 1996.

[SLL+97] S. Sekhavat, F. Lamiraux, J.-P. Laumond, G. Bauzil, and A. Ferrand. Motion planning and control for Hilare pulling a trailer : experimental issues. In *IEEE Int. Conf. on Robotics and Automation*, Albuquerque, NM (US), April 1997.

[SLN00] T. Simeon, J.-P. Laumond, and C. Nissoux. Visibility based probabilistic roadmaps for motion planning. *Advanced Robotics*, 14(6), 2000.

[SP07] M. Seder and I. Petrovic. Dynamic window based approach to mobile robot motion control in the presence of moving obstacles. In *Proc. of the Intl Conf. on Robotics & Automation*, April 2007.

[Ste93] Anthony Stentz. Optimal and efficient path planning for unknown and dynamic environments. *International Journal of Robotics and Automation*, 10 :89–100, 1993.

[THM+95] S. Tadokoro, M. Hayashi, Y. Manabe, Y. Nakami, and T. Takamori. Motion planner of mobile robots which avoid moving human obstacles on the basis of stochastic prediction. In *IEEE International Conference on Systems, Man and Cybernetics*, pages 3286–3291, 1995.

[THN89] A. Takahashi, T. Hongo, and Y. Ninomiya. Local path planning and control for AGV in positioning. In *IEEE-RSJ Int. Conf. on Intelligent Robots and Systems*, 1989.

[UB98] I. Ulrich and J. Borenstein. Vfh+ : Reliable obstacle avoidance for fast mobile robots. pages 1572–1577, 1998.

[UB00] I. Ulrich and J. Borenstein. Vfh* : Local obstacle avoidance with look-ahead verification. San Francisco, USA, 2000.

[VA09] Trung-Dung Vu and Olivier Aycard. Laser-based detection and tracking moving objects using data-driven markov chain monte carlo. In *IEEE Int. Conf. on Robotics and Automation*, 2009.

[vdBO05] Jur P. van den Berg and Mark H. Overmars. Roadmap-based motion planning in dynamic environment. *IEEE Transactions on Robotics*, 21(5), 2005.

[VF04] D. Vasquez and T. Fraichard. Motion prediction for moving objects : a statistical approach. In *IEEE Int. Conf. on Robotics and Automation*, pages 3931–3936, 2004.

[VFAL08] D. Vasquez, Th. Fraichard, O. Aycard, and C. Laugier. Intentional motion on-line learning and prediction. *Machine Vision and Applications*, January 2008.

[VHKT00] Kimon P. Valavanis, Timothy Hebert, Ramesh Kolluru, and Nikos C. Tsourveloudis. Mobile robot navigation in 2-d dynamic environments using an electrostatic potential field. *IEEE Trans. Systems, Man and Cybernetics*, 30(2), march 2000.

[VKA05] N. Vandapel, J. Kuffner, and O. Amidi. Planning 3-d path network in unstructured environments. In *IEEE Int. Conf. on Robotics and Automation*, pages 4624–4629, 2005.

[WAS99] S. A. Wilmarth, N. M. Amato, and P. F. Stiller. Maprm : A probabilistic roadmap planner with sampling on the medial axis of the free space. In *Proc. IEEE Int. Conf. on Robotics & Automation*, pages 1024–1031, 1999.

[YB06] Y. Yang and B. Brock. Elastic roadmaps : Globally task-consistent motion for autonomous mobile manipulation. In *Proc. of the Int. Conf. Robotics : Science and Systems*, Philadelphia PA (US), August 2006.

Abstract

This thesis presents a navigation method in uncertain and dynamic environment. More precisely, it consists in determining the motion of a robot from an initial position to a goal one, while preventing the robot to collide with the other agents evolving in its environment.

Between deliberative approaches - consisting in determining a priori a complete motion to the goal - and reactive approaches - computing a new motion to execute at each time step during the robot navigation - have arisen the motion deformation approaches, combining a motion planning method with a reactive obstacle avoidance process.

Their principle is simple : A priori complete motion is planned up to the goal and provided to the robotic system. During the course of the execution, the remaining part of the motion to execute is continually deformed in response to information provided by the sensors. The robot is consequently able to adapt its motion to the behaviour of the moving obstacles or to the incompleteness of its environment knowledge.

Most of the existing motion deformation methods only deform the geometric path followed by the robot. We propose thus to extend the previous approaches to a *trajectory deformation* approach that modify the followed motion either in space or time. To do it, trajectory deformation reason on an estimation of the future motion of the obstacles. By preventing the trajectory followed by the robot to collide with a forecast model of the future motion of the obstacles, the robotic system may *anticipate* their motion.

As the deformed trajectory is arbitrarily modified in time and space, one of the major difficulties of the approach is to keep the motion constraints of the robot satisfied along the trajectory. In that aim, a *trajectory generation approach with a final time constraint* has thus been developed. By discretizing the deformed trajectory in a sequence of state-times, the trajectory generation process allows to check if a feasible motion exists between each triplet of successive state-times, and should the opposite case occur they are modified to restore the connectivity of the deformed trajectory.

The trajectory deformation and trajectory generation with final time constraints have been illustrated by simulation results, and a few experiments have been proceeded on an automated wheelchair.

Keywords : Autonomous navigation ; obstacle avoidance ; trajectory deformation ; dynamic environments

Résumé

Cette thèse aborde le problème de navigation d'un système robotique en environnement dynamique et incertain. Plus particulièrement, elle s'intéresse à la détermination du mouvement pour un robot, permettant de rejoindre une position donnée tout en assurant sa propre sécurité et celle des différents agents qui l'entourent.

Entre approches délibératives - consistant à déterminer a priori un mouvement complet vers le but - et approches réactives - calculant au cours de la navigation un mouvement à suivre à chaque instant - ont émergé les approches de déformation de mouvement, combinant à la fois une planification de mouvement globale avec un évitement d'obstacles réactif local.

Leur principe est simple : un chemin complet jusqu'au but est calculé à priori et fourni au système robotique. Au cours de l'exécution, la partie du mouvement restant à être exécutée est déformée continuellement en réponse aux informations sur l'environnement récupérées par les capteurs. Le système peut ainsi modifier son parcours en fonction du déplacement d'obstacles ou de l'imprécision et l'incomplétude de sa connaissance de l'environnement.

La plupart des approches de déformations existantes se contentaient de modifier uniquement le chemin géométrique suivi par le robot. Nous proposons alors d'étendre les travaux précédents à une *déformation de trajectoire* modifiant le mouvement suivi à la fois dans l'espace et dans le temps. Pour ce faire, nous proposons de *raisonner sur le futur* en utilisant une estimation du comportement futur des obstacles mobiles. En éloignant la trajectoire suivie par le robot du modèle prévisionnel du comportement des obstacles, il est ainsi possible d'*anticiper* leur mouvement.

La trajectoire déformée étant modifiée arbitrairement dans l'espace et dans le temps, l'une des principales difficultés de cette approche consiste à maintenir le respect des contraintes sur le mouvement du robot le long de cette trajectoire et sa convergence vers le but. Une approche de *génération de trajectoire avec contrainte sur le temps final* a été développée dans ce but. En discrétisant la trajectoire déformée en une séquence d'états-temps successifs, le générateur de trajectoires permet de vérifier si un mouvement faisable existe entre chaque triplé d'états-temps de la trajectoire déformée, et dans le cas contraire de la modifier localement afin de restaurer sa faisabilité.

Les approches de déformation et de génération de trajectoire proposées ont été illustrées en simulation puis quelques expérimentations ont été réalisées sur une chaise roulante automatisée.

Mots clés : Navigation autonome ; évitement d'obstacles ; déformation de trajectoire ; environements dynamiques

(Abstract and keywords on previous page.)