Approche cognitive pour la planification de trajectoire sous contraintes

François Gaillard

Approche cognitive pour la planification de trajectoire sous contraintes

THÈSE

présentée et soutenue publiquement le 02 Février 2012

pour l'obtention du

Doctorat de l'Université Lille 1 - Sciences et Technologies

(spécialité informatique)

par

François Gaillard

Composition du jury

Directeur de la thèse :	Philippe MATHIEU, Pr, LIFL Université Lille1
Co-encadrant :	Cédric DINONT, Dr, ISEN Lille
Rapporteurs :	Rachid ALAMI, DR CNRS LAAS, Toulouse François CHARPILLET, DR INRIA, Nancy
Examinateurs :	Michael DEFOORT, MCF LAMIH, Université de Valenciennes Zahia GUESSOUM, HDR, LIP6, Université de Reims Patrick TAILLIBERT, Thales Systèmes Aéroportés Philippe VANHEEGHE, HDR, LAGIS Centrale Lille
Invité :	Emmanuel DRUON, Dr, ISEN, Lille

UNIVERSITÉ LILLE 1 - SCIENCES ET TECHNOLOGIES
Laboratoire d'Informatique Fondamentale de Lille — UPRESA 8022
U.F.R. d'I.E.E.A. – Bât. M3 – 59655 VILLENEUVE D'ASCQ CEDEX
Tél. : +33 (0)3 28 77 85 41 – Télécopie : +33 (0)3 28 77 85 37 – email : direction@lifl.fr

Résumé

Je présente une approche cognitive pour la planification de trajectoire sous contraintes. Elle repose en premier lieu sur DKP : un planificateur de trajectoires par échantillonnage. DKP construit un arbre d'exploration dans les parties atteignables de l'environnement à la manière d'un A*. Les solutions sont des trajectoires splines quadratiques adaptées au contrôle de robots à deux roues indépendantes. La couche de propagation construit un espace des paramètres, contenant toutes les valeurs des paramètres laissés libres dans les solutions. Il est représenté sous la forme d'une surface contenant toutes les solutions locales qui respectent les contraintes du problème : vitesse, accélération, évitement d'obstacle... Ensuite, une recherche de solutions est effectuée sur l'espace des paramètres selon un critère d'optimisation. DKP a la propriété d'être entièrement déterministe : les résultats sont reproductibles et son comportement est complètement contrôlable. Ce contrôle me sert à définir des comportements de pilotage. Ils sont exprimés sous la forme d'un arbre de comportements : chaque comportement agit sur la manière dont l'arbre d'exploration produit par DKP progresse dans l'environnement. De plus, les comportements sont appliqués en fonction de la partie explorée. Ainsi, seuls les comportements faisables sont développés. TÆMS permet de décrire ces comportements de pilotage puis d'évaluer les solutions ainsi construites. Pour résumer, mon approche cognitive repose sur l'évolution conjointe d'un arbre de comportements de pilotage et d'un arbre d'exploration : elle fait ainsi le lien entre planification classique et planification de trajectoire sous contraintes.

Mots-clés: planification de trajectoire, comportements, contraintes cinématiques, optimisation

Abstract

A cognitive approach for kinodynamic trajectory planning

In this thesis, I present two new contributions. First, I provide DKP: a sample-based approach for trajectory planning. DKP uses a selection/propagation architecture to build an exploration tree in the reachable parts of the environment, guided in an A* manner. Solutions are quadratic spline trajectories that are immediately executable by two-wheeled robots. The propagation level builds a parameter space which contains all the values of the free parameters in the solution. It is represented as a surface containing all local solutions which respect kinodynamic constraints: speed, acceleration, obstacle avoidance... Then, a search is applied on the parameter space using an optimization criterion. DKP is deterministic: every result produced by DKP may be repeated. Second, this control is used to define steering behaviors. These are expressed within a steering tree: every behavior acts on the way the exploration tree built by DKP progresses in the environment. Steering behaviors are applied according to the explored part. Thereby, TÆMS is used to describe the steering behaviors and to evaluate the solutions. To sum up, my cognitive approach takes advantage on the common building of a steering tree and an exploration tree which validates respect of constraints: thus, we get a link between classing planning and trajectory planning under constraints.

Keywords: trajectory planning, symbolic reasonning, kinematics, optimisation

Remerciements

Mes remerciements vont en premier lieu à Philippe Mathieu. Il m'a toujours poussé scientifiquement et humainement depuis les premiers instants du master jusqu'à l'achèvement de ma thèse. Tu resteras un exemple pour le restant de ma carrière scientifique.

Cette thèse n'aurait pas été possible sans l'acharnement de Cédric Dinont et de Emmanuel Druon. Merci pour la confiance, les conseils, les échanges, l'encadrement et la patience que vous m'avez apporté au quotidien.

Toute ma reconnaissance va aux rapporteurs et aux membres du jury pour l'intérêt qu'ils ont montré pour mes travaux.

Tout naturellement, ma gratitude va à l'équipe SMAC pour tout ce qu'elle m'a offert. Je remercie tout les membres que j'ai pu côtoyer : Sébastien, Yoann, Tony, Antoine, Laëtitia, Iryna, Omar, Benoît, David, Maxime, Jean-Christophe, Yann, Bruno, Patricia et Jean-Paul.

Un immense merci à l'ISEN et tous mes collègues grâce à qui ma thèse s'est déroulée dans des conditions idéales, que ce soit en terme d'épanouissement professionnel et d'esprit d'équipe. J'ai conscience d'avoir été un doctorant très privilégié de ce point de vue.

Merci tout particulièrement à Hugues avec qui j'ai partagé bien plus que deux bureaux au cours de ces trois années, tout comme Jordan, Xin, Jorge, Dominique, François et Olivier.

Toute ma gratitude va à Michaël "Nut", Benjamin, Gabriel et Olivier : ce travail n'aurait pas été le même sans vous. La thèse s'achève mais d'autres défis nous attendent !

Je remercie de tout mon cœur Annélie. La réalisation de cette thèse est à jamais indissociable du bonheur que je partage avec toi.

J'embrasse tendrement chaque membre de ma famille. Je ne serai pas là sans l'amour et le soutien que vous m'avez offert depuis toujours.

Je dis un grand merci à tous mes amis qui m'ont soutenu dans cet important épisode de ma vie. Je suis heureux d'avoir parcouru ces 40 mois avec vous tous. Vivement la suite !

À Annélie, ma famille, mes amis.

Table des matières

Chapitre 1

Introduction générale

Comme l'indique la Bibliothèque des Sciences et de l'Industrie[1], d'après l'International Federation of Robotics, le nombre de robots industriels recensés dans le monde a dépassé le million d'unités fin 2010. Toutefois, les robots industriels ne constituent plus la majorité de la production du fait de l'évolution des besoins en robots de service et de loisir : la robotique est devenue un enjeu majeur pour de nombreux pays industriels, en particulier ceux qui font face au vieillissement de leur population. Ainsi, pour nous assister, les robots prennent une place croissante dans le quotidien : ce ne sont pas moins de 5 millions de robots Roomba qui ont été vendus jusque maintenant. L'IFR s'attend à ce que le nombre de robots de ce types vendus sur la période 2011-2014 dépasse les 14 millions d'unités. Cette présence grandissante soulève des problèmes à la fois en terme de sécurité mais aussi en terme d'éthique : il faut coder dans nos robots des mécanismes prouvant qu'ils ne pourront pas faire du mal aux humains qu'ils côtoient, comme l'indique la première loi d'Asimov. De même, il faut déterminer les déplacements pour lesquels un robot nous semble se comporter naturellement et ainsi éviter cette sensation de malaise décrite par la « uncanny valley »[Mori, 1970]. Derrière cette notion de vallée dérangeante se trouve la notion d'acceptation d'un robot : celui-ci doit avoir des réactions qui paraissent logiques pour l'Homme. Nos seuls référentiels de comportement sont ceux des animaux et de nos semblables. Alors que les robots réalisent des tâches toujours plus complexes, il faut parvenir à représenter et appliquer de tels comportements.

Pour décider de ses actions, un robot doit établir des plans d'actions satisfaisant ses besoins. L'objectif de cette thèse est de traduire ces plans d'action en déplacements et vérifier qu'ils sont réellement réalisables par le robot. Cela requiert la description des actions d'un robot impliquant des déplacements sous la forme de comportements de pilotage et des mécanismes de planification de trajectoire permettant l'application de ces comportements de pilotage tout en préservant le respect du modèle physique du robot. Pour atteindre cet objectif, il faut faire un travail de rapprochement entre tous les domaines impliqués dans la conception d'un robot. Les couches hautes sont généralement attribuées aux raisonnements du robot, c'est-à-dire à la partie qui régit ses comportements et qui établit les plans d'actions. Sont attribuées aux couches basses le calcul des trajectoires et l'application physique des déplacements. Pour parvenir à réaliser les mouvements recherchés, l'un des enjeux du domaine est donc de trouver des mécanismes pour faire cohabiter tous ces modules qui régissent le déplacement d'un robot. Il faut donc coupler la planification symbolique qui est utilisée comme couche haute et la planification de trajectoire qui permet le calcul des déplacements du robot.

1.1 Problématiques

Ce travail se place dans le cadre du problème du déplacement de robots à deux roues indépendantes dans un environnement réel à deux dimensions. La robotique étant un domaine vaste, je ne m'intéresse pas ici à la découverte de l'environnement et l'obtention d'informations : les robots dans mes applications disposent d'une connaissance parfaite et absolue du monde dans lequel ils évoluent sur toute la durée de la planification de trajectoire. En particulier, les trajectoires des obstacles mobiles sont entièrement connues. Dans les expérimentations, le positionnement des robots est considéré comme parfait lors du calcul des trajectoires. De plus, le suivi des trajectoires s'appuie sur les travaux de [Defoort *et al.*, 2007]. Cette thèse est focalisée sur le respect des contraintes qui s'appliquent sur les déplacements du robot. Celles-ci sont souvent formalisées sous la forme d'un modèle cinématique.

Pour mettre en place ma solution, il existe dans la littérature plusieurs manières de relier la planification symbolique avec la planification de trajectoire :

Approches par fusion Les approches par fusion considèrent la planification symbolique et la planification de trajectoire directement dans le même ensemble. Nous considérons alors le problème général de la planification de trajectoire dont la difficulté est prouvée [Reif, 1987]. Toutefois, il est difficile d'exprimer dans le même contexte à la fois les actions des robots et leurs modèles physiques. En effet, les données sont très différentes : la planification des actions se fait dans le domaine symbolique alors que la planification des trajectoires se fait dans le domaine continu.

Approches découplées Les approches découplées considèrent séparément les tâches de planification. Nous pouvons nous inspirer du modèle de Reynolds [Reynolds, 1999] : le niveau haut décide d'actions qu'il faut ensuite traduire en une ou plusieurs trajectoires grâce à la couche de planification de trajectoire. Toutefois, il apparaît qu'un tel découplage n'offre aucune garantie sur la faisabilité réelle des actions décidées par le niveau haut. Cela requiert un langage commun pour décrire à la fois les tâches des robots et représenter leurs modèles de déplacement.

Approches par collaboration Cette approche consiste à isoler des données communes aux deux niveaux de planification. Chaque niveau agit sur ces données dont les modifications ont des répercussions sur l'autre couche. Ainsi, chaque niveau peut utiliser les techniques usuelles propre à son domaine d'application. Cette approche est utilisée dans la thèse.

Enjeux Le but premier de cette thèse n'est pas de dénicher une méthode ultime permettant de trouver systématiquement une solution pour tout modèle de robot en un minimum de temps. Des approches récentes fournissent des avancées intéressantes en la matière, par exemple RRT* [Karaman et Frazzoli, 2010] en terme d'optimalité ou RRT sur un espace d'états hybride [Branicky *et al.*, 2006] pour l'exploration de modèles d'état. À mon sens, le principal bénéfice à introduire une approche par collaboration serait le gain en performance dans la planification de trajectoire grâce aux savoir-faire que nous pourrions décrire.

Définitions et notations Pour montrer qu'une approche par collaboration est possible, les sections suivantes montrent que les approches discrètes et continues de la planification de trajectoire partagent les mêmes données : seules diffèrent les points de vue sur la manière de

trouver une solution. On décrit dans un contexte discret et un contexte continu les éléments suivants :

- l'espace des configurations dans la Section 1.2 ;
- l'espace de travail dans la Section 1.3 ;
- l'espace des configurations libres dans la Section 1.4 ;
- un chemin dans l'espace des configurations dans la Section 1.5 ;
- une trajectoire dans l'espace de travail dans la Section 1.6 ;
- la prise en compte des contraintes dans la Section 1.7.

1.2 Espace des configurations

D'après [Brock, 1999] et [LaValle, 2006], un robot est d'abord décrit dans son environnement par un torseur cinématique contenant dans le cas général :

- 3 paramètres pour décrire la position relative d'un point précis du robot dans un repère général ;
- 3 paramètres pour décrire l'orientation.

Sur cette base, il est possible de décrire tous les états que le robot peut prendre. Toutefois, les robots sont des solides et ne se limitent donc pas au seul torseur : il faut tenir compte de leur corps tout entier. Le robot est alors vu comme un corps articulé composé d'une série d'éléments rigides connectés entre eux par des liaisons (dont l'une relie le robot à son environnement). Chaque type de liaison cinématique permet d'obtenir un certain nombre de degrés de liberté (NF EN 23952, ISO 3952-1) entre deux corps ainsi reliés. Après la réduction du graphe de liaisons rassemblant tous les éléments du robot, les n degrés de liberté restants permettent de décrire une configuration q du robot. L'avantage de cette approche est de réduire grandement le nombre de variables nécessaires à la description du robot étudié.

1.2.1 Configurations admissibles

L'ensemble des valeurs prises par les n degrés de liberté décrivent toutes les configurations q admissibles. Des contraintes peuvent limiter des valeurs des n degrés de liberté. Cet ensemble est appelé « espace des configurations », noté $\mathbf{C}$. Un point q pris dans l'espace des configurations décrit alors une configuration unique du robot dans son environnement.

1.2.2 Exemples de robots

Robot à deux roues indépendantes

Suivant la vision de [Defoort, 2007], je considère les configurations du robot à deux roues indépendantes selon sa position (x, y) (au centre de l'entraxe) et son orientation θ dans un environnement à deux dimensions. L'espace des configurations du robot est alors constitué par tous les triplets de valeurs $(x, y, \theta) \in \mathbb{R} \times \mathbb{R} \times [0, 2\pi[$. La figure 1.1 en présente une.

Robot à deux roues indépendantes avec m chariots

On considère les configurations d'un robot, composé de m chariots, tirés par un robot à deux roues indépendantes. Chaque chariot k possède un angle de débattement θ_k avec celui qui le précède. Il est possible, naturellement, de limiter le débattement θ_k dans l'intervalle $[\theta_k^-; \theta_k^+]$ pour éviter la collision d'un chariot avec le précédent. La position (x, y) (au centre de l'entraxe) et l'orientation θ du robot à deux roues tractant ses chariots sont choisis comme référence dans

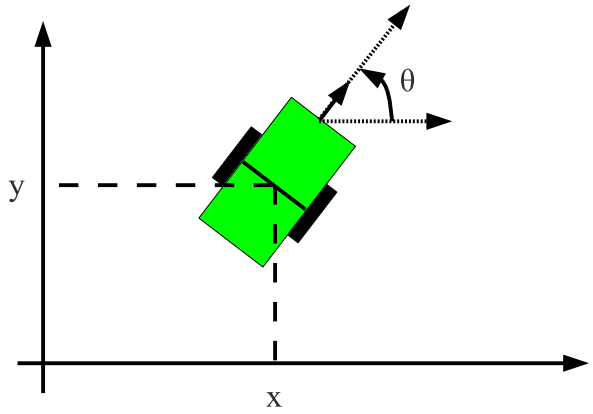

FIGURE 1.1 – Pour un robot à deux roues indépendantes, les variables permettant de décrire le robot dans un environnement ne sont que la position et l'angle du robot dans l'environnement. Le robot ne porte pas d'articulation : il ne possède donc pas de degré de liberté supplémentaire.

l'environnement. L'espace des configurations du robot à m chariots est alors constitué par tous les n-uplets de valeurs $(x, y, \theta, \theta_1, \ldots, \theta_m)$. La figure 1.2 en schématise une parmi tout l'espace des configurations possibles.

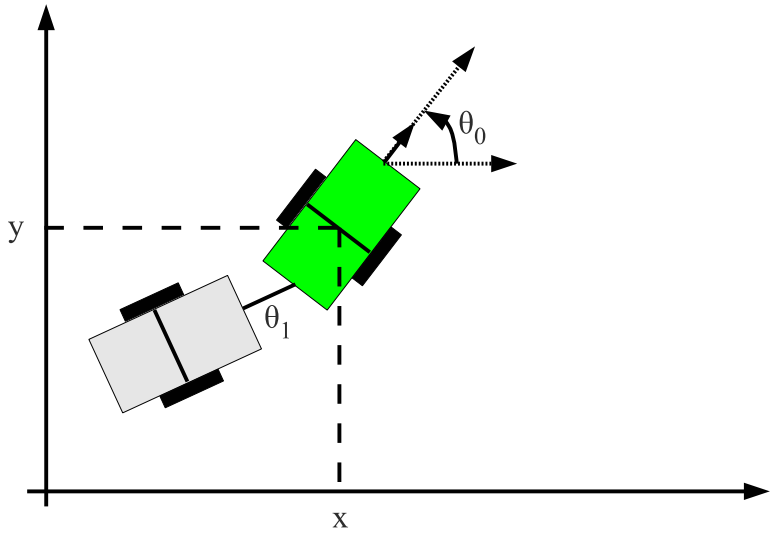

FIGURE 1.2 – Pour un robot à deux roues indépendantes tirant un chariot, les variables permettant de décrire le robot dans un environnement sont la position et l'angle θ_0 du robot dans l'environnement, ainsi que l'angle θ_1 entre le robot et son chariot.

Bras articulé

Pour l'exemple développant une approche continue de la planification de trajectoire de cette section, je considère un bras manipulateur[2] qui possède pour simplifier deux degrés de liberté : θ_1 permet le pivot de l'« épaule » et θ_2 permet le pivot du « coude ». Son espace des configurations est développé dans la sous-section 1.2.4.

Robot utilisé dans la thèse

Dans la thèse, je ne considère que les robots à deux roues indépendantes. Les robots utilisés dans les expérimentations sont décrits dans la Section 4.7.

1.2.3 Application au cas discret

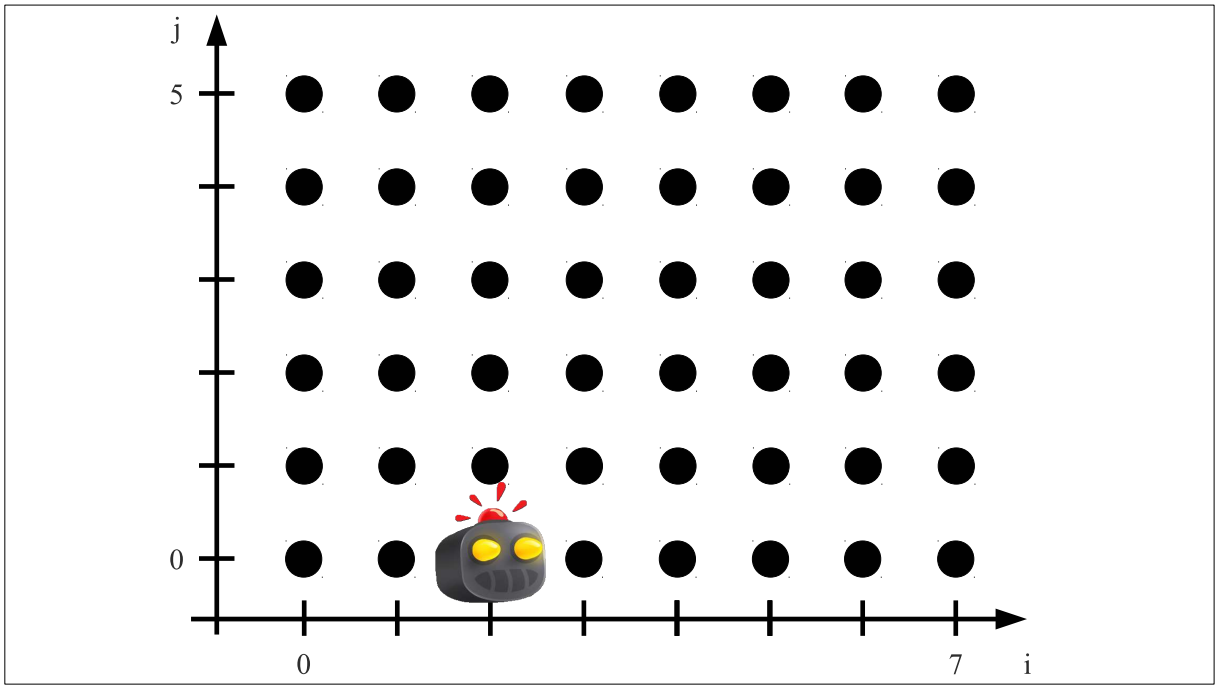

FIGURE 1.3 – Espace des configurations pour un robot dans un exemple discret.

Dans l'approche discrétisée, je me référerai régulièrement à l'exemple d'un robot[3] réduit à un point dont l'orientation n'est pas prise en compte. Dans la Figure 1.3, le robot se déplace sur un graphe dont les nœuds (les points noirs) représentent les points atteignables par le robot. Une configuration du robot est donc décrite par le couple $q = (x, y)$. Sa configuration initiale est la configuration $q_{init} = (2, 0)$. L'espace des configurations est défini par l'ensemble suivant :

$$\{\ (i,j) \mid i\ \in\ \{0;1;\ldots;7\},\ j\ \in\ \{0;1;\ldots;5\}\ \} \tag{1.1}$$

Cet exemple contient donc un nombre fini de configurations possibles : 48 dans ce cas.

1.2.4 Application au cas continu

Dans la Figure 1.4, le bras mobile peut se déplacer librement dans un environnement continu mais uniquement dans le plan de l'image. Une configuration du robot est donc décrite par :

$$q = (\theta_1, \theta_2) \tag{1.2}$$

Sa configuration initiale est la configuration $q_{init} = (\frac{\pi}{3}, \frac{\pi}{2})$. L'espace des configurations est défini par l'ensemble suivant :

$$\{\ (\theta_1,\theta_2) \mid \theta_1\ \in\ [0;\pi],\ \theta_2\ \in\ [0;\pi]\ \} \tag{1.3}$$

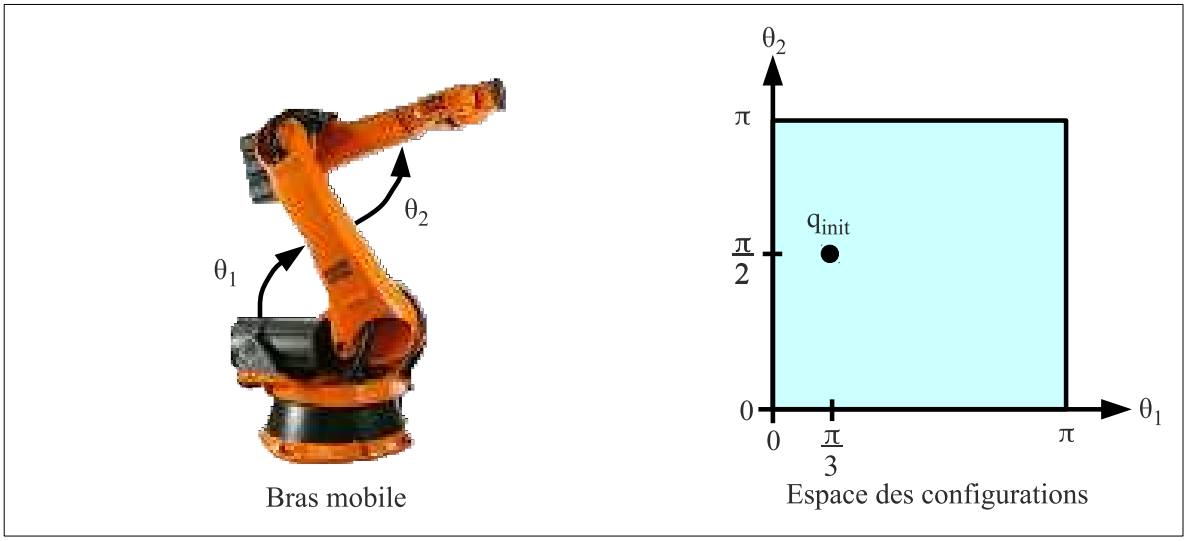

FIGURE 1.4 – Espace des configurations pour un bras manipulateur dans un exemple continu.

Bien que l'espace des configurations soit borné, il y a une infinité de configurations possibles à l'intérieur de cet espace.

1.3 L'espace de travail

L'espace de travail, noté **W**, correspond à l'espace réel physique dans lequel opère le robot. Mathématiquement, ce sera toujours un sous-ensemble de $\mathbb{R}^3$. Dans le cas du déplacement d'un robot mobile dans le plan (voiture, robot à roues...), il est possible de se ramener à un espace de dimension 2 en projetant les obstacles au sol. Cet espace peut être également de dimension 3 si le robot considéré se déplace dans l'espace, par exemple un drone. Dans l'espace de travail, au lieu d'être représenté par une seule configuration, le robot est représenté selon le volume complet qu'il occupe dans le monde physique. Plusieurs configurations q de l'espace des configurations C peuvent alors amener le robot à occuper des positions communes dans l'espace de travail.

1.3.1 Application au cas discret

L'espace de travail peut être obtenu par projection de l'espace des configurations sur tous les points constituant l'entité qui se déplace. Dans le cas discret précédemment développé, le robot est réduit à un point. L'espace des configurations contient donc directement toutes les positions admissibles du robot. Dans ce cas, l'espace des configurations se confond avec l'espace de travail, comme le montre la Figure 1.5 à comparer avec la Figure 1.3.

1.3.2 Application au cas continu

L'espace de travail peut être obtenu par l'application de toutes les configurations admissibles sur les degrés de liberté du robot, lui faisant prendre toutes les positions admissibles. Dans le cas continu, le bras robotisé possède une surface. Ainsi, construire l'espace de travail revient ici à imprimer l'empreinte du bras robotisé dans l'espace physique selon chaque configuration contenue dans l'espace des configurations. Pour donner une idée de la forme de l'espace de

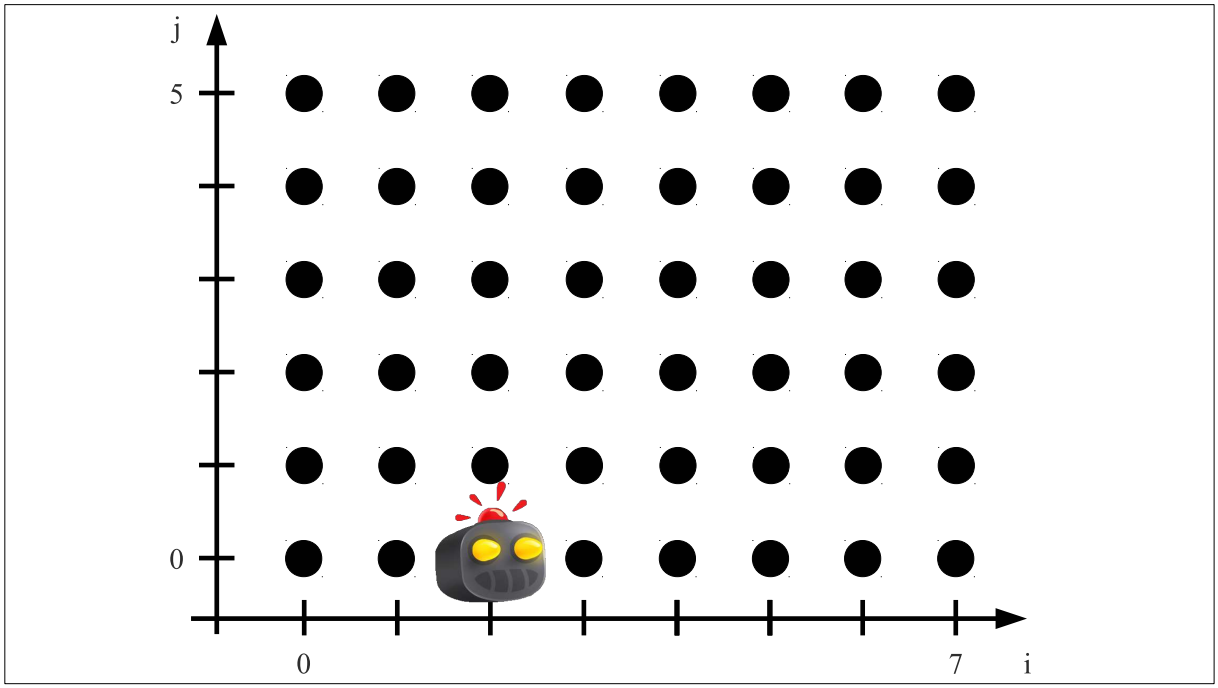

FIGURE 1.5 – Espace de travail dans un exemple discret : il se confond avec l'espace des configurations.

travail, la Figure 1.6 représente, dans la surface grisée, l'ensemble des positions que peuvent prendre les pinces situées au bout du bras robotisé. La configuration q_{init} correspond alors à la position initiale des pinces dans cet exemple. La configuration $(0, \pi)$ provoque la fermeture de l'*épaule* et l'ouverture complète du *coude*. La configuration $(\frac{\pi}{2}, \pi)$ déploie le bras vers le haut. La configuration (π, π) amène à l'ouverture complète du bras vers l'avant. Pour simplifier le travail de représentation, le bras et l'avant-bras possède la même longueur. Dans ce cas, les configurations $(0, 0)$ et $(\pi, 0)$ sont confondues dans l'espace de travail : plusieurs configurations peuvent correspondre à une même position dans l'espace de travail.

1.4 L'espace des configurations libres

Lors d'une planification d'un chemin, l'une des contraintes qui est posée dans le planificateur de trajectoire est de déterminer un chemin qui évite les obstacles. L'espace des configurations libres représente donc l'ensemble des configurations pour lesquelles les points correspondants dans l'espace de travail ne rentrent pas en collision avec un obstacle. L'espace des configurations libres est noté $\mathbf{C}_{free} \subset \mathbf{C}$. La navigation dans un environnement avec obstacles mobiles est un problème encore plus difficile [Stilman et Kuffner, 2004] : l'espace des configurations libres évolue alors en fonction du temps et des éventuelles interactions du robot avec les obstacles mobiles.

1.4.1 Application au cas discret

Comme vu précédemment, l'espace des configurations et l'espace de travail se confondent. À toute configuration unique de l'espace des configurations correspond une unique position dans

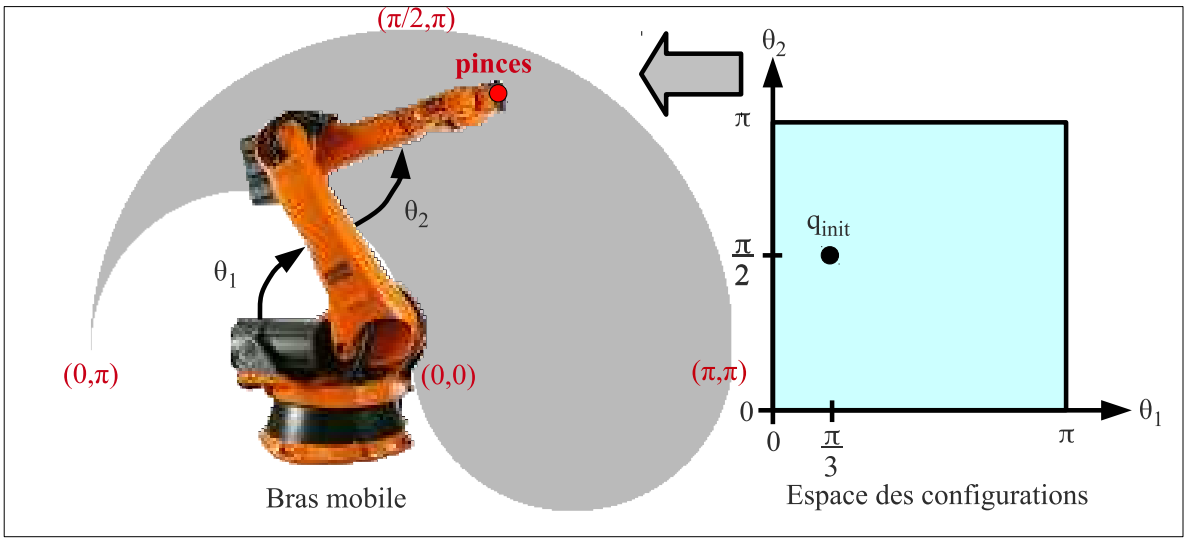

FIGURE 1.6 – Espace de travail dans un exemple continu : c'est la localisation possible du point de référence (ici, le centre des pinces) pour toutes les configurations (θ_1, θ_2).

l'espace de travail. Inversement, à toute position de l'espace de travail correspond une seule configuration. Il est donc facile de retrouver les configurations libres à partir de l'espace de travail. D'après la Figure 1.7, les positions grisées entrent en collision avec les obstacles dans l'espace de travail. Ainsi, l'espace des configurations libres est alors :

$$\mathbf{C}_{free} = \mathbf{C} \setminus \{(1,2),(2,2),(3,2),(4,4),(5,4),(6,4)\} \tag{1.4}$$

1.4.2 Application au cas continu

Dans cet exemple, l'espace des configurations contient des points qui correspondent parfois à des positions communes dans l'espace de travail. Si la projection de l'espace des configurations sur l'espace de travail est possible, cette transformation n'est pas surjective. Dans le cas où un obstacle occupe l'espace des positions, il devient alors plus difficile d'obtenir les configurations qui entrent en collision avec cet obstacle.

Ce problème est résumé à titre d'illustration par la Figure 1.8. Dans mon exemple, un mur constitue un obstacle aux déplacements du bras. Les positions occupées par le mur dans l'espace de travail correspondent à un ensemble de configurations $\mathbf{C}_{Mur}$, ensemble illustré par la surface coloriée en rouge dans l'espace des configurations. Ces configurations sont celles pour lesquelles il existe une collision entre le bras mobile et le mur. L'espace des configurations libres est alors :

$$\mathbf{C}_{free} = \mathbf{C} \setminus \mathbf{C}_{Mur} \tag{1.5}$$

Cette étape de construction de l'espace des configurations libres, préalable à la planification de mouvement, est un problème en soit, d'autant plus difficile que le robot possède beaucoup de degrés de liberté. Par conséquent, l'espace des configurations peut être d'une dimension très largement supérieure à l'espace de travail : il peut y avoir de nombreuses configurations, parfois très éloignées entre elles, amenant tout ou partie du robot à occuper une même portion de l'espace de travail.

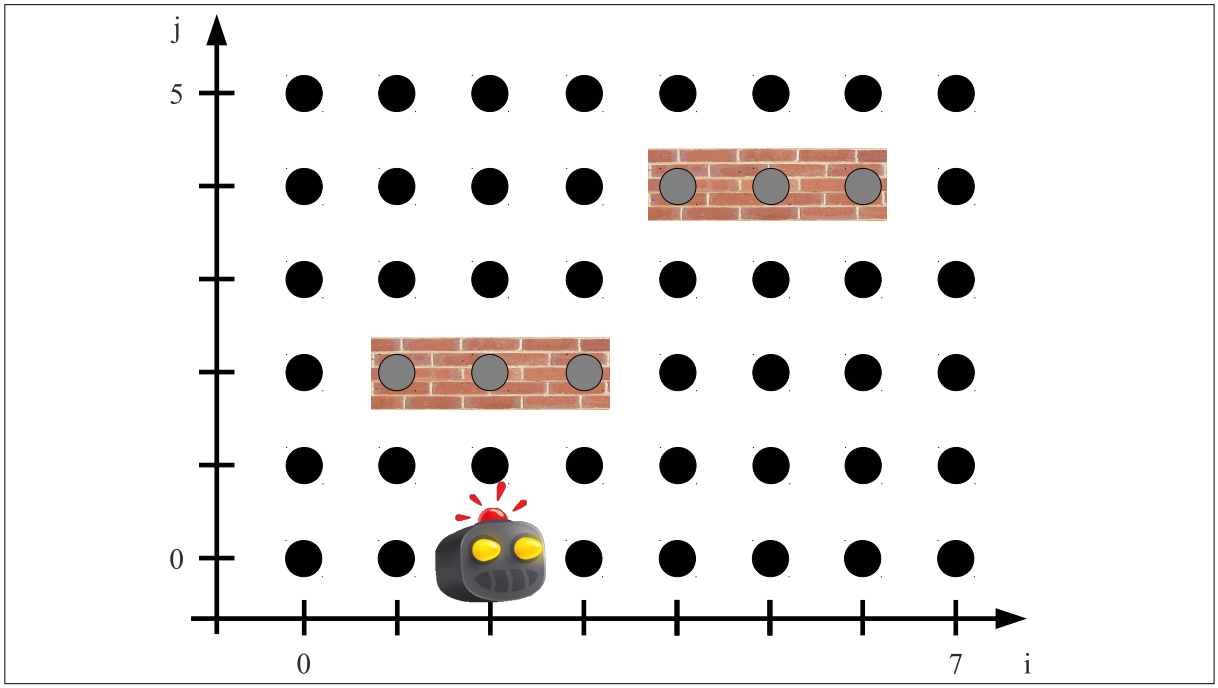

FIGURE 1.7 – Espace des configurations libres dans un exemple discret : les points gris sont des configurations non autorisées.

1.4.3 Empattement du robot

Dans le cas où le robot est ramené à un point dans le processus de planification de trajectoire, il faut tenir compte de son empattement réel pour assurer l'évitement des obstacles présents dans l'environnement. Soit r le rayon du disque englobant la surface réelle du robot (projetée au sol). Le point de référence sur cette surface est également le centre du cercle. La taille de tous les obstacles est alors augmentée du diamètre $2 \times r$ (région qui sera généralement grisée sur les schémas). Cette règle permet de garantir l'évitement des obstacles en toute situation. Toutefois, elle surestime largement la taille du robot : certaines configurations, pourtant acceptables dans l'espace de travail, peuvent devenir inaccessibles, comme le montre la Figure 1.9. La gestion de la forme exacte est un problème extrêmement complexe à formuler dans l'espace des configurations.

Distance de sécurité

Dans la suite, le terme distance de sécurité est utilisé en référence à la distance d'évitement du robot aux obstacles fixes ou mobiles. Cette distance tient compte au minimum de l'empattement du robot. Toutefois, un planificateur de trajectoire prenant en compte cette seule distance peut générer des solutions amenant le robot à frôler les obstacles. La distance de sécurité d_{obst} ajoute un écartement e supplémentaire au rayon r servant à l'estimation de la taille du robot. Cette distance de sécurité ainsi majorée permet d'obtenir des solutions présentant des évitements plus larges. Compte tenu de l'incertitude qu'il peut y avoir lors des déplacements des robots en expérimentation, cet écartement se révèle souvent nécessaire.

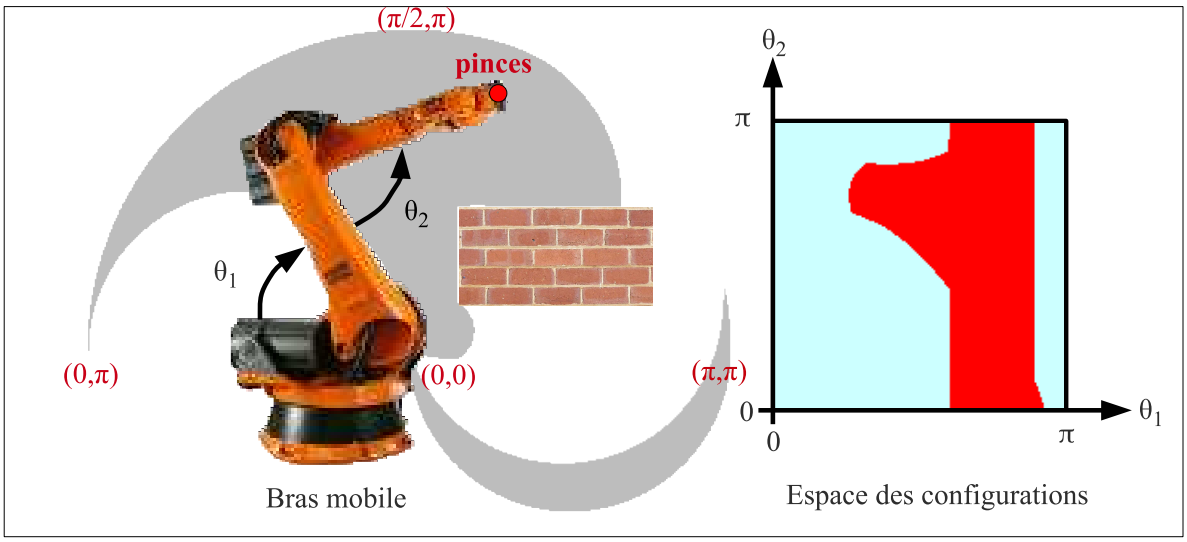

FIGURE 1.8 – Espace des configurations libres dans un exemple continu : la surface rouge illustre les configurations pour lesquelles le bras n'entre pas en collision avec l'obstacle (voir le chapitre 4 de [LaValle, 2006] pour les méthodes de calcul).

1.5 Le problème de planification de chemin

Pour déplacer le robot, il faut trouver une succession de configurations admissibles lui permettant d'atteindre son objectif. Le problème de planification de chemin correspond à la recherche d'une séquence de configurations appartenant à $\mathbf{C}_{free}$ depuis un état de départ *Start* jusqu'à un état d'arrivée *Goal*. Cette séquence est appelée chemin. Il est noté *path* tel que :

$$path(n) : [0; 1] \rightarrow \mathbf{C}_{free} \tag{1.6}$$

Ainsi, les états initiaux et finaux sont :

$$path(0) = Start \tag{1.7}$$
$$path(1) = Goal \tag{1.8}$$

1.5.1 Application au cas discret

Dans la suite des exemples sur le cas discret, l'espace des configurations contient un ensemble de positions qui correspond au maillage d'une grille. Comme représenté par la Figure 1.10, l'état de départ correspond à la position du robot tel qu'il est représenté. Le chemin correspond à la séquence de points en bleu rejoignant le point d'arrivée représenté en rouge. Dans le cas discret,

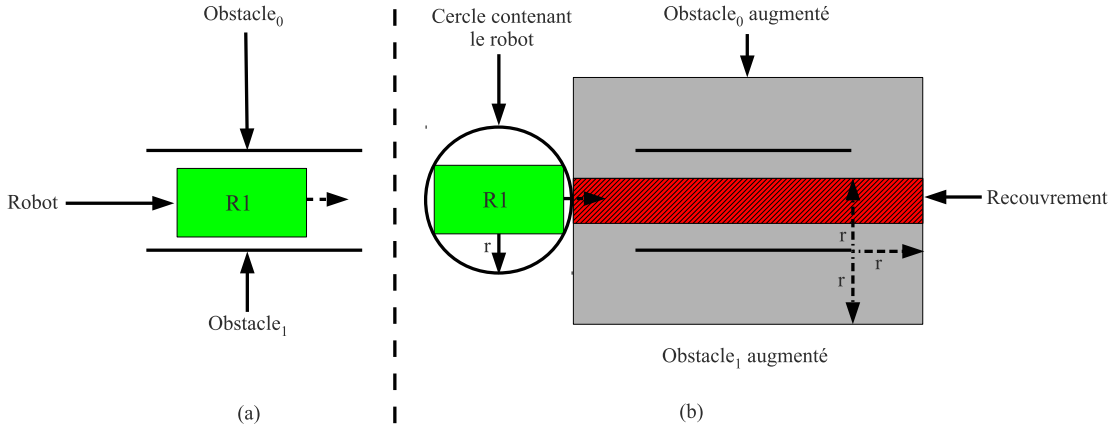

FIGURE 1.9 – Défaut dans l'approximation de la taille d'un robot dans le cas d'un passage exigu : (a) si sa forme exacte est prise en compte, le robot peut passer dans le couloir et (b) si le robot est ramené à un point, soit r le rayon du disque recouvrant complètement sa surface, si la taille des obstacles est augmentée de ce rayon, alors le passage n'apparaît plus franchissable car les obstacles se recouvrent dans la surface hachurée en rouge.

le chemin $path_{discrete}$ est une séquence discrète de $N+1$ configurations libres :

$$\begin{aligned} path_{discrete} : 0 &\rightarrow (2,0) = Start, \\ \frac{1}{N} &\rightarrow (3,0), \\ \frac{2}{N} &\rightarrow (4,0), \\ \frac{3}{N} &\rightarrow (4,1), \\ &\vdots \quad , \\ 1 &\rightarrow (5,5) = Goal \end{aligned}$$

1.5.2 Application au cas continu

Dans le cas continu, la manière de procéder à la planification de chemin ne diffère pas du cas discret du point de vue des définitions. Simplement, l'espace des configurations contient un continuum de configurations, ce qui augmente démesurément l'espace de recherche. Une solution dans le cas continu est illustrée dans la Figure 1.11. Le chemin $path_{continuous}$ est une séquence continue de configurations libres telle que :

$$path_{continuous}(0) = Start \tag{1.9}$$
$$path_{continuous}(1) = Goal \tag{1.10}$$

1.6 Le problème de planification de trajectoire

Si la résolution du problème de planification de chemin se fait dans l'espace des configurations, on ne fait que projeter le résultat, le chemin, dans l'espace de travail en fonction du temps. Ainsi, une trajectoire est l'ensemble des points de l'espace de travail associés aux

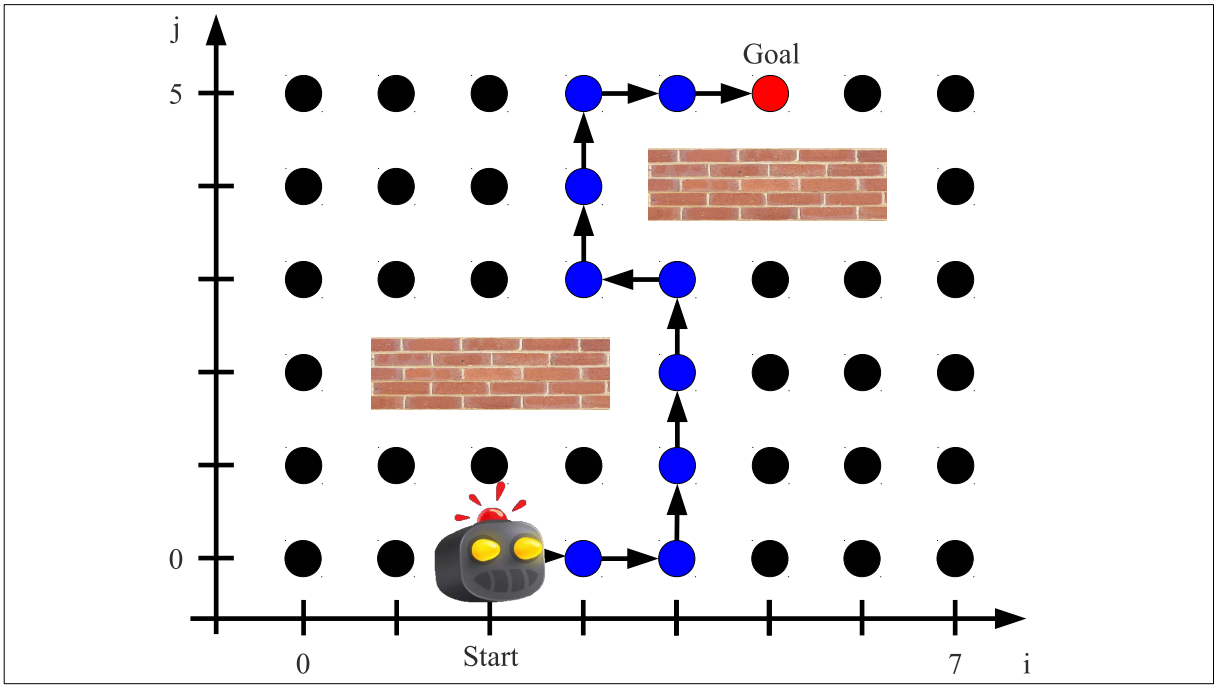

FIGURE 1.10 – Chemin planifié dans le cas discret.

configurations du chemin planifié. Dans le cadre de cette thèse, nous limitons l'espace de travail à un environnement à 2 dimensions. Soit T la durée d'exécution d'une trajectoire. La trajectoire est notée p, telle que :

$$p(t) : [0; T] \rightarrow \mathbb{R}^2 \tag{1.11}$$

Les états initiaux et finaux dans l'espace de travail sont encore notés, par abus de langage, *Start* et *Goal* :

$$p(0) = Start \tag{1.12}$$
$$p(T) = Goal \tag{1.13}$$

1.6.1 Application au cas discret

Dans le cas discret présenté jusqu'ici, l'espace des configurations et l'espace de travail sont confondus. Les configurations empruntées par le chemin *path* correspondent directement aux positions dans l'espace de travail. Dans ce cas, le temps est également discrétisé, par exemple :

$$t \in \{0, 1, \ldots, T\} \tag{1.14}$$

La trajectoire $p_{discrete}$, représentée par la Figure 1.12, passe donc par les positions suivantes :

$$\begin{aligned} path_{discrete} : 0 &\rightarrow (2,0) = Start, \\ 1 &\rightarrow (3,0), \\ 2 &\rightarrow (4,0), \\ 3 &\rightarrow (4,1), \\ &\vdots \quad , \\ T &\rightarrow (5,5) = Goal \end{aligned}$$

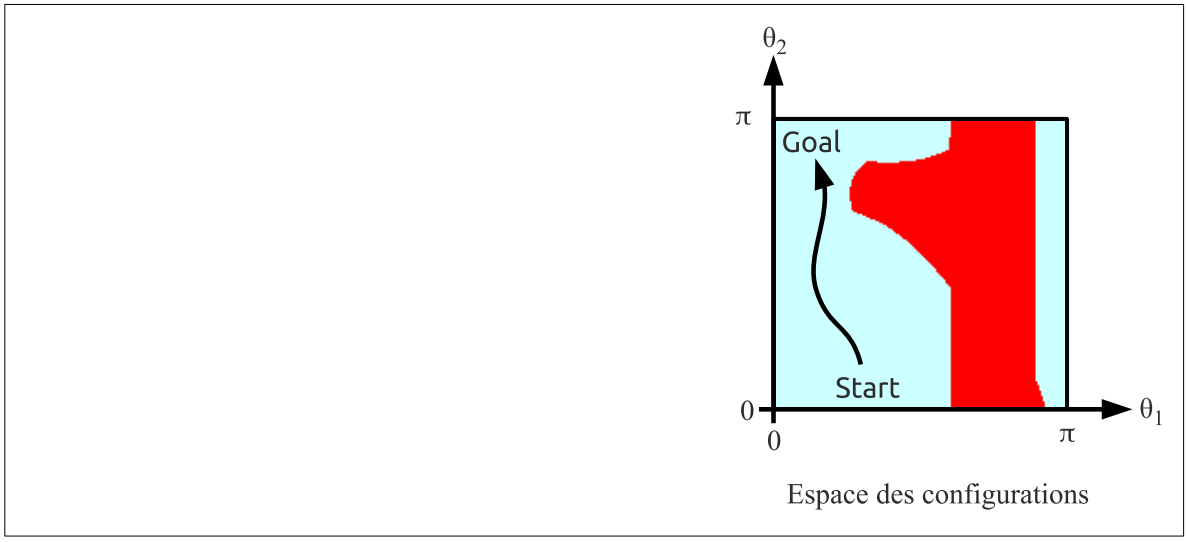

FIGURE 1.11 – Chemin planifié dans le cas continu.

1.6.2 Application au cas continu

Dans le cas continu présenté jusqu'ici, l'espace des configurations et l'espace de travail sont deux espaces différents. Les configurations empruntées par le chemin *path* doivent être projetées dans l'espace de travail pour obtenir les positions empruntées par le robot. Dans ce cas, le temps est également continu :

$$t \in [0; T] \tag{1.15}$$

La trajectoire $p_{continuous}$, représentée par la Figure 1.13, emprunte donc une suite continue de positions telles que :

$$p_{continuous}(0) = Start \tag{1.16}$$
$$p_{continuous}(T) = Goal \tag{1.17}$$

1.7 Le problème de planification de trajectoire sous contraintes et optimisation

Un robot est limité par ses capacités de mouvement décrites selon son modèle cinématique. Un robot holonome peut ainsi effectuer des mouvements de translation qui sont impossibles pour un robot de type unicyle. La succession des actions est également soumise à des contraintes cinématiques et dynamiques liées à la nature même de l'entité commandée : un robot est un assemblage de composants mécaniques et électriques dont la capacité à changer d'état, la capacité à soutenir des efforts et dont l'énergie disponible sont limitées. Des contraintes s'imposent donc, limitant les transitions entre deux configurations. Ce problème est désigné sous le terme de « kinodynamic planning » [Donald *et al.*, 1993].

Contraintes cinématiques

Une contrainte cinématique restreint les possibilités de passage d'une configuration à une autre configuration. Soit $c(t)$ une contrainte cinématique. Elle est vérifiée si la trajectoire p

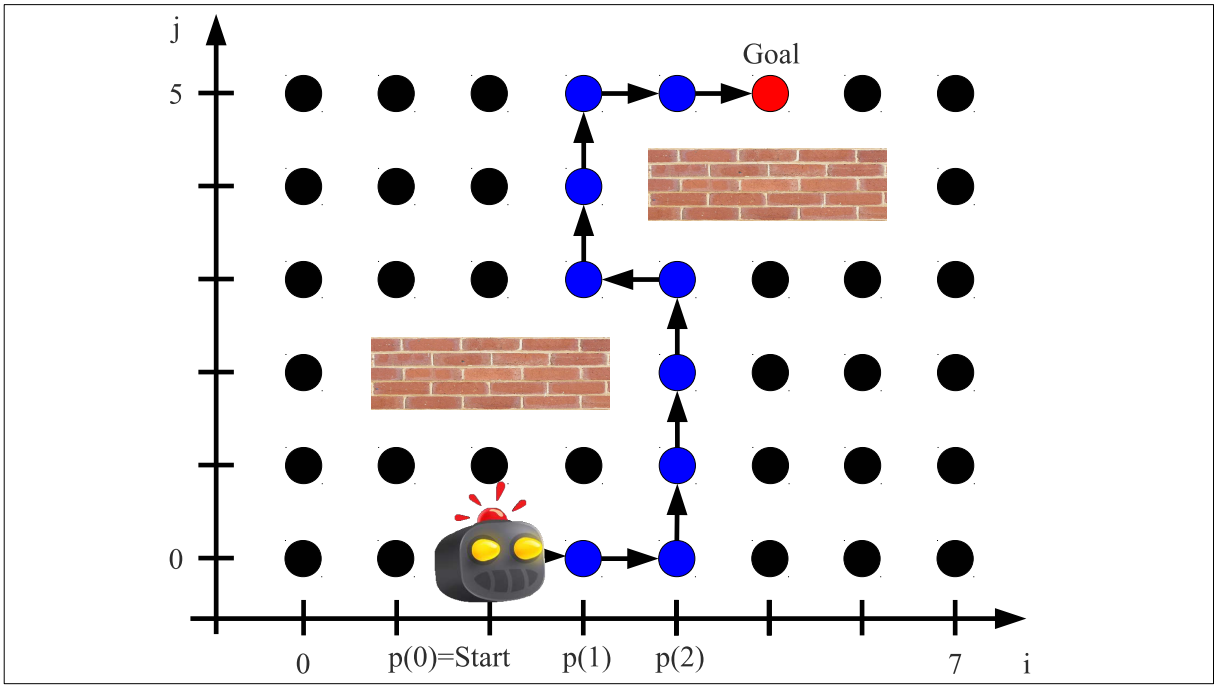

FIGURE 1.12 – Trajectoire planifiée dans le cas discret.

vérifie :

$$\forall t \in [0, T], \qquad c(trajectory(t)) \in [D^-, D^+] \tag{1.18}$$

Les contraintes cinématiques imposent des contraintes dites non-intégrables. Typiquement, ce sont des équations différentielles reflétant le modèle cinématique du robot considéré pour lesquelles il n'existe en général pas de solutions explicites. Entre autres, les contraintes cinématiques sont la vitesse linéaire, l'accélération linéaire, la vitesse angulaire, l'accélération angulaire, les contraintes pour éviter le glissement des roues...

Contraintes dynamiques

Nous considérons dans ce manuscrit deux types de contraintes dynamiques. La première est la continuité entre deux trajectoires. Celle-ci peut concerner la continuité sur la position, la vitesse, l'accélération...

Le niveau de continuité dépend alors de la dérivabilité de la trajectoire employée par le planificateur. La seconde contrainte vérifiée est la commandabilité du robot : les solutions déterminées doivent être transformables en commandes effectivement réalisables par le robot.

Autres contraintes

Bien entendu, les contraintes ne sont pas uniquement centrées sur le modèle de fonctionnement du robot. D'autres contraintes existent selon le domaine d'application, par exemple, des contraintes liées aux interactions avec des humains [Mainprice *et al.*, 2011].

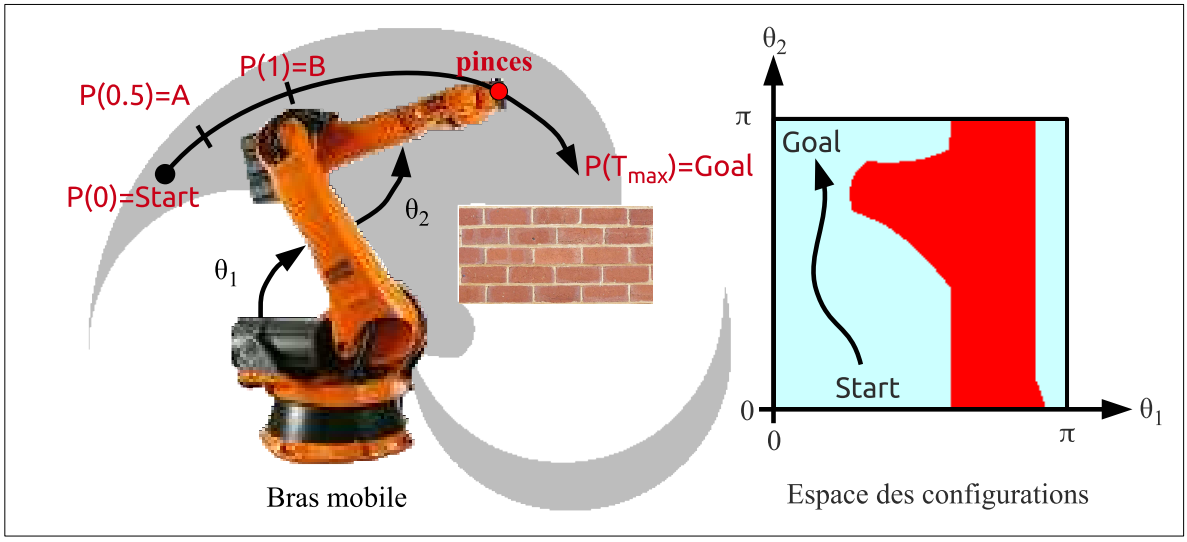

FIGURE 1.13 – Illustration d'une trajectoire planifiée dans l'espace des configurations et sa conséquence sur la position des pinces dans l'espace de travail.

1.7.1 Application au cas discret

Comme l'illustre la figure 1.14, lorsque les mouvements d'un robot dans le cas discret sont planifiés, le planificateur limite naturellement les successions de configurations effectivement réalisables. Par exemple, un robot ne peut se déplacer que dans le voisinage de Moore ou dans le voisinage de von Neumann. À partir du graphe ainsi généré, il existe de nombreuses approches permettant d'obtenir le meilleur chemin reliant un état de départ *Start* à un état d'arrivée *Goal* tels que l'algorithme de Dijkstra ou A*.

Dijkstra Le principe de cet algorithme est de diffuser le long des nœuds du graphe une fonction de coût qui évalue le coût du déplacement d'un nœud vers le suivant. Si cette fonction est, par exemple, la distance entre deux nœuds et que la diffusion commence à partir du nœud de départ, le chemin le plus court pour atteindre tous les nœuds du graphe peut être déterminé.

A* L'algorithme A* ne fait qu'ajouter une fonction heuristique pour n'explorer qu'une partie du graphe utile à la recherche de la solution optimale : classiquement, l'éloignement du nœud exploré à l'objectif sert d'heuristique minorante qui permet de garantir l'optimale au moment de l'arrêt de l'algorithme.

Illustration La contrainte de vitesse s'exprime en limitant les mouvements dans un certain voisinage, ici le voisinage de von Neumann :

$$\forall t \in [0, T_{max}], \qquad trajectory_{discrete}(t+1) - trajectory_{discrete}(t) \in \{(0,1),(1,0),(0,-1),(-1,0)\} \tag{1.19}$$

Prenons le second voisinage de l'exemple de la figure : selon cette contrainte, passer de la configuration $(2,0)$ à la configuration $(3,0)$ est acceptable. Par contre, passer de la configuration $(3,0)$ à la configuration $(5,2)$ n'est pas acceptable.

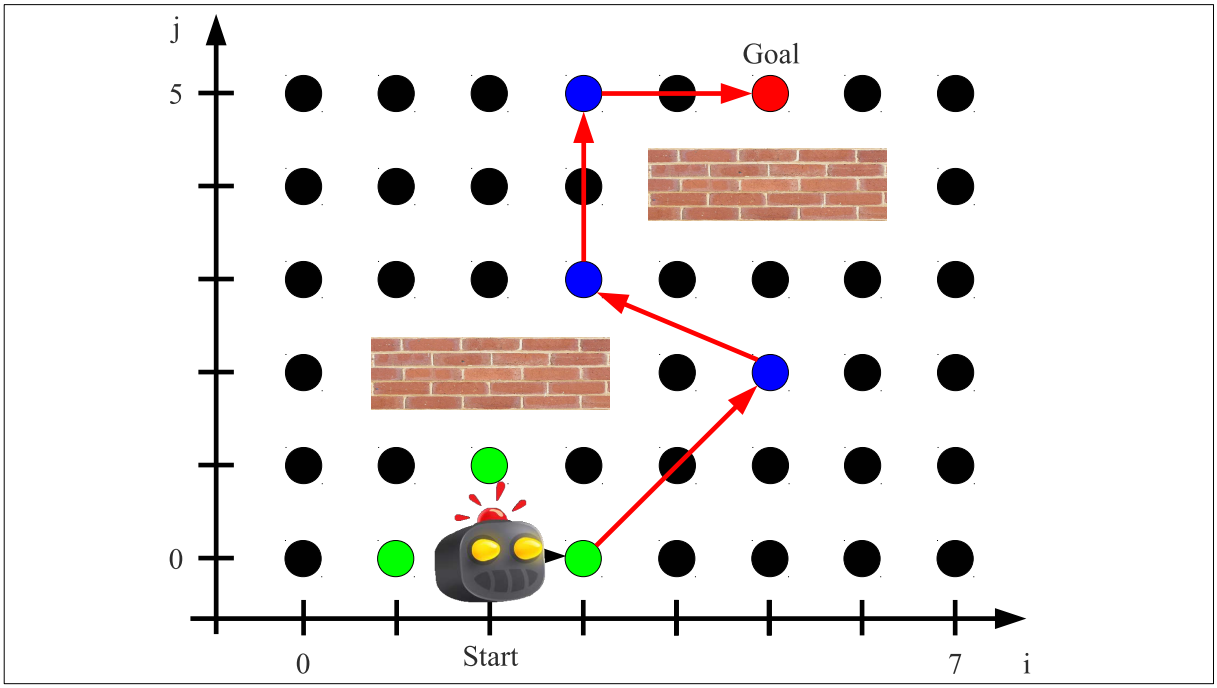

FIGURE 1.14 – Trajectoire planifiée dans le cas discret avec un voisinage de Von Neumann : le robot se déplace uniquement selon une suite de nœuds voisins les uns des autres.

Limites

Le graphe des déplacements possibles ne rend pas compte de toute la complexité des déplacements du robot. Il n'y a pas de vérification explicite des contraintes : tout repose sur le choix des déplacements faisables et la construction du graphe des déplacements. Le problème ne profite donc pas de tous les déplacements qui sont en réalité réalisables. Or, répercuter plus de possibilités de déplacement augmente la finesse du graphe des déplacements et donc le temps de calcul pour obtenir une solution. Enfin, générer une trajectoire discrète acceptable n'assure pas que les contraintes ne soient pas violées par le robot réel entre les points de passage.

1.7.2 Application au cas continu

Comme l'illustre la figure 1.15, le bras robotisé se déplace uniquement en continu le long d'une trajectoire. Les discontinuités induisant une accélération infinie sont donc interdites. De même, bien que cela soit plus difficile à illustrer, la vitesse de déplacement du robot est limitée par la vitesse de rotation des liaisons pivots. Il y a donc plusieurs manières de considérer une contrainte cinématique comme limiter la succession des positions d'un ou plusieurs points sur le robot ou imposer des valeurs admissibles aux degrés de liberté. Selon la manière de représenter le robot et de planifier, les approches pour la planification de trajectoire choisissent l'une ou l'autre des solutions.

Pour résumer, toute transition entre deux configurations n'est pas nécessairement acceptable. Ainsi, en ajoutant ces contraintes au fait que l'espace des configurations contient une infinité de points, les solutions mises en œuvre pour établir le chemin sont bien différentes du cas discret.

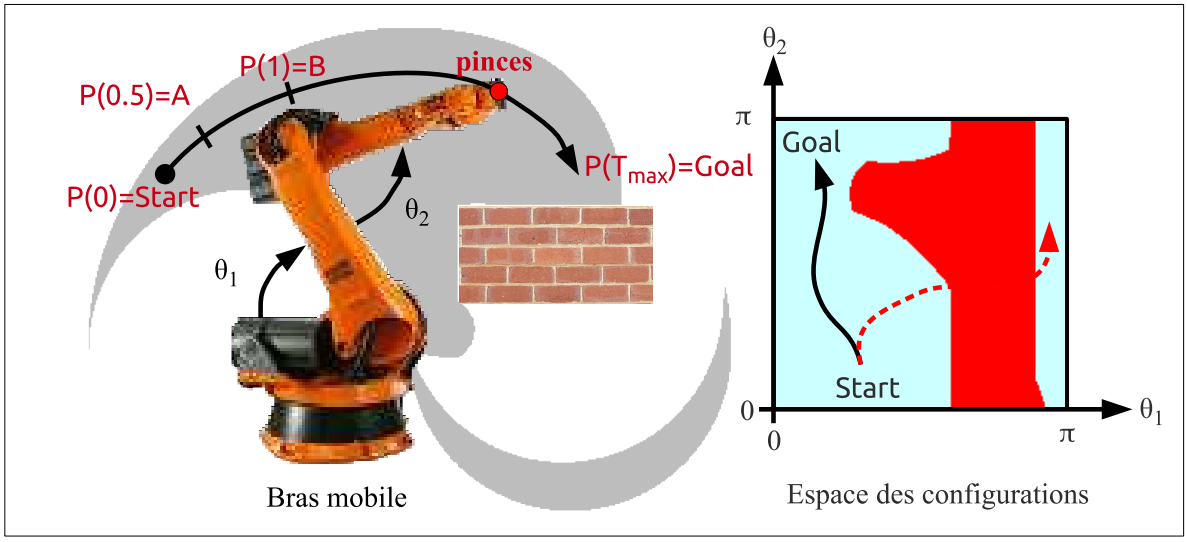

FIGURE 1.15 – Trajectoire planifiée dans le cas continu : le bras ne peut se déplacer qu'à une vitesse continue et finie dans l'espace de travail.

Limites

La principale faiblesse des approches continues vient du grand nombre de degrés de liberté du système étudié. L'espace des configurations possède alors de nombreuses dimensions. L'application des contraintes devient de ce fait plus compliquée : il est difficile de déterminer quelles sont les successions de configurations qui respectent réellement les contraintes. Ainsi, la principale limite de ces approches vient de l'impossibilité d'explorer le continuum de configurations et de la difficulté de transcrire les contraintes dans l'espace des configurations.

1.7.3 Critère d'optimisation

Dans un problème de planification de trajectoire, il y a très souvent plusieurs solutions permettant de rejoindre l'objectif *Goal* depuis un état de départ *Start* en respectant les contraintes. Cet ensemble est noté *Candidates*.

Je définis alors un critère permettant de discriminer les solutions entre elles. Il sera noté ρ. L'optimisation de la solution consiste à trouver, parmi les solutions candidates, la trajectoire $trajectory^*$ qui minimise le critère de recherche ρ :

$$trajectory^* = \underset{trajectory \in Candidate}{Argmin} \Big(\rho(trajectory)\Big) \tag{1.20}$$

1.8 Approches communes

Dans l'absolu, les problèmes de planification de trajectoire dans les approches discrètes et continues répondent aux mêmes définitions :

- un espace des configurations pour décrire les états d'un robot ;
- un espace de travail dans lequel le robot évolue ;
- un espace des configurations libres qui ne contient que les configurations ne provoquant pas de collision entre le robot et des obstacles ;

– un chemin dans l'espace des configurations ;
– une trajectoire dans l'espace de travail.

Ce qui varie, c'est la représentation de l'espace des configurations dans lequel les solutions sont établies et la représentation des contraintes.

1.8.1 Le problème de planification de trajectoire sous contraintes

Planifier des trajectoires faisables est fondamental en robotique. Ces trajectoires doivent être planifiées en prenant en compte la mécanique du robot, des environnements complexes, emplis d'obstacles statiques ou mobiles avec des formes variées. De plus, ce type de problème devient encore plus compliqué dès que des robots réels sont utilisés. Dans ce cas, les trajectoires doivent respecter des contraintes cinématiques et dynamiques, telles que le respect de limites sur la vitesse ou sur l'accélération : il est nécessaire que la séquence de déplacements générée par le planificateur de trajectoire soit exécutable par le système [Frazzoli *et al.*, 2002].

Les états du robots sont continus dans l'espace des configurations, duquel sont enlevées les configurations provoquant des collisions avec les obstacles. Toutefois, certaines de ces séquences de configurations peuvent être interdites dès que les contraintes sont appliquées. Ce problème est désigné comme étant un problème de planification de trajectoire sous contraintes cinématiques et dynamiques (*kinodynamic planning*).

1.8.2 Difficulté du problème de planification de trajectoire

Complexité

La complexité du problème de planification de trajectoire a été démontrée à l'époque où des solutions directes étaient recherchées. Ce problème est de la classe PSPACE-hard[Reif, 1987]. S'il faut n degrés de liberté pour décrire la mécanique du robot, l'espace des configurations est de dimension n. Comme l'a résumé [Frazzoli *et al.*, 2002], si les obstacles sont définis avec m contraintes polynomiales de degré d, la complexité en temps de l'algorithme est deux fois exponentielle sur n, polynomiale sur m (la littérature parle de complexité géométrique) et polynomiale sur d (complexité algébrique).

Abandon de la complétude

Si des contraintes non intégrables sont ajoutées au problème, par exemple des limites sur la vitesse linéaire ou sur l'accélération linéaire, la complexité devient encore plus importante, en particulier pour des problèmes présentant beaucoup de degrés de liberté. C'est pourquoi la propriété de complétude, c'est-à-dire obtenir une solution si elle existe quelle que soit la situation, est totalement abandonnée dans les approches directes pour la planification de trajectoires sous contraintes cinématiques et dynamiques. Dans ces approches, l'idée est plutôt de trouver des bornes aux temps de calcul.

1.8.3 Approches discrètes

D'un côté, les approches discrètes profitent d'un espace de recherche limité pour trouver efficacement une solution optimale. Dans le contexte d'applications en temps réel comme le jeu vidéo, où la faisabilité physique des solutions est peu importante, des solutions sont proposées [Botea *et al.*, 2004] ou [Duc *et al.*, 2008]. Toutefois, la complexité des approches discrètes augmente rapidement avec le nombre de configurations constituant les nœuds d'un

graphe de déplacement. C'est d'ailleurs sur ce graphe de déplacement et sur l'espace des configurations que repose la prise en compte des contraintes. Ainsi, deux alternatives sont possibles : soit le graphe tient compte des contraintes tout en maintenant un ensemble discret de configurations pour que les solutions soient applicables mais ces approches prennent le risque de s'interdire des configurations pourtant acceptables ; soit le graphe ne tient pas compte de toutes les contraintes et ces approches prennent le risque de générer des solutions non-acceptables.

1.8.4 Approches continues

De l'autre, les approches continues se focalisent sur les méthodes d'exploration de l'espace des configurations. Elles découvrent au fur et à mesure les solutions tout en vérifiant les contraintes et en optimisant la solution. Elles souffrent de la complexité de l'exploration : l'espace des configurations peut présenter de nombreux degrés de liberté. De plus, il n'est pas possible d'exprimer dans cet espace les contraintes non-intégrables. Des solutions plus simples sont recherchées mais il faut limiter le nombre de contraintes à vérifier, d'où une efficacité de ces approches souvent limitée à des problèmes de planification très locaux (engageant peu de contraintes, en particulier d'évitement d'obstacle).

L'état de l'art, dans le chapitre suivant, présente sous la forme d'un historique les principales approches pour la planification de trajectoires sous contraintes. Cet historique montre comment les approches présentes dans la littérature ont pu progressivement relier les approches continues et discrètes pour la planification de trajectoire afin de relever les défis de ce domaine.

1.9 Apports de la thèse

Dans ma thèse, je propose de me focaliser sur la manière de coupler la planification symbolique avec la planification de trajectoire. Les actions décidées dans les couches hautes ont pour conséquence le déplacement du robot. Pourtant, ces mouvements ne peuvent être réellement déterminés que par les couches basses de planification de trajectoire en cohérence avec l'action planifiée. De plus, pour maintenir le respect du modèle du robot, le processus de planification de trajectoire doit lui-même influencer le modèle de décision des actions. Je propose donc une représentation commune des mouvements dans les deux couches permettant des interactions à la fois descendantes et ascendantes entre les deux niveaux. Cette approche permet de réaliser ce que j'appelle un processus cognitif de planification de trajectoire sous contraintes. En m'inspirant de l'architecture utilisée dans des travaux récents comme RRT [LaValle et Kuffner, 2001], celle que j'ai retenue utilise des mécanismes de sélection/propagation. Avec ce modèle, on construit la trajectoire en rassemblant des morceaux générés par la couche de propagation. Le choix des morceaux à prolonger est décidé par la couche de sélection dans le but de générer un arbre d'exploration.

Couche de propagation La propagation génère des morceaux de trajectoire qui respectent les contraintes physiques s'appliquant sur le modèle du robot : c'est l'apport des approches continues. Dans cette couche, je formalise d'abord un espace des paramètres : un espace qui décrit tous les échantillons de trajectoire pouvant être formées. En adoptant les opérations ensemblistes utilisées en géométrie constructive de surface, cet espace est progressivement construit en tenant compte des contraintes cinématiques et dynamiques du problème. À la fin de cette étape, cet espace contient un ensemble d'échantillons ensuite sélectionnés en fonction des critères d'optimisation de l'application.

Couche d'exploration Je fournis ma propre couche de sélection/propagation, appelée DKP, qui est inspirée des approches discrètes. En jouant sur la durée des trajectoires, le niveau de propagation crée des échantillons pour la construction d'un arbre d'exploration de l'environnement. DKP utilise une variation de l'algorithme A* pour sélectionner les parties de l'arbre à prolonger en priorité et ainsi construire des trajectoires complexes. DKP fournit un comportement totalement déterministe : les résultats de DKP sont reproductibles.

Couche cognitive Le rôle de la partie cognitive est de construire un arbre de comportements de pilotage : un comportement de pilotage est constitué d'un ensemble de paramètres fournis par DKP. Pour faire interagir les couches cognitives et les couches de sélection/propagation, les comportements de pilotage agissent sur DKP et contrôlent donc la croissance de l'arbre d'exploration de DKP. Celui-ci guide en retour la croissance de l'arbre de comportements. Il y a donc deux niveaux d'interactions. Dans DKP, les couches de propagation et de sélection collaborent pour contrôler l'extension de l'arbre d'exploration. Dans l'approche cognitive, nous établissons les relations entre les comportements de pilotage et DKP. Les comportements de pilotage expriment des hiérarchies de tâches permettant de résoudre des situations usuelles de la planification de trajectoire : évitement d'obstacle, problème du U, problème du dépassement. Ces exemples seront développés au cours de la thèse. Enfin, j'utilise le formalisme TÆMS pour évaluer les trajectoires et décider de la meilleure action à effectuer.

1.10 Organisation de la thèse

Le manuscrit est organisé en quatre chapitres principaux.

Le Chapitre 2 présente les différentes approches pour la planification de trajectoire. Chaque section s'achève sur une analyse des propriétés que je recherche pour construire l'approche cognitive pour la planification de trajectoire. Il en ressort que les approches par échantillonnage explorent efficacement l'environnement grâce aux interactions entre les couches de propagation et de sélection : la première génère des morceaux de trajectoire qui respectent les contraintes et la seconde utilise ces échantillons pour bâtir un arbre d'exploration dans les parties atteignables de l'environnement. En revanche, il n'existe pas encore de solutions mêlant efficacement une approche cognitive avec le respect des contraintes cinématiques et dynamiques. Je remarque que les HTN permettent de construire des plans d'actions sous la forme d'arbres qui peuvent constituer une donnée commune avec les arbres d'exploration des approches de planification de trajectoire par échantillons. J'exploite ce rapprochement pour construire l'approche cognitive pour la planification de trajectoire. Cette approche requiert une maîtrise de toutes les étapes de la planification. C'est pourquoi j'ai conçu mes propres couches de propagation et de sélection permettant un tel contrôle.

Le Chapitre 3 est consacré au niveau de propagation. Ce niveau génère des morceaux de trajectoire qui respectent le modèle physique du robot. Pour garantir cette propriété, notre approche centrée sur le modèle de solution coupe le processus de génération en deux parties : tout d'abord, le modèle physique est formalisé dans le but de construire un espace des paramètres contenant toutes les valeurs des paramètres laissés libres dans les solutions. On indique ensuite comment délimiter cet espace en tenant compte des contraintes. Une fois l'espace construit, une recherche de solution peut être effectuée dessus. Cette méthode peut être utilisée avec des ensembles continus ou discrets de solution. Le point central de ce chapitre est l'instanciation de ces concepts pour des échantillons à base de quadratiques. Ces morceaux sont des solutions suffisantes pour le contrôle de robots à deux roues indépendantes. Dans ces échantillons, deux

paramètres sont laissés libres : l'espace des paramètres est alors représenté sous la forme d'une surface contenant toutes les solutions locales du problème. L'application des contraintes du problème telles que la vitesse, accélération ou l'évitement d'obstacles est réalisée grâce à des opérations géométriques. L'avantage est de maintenir une représentation complète de l'espace des paramètres jusqu'à l'étape d'optimisation. Un exemple de construction est donné en fin de chapitre.

Le Chapitre 4 est consacré à DKP, une nouvelle approche par échantillonnage pour la planification de trajectoire. Il génère un arbre d'exploration de l'environnement pour déterminer des trajectoires complètes. La couche de sélection est inspirée de l'algorithme A* pour sélectionner les meilleures branches à prolonger. DKP s'appuie sur le niveau de propagation du chapitre 3 pour générer de la diversité en échantillons. Cette diversité est basée sur la durée des trajectoires et la recherche de solutions alternatives dans l'espace des paramètres. De plus, il sert aussi à ne sélectionner que les parties réellement prolongeables de l'arbre d'exploration. Les solutions sont des trajectoires splines adaptées au contrôle de robot à deux roues indépendantes. DKP a la propriété d'être entièrement déterministe : les résultats sont reproductibles et son comportement est entièrement contrôlable. Ce chapitre propose également une étude des différents modes de fonctionnement de DKP. En plus des modes optimaux et gloutons issus du fonctionnement de A*, je présente le mode backtrack qui améliore l'efficacité de DKP en modifiant à la fois l'arbre de trajectoires et la structure de l'environnement en ajoutant des obstacles virtuels dans les minima locaux, ce qui permet de s'en extraire. L'adaptation de l'évolution de l'arbre d'exploration permet de faire le lien avec les couches de raisonnement : c'est à cet endroit que se place l'approche cognitive pour la planification de trajectoire sous contraintes.

Le Chapitre 5 est consacré à l'explication du fonctionnement de cette approche cognitive. L'élément clé est d'avoir les deux niveaux de sélection/propagation de DKP qui interagissent avec une couche cognitive dans le but de faire évoluer conjointement l'arbre d'exploration de l'environnement avec un arbre de comportements de pilotage qui décrit les actions du robots. J'obtiens des trajectoires correspondant aux comportements de pilotage, réalisables par des robots à deux roues indépendantes grâce aux échantillons du niveau de propagation. Les comportements de pilotage contrôlent le fonctionnement de DKP qui fait grandir un arbre d'exploration en suivant les paramètres identifiés dans ce chapitre. L'arbre d'exploration se développe dans l'environnement et permet en retour de faire évoluer l'arbre de comportements en fonction des portions explorées de l'environnement. Ce chapitre montre comment utiliser le formalisme TÆMS pour décrire et évaluer les solutions ainsi bâtis. Deux méthodes de guidage sont développées au travers d'exemples pour montrer les bénéfices de cette architecture.

Au final, nous avons une approche qui repose sur deux niveaux de collaboration : interactions entre la sélection et la propagation dans DKP et interactions entre DKP et la couche cognitive. Cette architecture est représentée par la Figure 1.16.

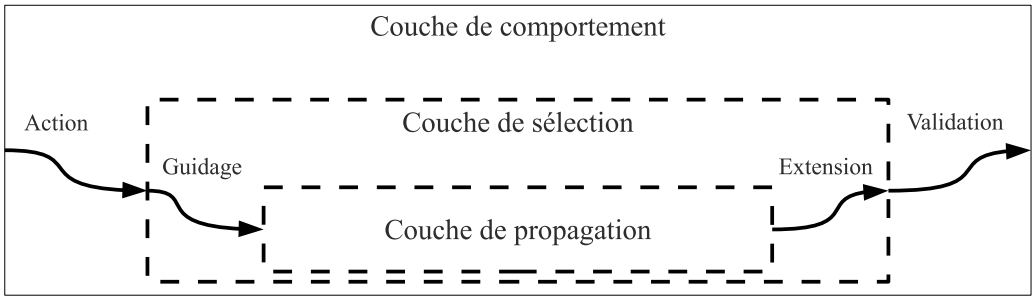

FIGURE 1.16 – Architecture de l'approche cognitive pour la planification de trajectoire développée dans cette thèse. Je me suis attaché à trouver les données communes à chacune des couches de l'architecture. La couche de comportement fournit les actions sous forme de paramètres à DKP. La couche de sélection de DKP guide le niveau de propagation qui, en retour, fournit les morceaux de trajectoires qui permettent l'extension de l'arbre de trajectoires. Enfin, les actions sont validées selon le développement de l'arbre de trajectoires.

Chapitre 2

État de l'art

Le problème abordé est le suivant : comment décrire le comportement de robots réels en tenant compte des contraintes physiques qui s'appliquent sur leurs déplacements ? Pour répondre à cette question, il faut donner des solutions aux équations du modèle cinématique du robot. Ensuite, il faut identifier les différents mécanismes de planification de trajectoire et leurs limites. Enfin, ce problème requiert un formalisme permettant de décrire les situations et les comportements impliquant des déplacements.

Modèle du robot Le cœur du problème concerne le déplacement de robots réels. Ceux-ci sont soumis à des contraintes imposées par leur modèle physique : leurs moteurs ne peuvent produire des accélérations infinies, leurs composants ont un poids, les roues glissent, le robot se déplace dans un environnement continu... Autrement dit, ils sont souvent incapables de produire certains mouvements auxquels nous aimerions les soumettre. Malgré cela, cet espace limité par leurs capacités de mouvements contient une infinité de trajectoires possibles. Remarquons tout de suite que je m'intéresse à des robots réels, ce qui signifie que les robots sont sujets à d'inévitables perturbations.

Trajectoires pour le robot Le domaine de la planification de trajectoire cherche donc à produire des solutions respectant le modèle du robot. Ainsi, pour bien choisir les déplacements, il faut vérifier les contraintes cinématiques, la continuité dans les trajectoires et leur transformation en commandes permettant le déplacement effectif du robot.

Optimisation de la trajectoire En plus de cela, une telle solution est souvent optimisée en fonction d'un ou plusieurs critères dépendant du domaine pour répondre à des besoins spécifiques : minimisation de la distance parcourue entre deux points, maximisation de la vitesse, économie sur l'usage de la batterie, optimisation de la sécurité en s'écartant au maximum des obstacles, par exemple pour obtenir des déplacements d'un robot mobile en intérieur favorisant les trajectoires au milieu des couloirs...

Approches discrètes pour la planification de trajectoire Les approches usuelles de planification de trajectoire issues du domaine de l'intelligence artificielle reposent sur un espace fini contenant des déplacements que le robot est capable d'exécuter. L'ensemble des déplacements sont par exemple constitués des mouvements dans le voisinage de von Neumann : aller en avant, en arrière, à gauche et à droite d'une unité de distance. À partir de ces déplacements, nous pouvons construire un graphe de toutes les positions atteignables par le

robot dans l'environnement considéré. Ensuite, il existe des approches bien connues pour établir le déplacement optimal en distance (Dijkstra [Dijkstra, 1959] ou A* [Hart *et al.*, 1968] pour ne citer que les plus connues). Dans ces travaux, nous supposons toujours qu'un déplacement est effectivement faisable physiquement. Or, dès que des contraintes du modèle physique sont introduites, un problème se pose : comment un robot mobile, lancé à 20cm/s, pourrait-il instantanément tourner sur lui-même de 90 ° ?

De plus, dès que le problème est projeté dans le domaine continu avec contraintes physiques, les approches usuelles de l'intelligence artificielle ont du mal à retranscrire le coût réel des déplacements : chaque déplacement possède une durée et n'est pas toujours faisable. De surcroît, en ne considérant qu'un ensemble discret de déplacements, ces approches prennent le risque de ne pas couvrir toutes les possibilités de contournement des obstacles. En tenant compte d'un plus grand ensemble de déplacements, le graphe des positions atteignables gagne en précision. Mais, en contrepartie, le temps d'exécution de ces planificateurs explose. *In fine*, ces approches ne sont donc pas assurées de produire le chemin optimal et/ou faisable par le robot.

Approches continues pour la planification de trajectoire

À l'inverse, les approches continues, qui partent du modèle cinématique, visent à établir localement des trajectoires satisfaisant toutes les contraintes. Il faut donc établir l'ensemble des déplacements possibles puis trouver une solution en optimisant un critère de recherche. La planification de trajectoire dans ce cadre peut être résumée de la manière suivante :

1. le choix d'un modèle de solution ;
2. ensuite l'application des contraintes sur cette solution, limitant les formes possibles ;
3. enfin, l'optimisation de la solution, en faisant varier les degrés de liberté pour favoriser un critère de recherche.

Les approches continues usuelles ne font qu'appliquer ce schéma plus ou moins explicitement. Dans ces approches, on y retrouve la mise en équation directe des éléments sus-cités, dans des approches telles que [Milam *et al.*, 2000] ou [Defoort, 2007]. Ceux-ci sont fournis ensuite à un programme de résolution de problèmes d'optimisation non-linéaires. Les autres approches personnalisent plusieurs de ces éléments : par exemple, le programme de résolution pour le problème de déplacement (champ de potentiel [Khatib, 1986]). La prise en compte des contraintes peut également se faire de manière progressive comme dans Elastic Strips [Brock et Khatib, 1997] ou Elastic Bands [Quinlan et Khatib, 1993]). Enfin, il existe des méthodes variées d'exploration de l'espace des solutions dont font partie les approches par échantillonnage telles que RRT [LaValle et Kuffner, 2001].

La principale faiblesse de ces approches vient du grand nombre de degrés de liberté présents dans le modèle des robots étudiés. L'espace des déplacements possibles est, dans ce cas, coûteux à explorer à cause de sa dimension égale au nombre de degrés de liberté. De plus, cette difficulté augmente avec l'ajout des contraintes non-intégrables sur le modèle physique du robot.

Dans les approches continues, la complexité est donc un frein à l'exploration de environnement, en particulier lorsqu'il présente beaucoup de contraintes d'évitement d'obstacle. C'est pourquoi les approches continues travaillent sur le guidage et la qualité des solutions. Nous pouvons citer, par exemple, le guidage par roadmap, l'exploration aléatoire de l'environnement (RRT [LaValle et Kuffner, 2001]) ou l'affinage des solutions (RRT* [Karaman et Frazzoli, 2010]).

Description des situations et des comportements de pilotage

Les plus hauts niveaux de raisonnement dans les applications robotiques doivent gérer le problème général de la planification de tâches complexes en décrivant : les robots et leurs capacités, l'environnement et les savoir-faire liés aux déplacements.

Les approches usuelles sont basées soit sur une description de type STRIPS, comportant une description du monde et de toutes les actions possibles, soit sur une description en réseaux hiérarchiques de tâches (HTN pour Hierarchical Task Networks) pour décomposer un objectif en tâches composées. Des algorithmes existent pour déterminer les séquences d'actions à effectuer pour résoudre des tâches spécifiques. Le problème est de pouvoir exprimer à ce niveau de raisonnement les contraintes de mouvements s'appliquant aux robots afin de ne pas décider des actions qui ne sont pas réalisables. C'est pourquoi une approche comme celle de C. Reynolds pose des problèmes qui sont détaillés plus tard. Toutefois, Reynolds [Reynolds, 1999] propose des comportements de pilotage pour transmettre des objectifs depuis les couches cognitives vers les couches de planification de mouvements. C'est cette idée que je veux réutiliser dans mon approche cognitive pour la planification de trajectoire sous contraintes.

Organisation de l'état de l'art

La planification de trajectoire dans le domaine continu et dans le domaine discret posent deux problèmes identiques : elles sont complémentaires lorsque leurs forces et leurs faiblesses sont mises en vis-à-vis. N'oublions pas qu'un enjeu de la planification de trajectoire est de satisfaire les besoins des applications en temps réel, ce qui implique aussi de limiter la complexité du planificateur. Je montre comment les architectures ont concilié les points suivants :

- garantir sur le temps de calcul et l'obtention d'un résultat pour planifier dans un contexte temps réel ;
- explorer au mieux l'espace des configurations ;
- respecter les contraintes et le modèle de fonctionnement du robot ;
- s'assurer de la bonne application des commandes.

Ces points seront repris lors de la présentation de chaque approche pour la planification de trajectoire.

L'état de l'art présente une vision quasiment chronologique de l'évolution des algorithmes de planification de trajectoire face à l'enjeu de la complexité des robots et de la prise en compte des contraintes. Comme le montre [Choset, 2005], les approches pour la planification de trajectoire ont évolué au cours des années : les approches directes durant les années 1980, les approches découplées durant les années 1990 et les approches par échantillonnage et leurs améliorations des années 2000 jusqu'aujourd'hui. Ainsi, l'état de l'art présente quelques approches révélatrices de chacune de ces époques.

Dans ce chapitre, nous présentons les différentes méthodes pour effectuer la planification de trajectoire dans un contexte temps réel. Plusieurs défis se présentent : tout d'abord, il faut avoir des garanties sur l'obtention de résultats, en particulier des garanties sur les temps de calcul, pour planifier dans un contexte temps réel. Ensuite, l'efficacité de la planification de trajectoire repose sur sa capacité à explorer l'espace des configurations. Le planificateur doit assurer le respect les contraintes et le modèle de fonctionnement du robot. Enfin, il faut s'assurer de la bonne application des commandes.

Ce sont des propriétés qui devront se retrouver dans l'approche cognitive pour la planification de trajectoire. La littérature présente de nombreuses solutions dont nous pouvons nous inspirer et qui possèdent au moins une partie de ces propriétés. Toutefois, aucune d'entre elles n'offre de

véritable cadre formel pour une approche cognitive de la planification de mouvement. Ainsi, une dernière section montre les tentatives pour apporter des couches cognitives dans la planification de trajectoire. Pour construire l'approche cognitive pour la planification de trajectoire présentée ici, deux autres éléments sont évaluées : la compatibilité du planificateur avec des données « gros grain » est évaluée : chemins pré-calculés, changements de paramètres décidés à haut niveau. La capacité d'ajouter de nouvelles contraintes est également une propriété recherchée en vue de modifier le comportement du planificateur ou l'adapter au domaine appliqué ou au type d'environnement.

2.1 Les approches directes

Reprenons le cheminement pour résoudre un problème de planification de trajectoire :
- choix d'un modèle de solution ;
- application des contraintes sur cette solution, limitant les formes possibles ;
- optimisation de la solution, en faisant varier les degrés de liberté pour favoriser un critère de recherche.

Champs de potentiel

L'approche par champs de potentiel [Khatib, 1986] se retrouve directement chacune de ces composantes. Elle s'applique parfaitement à un monde continu. L'évitement d'obstacles est traité directement au niveau de la commande. Ainsi la navigation par champs de potentiel consiste en la navigation d'un robot le long d'un gradient défini dans tout l'espace à explorer. Ce gradient est engendré par les obstacles et le point but.

Soit q l'état courant du robot (sa position en $(x; y)$ par exemple). La force artificielle qui le guide est générée par le gradient de la somme des potentiels s'exerçant sur le robot :

$$F(q) = -\nabla U(q) = \frac{\partial U}{\partial x}\frac{\partial U}{\partial y^T} \tag{2.1}$$

Ce gradient est déterminée par deux type de potentiels :

Les potentiels attracteurs sont notés :

$$U_t(q) = \frac{1}{2}k_t \times \|q - q_t\|^2 \tag{2.2}$$

avec k_t un gain en position. Ces potentiels attracteurs induisent une pente descendante vers un objectif situé en $q_t = (x_t; y_t)$.

Les potentiels répulsifs sont notés :

$$U_r(q) = \frac{1}{2}\eta(\frac{1}{\rho} - \frac{1}{\rho_0})^2$$
$$\text{si } \rho \leq \rho_0,\ U_r(q) = 0$$

η est un gain et ρ_0 est la distance limite d'influence de l'obstacle.
$\rho = \|q - q_r\|$ désigne la plus petite distance euclidienne entre le robot et l'obstacle. Ils induisent une pente ascendante et rendent coûteux le passage par la zone où est définie un tel potentiel.

L'analogie avec la physique provient de l'utilisation de champs où l'apparition d'une distance euclidienne rappelle les champs rencontrés en physique newtonienne. Le champ résultant est donné par la somme des forces attractives et répulsives :

$$U(q) = \sum U_t(q) + \sum U_r(q) \tag{2.3}$$

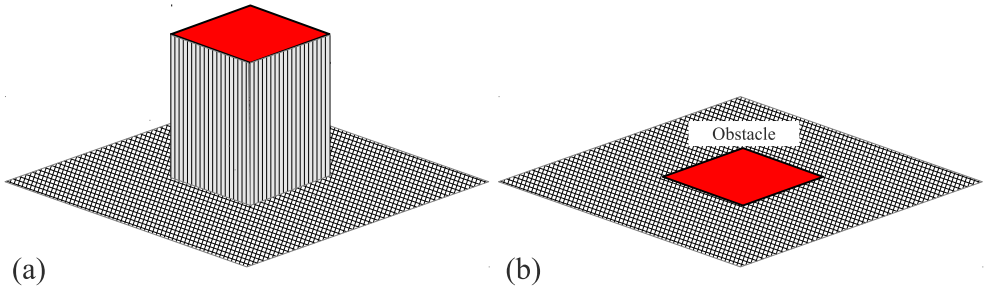

FIGURE 2.1 – Image présentant un potentiel artificiel (a) engendré par un obstacle de forme carrée (b), selon [Khatib, 1986].

Ce type de navigation peut être représenté par un champ de vecteurs. Chaque vecteur est alors orienté dans la direction de navigation à favoriser. Celui engendré par un obstacle est représenté par la figure 2.1.

Les obstacles manipulés devant être représentés de manière analytique, seules quelques formes peuvent être réellement employées : parallélépipède, cylindre... De la même manière que les obstacles fixes, les potentiels pouvant être superposés, il est possible de gérer l'évitement d'obstacles mobiles en plaçant des champs de potentiel sur des points judicieusement choisis de sa trajectoire.

Le vecteur force résultant sert à commander le robot. La réitération de ce processus, c'est-à-dire les calculs successifs des champs de potentiel dans les états q successifs, permet au robot de naviguer en évitant les obstacles et de construire une solution complète.

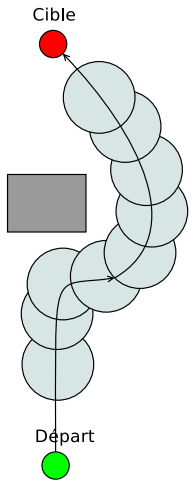

FIGURE 2.2 – Un chemin proposé par une navigation le long des champs de potentiel.

Critiques

La répétition de ce processus de planification de trajectoire rend cette approche réactive aux changements dans l'environnement. Ce procédé rend compte de beaucoup de difficultés usuellement rencontrées en planification de trajectoire.

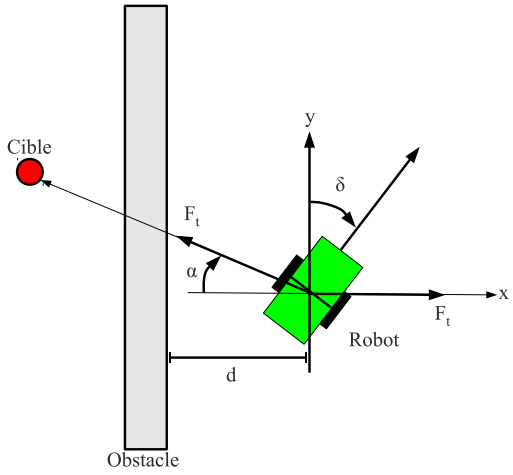

FIGURE 2.3 – Une situation généralisée de l'application des champs de potentiel.

Stabilité du modèle [Koren et Borenstein, 1991] montrent que les approches à base de champs de potentiel souffrent de nombreux défauts. Soit la situation représentée par la figure 2.3 qui montre le cas général de la commande d'un robot dont la stabilité est étudiée.

Soit $\Omega = k[\delta - \theta]$ la commande de direction avec k le gain sur la commande, θ l'angle du robot et δ l'angle commandé. Partant d'un modèle simplifié de voiture, le déplacement d'un tel robot peut être vu à travers la commande du moteur de direction dont le modèle différentiel est :

$$\tau\dot{\omega} + \omega = \Omega \tag{2.4}$$

où τ est la constante de temps du moteur.

Une fois appliqué le modèle de commande de direction, cette équation différentielle devient alors :

$$\tau\ddot{\theta} + \dot{\theta} + k \times \theta = k \times \delta \tag{2.5}$$

[Koren et Borenstein, 1991] déterminent la stabilité de ce système en présence d'une cible à atteindre provoquant une force attractive F_{ct} et d'un obstacle provoquant une force répulsive F_{cr} Un angle α oppose les deux forces (voir la Figure 2.3).

L'angle commandé δ dans cette situation est :

$$\tan\delta = \frac{F_{cr} - F_{ct}\cos\alpha}{F_{ct}\sin\alpha} \tag{2.6}$$

Appliquée à l'équation différentielle provenant du modèle de fonctionnement du moteur, [Koren et Borenstein, 1991] obtiennent le critère de stabilité suivant :

$$1 - NV(\tau + T) > 0 \tag{2.7}$$

où :

- V est la vitesse du robot ;
- N est une constante proportionnelle à la largeur du robot et inversement proportionnelle à la distance aux obstacles/cibles ;
- T est le délai entre deux prises de mesures.

Ainsi, un tel système n'est pas stable lorsqu'il est appliqué au déplacement d'un robot circulant à grande vitesse et en prenant des mesures espacées dans le temps. Ce phénomène est particulièrement présent lorsque des obstacles sont proches. C'est pourquoi les champs de potentiel ne sont dans la pratique utilisés qu'avec des robots circulant à faible vitesse tout en nécessitant un grand nombre de prises de mesures qu'un système réel ne peut facilement offrir (contrairement à la simulation). Cette difficulté peut être retrouvée dans beaucoup de planificateurs de trajectoire : en effet, les approches directes ne peuvent pas fonctionner sur un horizon de planification élevé, afin de ne pas être confronté à trop de contraintes d'évitement d'obstacles. Il faut donc réitérer ce processus de planification : cette répétition est ici la source d'instabilité.

Autres défauts De plus, [Koren et Borenstein, 1991] identifient de nombreux défaut dont le principal est que les potentiels artificiels ne peuvent se sortir des situations pièges provoquées par la présence de minima locaux (tel que l'obstacle en U montré dans la figure 2.4) de l'aveu même de [Khatib, 1986]. La correction passe par l'emploi d'une approche découplée à base de planificateur global puis de planificateur local [Krogh et Thorpe, 1986] qui calcule un chemin évitant ces minima locaux (si un tel chemin existe). Ils recensent d'autres défauts tels que l'absence de solution permettant le passage entre deux obstacles rapprochés, l'apparition d'oscillations en présence d'obstacles, la présence d'oscillations lors de la navigation dans des passages fins (typiquement des corridors) et le risque de divergences numériques provoquées par $\frac{1}{\rho}$ à mesure que le robot se rapproche de l'obstacle.

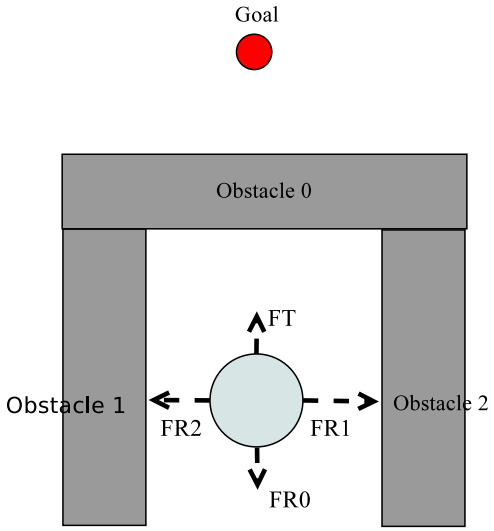

FIGURE 2.4 – Minimum local provoqué par un obstacle en forme de U, problème soulevé par [Khatib, 1986].

2.1.1 Programmation non-linéaire

Parmi les autres approches, la programmation non-linéaire présente l'intérêt de ramener le problème de planification de trajectoire à une définition entièrement mathématique, comme dans [Defoort, 2007]. Le cheminement pour résoudre un problème de planification de trajectoire peut alors être directement traduit en un problème non-linéaire. Il faut d'abord faire le choix d'un

modèle de trajectoire p présentant n degrés de liberté, notés $\alpha_0, \ldots, \alpha_{n+1}$. Sur une trajectoire sont vérifiées les conditions de continuité à partir d'une situation initiale. La solution globale est vue comme une succession de solutions locales. La situation initiale d'un morceau est alors basée sur l'état final du morceau précédent. Les n degrés de liberté de la trajectoire constituent les variables continues du programme non-linéaire. Chacune des n variables est initialisée avec une configuration acceptable du problème (qui vérifie les contraintes).

Ce type de problème est confié à un programme de résolution de problèmes d'optimisation non-linéaire, tel que CFSQP dans le cas général [Lawrence *et al.*, 1994] pour les problèmes d'optimisation non-linéaire à variables continues sous contraintes non-linéaires ou BLMVM [Benson et More, 2001] de la librairie AMPL [Fourer *et al.*, 2002] dans le cas plus simple des problèmes d'optimisation non-linéaire à variables continues bornées. Celui-ci explore l'espace des configurations en faisant varier les variables $\alpha_0, \ldots, \alpha_{n+1}$ et le temps le long de la trajectoire $p_{\alpha_0,\ldots,\alpha_{n+1}}$. Le but de ce programme est de minimiser une fonction objectif ρ calculée à partir de la solution $p_{\alpha_0,\ldots,\alpha_{n+1}}$. Les contraintes du problème, notées $c(t)$, sont vues à travers une fonction $f_c(p(t)) \leq 0$ validant ou non chacune des contraintes pour tout instant t du problème (une discrétisation est appliquée sur le temps pour limiter le nombre de vérifications). Durant la recherche de solutions, à chaque nouvel ensemble de valeurs $\alpha_0, \ldots, \alpha_{n+1}$, les contraintes sont ainsi vérifiées, limitant du coup l'espace des solutions.

Comme dans les potentiels artificiels, le mécanisme de recherche est une descente de gradient, ici généralisée à tout type de contraintes exprimable dans le programme de résolution.

Critique

La principale critique faite à cette approche est que le mécanisme de résolution repose entièrement sur le fonctionnement du programme permettant la résolution de problèmes non-linéaires à variables continues sous contraintes non-linéaires (vitesse, accélération...). Pour ce type de problème, à moins que les contraintes présentent un minimum global dans une région commune de l'espace des configurations, il est difficile d'obtenir une solution optimale. En effet, de tels problèmes peuvent présenter des solutions certes bonnes mais souvent sous-optimales. Tout l'enjeu est de sortir de ces minima locaux provoqués par les fonctions de contraintes. Par exemple, CFSQP utilise un bruit aléatoire sur les variables du problème pour sortir la solution d'un minimum local, rendant difficilement prévisible la solution qui va être obtenue. Pour initialiser correctement les variables du problème, il faut trouver une configuration qui vérifie les contraintes, or, il faut pour cela déterminer l'espace des configurations avant de commencer à résoudre le problème. Selon la topologie de l'espace des solutions, le choix de la configuration initiale va influencer le type de solutions vers lequel le programme va converger : il faut donc estimer correctement la première solution pour espérer obtenir une solution optimale, ce qui revient à vouloir de nouveau résoudre le problème à l'avance. Il y a donc un besoin de guidage à haut niveau de la planification de trajectoire qui est progressivement apparu dans la littérature, comme en témoigne [Krogh et Thorpe, 1986]. Un résumé des critères que je recherche pour créer mon approche cognitive est donné dans le Tableau 2.1.

2.2 Les approches découplées

Les approches découplées séparent usuellement les problèmes de planification de trajectoire sous contraintes cinématiques et dynamiques en deux problèmes successifs. D'abord, calculer un chemin qui prend en compte une partie des contraintes du problème (classiquement : les

Propriété	Approches directes
Planification en temps réel	Oui, méthode réactive
Exploration de l'espace des solutions	Non, sensible aux minima locaux
Respect du modèle du robot et des contraintes	Oui, dans la limite de l'existence de solutions au modèle considéré
Application directe des commandes	Oui
Compatibilité avec un planificateur de chemin	Oui, par des points intermédiaires évitant les minima locaux
Ajout de nouvelles contraintes	Difficile, à transcrire sous forme de champs de potentiel

TABLE 2.1 – Propriétés attendues d'une approche cognitive pour la planification de trajectoire et solutions apportées par les approches directes.

obstacles). Deuxièmement, adoucir le chemin en prenant les contraintes restantes pour rendre admissibles les solutions ce qui permet de les appliquer aux robots.

Ces approches exploitent des méthodes efficaces pour planifier des trajectoires au niveau local, bien qu'elles ne puissent pas être étendues. Pour améliorer le processus d'exploration de l'environnement, les approches discrètes pour la planification de trajectoire fournissent quant à elles des solutions pour des problèmes plus étendus quand il s'agit de tenir compte des obstacles.

Modèle

Les approches usuelles pour marier les deux domaines se basent sur des modèles hiérarchiques comme dans OpenSteer [Reynolds, 1999]. Reynolds identifie un comportement de déplacement comme un modèle Top Down en trois couches, illustré par la figure 2.5 :

1. La sélection d'action, c'est-à-dire le suivi d'une stratégie.
2. La conduite, c'est-à-dire la détermination des actions à effectuer, est partagée entre la planification de chemin, c'est-à-dire le chemin que le robot va suivre dans l'environnement et la planification de trajectoire, qui correspond à la détermination des mouvements à effectuer pour parcourir ce chemin.
3. La locomotion, le moyen d'exécuter les actions de mouvement.

Ce modèle est illustré par l'exemple de la figure 2.6, qui montre la succession de réflexions nécessaires à l'établissement d'un déplacement dans le cadre d'une maison.

Découplage chemin/trajectoire

Dans [Jaesik et Eyal, 2009], les auteurs soulignent les problématiques de ce type de solution. L'efficacité de ce type d'approche, en particulier les variantes autour des champs de potentiel et des bandes élastiques, est expliqué par le fait qu'elles sont généralement fortement spécialisées pour un type particulier de problème de planification sous contraintes cinématiques et dynamiques. Ce type d'approches fournit également des bornes sur les temps de calcul, ce qui permet leur mise en œuvre dans un cadre opérationnel et ce qui explique leur usage répandu au cours des années 90.

Ainsi, les approches discrètes s'inscrivent préférentiellement dans le niveau supérieur de conduite qui fournit des objectifs aux approches continues pour déterminer les trajectoires.

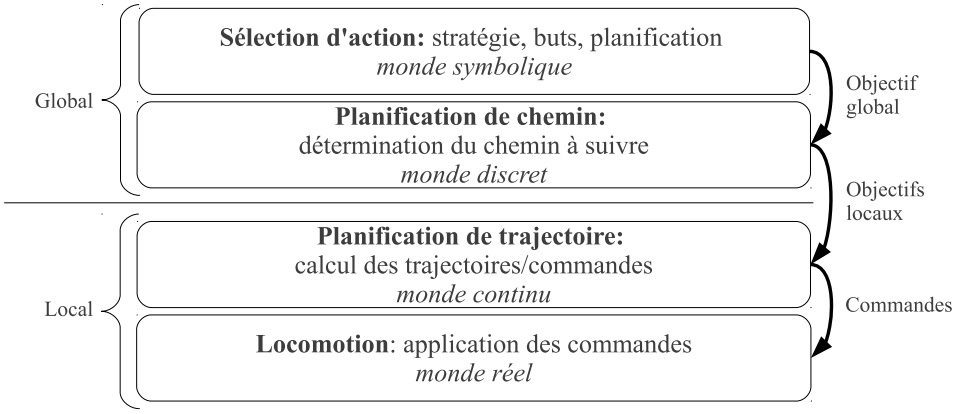

FIGURE 2.5 – Modèle Top Down établi par Craig Reynolds illustrant la succession de raisonnements nécessaires à l'établissement d'un déplacement : une première étape détermine l'action, c'est-à-dire l'objectif et les conditions du déplacement à réaliser. Ces informations sont fournies à l'étage de planification de chemin qui établit le chemin à suivre pour atteindre l'objectif. Ensuite, ce chemin est fourni à l'étape de planification de trajectoire qui établit les trajectoires à effectuer pour suivre ce chemin. Enfin, la couche de locomotion exécute les trajectoires ainsi déterminées, éventuellement avec un suivi de trajectoire, pour réaliser ce mouvement global et atteindre l'objectif désiré.

La locomotion est décrite comme une sous-couche dans laquelle les trajectoires sont traduites en commandes pour le robot. L'approche continue modélise et cache le modèle physique du robot vis-à-vis des approches discrètes. Une telle séparation souffre d'un défaut important : avec ce type de modèle descendant, on peut aboutir à des situations infaisables pour le robot.

Les approches découplées présentent des difficultés à résoudre des problèmes compliqués, en particulier présentant beaucoup de degrés de liberté. De plus, ces approches souffrent de problèmes d'incomplétude : à partir du moment où le chemin initial n'est pas généré en tenant compte de toutes les contraintes du problème, l'étape d'affinage du chemin peut échouer à satisfaire les contraintes restantes. Pire, l'ensemble peut échouer à fournir une solution, même si elle existe.

2.2.1 Roadmaps et segmentations de l'environnement

Le principe des algorithmes de génération de roadmaps est de trouver des points de connexion dans l'espace des configurations libre C_{free}. Ces configurations constituent alors les nœuds d'un graphe, auxquels sont également joints les configurations finales et initiales. L'idée est alors de déterminer la connexion entre ses configurations en déterminant un chemin dans le graphe. Le déplacement de configuration en configuration le long du chemin déterminé peut alors être effectué par l'utilisation d'une approche directe, en considérant que le problème est suffisamment simple. La recherche du chemin peut s'appuyer sur des méthodes du type Dijkstra tant que une fonction de coût peut être exprimée pour aller de configuration en configuration.

Critique

L'avantage par rapport aux premières approches découplées est que le graphe s'appuie sur des configurations atteignables. Toutefois, il n'y a pas de prise en compte des contraintes

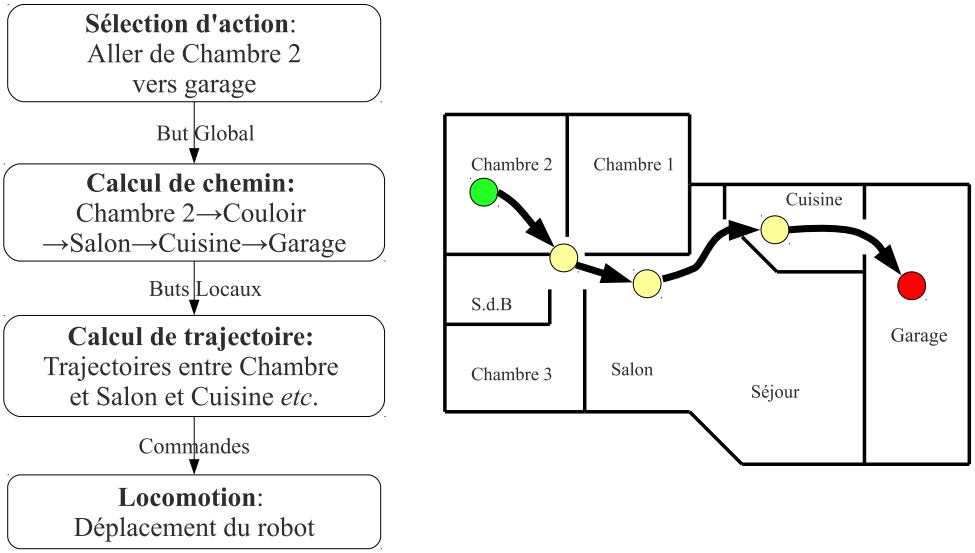

FIGURE 2.6 – Étapes de raisonnement à effectuer pour réaliser un déplacement selon le modèle Top Down de Craig Reynolds. Un robot est situé dans la chambre 2 ; le planificateur décide de l'objectif : aller au garage. Le niveau de planification discrète décide du chemin : passer par le couloir puis le salon, la cuisine et enfin le garage. Ensuite, la couche de planification continue établit les trajectoires entres ces points de passage : trajectoire entre la chambre 2/couloir, couloir/salon, salon/cuisine et cuisine/garage. Enfin, le niveau de locomotion effectue le déplacement effectif du robot depuis la chambre 2 jusqu'au garage en suivant les trajectoires déterminées par le niveau précédent.

cinématiques non intégrables : c'est le travail de l'approche directe de les respecter. Cela implique une nouvelle fois que le chemin ainsi déterminé n'est pas forcément totalement applicable, à moins d'avoir la certitude que les configurations intermédiaires sont des états acceptables. De plus, la complexité de cette approche repose essentiellement sur la construction de la roadmap.

Construction des roadmaps

Graphes de visibilité Les graphes de visibilité, proposés par Nilsson en 1969, constituent l'une des plus anciennes approches pour la construction de roadmap. L'idée est que le chemin le plus court pour aller d'une configuration à une autre passe par les sommets des obstacles polygonaux placés dans l'environnement. Pour générer ce graphe, il faut relier avec des droites tous les sommets des obstacles entre eux, sans que ces droites ne coupent d'autres obstacles. Un graphe est alors obtenu en reliant les sommets entre eux, comme illustré par la Figure 2.7.

Décomposition cellulaire L'idée de la décomposition cellulaire est de découper l'espace des configurations libres C_{free} en morceaux simples (N-cubes ou carrés selon les dimensions de l'espace). Si la décomposition est exacte, l'union de ces morceaux permet de retrouver l'espace des configurations. Un graphe de connexion peut alors être décrit à partir des côtés des cellules issues de la décomposition.

Le Quadtree est une méthode de pavage de l'espace des configurations avec des carrés. L'idée est de subdiviser une surface en 4 carrés. Chaque carré est lui-même subdivisé en 4 carrés jusqu'à

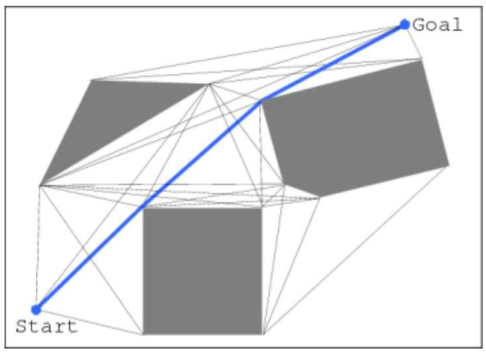

FIGURE 2.7 – Plus court chemin obtenu à partir d'un graphe de visibilité [Kim et Kim, 2009].

atteindre une profondeur voulue. Le résultat est alors un pavage uniforme de la surface. Cela donne un pavage tel que représenté par la Figure 2.8.

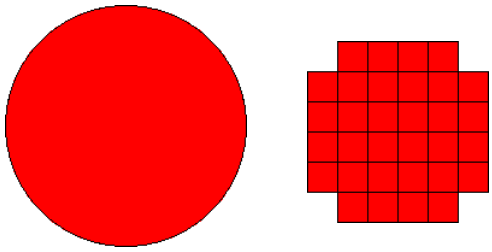

FIGURE 2.8 – Un disque et son pavage uniforme de 8×8 carrés, seuls les carrés présentant une intersection au moins partielle avec le disque sont conservés.

Toutefois, la subdivision peut n'être faite que sur les carrés ne présentant qu'une intersection incomplète avec la surface. Ainsi, le Quadtree fait alors un pavage uniquement sur le bord de la surface, comme le montre la Figure 2.9.

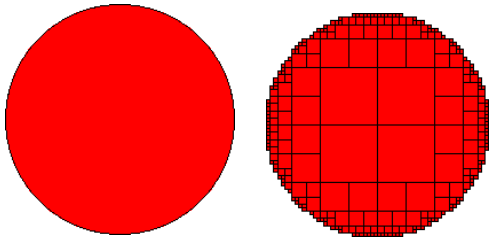

FIGURE 2.9 – Un disque et les carrés issus du Quadtree de profondeur 6.

Graphes de Voronoï Une autre manière de construire une roadmap est d'utiliser des diagrammes de Voronoï [Takahashi et Schilling, 1989] : ceux-ci ont la propriété de pouvoir maintenir les robots à distance des obstacles (voir Figure 2.10[4] et Figure 2.11). La méthode pour générer ces figures est donnée dans la Section 5.4. En effet, les chemins composant la roadmap sont équidistants des frontières des obstacles et du bord de l'environnement exploré. Cette distance aux obstacles permet donc de fournir naturellement une bonne roadmap servant

au guidage du robot, validant naturellement la distance de sécurité. De surcroît, cela permet également au planificateur de trajectoire de dévier du chemin pour aller dans l'espace libre. Les diagrammes de Voronoï sont même désormais générés très rapidement grâce à un usage détourné des GPU équipant les machines récentes [Nielsen, 2008].

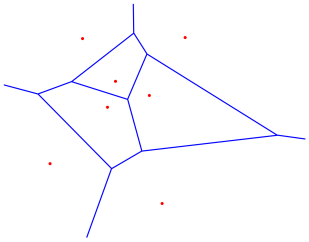

FIGURE 2.10 – Naviguer sur un diagramme de Voronoï revient à circuler sur les médianes entre deux points obstacles.

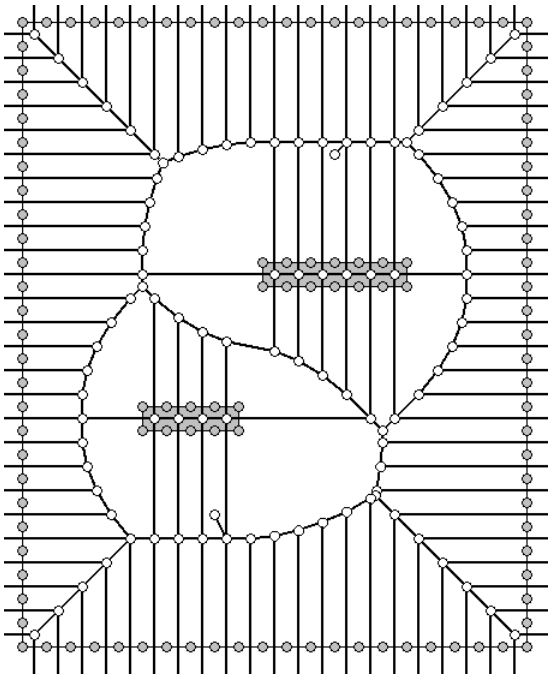

FIGURE 2.11 – Construction du diagramme de Voronoï à partir de la discrétisation du bord de l'environnement et des obstacles, obtenu à partir d'une triangulation de Delaunay.

Méthode silhouette La méthode Silhouette [Canny, 1988] est citée pour sa volonté de généralisation. Comme l'indique [Brock, 2000], son objectif est de généraliser la construction de roadmap à tout problème de planification de trajectoire de n'importe quelle dimension. L'idée est de réduire un espace de configuration en utilisant un algorithme qui projette cet espace de dimension moindre jusqu'à obtenir plusieurs projections en 2D. Pour chaque projection ainsi obtenue, une recherche de points critiques permet d'établir les configurations qui font partie de la roadmap. En rassemblant tous ces points, la méthode permet d'obtenir une roadmap dans l'espace des configurations. Plus de détails sont donnés dans [LaValle, 2006].

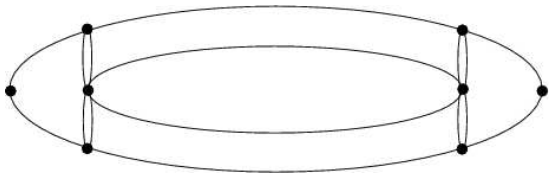

FIGURE 2.12 – Points critiques obtenus dans un tore par la méthode silhouette, image tirée de [LaValle, 2006].

Probalistic Roadmap Method Dans PRM [Kavraki *et al.*, 1996], des configurations sont tirées au hasard dans l'espace des configurations libres. Ensuite, une approche directe pour la planification de trajectoire est utilisée pour relier les configurations voisines. Si deux configurations sont trop éloignées ou ne peuvent être reliées, alors des nouvelles configurations peuvent être tirées aléatoirement afin d'augmenter localement la densité de la roadmap. Cette méthode exploite donc l'idée que les approches directes peuvent fonctionner correctement dans un problème local. De plus, les échantillons aléatoires permettent d'envisager la planification dans des espaces de configurations de grande dimension sans s'inquiéter de sa structure. Toutefois, PRM ne prend pas en compte toutes les contraintes cinématiques et dynamiques du problème pour passer d'une solution reliant une même configuration à une autre dans la construction de la roadmap. Le choix des nœuds à relier a un important impact sur l'efficacité de PRM, comme l'indique [Marble et Bekris, 2011], ce qui limite les recours aux planifications locales de trajectoire. Enfin, PRM trouve difficilement des solutions lorsque l'environnement comporte des passages étroits [Hsu *et al.*, 1998].

La variante Lazy PRM [Bohlin et Kavraki, 2000] témoigne bien de la subtilité du niveau de placement des contraintes dans les couches : au lieu de tirer des points dans l'espace des configurations libres, espace qui tient compte de la contrainte d'évitement d'obstacles, les configurations sont tirées dans l'espace des configurations. C'est donc à l'approche directe pour la planification de trajectoire de relier ces configurations. Si cela entraîne une collision, cette configuration est retirée de la roadmap. Cette variante obtient de bien meilleures performances grâce à la réduction du coût des tests de collision qui existent dans la version originale.

2.2.2 Bandes élastiques

Les bandes élastiques, créées par [Quinlan et Khatib, 1993], sont conçues pour faire le lien entre deux grands types d'algorithmes de planification : d'un côté, les algorithmes de planification de chemin peuvent calculer un chemin garanti pour atteindre l'objectif, éventuellement de manière optimale et gérant de nombreux degrés de liberté (voir PRM et RRT décrits plus loin), au prix d'une connaissance complète de l'environnement et du temps pour calculer le nouveau chemin. De l'autre, des algorithmes réactifs, tels que les potentiels artificiels, permettent de naviguer dans un contexte temps réel sans avoir à planifier longuement. Toutefois, ces méthodes étant très sensibles à l'existence de minima locaux, elles ne garantissent pas de pouvoir rejoindre le but.

Principe

Les bandes élastiques ont été d'abord proposées pour les robots qui se déplacent dans un plan 2D puis ont été étendues dans [Brock et Khatib, 1997] aux robots disposant d'un bras manipulateur dans un environnement en 3D. Cette approche étend le fonctionnement des

potentiels artificiels pour l'appliquer à des trajectoires pré-calculées. L'idée est de déformer un chemin qui vérifie les conditions de non collision aux obstacles soit pour l'affiner soit pour tenir compte des modifications de l'environnement en cours de déplacement. Pour limiter l'espace de recherche dans l'espace des configurations libres, chaque configuration d'un chemin pré-calculé est entourée d'une bulle. Chaque bulle contient un ensemble de configurations localement jugées comme atteignables. Le rayon de ces bulles est limité par les obstacles. Aussi, tant que les bulles de deux configurations voisines $q1$ et $q2$ partagent une partie commune, il est considéré comme possible de passer d'une configuration à l'autre par l'intermédiaire d'une configuration ou plusieurs configurations intermédiaires situées dans cette partie commune grâce à des courbes de Bézier.

Le principe d'Elastic Bands est de déplacer les bulles en fonction du déplacement des obstacles ou de la découverte de nouveaux obstacles. Ce déplacement des bulles est effectué en faisant un bilan des forces imposées par les obstacles et repoussant les bulles. Ce processus est illustré par la Figure 2.13.

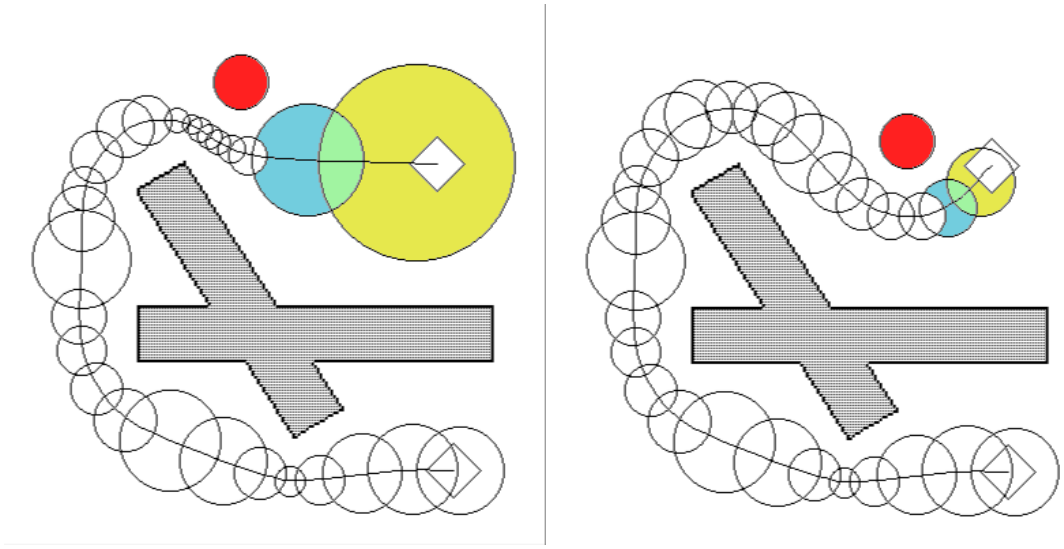

FIGURE 2.13 – Illustration provenant de [Quinlan et Khatib, 1993] montrant la déformation d'un chemin grâce aux forces créées par le déplacement d'un obstacle en rouge. Ces forces déplacent les bulles disposées sur le chemin : la déformation est acceptable tant que les bulles voisines, par exemple celles en couleur, partagent une partie commune.

Critique

Le processus de déformation des trajectoires est censée maintenir des solutions faisables. Or, le processus de déformation ne tient pas compte des contraintes cinématiques. De plus, en décidant que les bulles contiennent uniquement des configurations libres, cette approche utilise une approximation, parfois importante, de l'espace réellement atteignable. La consistance de la tâche durant la déformation est devenue un enjeu de cette approche, comme en témoigne [Brock *et al.*, 2002].

2.2.3 Synthèse des approches découplées

En voyant comme des boîtes noires les passages successifs permettant de passer d'un plan d'actions à une succession de trajectoires, les approches découplées gagnent en rapidité et les problématiques soulevées lors d'un problème de mouvement sont résolues séparément. Toutefois, ces approches souffrent de plusieurs défauts dont les risques d'incomplétude, la faisabilité réelle de l'application des commandes *in fine*, l'absence de formalisme commun aux différentes couches et enfin l'absence d'interactions dans le fonctionnement de chaque couche. De plus elles restent très spécialisées pour des problèmes de planification mettant en œuvre des entités spécifiques (robots, plate-formes avec bras manipulateur). C'est par exemple le cas du vainqueur 2005 du Challenge Darpa [Urmson *et al.*, 2006] : dans leur approche, un chemin (des coordonnées GPS à rejoindre) est affiné pour tenir compte de la route. Puis une transformation supplémentaire est appliquée pour obtenir des commandes en vitesse acceptables (et gérer l'évitement d'obstacle par adaptation de vitesse en cours d'exécution).

Pour répondre à ces problèmes, la littérature propose de passer à des approches ascendantes, dans lesquelles une couche de génération de trajectoire, appelée propagation, est une partie intégrée dans la couche de niveau supérieure, appelée sélection. Elles sont désignées comme étant les approches par échantillonnage.

Un résumé des critères que je recherche pour créer mon approche cognitive est donné dans le Tableau 2.2. Nous ajoutons que les bandes élastiques sont adaptées aux applications réelles grâce au mécanisme de déformation de trajectoire en cours de déplacement.

Propriété	**Approches découpléees**
Planification en temps réel	Oui si temps de calcul est borné
Exploration de l'espace des solutions	Oui, segmentation de l'environnement
Respect du modèle du robot et des contraintes	Oui sauf dans l'étape de smoothing si l'approximation de l'espace localement atteignable est trop importante
Application directe des commandes	Oui
Compatibilité avec un planificateur de chemin	Oui, construction d'une roadmap
Ajout de nouvelles contraintes	Difficile car cela dépend des différentes couches employées

TABLE 2.2 – Propriétés attendues d'une approche cognitive pour la planification de trajectoire et solutions apportés par les approches découplées.

2.3 Les approches par échantillonnage et leurs améliorations

Itération contre Exploration Le principe des approches itérées est de générer une solution complexe à partir des résultats de chaque planification. Ce processus de planification de trajectoire est effectué sur un horizon de planification limité de manière à explorer qu'une partie restreinte de l'environnement et donc à limiter le nombre de contraintes à respecter, en particulier des contraintes d'évitement d'obstacles. Bien que réactives aux modifications de l'environnement, ces approches itérées souffrent des défauts évoquées dans la Section 2.1 à propos des approches directes.

Dans les approches directes qui sont itérées, ne générer qu'un seul successeur à la solution d'une première planification peut conduire à l'échec de la construction d'une trajectoire

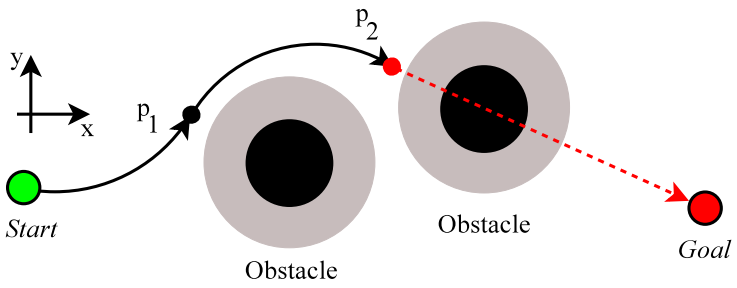

FIGURE 2.14 – Illustration d'une planification itérée : avec un seul morceau polynomial généré à la fois, l'algorithme tend vers une situation insoluble.

complète : en effet, il faudrait que le guidage puisse anticiper l'évitement d'un obstacle ou le non respect d'une contrainte au-delà de l'horizon d'une planification.

Ce type de problème est illustré par la figure 2.14. Le robot trouve successivement deux solutions p_1 et p_2. Toutefois, une fois arrivé au point final de p_2, la vitesse du robot ne lui permet plus d'éviter l'obstacle. Ces approches manquent donc d'anticipation dans leur processus d'exploration de l'environnement.

Les approches par échantillonnage étendent donc le principe de la planification itérée à la construction d'un arbre d'exploration. Chaque trajectoire est alors vue comme un nœud d'un arbre. Les fils sont construits comme étant successeurs du nœud père. La diversité dans les échantillons générés à chaque planification devient alors la clé pour faire face à des environnements fortement contraints, comme le montre la figure 2.15.

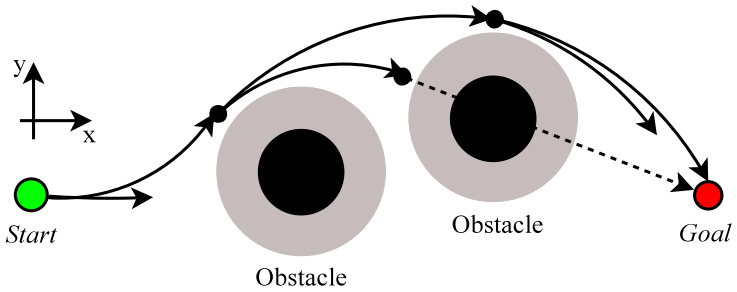

FIGURE 2.15 – Illustration d'une planification avec exploration : une situation insoluble est proposée mais l'algorithme peut générer une trajectoire alternative.

Fonctionnement Les approches de planification de trajectoire reposent sur le même modèle de construction d'un arbre d'exploration (ou d'un graphe d'exploration). Premièrement, une couche de sélection détermine un morceau de l'arbre d'exploration selon un critère de guidage. Ce morceau est prolongé par une couche de propagation qui génère des échantillons de trajectoire. Enfin, ces nouveaux échantillons sont placés dans l'arbre d'exploration et le processus est itéré.

L'avantage de cette approche est d'utiliser au cœur de son fonctionnement des approches exactes pour la création de trajectoires locales. Contrairement aux approches découplées, la construction de l'arbre/graphe d'exploration se fait conjointement avec la propagation plutôt que séparément. Ainsi, l'arbre d'exploration contient nécessairement des solutions faisables en s'appuyant un mécanisme de propagation qui respecte les contraintes du problème.

2.3.1 Une couche de propagation...

Réutilisation des approches directes et optimisation du coût

Une des voies pour générer de la diversité est de tout simplement se reposer sur une approche directe pour générer les échantillons de trajectoire. Toutefois, leur utilisation a un coût en temps de calcul. La répétition de leur utilisation peut alors conduire à une complexité désastreuse. Certaines approches envisagent donc le pré-calcul des solutions : dans un contexte de planification fortement contraint, [Howard et Kelly, 2007] proposent d'explorer l'espace des configurations à partir d'initialisations connues pour donner de bon résultats afin de générer des solutions pré-établies. Dans leur cas, il s'agit de solutions polynomiales à 8 degrés de liberté qui sont illustrées par la Figure 2.16. Ensuite, l'application de contraintes supplémentaires permet d'affiner les solutions en fonction de l'application : par exemple, la prise en compte du glissement ou de pentes.

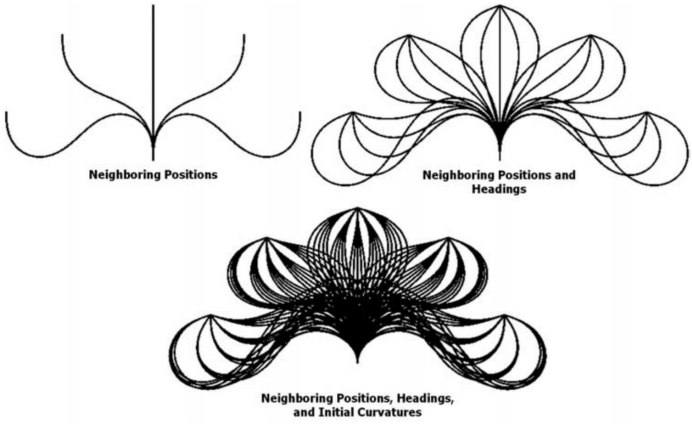

FIGURE 2.16 – Tirée de [Howard et Kelly, 2007], cette figure montre l'évolution de la base de solutions en fonction des informations initiales prises en compte dans le problème.

Méthode de Runge-Kutta

L'utilisation d'un intégrateur est une approche très courante dans les planificateurs de trajectoire par échantillonnage. L'idée est, tout simplement, d'estimer l'état suivant d'une système à partir d'une situation initiale et des équations différentielles traduisant le fonctionnement du système modélisé.

De nombreuses techniques d'intégration en analyse numérique existent mais l'une des plus employées est la méthode de Runge-Kutta. La méthode peut être itérée suivant la qualité d'estimation recherchée. Les équations suivantes donnent le Runge-Kutta du quatrième ordre :

$$x(\Delta t) \simeq x(0) + \frac{\Delta}{6}(w_1 + 2w_2 + 2w_3 + w_4) \tag{2.8}$$

Avec :

$$\begin{aligned} w_1 &= f(x(0), u(0)) \\ w_2 &= f(x(0) + \frac{1}{2}\Delta t w_1, u(\frac{1}{2}\Delta t)) \\ w_3 &= f(x(0) + \frac{1}{2}\Delta t w_2, u(\frac{1}{2}\Delta t)) \\ w_4 &= f(x(0) + \Delta t w_3, u(\Delta t)) \end{aligned} \tag{2.9}$$

Courbes de Dubins/Courbes de Reeds-Shepp

L'utilisation d'un intégrateur local peut s'appliquer à tout type de modèle localement intégrable pour générer des solutions locales. C'est exactement ce que sont les courbes de Dubins/Reeds-Shepp qui sont des solutions pour le déplacement de robots unicycles. L'idée principale est de construire des trajectoires à partir de paramètres pré-établis. Le modèle cinématique du robot unicycle est :

$$\begin{aligned} w_1 &= f(x(0), u(0)) \\ \dot{x} &= v \times \cos(\theta) \\ \dot{y} &= v \times \sin(\theta) \\ \dot{\theta} &= \omega \end{aligned} \tag{2.10}$$

Courbes de Dubins Elles décrivent des mouvements unitaires, réalisés à vitesse constante et différentiés selon l'angle à atteindre. Ces mouvements sont inscrits dans le Tableau 2.3.

Symbole	Commande ω	Mouvement associé
S	0	Tout droit
L	1	Vers la gauche
R	-1	Vers la droite

TABLE 2.3 – Mouvements unitaires pour les courbes de Dubins.

L'espace des paramètres est donc un espace discret à une dimension dont le paramètre est ω qui vérifie implicitement une contrainte en vitesse linéaire et une contrainte en vitesse angulaire. L'espace des paramètres contient les mots S, L et R.

Courbes de Reeds-Shepp Au lieu de s'en tenir simplement à l'ensemble limité de déplacements dans les courbes de Dubins, ces courbes considèrent tous les mouvements en vitesse linéaire et vitesse angulaire pour le modèle de robot unicycle. C'est ce qui est fait dans les courbes de Reeds-Shepp. Tout l'intérêt de ces courbes est de fournir des ensembles finis de trajectoires qu'il suffit de développer pour construire des solutions complexes respectant le modèle du robot, à l'image des exemples de la Figure 2.17.

2.3.2 ...Intégrée dans une couche de sélection

RRT

RRT est une approche par échantillonnage qui génère aléatoirement un arbre explorant l'espace des états d'un robot, depuis un état de *Start*, jusqu'à se rapprocher suffisamment de

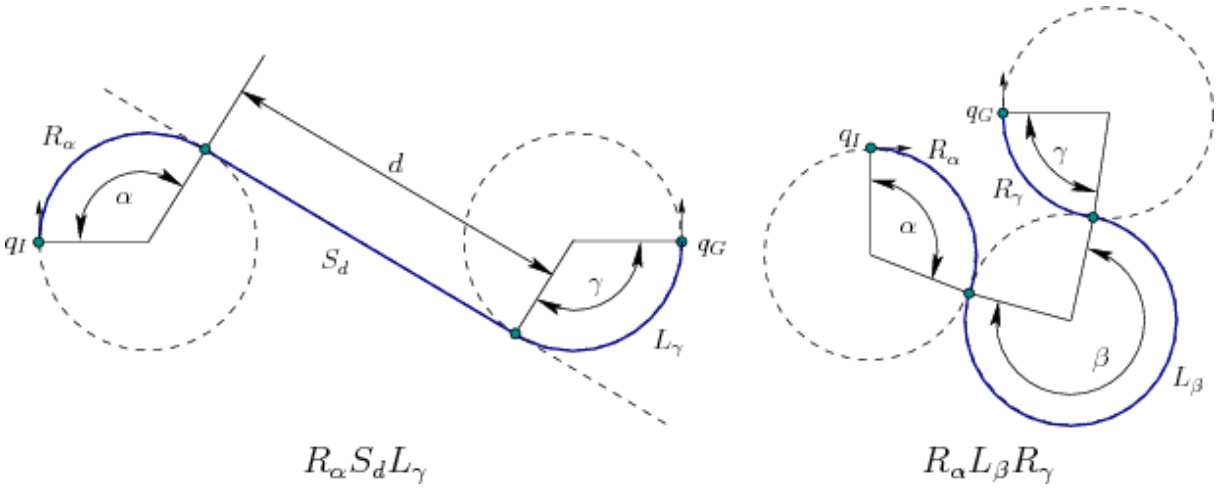

FIGURE 2.17 – Exemples de déplacements produits avec des courbes de Dubins/Reeds-Shepp, tirée de [LaValle, 2006].

l'état *Goal*. Comme expliqué précédemment, les approches par échantillonnage incorporent une couche de propagation à l'intérieur d'une couche de sélection. Le planificateur global choisit aléatoirement un état S_{rand} dans l'espace des états, puis l'état le plus proche S_{near} est sélectionné parmi les états de l'arbre. Ensuite, à partir de S_{near}, le planificateur local intègre tous les contrôles possibles du robot sur un intervalle de temps fixé ΔT, créant ainsi des branches candidates. Enfin, parmi les branches candidates, la plus proches de S_{rand}, qui n'engendre pas de collision, est sélectionnée. Cette branche est ensuite ajoutée à l'arbre d'état en tant que successeur de S_{near}. Un exemple est fourni dans la figure 2.18[5].

La simplicité de RRT explique son succès. Il s'explique également par sa capacité à trouver des solutions dans des problèmes pourtant de grande complexité où il est difficile d'établir une heuristique de guidage pertinente, par exemple : le déplacement de robots humanoïdes et la résolution de casse-tête dans [Kuffner *et al.*, 2005].

Critique de RRT RRT, par son principe de fonctionnement, possède d'excellentes propriétés : RRT est adapté à tout type de problème et possède la propriété de complétude. Elle vient du fait que la probabilité de trouver une solution augmente à mesure que RRT fait aléatoirement grandir l'arbre d'exploration dans les parties atteignables de l'environnement. Toutefois, RRT montre des problèmes de convergence vers la configuration objectif. Une solution est proposée dans RRT-Connect [Lamiraux *et al.*, 2004], généralisable aux autres approches par échantillonnage qui développent un arbre d'exploration. Dans cette variante, deux arbres grandissent : l'un à l'endroit depuis l'état *Start* et l'autre à l'envers depuis l'état *Goal*. Lorsque les branches des deux arbres sont proches, un processus permet de les relier et donc d'obtenir des solutions allant de *Start* à *Goal*

Le guidage de l'arbre d'exploration dans RRT ne convient pas nécessairement aux besoins temps réel. Les processus aléatoires rendent son comportement peu prévisible, comme le précise [LaValle et Konkimalla, 2001], en terme de temps de calcul, basé sur le nombre de propagations nécessaires à l'obtention d'une solution ; et en terme de qualité de la solution [6].

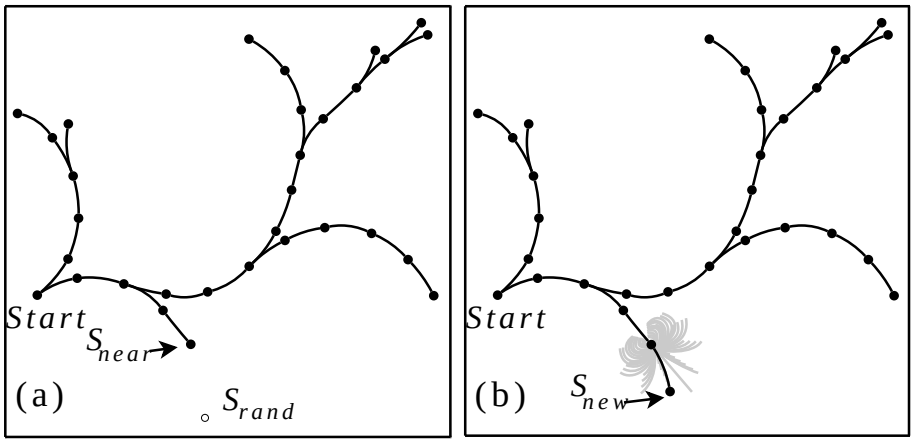

FIGURE 2.18 – Exploration de l'espace des états avec RRT. Après tirage aléatoire d'une configuration S_{rand} (a) le niveau de sélection choisit un nœud S_{near} pour son expansion ; (b) le couche de propagation génère un nœud suivant valide S_{new}.

Guidage D'autres approches que RRT s'appuient sur des processus aléatoires pour offrir d'autres perspectives d'exploration. À l'image de l'importance du guidage dans PRM [LaValle *et al.*, 2004], RRT tire profit de meilleures stratégies de guidage, telles que la stratégie nommée « Goal bias » [Ferguson et Stentz, 2006] ou la mise en place d'un coût sur l'espace des configurations dans Transition-based RRT [Jaillet *et al.*, 2010] ou Environnement-guided RRT [Jaillet *et al.*, 2011].

EST [Hsu *et al.*, 2002] est vu comme le croisement de RRT et PRM : les points de passage placés aléatoirement dans l'espace des configurations servent au guidage de l'arbre d'exploration. Ils favorisent l'exploration de zones très chargées en obstacles.

PDST-EXPLORE [Ladd et Kavraki, 2005] améliore deux points : la propagation génère des échantillons de trajectoire, plutôt qu'une seule configuration suivante. Un mécanisme de subdivision de l'environnement découpe l'environnement en cellules. Puis, en donnant du poids aux échantillons, une densité en solutions est établie pour chaque cellule. Ensuite, le mécanisme de sélection est orienté vers les zones de l'environnement les moins denses en solutions. Il en résulte une exploration prouvée comme étant complète.

DSLx [Plaku *et al.*, 2007] améliore le principe précédent en défavorisant le guidage amenant l'arbre trop près des obstacles. Pour cela, DSLx utilise le niveau de propagation pour estimer quelles sont les régions les plus faciles à atteindre dans un environnement discrétisé au préalable.

SRT [Plaku *et al.*, 2005] mélange RRT et PRM : des arbres d'exploration sont étendus en parallèle depuis des configurations placées selon les règles édictées dans PRM.

Optimalité Dans [Karaman et Frazzoli, 2010], Frazzoli fait la preuve qu'avec RRT, une fois qu'une solution atteint la région objectif, la solution ne peut pas converger vers une solution optimale. Cela signifie qu'il ne sert à rien de poursuivre le processus d'exploration au-delà d'une première solution. RRT* améliore le mécanisme d'ajout des nouveaux échantillons dans l'arbre d'exploration en accumulant une fonction de coût. Ensuite, en s'inspirant de RRG [Karaman et Frazzoli, 2009], RRT* réorganise l'arbre d'exploration à chaque propagation. Ce processus reconnecte les nouveaux échantillons aux branches voisines qui peuvent être de plus améliorées en fonction du coût propagé. RRT* profite également des améliorations habituelles

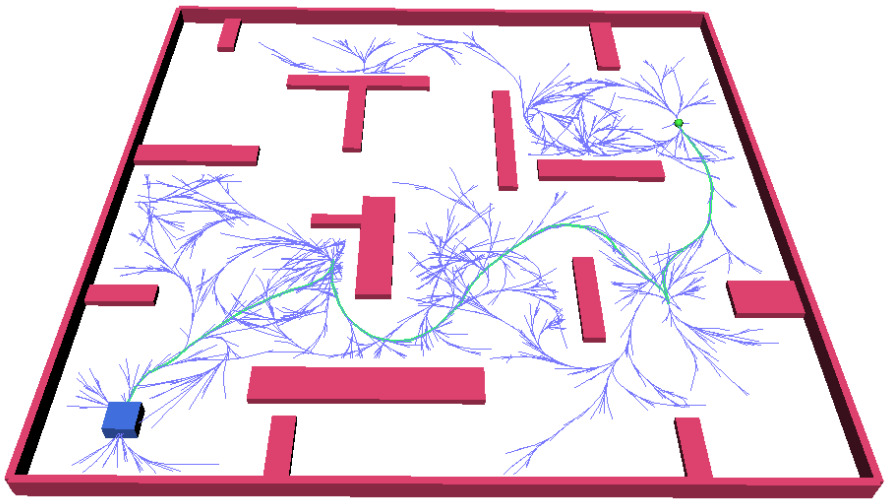

FIGURE 2.19 – RRT étend un arbre d'exploration dans l'espace des positions atteignables. Il est prouvé que les processus aléatoires de propagation et de sélection des morceaux à prolonger permettent d'obtenir nécessairement une solution (éventuellement en réitérant la planification).

de l'implémentation de RRT [Akgun et Stilman, 2011]. Comme RRG, la convergence de RRT* vers une solution optimale à mesure que de nouveaux échantillons sont ajoutés a été prouvée. Autrement dit, le temps passé après l'obtention d'une première solution est dépensé à son amélioration : l'avantage est considérable pour les applications en temps réel où la gestion des ressources peut devenir critique.

Approches multi-résolutions

Les approches multi-résolutions proposent la construction de trajectoires à plusieurs niveaux de résolution, comme [Knepper et Mason, 2011]. Le principe est que des échantillons de trajectoire peuvent servir à la production de solutions composées. Celles-ci servent alors de base à une production de trajectoires plus complexes à une échelle supérieure. Tout l'enjeu de ces approches est alors de jouer sur le facteur d'échelle pour accélérer l'efficacité de l'exploration de l'environnement tout en affinant la recherche quand cela est nécessaire. Le travail le plus notable est AD* [Likhachev et Ferguson, 2009] : il est conçu pour faire face à des environnements découverts dynamiquement tout en ayant un temps de calcul très limité. En se basant sur des mouvements à grande échelle, une pré-solution est établie puis affinée en fonction du temps de calcul restant et des modifications dans l'environnement en considérant que tout mouvement peut être décomposé à coup sûr en mouvements de petite échelle.

2.3.3 Conclusion

Les solutions décrites dans ce chapitre reposent sur des mécanismes aléatoires pour générer les solutions et éventuellement explorer l'espace des solutions. Si elles agissent directement sur le modèle, leur nature aléatoire rend difficile l'exploitation de ces résultats dans un contexte temps réel. Cela est dû à l'incertitude provenant de la durée des calculs mais aussi de la qualité de la solution. Si statistiquement, l'exploration de l'espace est complète au bout d'un nombre infini de propagations, dans une seule planification, il n'est pas certain d'obtenir une solution, surtout si

elle est difficile à atteindre. Les approches multi-résolutions abordent ce problème en adoptant un comportement déterministe, déterminisme qui semble améliorer l'efficacité des approches par échantillons [LaValle *et al.*, 2004] : l'idée est de constituer une base de trajectoires pré-établies, éventuellement planifiées sous des contraintes difficiles qui empêchent l'approche d'être utilisée en temps réel. Le problème est que cette base doit être suffisamment représentative de tous les mouvements possibles tout en limitant leur diversité pour réduire la complexité des approches multi-résolutions [Branicky *et al.*, 2008]

Le problème de la replanification est lui-même abordé : dans la veine des approches telles que D* [Stentz, 1995], des travaux proposent la transformation de l'arbre d'exploration issu d'une première planification de RRT [Ferguson *et al.*, 2006] plutôt qu'une replanification complète et certainement coûteuse.

Un résumé du fonctionnement de ces approches est proposé dans la Table 2.4.

Mon objectif est de pouvoir conserver un niveau suffisant de description des différentes entités. Cela doit être fait en ayant les garanties nécessaires sur l'exécution afin de contrôler la planification locale de trajectoire. Elle devient alors un mécanisme de vérification des mouvements pour les couches cognitives. Un résumé des critères recherchés pour créer l'approche cognitive est donné dans la Table 2.5.

2.4 Le mélange de la planification symbolique avec la planification de trajectoires

Cette section présente les travaux relatifs à l'utilisation de comportements de pilotage dans les applications robotiques.

2.4.1 Comportements en planification de mouvement

[Reynolds, 1999] propose une approche découplée pour la description des comportements de pilotage utilisés par des personnages autonomes afin qu'ils puissent achever des tâches de haut niveau. Un niveau de sélection d'action décide en premier lieu des objectifs de haut niveau. Ceux-ci sont ensuite transmis à un niveau de planification de mouvement (de locomotion) qui se charge d'effectuer le déplacement.

Les approches centrées sur les comportements proposent d'intégrer le domaine de la planification de comportements dans les applications robotiques [Jones, 2004]. Celles-ci sont pourtant limitées à des architectures réactives s'appuyant sur les capteurs et les contrôleurs des robots [Brooks, 1985]. Il n'y a aucune inférence à propos des capacités de mouvement du robot ou encore de l'accessibilité des objectifs sélectionnés.

2.4.2 Approches mêlant états discrets et continus

Approches cognitives pour bras articulé

Les problèmes de planification de mouvement n'impliquent pas nécessairement un déplacement libre du robot dans l'environnement et donc un espace des configurations potentiellement infini : c'est par exemple le cas des bras articulés et en général des problèmes de manipulation d'objets dont les enjeux sont révélés par [Lozano-Perez *et al.*, 1987]. Des contraintes peuvent apparaître sur les mouvements relatifs entre les éléments du bras, sur la manière d'attraper les objets à manipuler, sur l'équilibre du bras ...

Planificateur	**Sélection**	**Propagation**	**Source de diversité**
AD*	Amélioration de A*	Combinaisons d'actions élémentaires	Planificateur de [Howard et Kelly, 2007]
RRT	Configurations aléatoires	Intégration aléatoire	Commandes
EST	Points de passage aléatoires	Intégration aléatoire	Commandes
PDST -EXPLORE	Segmentation de l'environnement pour favoriser les zones non explorées	Intégration aléatoire sur plusieurs pas de temps + poids sur les trajectoires	Commandes
DSLx	Choix déterministe de branches à étendre en fonction de l'accessibilité des zones	Intégration aléatoire	Commandes et temps
RRT*	Configurations aléatoires + réorganisation de l'arbre	Intégration aléatoire	Commandes

TABLE 2.4 – Récapitulatif des propriétés des approches par échantillonnage citées dans cette section

Propriété	Approches par échantillonnage
Planification en temps réel	Problème de l'aléatoire
Exploration de l'espace des solutions	Oui, complète statistiquement pour RRT
Respect du modèle du robot et des contraintes	Oui
Application directe des commandes	Oui
Compatibilité avec un planificateur de chemin	Oui, mais guidage à formaliser
Ajout de nouvelles contraintes	Au niveau de la propagation

TABLE 2.5 – Propriétés attendues d'approche cognitive pour la planification de trajectoire et solutions apportés par les approches par échantillonnage.

Notons qu'un bras articulé ne peut s'étendre à l'infini. Dans ce contexte, les degrés de liberté sont définis par les angles entre chaque partie du bras dont les valeurs possibles sont limitées. L'espace des configurations qui en résulte est donc fini et peut être complètement calculé à l'avance. Cette possibilité est particulièrement intéressante puisqu'elle rend possible la vérification en ligne des états du robot. Dans ce contexte particulier, de récentes avancées dans [Jaesik et Eyal, 2009] ont ainsi été réalisées en associant la planification « classique » (raisonnement dans un monde symbolique) et la planification de mouvement. La partie descriptive utilise *PDDL* [McDermott *et al.*, 1998]. L'espace des configurations du bras est représenté dans le niveau de planification des actions, assurant au préalable la faisabilité physique des décisions prises. Ce raisonnement est illustré par la figure 2.20. Une action est décrite en particulier par la configuration de départ et la configuration à atteindre. L'appartenance de ces configurations à l'espace des configurations sert alors de pré-condition au mécanisme de décision. Dans ce travail, je vois la fusion réussie de mécanismes de prise de décision qui gèrent la vérification physique des états du robot. Toutefois, dans cette approche, les contraintes cinématiques ne sont pas considérées puisque seul l'espace des configurations est utilisé. Or, cet espace ne décrit par la faisabilité des transitions entre deux configurations.

Approches par échantillonnage dans un espace hybride

Certains travaux comme [Branicky *et al.*, 2006] tendent à utiliser des approches par échantillonnage pour explorer des espaces d'état. Leur approche est d'abord utilisée sur des espaces strictement discrets (s'inspirant de ce qui peut être fait dans la catégorie des algorithmes de renforcement). Puis leur approche est étendue à des espaces mêlant états continus et discrets. Toutefois, comme le montre avec pertinence [Jaesik et Eyal, 2009], l'espace des configurations peut grossir de manière exponentielle en faisant intervenir des états discrétisés. Dans le même ordre d'idée, [Agha-mohammadi *et al.*, 2011] proposent une approche qui généralise PRM pour tenir compte de l'évolution des connaissances du robot (incertitude sur les obstacles par exemple) et pour guider le niveau de planification de trajectoire en fonction de l'évolution des connaissances du robot.

2.4.3 Description de comportements

Le réalisme des comportements de conduite est pourtant devenu un important challenge, ainsi qu'en témoigne la réflexion menée dans le domaine de la simulation de trafic routier [Lacroix *et al.*, 2009]. Comme dans mon cas, la principale difficulté est de réussir à lier le haut

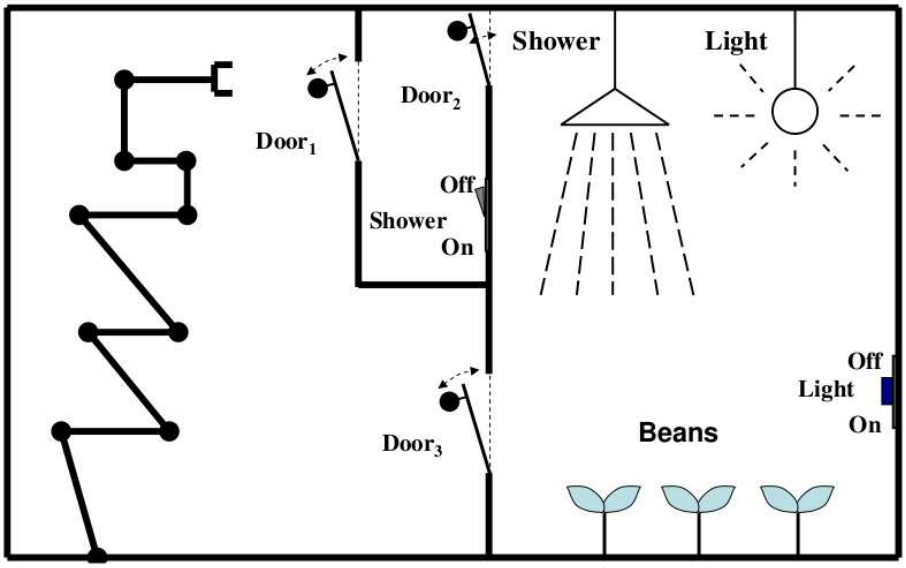

FIGURE 2.20 – Illustration, tirée de [Jaesik et Eyal, 2009], du problème de planification cognitive de trajectoire dans l'espace des configurations : tous les mouvements possibles d'un bras articulé sont établis au préalable et donc les zones de l'espace qu'il peut atteindre. Cet espace des mouvements possibles sert alors de pré-condition aux actions envisagées par le planificateur : si elles entraînent le robot dans des configurations physiques infaisables, elles sont rejetées du planificateur. Ce planificateur prendra donc uniquement les décisions de déplacement que l'on sait effectivement réalisables par le bras manipulateur.

niveau, attaché au domaine de la psychologie du conducteur, au bas niveau, qui lui concerne les données mesurables (vitesse, braquage...). J'entends par psychologie du conducteur les éléments qui ont une influence sur le choix des trajectoires à suivre. Par exemple, un conducteur « agressif » choisira des solutions plus risquées qu'un conducteur « calme ». Ce problème a été par exemple traité par [Lacroix *et al.*, 2007]. Celui-ci propose d'utiliser les distributions de probabilités sur les paramètres, tels que le *temps avant collision* ou encore le *temps avant de traverser la ligne* pour générer de la variété dans les comportements de conduite rencontrés dans le monde réel.

2.4.4 Planification de tâches complexes

Les plus hauts niveaux de raisonnement dans les applications robotiques doivent gérer le problème général de la planification de tâches complexes. Les approches usuelles sont soit basées sur une description de type STRIPS, comportant une description du monde et de toutes les actions possibles, soit sur une description en réseaux hiérarchiques de tâches (HTN pour Hierarchical Task Networks) pour décomposer un objectif en tâches composées. Les HTN ont été massivement utilisés car ils permettent de décrire des tâches de la même manière que les humains le font pour eux-mêmes, bien que leur expressivité dans la pratique ne soit pas meilleure que celle de STRIPS [Lekavy et Návrat, 2007].

Afin de planifier ces tâches tenant compte des capacités physique des robots, il faut disposer d'une représentation de l'environnement qui soit adaptée au contexte de la robotique. Il est pour cela nécessaire de construire des bases de connaissances orientées robotique, à l'image de celle proposée dans OpenRobotOntology [Lemaignan *et al.*, 2010].

Moteurs d'inférence

Les moteurs d'inférence s'appuient sur un langage de description à la STRIPS pour la planification de plans action. L'état du monde est décrit selon une suite de faits. Une action est décrite sous la forme d'un tuple contenant :

- le nom de l'action ;
- les paramètres de l'action ;
- une liste de pré-conditions nécessaires à la réalisation de l'action portant sur l'état du monde avant la réalisation de l'action ;
- l'effet de l'action, qui modifie l'état du monde.

Un planificateur permet alors d'établir la série d'actions nécessaires à la réalisation d'un objectif : un état à atteindre.

Un important travail de traduction est nécessaire pour faire rentrer la planification de trajectoire dans ce formalisme. En effet, il faut traduire une représentation continue du problème de planification de trajectoire sous la forme de faits : état du robot, obstacles. . .Il faut également vérifier l'appartenance de l'état du robot à l'espace des configurations, sous la forme de pré-condition ; De même, il est nécessaire d'exprimer le respect des contraintes cinématiques dans l'effet d'une action, ce qui implique une représentation des contraintes cinématiques et dynamiques dans la base de faits. Cela correspond à l'un des objectifs de la thèse : construire un pont entre un langage formel à la STRIPS et le problème de planification de trajectoire sous contraintes cinématiques et dynamiques.

Hierarchical Task Networks

Dans les HTN, les plans sont construits en établissant des hiérarchies d'actions sous la forme d'arbres. Les HTN proposent un mécanisme d'évaluation qui établit un coût, éventuellement temporel, de chaque alternative. Ils sont utilisés comme support à la décision.

L'approche par HTN développe des arbres d'actions comme les approches par échantillonnage créent des arbres d'exploration. Chaque trajectoire allant d'un nœud de l'arbre d'exploration jusqu'à l'une des racines est une trajectoire alternative pour atteindre une partie de l'environnement. J'exploite ce lien entre HTN et planification de trajectoire dans le Chapitre 5.

TÆMS

TÆMS [Decker, 1996] est un formalisme utilisé pour décrire des HTN. Il a été employé avec succès pour des applications orientées agents ou multi-agents dans des contextes temps réel, avec des planificateurs indépendants du domaine : Design-to-Time [Garvey et Lesser, 1993], Design-to-Criteria [Wagner et Lesser, 2000] et avec la plateforme de coordination GPGP [Decker et Lesser, 1995]. TÆMS et ses modules associés sont capables de décrire des tâches complexes utilisant des ressources, possédant une durée et possédant des inter-relations complexes, comme dans le cas des applications robotiques.

2.5 Résumé

Dans cet état de l'art, je montre que calculer une trajectoire qui prend en compte à la fois les contraintes cinématiques et dynamiques est un problème prouvé comme étant PSPACE-hard [Reif, 1987]. Partant de ce constat, les approches directes développées initialement sont condamnées à rester dans un contexte local ou à utiliser des modèles simples avec peu de

contraintes s'il faut planifier à long terme. Pour résoudre ce problème, les approches découplées ont séparé le problème de planification en plusieurs couches : chaque couche prend en compte une partie des contraintes puis le chemin déterminé par la première couche est donné à la suivante qui affine le chemin en fonction des contraintes manquantes. Bien que ces approches puissent être utilisées dans un cadre temps réel, elles souffrent d'incomplétude et risquent de fournir des solutions qui ne sont pas utilisables. Ainsi, les approches découplées incorporent le modèle de fonctionnement des approches directes dans une couche d'exploration afin de guider au mieux le processus d'exploration. Les solutions respectent par conséquent les contraintes du problème et peuvent être utilisées dans des cas complexes : des modèles avec beaucoup de degrés de liberté et de nombreuses contraintes dans l'environnement. Pour cela, elles s'appuient sur des processus aléatoires qui ont le défaut de rendre imprévisible le comportement du planificateur. Les approches multi-résolutions restent dans la même veine mais en s'appuyant sur un ensemble pré-déterminé de solutions, elles peuvent utiliser des processus déterministes pour construire des solutions complexes. De nos jours, la mise en lien d'une couche cognitive avec une couche de planification de trajectoire se fait par une approche découplée dont les défauts sont les mêmes que les approches pour la planification de trajectoire. En effet, il y a un risque d'inconsistance dans le processus de décision si la couche cognitive ne prend pas en compte le modèle du robot. Afin d'amener le modèle du robot dans les couches décisionnelles, soit l'espace des configurations est pris en compte dans le processus de sélection d'action soit l'espace de configuration contient à la fois des états discrets, le modèle d'action, les états continus et le modèle du robot.

Toutefois, ces efforts ne sont pas suffisants car dans le premier cas, les contraintes cinématiques ne sont pas prises en compte ; et dans le second cas, la complexité explose avec la dimension de l'espace de recherche. Selon moi, il est possible de reprendre la même architecture que les approches de planification de trajectoire par échantillonnage, à savoir l'incorporation d'une telle approche dans une couche cognitive pour construire un arbre de comportements de pilotage. Les comportements de pilotage servent alors à contrôler en retour, le comportement de la couche de planification de trajectoire pour guider l'évolution de l'arbre d'exploration. Ainsi, les couches interagissent entre elles pour créer un arbre d'exploration validant les contraintes cinématiques et dynamiques du problème tout en répondant aux besoins d'un arbre de comportements de pilotage. L'arbre décrivant les comportements de pilotage permet alors de décider quelle action effectuer, sachant que seules les actions physiquement faisables peuvent être développées par le planificateur de trajectoire. Cela implique que la couche de planification de trajectoire ait un comportement complètement prévisible et contrôlable.

2.6 Vers une approche cognitive pour la planification de trajectoire

Sur la base des travaux présentés dans ce chapitre, je distingue deux apports dans cette thèse : le premier apport est DKP qui est une couche de planification de trajectoire, développée dans le Chapitre 4. Elle est entièrement déterministe grâce au mécanisme de propagation développé dans le Chapitre 3. Le second apport est la mise en œuvre de comportements de pilotage développés dans le Chapitre 5, ce qui concrétise l'approche cognitive pour la planification de trajectoire.

DKP

L'approche que je propose pour la planification de trajectoire par échantillonnage, DKP, construit dans un environnement 2D un arbre d'exploration dans lequel les branches sont des

échantillons quadratiques guidées par un critère d'optimisation. DKP est en particulier construit pour les robots à deux roues indépendantes des solutions à base de quadratiques, lesquelles suffisent pour obtenir des commandes respectant leur modèle.

Dans DKP, les robots sont définis selon leurs contraintes cinématiques, telles que des limites sur la vitesse linéaire ou angulaire. De plus, DKP gère l'évitement d'obstacles fixes ou mobiles. L'originalité du processus de propagation provient de la représentation géométrique du problème. Ce processus définit localement un espace atteignable et continu, appelé espace des paramètres, qui modélise toutes les trajectoires qui peuvent satisfaire toutes les contraintes du problème. D'autres contraintes peuvent être définies pour un robot particulier tant qu'une restriction de l'espace des paramètres peut être exprimée. Dans un processus déterministe, DKP produit alors des échantillons localement optimaux ayant des durées variées et qui respectent les contraintes du problème. Ce processus de propagation permet d'obtenir une exploration expansive de l'espace de travail, évitant l'usage de processus aléatoires pour éviter les minima locaux.

DKP tire avantage de la simplicité de ses échantillons pour totalement séparer le processus de calcul de l'espace atteignable du processus de recherche de solution. Cette simplicité des morceaux de trajectoire, couplée à un processus d'exploration efficace, est largement suffisante pour produire des manœuvres complexes. Comme dans les approches par échantillonnage, des garanties sur la faisabilité des trajectoires sont fournies, tout en obtenant des solutions de bonne qualité. Le point important à mon sens est que les résultats sont reproductibles et que l'on peut établir dans DKP des bornes sur le temps de calcul du niveau de propagation. Le processus de sélection est une variation de l'algorithme A^* : DKP optimise un critère d'exploration. Enfin, l'espace des paramètres est une manière efficace de créer des contraintes spécifiques sur nos robots tout en identifiant clairement l'espace localement atteignable.

Comportements de pilotage

L'approche cognitive de planification de trajectoire utilise deux couches bien différentes qui interagissent entre elles. Chacune d'elles crée un arbre dont la construction influence la construction de l'autre : l'arbre d'exploration de l'environnement et l'arbre de pilotage. L'arbre de trajectoires contient des échantillons dynamiquement étendus en utilisant les paramètres de pilotage fournis dans les comportements de pilotage exprimés dans l'arbre de pilotage. DKP est utilisé pour la génération des échantillons et la construction de l'arbre d'exploration. La couche cognitive utilise le déterminisme de DKP pour ne pas avoir à répéter la planification de trajectoire jusqu'à ce qu'une solution soit éventuellement trouvée. L'arbre de pilotage est fabriqué à partir de l'instanciation de comportements de pilotage, exprimés sous la forme de Hierarchical Task Network. J'utilise pour cela TÆMS qui me permet à la fois de décrire des tâches complexes et d'avoir des mécanismes pour évaluer ces tâches. Ainsi, mon approche cognitive combine des interactions à la fois descendantes et ascendantes pour construire ses solutions au problème de la planification de trajectoire. Les interactions ascendantes permettent à DKP de fournir des informations à la couche cognitive servant ensuite à décider de l'action à entreprendre parmi les alternatives explorées.

Chapitre 3

Approche géométrique pour la planification locale de trajectoire sous contraintes cinématiques et dynamiques

D'après l'état de l'art, il apparaît qu'un planificateur générant des trajectoires en vérifiant les contraintes cinématiques et dynamiques doit également prendre en compte les conditions de continuité entre deux mouvements successifs. La continuité doit être vérifiée en position, en vitesse linéaire... Il faut également respecter des contraintes cinématiques issues du modèle du robot à simuler, ainsi que l'évitement des obstacles fixes ou mobiles et contraintes spécifiques au modèle étudié (distance minimale pour communiquer, maintien de groupe...). Enfin un planificateur de trajectoire doit prendre en compte de la commandabilité du robot.

Les approches pour la planification de trajectoires mises en avant dans la section 2.2 vérifient ces contraintes soit durant l'étape d'optimisation des solutions soit avant l'étape d'optimisation des solutions.

Je distingue deux types d'approches dans ce cas de figure : les approches par descente de gradient (champs de potentiel) et les approches qui expriment des problèmes d'optimisation non-linéaire avec contraintes non-linéaires [Milam *et al.*, 2000] ou [Defoort, 2007]. Les deux approches sont similaires car elles reposent ou cachent un programme de résolution de problèmes d'optimisation non-linéaires. Ce type de programme évalue un critère de recherche sous contraintes. La différence est que dans le premier cas, le programme de résolution est directement codé dans le problème de planification de trajectoire alors que dans le second cas, le programme de résolution est un programme générique pouvant résoudre tout type de problème non-linéaire.

Vérification des contraintes avant l'étape d'optimisation

Cette vérification se fait après génération d'un premier ensemble de trajectoires qui sont potentiellement des solutions. Je distingue les approches suivantes :

Approches avec génération aléatoire d'échantillons Dans ces approches, on distingue notamment les planificateurs de trajectoires tels que RRT ou EST. Un intégrateur local vérifie une première partie des contraintes (par exemple la vitesse bornée) durant la génération

des solutions aléatoires. Ensuite, la solution est vérifiée en tenant compte de l'évitement des obstacles, voire d'autres contraintes non-intégrables.

Approches avec ensemble d'échantillons à éluder Ces approches correspondent par exemple aux courbes de Dubins, aux courbes de Reeds-Shepp ou encore à AD*. Dans ces approches, un ensemble de trajectoires pré-établies est construit à l'avance. Par exemple, dans AD*, les solutions sont obtenues par programmation non-linéaire en offline. Ces solutions sont souvent valables uniquement pour un modèle donné et/ou une situation type. Elles vérifient à l'avance certaines contraintes qu'il est difficile de prendre en compte en temps réel pour des raisons de temps de calcul (contraintes de glissement, complexité inhérente au modèle). Cet ensemble est finalement éludé en cours de planification en fonction des contraintes restantes.

Approches avec smoothing Ces approches découplées correspondent par exemple à Elastic Bands. Dans ce type d'approche, une première trajectoire est le résultat d'un premier niveau de planification vérifiant une partie des contraintes et/ou un critère d'optimisation. Par exemple, le plus court chemin évitant des obstacles fixes est un résultat fourni par l'application de l'algorithme de Dijkstra ou par l'algorithme A*. Cette première solution est ensuite affinée si possible pour vérifier les contraintes restantes du problème. Cette étape dite de smoothing ne doit pas altérer la solution sous peine de ne plus vérifier les premières contraintes. Au final, le résultat est une solution vérifiant toutes les contraintes.

Optimisation Si la couche de vérification des contraintes maintient un ensemble de solutions candidates, alors une étape d'optimisation permet de choisir la solution candidate.

Critique

Ces approches sont toutes confrontées à la même difficulté : il est difficile d'exprimer les contraintes cinématiques et les contraintes dynamiques dans l'espace des configurations.

D'après moi, les méthodes ci-dessus souffrent de défauts dont le premier est provoqué par la vérification des contraintes durant l'étape d'optimisation. Dans ce cas de figure, un espace des solutions est implicitement parcouru par le programme de résolution. Trois problèmes apparaissent : d'abord, il faut correctement initialiser la recherche de solutions avec un point appartenant à coup sûr à cet espace des solutions, sinon la vérification des contraintes échoue. Ce problème revient à devoir déterminer au préalable l'espace des solutions ! Ensuite, si l'espace des solutions présente des disjonctions, il faut initialiser la recherche dans la partie qui contient la solution optimale. S'il est possible de déterminer ces parties, il faudrait alors lancer des recherches de solutions sur chaque espace. Cela implique à nouveau de déterminer au préalable l'espace des solutions. Enfin, la solution obtenue dans ces approches dépend de la descente de gradient effectuée sur l'espace des solutions. Ce type de recherche est sensible aux minima locaux. De plus, la solution obtenue dépend grandement du comportement du programme de résolution pour ce type de problème. Par exemple, CFSQP utilise un bruit aléatoire pour sortir des minima locaux. Enfin, l'ordre d'évaluation des contraintes peut influencer le cours de la recherche de solutions.

Vérification des contraintes avant l'étape d'optimisation Dans ce cas de figure, un espace des solutions est soit généré pour servir de base de solutions pré-existantes, soit testé, par exemple avec un intégrateur local, puis réduit en fonction des contraintes restantes. Les approches pour la planification de trajectoire réalisent ce processus de plusieurs manières.

Sauf tirage infini, les approches aléatoires ne permettent pas d'explorer tout l'espace des solutions, bien que ce soit la seule méthode quand le modèle présente beaucoup de degrés de liberté ;

Les approches avec ensemble d'échantillons à éluder ne fournissent qu'une partie de l'espace des solutions. Il faut être certain que cet ensemble couvre toutes les situations auxquelles le robot peut être confronté.

Les approches avec smoothing peuvent tout simplement échouer à fournir des solutions tenant compte de toutes les contraintes. En effet, ce processus peut restreindre trop fortement l'espace des solutions afin de ne pas altérer le respect des premières contraintes.

L'architecture

Le premier apport de cette partie consiste à formaliser le processus de génération de solutions. Une architecture émerge des critiques précédentes. D'abord, on construit un espace contenant tous les échantillons de trajectoire pouvant être formés selon un modèle adapté au robot à contrôler. Cet espace a pour base les paramètres laissés libres par les conditions de continuité dans le modèle de trajectoire sélectionné. Cet espace est réduit en vérifiant les contraintes restantes. Enfin sur cet espace, une solution est déterminée par un processus d'optimisation. Dans cette architecture, la résolution des contraintes est donc effectuée avant toute recherche de solutions. Le résultat est un échantillon de trajectoire vérifiant les contraintes du problème et qui optimise un critère de recherche.

Espace des paramètres L'espace des paramètres correspond aux paramètres identifiés d'un modèle de solution adapté au robot étudié. La variation de ces paramètres permet de générer l'ensemble des solutions : par intégration, ensemble discret ou continu de valeurs. Chacun des points de cet espace définit alors un morceau possible dans l'espace des solutions.

Contraintes Nous formulons ensuite les contraintes en représentant l'ensemble des valeurs admissibles dans leurs bases respectives. Les contraintes cinématiques usuelles sont développées : contraintes sur la vitesse linéaire ou la vitesse angulaire, contraintes sur l'évitement d'obstacles fixes et mobiles. Toute contrainte peut être employée tant que mon approche reçoit sa représentation dans l'une des bases du problème. L'enjeu pour l'approche cognitive que nous construisons est de passer d'une représentation symbolique des contraintes, par exemple en posant des bornes sur des contraintes, à une version continue adaptée à la planification de trajectoire. Le point essentiel est qu'il doit exister des fonctions de changement de base entre l'espace de chaque contrainte et l'espace des paramètres.

Réduction de l'espace des paramètres L'objectif est de réduire l'espace des paramètres afin de ne conserver que des points créant des solutions qui respectent les contraintes. Pour cela, mon approche dispose de deux méthodes suivant l'existence des transformations de base : soit les ensembles de valeurs admissibles de chaque contrainte sont projetés dans la base de l'espace des paramètres puis l'intersection est faite avec l'espace des paramètres ; soit l'espace des paramètres est projeté dans les bases de chaque contrainte puis l'intersection est faite avec les valeurs admissibles de chaque contrainte. L'espace des paramètres est donc réduit afin de ne contenir que des solutions respectant les contraintes.

Recherche de solutions Le premier avantage de cette approche est d'abandonner la recherche de solutions dès que l'espace des paramètres est vide. Sur l'espace des paramètres

restant, l'optimisation d'un critère de recherche permet de retenir une solution permettant un déplacement optimal selon le critère considéré. De plus, si la représentation de l'espace des paramètres le permet, il est envisageable d'identifier en premier lieu des disjonctions dans l'espace des paramètres puis d'effectuer une recherche de solutions sur chacune des parties ainsi révélées. Des solutions localement optimales sont ensuite recherchées sur chacun des sous-espaces des paramètres : ce cas de figure apparaît notamment lorsqu'il existe localement des alternatives de contournement d'un obstacle. Ces solutions sont utiles au processus d'exploration qui est développé dans le Chapitre 4 consacrée à DKP.

Contraintes et espace des paramètres

Dans ce chapitre, je fournis trois exemples de trajectoire :
- échantillons quadratiques pour des robots à deux roues indépendantes ;
- courbes de Dubins/Reeds-Shepp pour des robots de type voiture ;
- échantillons cubiques pour des robots à deux roues indépendantes.

L'approche de propagation est ensuite développée pour le cas des échantillons de trajectoire à base de quadratiques Ceux-ci possèdent des paramètres fixés par continuité en position et vitesse et deux paramètres laissés libres, ce qui définit un espace des paramètres. Cet espace est ensuite construit en faisant l'intersection des surfaces représentant chaque contrainte projetées depuis leurs bases respectives. Chaque point de l'espace des paramètres fixe les paramètres libres des échantillons : cela permet de générer des morceaux de trajectoire qui respectent les contraintes du problème telles que les contraintes cinématiques ou les contraintes d'évitement d'obstacle. Le meilleur échantillon est enfin sélectionné grâce à une étape d'optimisation dans l'espace des paramètres. Cette approche pour la propagation est utilisé par défaut dans DKP, qui est présenté dans la partie suivante.

Ce chapitre formalise des trajectoires sous la forme d'un espace des paramètres contenant les trajectoires étant potentiellement solutions. Je fournis ensuite une définition des contraintes agissant sur ces trajectoires. L'application des contraintes sur l'espace des paramètres passe par l'application de transformations et d'opérations ensemblistes. Cette étape engendre la restriction des valeurs admissibles des paramètres des trajectoires. La recherche d'une solution par optimisation d'un critère est alors envisageable : la solution garantie le respect des contraintes du problème.

Organisation

Ce Chapitre développe l'approche pour la propagation de morceaux de trajectoire. Tout d'abord, ce manuscrit développe dans la Section 3.1 la représentation des trajectoires. Puis, on définit les contraintes qui s'appliquent aux trajectoires dans la Section 3.2. Ensuite l'architecture de propagation est présentée dans la Section 3.3. Cette couche s'intègre dans l'architecture du type sélection/propagation qui interagit à terme avec une couche de raisonnement (voir la Figure 3.1).

Les sections suivantes fournissent des exemples d'utilisation de la couche de propagation. Deux types de trajectoire y sont décrits : des trajectoires conduisant à un ensemble continu de solutions dans la Section 3.5 et des trajectoires amenant à un ensemble fini de solutions dans la Section 3.6. Enfin, la Section 3.7 détaille complètement la mise en œuvre de l'architecture de propagation pour des trajectoires paramétriques basées sur des quadratiques.

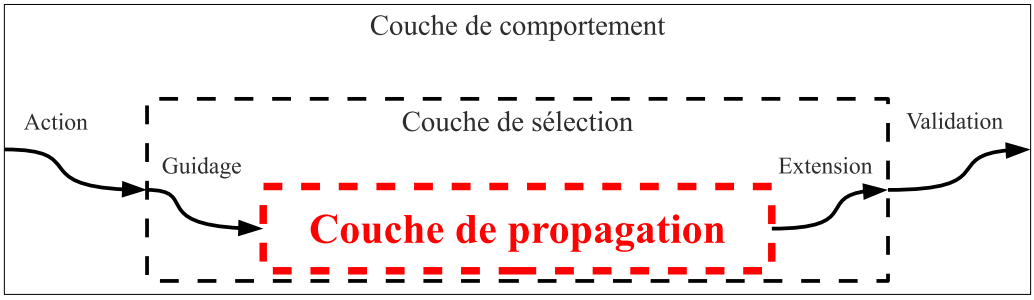

FIGURE 3.1 – Ce chapitre se focalise sur le niveau de propagation.

3.1 Modèles de trajectoire

Une trajectoire $P(t)$ est composé d'une succession de $n+1$ morceaux de trajectoire dans le plan $(x;y)$ Un morceau de trajectoire est notée $p_i(t)$ avec i entier et $i \in [0, n+1]$. $p_i(t)$ définit la position du centre du robot pour tout instant $t \in [0, T_{p_i}]$:

$$p_i(t) = \begin{cases} x_{p_i}(t) \\ y_{p_i}(t) \end{cases} \tag{3.1}$$

T_{p_i} désigne la durée du morceau de trajectoire p_i. Le morceau de trajectoire $p_{i+1}(t)$ succède au morceau $p_i(t)$. La durée totale d'une trajectoire $P(t)$ vaut $\sum_{i=0}^{n} T_{p_i}$.

3.1.1 Espace des paramètres

Dans cette partie du chapitre, un morceau de trajectoire $p_i(t)$ tel que $x(t)$ et $y(t)$ contient au total $2n+2$ paramètres :

$$p_i(t) = \begin{cases} x_{p_i}(t) = x_{\alpha_0^x,\ldots,\alpha_n^x}(t) \\ y_{p_i}(t) = y_{\alpha_0^y,\ldots,\alpha_n^y}(t) \end{cases} \tag{3.2}$$

On désigne par espace des paramètres, noté E, l'ensemble des valeurs des paramètres $\alpha_0^x, \ldots, \alpha_n^x, \alpha_0^y, \ldots, \alpha_n^y$ qui définissent la forme des morceaux de trajectoire. Il repose dans un sous-ensemble de $\mathbb{R}^n$ et de base $\alpha_0^x, \ldots, \alpha_n^x, \alpha_0^y, \ldots, \alpha_n^y$. Un point $(\alpha_0^x, \ldots, \alpha_n^x, \alpha_0^y, \ldots, \alpha_n^y)$ de l'espace des paramètres définit donc une trajectoire unique $p_{\alpha_0^x,\ldots,\alpha_n^x,\alpha_0^y,\ldots,\alpha_n^y}(t)$ validant tout ou partie des contraintes. Dans le domaine de la robotique, le terme espace des paramètres se retrouve dans [Merlet, 2001] : cet article propose un espace des paramètres pour représenter l'espace de toutes les formes d'un robot et trouver sur cet espace le design optimal d'un robot. Chaque point de cet espace des paramètres permet de générer un espace des solutions associé.

3.1.2 Dérivabilité

La trajectoire $p(t)$ est n fois dérivable et continue. Le vecteur vitesse instantanée est donc une fonction de tout ou partie des paramètres $\alpha_0^x, \ldots, \alpha_n^x, \alpha_0^y, \ldots, \alpha_n^y$:

$$\overrightarrow{S}_{p_i}(t) = \begin{cases} \dot{x}_{p_i}(t) = \dot{x}_{\alpha_0^x,\ldots,\alpha_n^x}(t) \\ \dot{y}_{p_i}(t) = \dot{y}_{\alpha_0^y,\ldots,\alpha_n^y}(t) \end{cases} \tag{3.3}$$

De même, le vecteur accélération instantanée est donc également une fonction de tout ou partie des paramètres $\alpha_0^x, \ldots, \alpha_n^x, \alpha_0^y, \ldots, \alpha_n^y$:

$$\overrightarrow{A}_{p_i}(t) = \begin{cases} \ddot{x}_{p_i}(t) = \ddot{x}_{\alpha_0^x,\ldots,\alpha_n^x}(t) \\ \ddot{y}_{p_i}(t) = \ddot{y}_{\alpha_0^y,\ldots,\alpha_n^y}(t) \end{cases} \tag{3.4}$$

3.1.3 Respect de la continuité

Soit $p_{i-1}(t)$ un morceau de trajectoire précédent le morceau de trajectoire $p_i(t)$. Je peux assurer la continuité en position entre ces deux morceaux selon la règle :

$$p_{i-1}(T_{p_{i-1}}) = \begin{cases} x_{p_{i-1}}(T_{p_{i-1}}) = x_{p_i}(0) \\ y_{p_{i-1}}(T_{p_{i-1}}) = y_{p_i}(0) \end{cases} \tag{3.5}$$

De même, je peux également assurer la continuité en vitesse entre ces deux morceaux selon la règle :

$$\overrightarrow{S}_{p_{i-1}}(T_{p_{i-1}}) = \begin{cases} \dot{x}_{p_{i-1}}(T_{p_{i-1}}) = \dot{x}_{p_i}(0) \\ \dot{y}_{p_{i-1}}(T_{p_{i-1}}) = \dot{y}_{p_i}(0) \end{cases} \tag{3.6}$$

La continuité permet d'avoir des échantillons se collant entre eux avec le plus de fluidité possible, comme l'illustre la Figure 3.2. Si la dérivabilité du modèle de trajectoire le permet, la continuité peut être établie pour l'accélération instantanée et les vecteurs dérivés suivants.

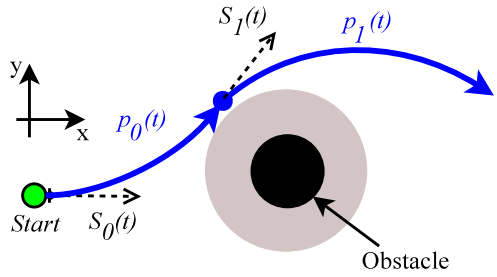

FIGURE 3.2 – Contrainte sur la continuité en vitesse et position entre deux morceaux.

3.1.4 État de départ

L'état de départ *Start* définit la position de départ du problème (x_{Start}, y_{Start}) et le vecteur vitesse initial (v_{Start}, v_{Start}).

3.1.5 État d'arrivée

De la même manière, l'état d'arrivée *Goal* est défini comme étant le point (x_{Goal}, y_{Goal}) à atteindre dans l'environnement. On n'impose pas de vecteur vitesse final, bien que je montre plus tard comment amener le robot à atteindre un intervalle de vitesse spécifié à l'avance. Pour des raisons autant numériques que des difficultés de convergence quand un processus de planification est itéré, une distance $d_{Goal,min}$ est défini entre l'objectif et le point final d'un morceau de trajectoire $p_i(T_{p_i})$ en dessous de laquelle l'objectif est considéré comme atteint. Pour des traitements homogènes au reste du problème, la trajectoire $p_{Goal}(t)$ décrit la position de l'objectif *Goal* à tout instant de la planification. Cette définition permet de spécifier des cibles mouvantes.

3.1.6 Problème de planification de trajectoire

Le problème de planification de trajectoire, illustré par la figure 3.3, consiste à établir la trajectoire entre l'état de départ *Start* et l'état d'arrivée *Goal*. La solution doit respecter les conditions suivantes : la continuité entre deux planifications successives, en position, vitesse linéaire et plus selon la dérivabilité des solutions, le respect des contraintes non intégrables du problème et la prise en compte de la commandabilité du robot, qui repose sur le bon choix du type de solution en fonction du robot à déplacer.

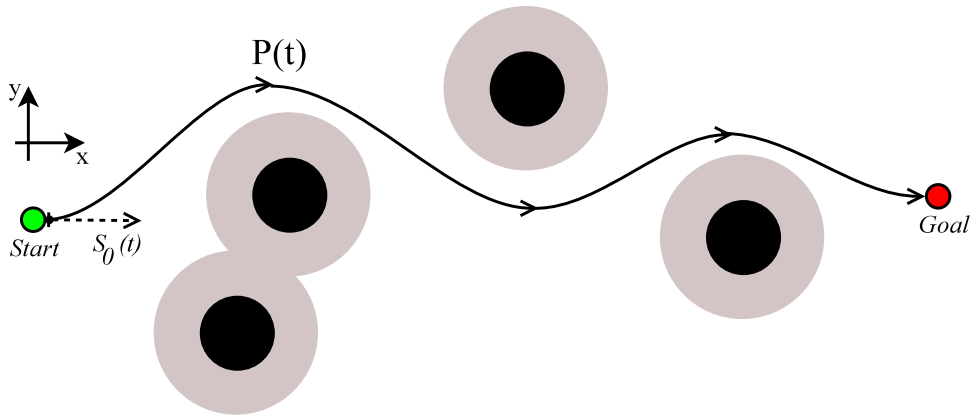

FIGURE 3.3 – Problème complet de planification de trajectoire depuis l'état de départ *Start* jusque l'état d'arrivée *Goal*.

3.2 Contraintes

3.2.1 Contraintes sur les trajectoires

Espace de la contrainte

Une contrainte $c(t)$ est définie par la fonction $f_c(p_i(t), t)$ qui s'applique à un échantillon de trajectoire $p_i(t)$ à l'instant t. Une contrainte $c(t)$ peut être vue comme un ensemble de valeurs admissibles dans l'espace de la contrainte de base B_c. Cet ensemble est noté G_c. Lorsque la dimension de l'espace de la contrainte vaut 2 (cas rencontré dans les contraintes présentées dans la thèse), cet ensemble peut être composé d'une ou plusieurs surfaces contenant les valeurs autorisées. Pour les représenter et les manipuler, j'utilise des outils de géométrie constructive de surface, dont les fondements sont rappelés dans l'Annexe 6.0.0.0. Si l'application $f_c(p_i(t), t)$ vérifie ces valeurs à tout instant t, alors l'échantillon de trajectoire respecte la contrainte.

Espace des paramètres

Du point de vue de l'espace des paramètres, une contrainte $c(t)$ peut être vue comme la fonction de contrainte appliquée au morceau de trajectoire $p_i(t)$ à l'instant t, également noté $g_c(\alpha_0^x, \ldots, \alpha_n^x, \alpha_0^y, \ldots, \alpha_n^y, t)$. Par conséquent, une contrainte $c(t)$ restreint les valeurs possibles des paramètres $(\alpha_0^x, \ldots, \alpha_n^x, \alpha_0^y, \ldots, \alpha_n^y)$ du morceau $p_i(t)$. Ces valeurs définissent ensuite toutes les formes de trajectoires permises par les contraintes du problème.

Changements de base

Dans la suite de la thèse, nous n'employons que des espaces de paramètres pour lesquels il existe les fonctions de changement de base vers l'espace des paramètres de base $\alpha_0^x, \ldots, \alpha_n^x, \alpha_0^y, \ldots, \alpha_n^y$, noté $M_p(t)$ (spécifiques au type de trajectoire choisi), depuis :

- la base de l'espace des positions (x, y) ;
- la base de l'espace des vitesses linéaires $(\dot{x}, \dot{y})$;
- la base de l'espace des vitesses angulaires $(\dot{\theta})$;
- la base de l'espace des accélérations $(\ddot{x}, \ddot{y})$;
- toute autre base atteignable par dérivations supplémentaires.

De même, j'emplois des espaces de paramètres pour lesquels il existe les fonctions de changement de base, noté $M_p^{-1}(t)$ depuis l'espace des paramètres.

3.2.2 Contraintes cinématiques bornées

Afin de manipuler plus facilement certaines contraintes cinématiques, j'ajoute à ces contraintes le domaine de valeurs $[D_-, D_+]$ dans lequel le morceau $p_i(t)$ est contraint par la fonction de $f_c(p_i(t), t)$. Ainsi, une contrainte est constituée des inégalités :

$$D_- \leq g_c(\alpha_0^x, \ldots, \alpha_n^x, \alpha_0^y, \ldots, \alpha_n^y, t) \leq D_+ \tag{3.7}$$

Vitesse linéaire

La vitesse linéaire est définie comme la norme du vecteur vitesse instantanée $\overrightarrow{S}(t)$:

$$S_p(t) = \sqrt{\dot{x}_p(t)^2 + \dot{y}_p(t)^2} \tag{3.8}$$

Soit $[S_-, S_+]$ le domaine dans lequel la contrainte de vitesse linéaire $c_{Speed}(t)$ est définie. Cette contrainte est représentée par un anneau de rayons S_- et S_+ dans la base des vitesses $(\dot{x}; \dot{y})$. Par exemple, dans le cas où $S_- = 0$ et $S_+ = 1\text{m/s}$, la contrainte est un disque centré en $(0, 0)$ dans la base des vitesses tel que représenté par la figure 3.4.

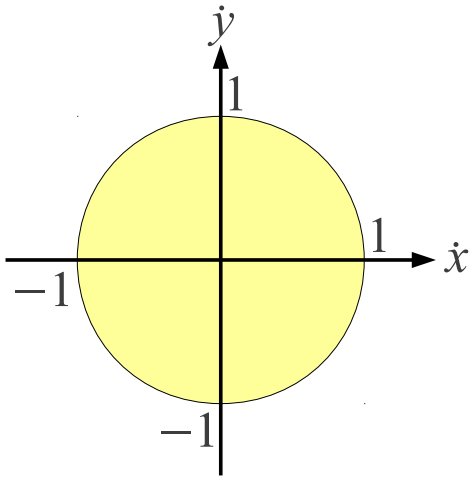

FIGURE 3.4 – Contrainte sur la vitesse linéaire bornée par $[0, 1]$ représentée dans la base des vitesses.

Accélération linéaire

L'accélération linéaire est définie comme la norme du vecteur accélération instantanée $A(t)$:

$$\overrightarrow{A}_p(t) = \sqrt{\ddot{x}_p(t)^2 + \ddot{y}_p(t)^2} \tag{3.9}$$

Soit $[A_-, A_+]$ le domaine dans lequel la contrainte d'accélération linéaire $c_{Acceleration}(t)$ est définie. Cette contrainte est représentée par un anneau de rayons A_- et A_+ dans la base des accélérations $(\ddot{x}; \ddot{y})$. Dans le cas où $A_- = 0$ et $A_+ = 1\text{m/s}^2$, la contrainte est un disque centré en $(0, 0)$ dans la base des accélérations tel que représenté par la figure 3.5.

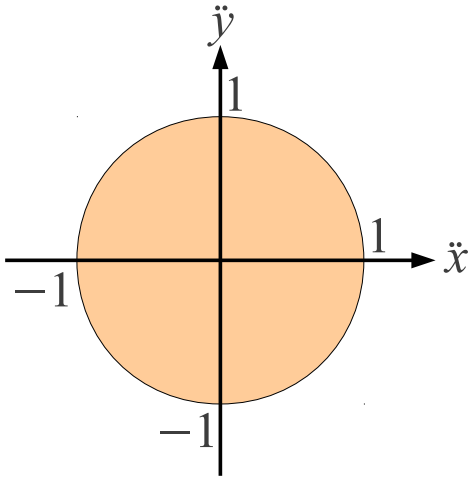

FIGURE 3.5 – Contrainte sur l'accélération linéaire bornée par $[0, 1]$ représentée dans la base des accélérations.

3.2.3 Contraintes d'évitement d'obstacle

Représentation d'un obstacle

Les obstacles sont représentés dans l'espace des positions par une surface, notée A_{obs}, qui est espace qu'ils occupent dans l'environnement à l'instant t de la planification. Une contrainte d'évitement d'obstacle est une partie de l'environnement, donc une partie de l'espace des positions de base $(x; y)$, dont la trajectoire ne doit pas avoir de points en commun.

Comme discuté dans le Chapitre 2, dans le planificateur de mouvement, le robot considéré est vu comme un point circulant le long de sa trajectoire $p(t)$. Afin d'assurer l'évitement d'obstacles fixes ou mobiles, les formes d'obstacles sont grossies selon un écartement qui prend en compte à minima l'empattement du robot. Soit r_{robot} le rayon du disque couvrant la forme du robot. Le grossissement des obstacles est effectué par l'homothétie $TA_{Ho}(A_{obs}, r_{robot})$ (cf Annexe 6.0.0.0) comme le montre la Figure 3.7.

Obstacle mobile

Un obstacle mobile *obs* est défini par sa surface A_{obs} à $t = 0$ et sa trajectoire $p_{obs}(t)$.

Soient (voir l'Annexe 6.0.0.0 pour les opérateurs) $\Theta p_{obs}(t)$ l'orientation de l'obstacle mobile et $p(t)$ la trajectoire du robot considéré. On note $(TA_{Tr} \circ TA_{Ro})(A_{obs}, p_{obs}(t), \Theta p_{obs}(t)) =$

$TA_{obs}(A_{obs}, t)$ la transformation affine associée au déplacement de l'obstacle mobile, composition d'une translation le long de $p_{obs}(t)$ et d'une rotation selon son orientation $\Theta p_{obs}(t)$. Cette transformation permet de déplacer la forme de l'obstacle le long de sa trajectoire selon l'instant t utilisé dans la planification, comme l'illustre la Figure 3.6. La contrainte d'évitement d'un obstacle mobile est assurée si :

$$\forall t \in [0, T_{p_i}], p_i(t) \ni TA_{obs}(A_{obs}, t) \tag{3.10}$$

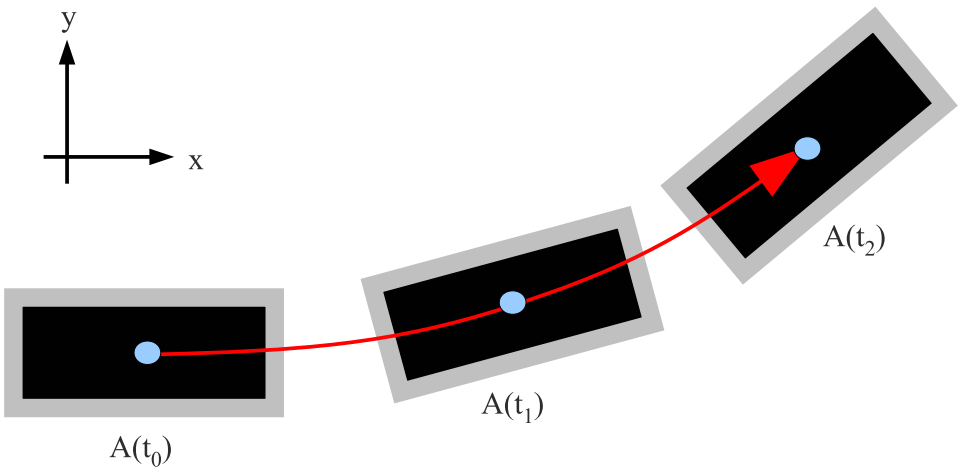

FIGURE 3.6 – Contrainte sur un obstacle mobile, avec déplacement le long de la trajectoire de l'obstacle et mise à l'échelle de l'obstacle en fonction d'une distance de sécurité.

Obstacle fixe

Un obstacle fixe *obs* est défini par sa surface A_{obs} située dans l'environnement. Sa trajectoire $p_{obs}(t)$ est fixe au cours du temps. Soit $p(t)$ la trajectoire du robot considéré. La contrainte d'évitement d'un obstacle fixe est assurée si :

$$\forall t \in [0, T_{p_i}], p_i(t) \ni A_{obs} \tag{3.11}$$

3.2.4 Restriction à une zone de déplacement

Représentation de la zone de déplacement

Les zones de déplacement sont représentées de la même manière que les obstacles. Une zone de déplacement est tout ou partie de l'environnement, donc une portion de l'espace des positions dont la base est $(x; y)$, dans laquelle le robot doit se déplacer.

Zone mobile

Un zone mobile $zone_{mobile}(t)$ est définie par sa surface A_{zone} à l'instant initial et sa trajectoire $p_{zone}(t)$. Soient (voir l'Annexe 6.0.0.0 pour les opérateurs) $\Theta p_{zone}(t)$ l'orientation de la zone mobile et $p(t)$ la trajectoire du robot considéré. On note $(TA_{Tr} \circ$

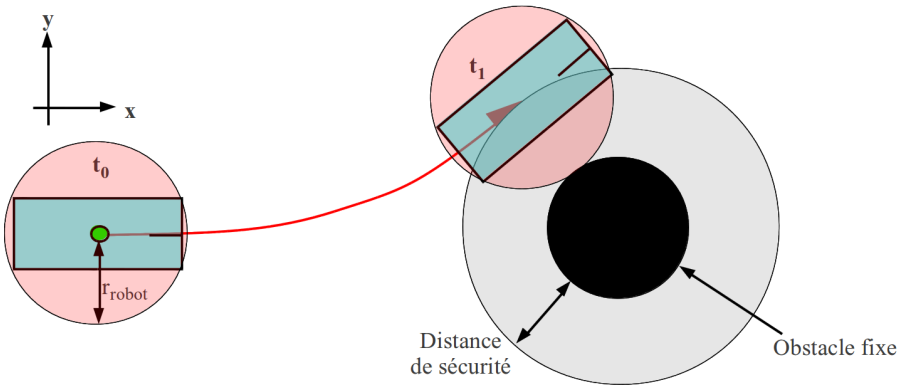

FIGURE 3.7 – Contrainte sur un obstacle fixe, avec grossissement de l'obstacle en fonction d'un écartement prenant en compte à minima l'empattement du robot.

$TA_{Ro})(A_{zone}, p_{zone}(t), \Theta p_{zone}(t)) = TA_{zone}(A_{zone}, t)$ la transformation affine associée au déplacement de la zone mobile, composition d'une translation le long de $p_{zone}(t)$ et une rotation selon son orientation $\Theta p_{zone}(t)$. La contrainte de restriction du déplacement à une zone mobile est assurée si :

$$\forall t \in [0, T_{p_i}], p(t) \in TA_{zone}(A_{zone}, t) \tag{3.12}$$

Une telle contrainte peut servir à définir une contrainte de déplacement groupé pour un ensemble de robots.

Zone fixe

Une zone fixe *zone* est représentée dans l'espace des positions par la surface A_{zone} qu'elle occupe dans l'environnement à tout instant t de la planification (la trajectoire $p_{zone}(t)$ est fixe au cours du temps). La contrainte de restriction du déplacement à une zone fixe est assurée si :

$$\forall t \in [0, T_{p_i}], p(t) \in A_{zone} \tag{3.13}$$

Une telle zone de déplacement sert en particulier à délimiter l'environnement dans lequel le robot doit se déplacer.

3.2.5 Environnement statique

En présence de contraintes d'évitement d'obstacles fixes, il est possible de rassembler toutes les contraintes d'obstacle fixe et de zone fixe au sein d'une seule et même contrainte de zone fixe. Soient $A_{environnement}$ la surface englobant la partie de l'environnement dans lequel le robot va planifier ses mouvements, $\{A_{obs}\}$ l'ensemble des surfaces représentant les obstacles fixes et $\{A_{zone}\}$ l'ensemble des surfaces représentant les zones fixes.

La surface $A_{environnement} \cap A_{k,obs} \cap A_{k,zone}$ avec $A_{k,obs} \in \{A_{obs}\}$ et $A_{k,zone} \in \{A_{zone}\}$ représente la surface réellement navigable pour toutes les contraintes statiques. Cette surface est notée $A_{autorisee}$. J'en déduit une contrainte de zone fixe basée sur cette surface, nommée $C_{autorisee}$.

3.3 Architecture de propagation

Je présente dans cette section l'architecture que j'ai retenue pour générer des trajectoires. D'abord, je construis un espace contenant les solutions : l'espace des paramètres. Cet espace

est ensuite réduit avec tout ou partie des contraintes restantes, parmi les contraintes présentées dans la section précédente ou celles spécifiques au problème étudié. Enfin sur cet espace, une solution est déterminée par un processus d'optimisation.

3.3.1 Application des contraintes sur les trajectoires

L'espace des paramètres représente les parties atteignables de l'environnement par des morceaux de trajectoire qui respectent toutes les contraintes. Soit C l'ensemble des contraintes de mouvement mises en œuvres dans un problème. Cet ensemble peut contenir en particulier les contraintes cinématiques et d'évitement d'obstacle, bien qu'il soit possible de décrire d'autres types de contraintes par une approche géométrique.

Construction de l'espace des paramètres dans sa base

Soit la projection d'une contrainte dans la base de l'espace des paramètres $M(t) \times G_c$ à l'instant t. L'intersection de toutes les contraintes projetées dans la base de l'espace des paramètres assure la vérification des contraintes à l'instant t :

$$I(t) = \{\bigcap M(t) \times G_c \mid c \in C\} \tag{3.14}$$

E représente alors l'espace des paramètres pour l'horizon de planification T_{p_i} et le pas de planification *step* tel que :

$$E(T_{p_i}) = \{\bigcap I(t) \mid t \in \{0, step, \ldots, T_{p_i}\}\} \tag{3.15}$$

Cela signifie que toutes les contraintes de C sont strictement respectées par $p_i(t)$ à chaque instant $t \in \{0, step, 2 \times step, \ldots, T_{p_i}\}$.

C'est cette solution que je retiens lorsque je dispose d'une représentation continue de l'espace des paramètres (Section 3.5 du chapitre suivant).

Construction de l'espace des paramètres dans la base des contraintes

Une autre méthode pour construire l'espace des paramètres Il est également possible de construire l'espace des paramètres en ramenant l'espace des paramètres dans la base de chaque contrainte.

La définition d'une contrainte $c(t)$ est constituée des inégalités :

$$D_- \leq g_c(\alpha_0^x, \ldots, \alpha_n^x, \alpha_0^y, \ldots, \alpha_n^y, t) \leq D_+ \tag{3.16}$$

Par conséquent, une contrainte $c(t)$ restreint les valeurs possibles des paramètres $(\alpha_0^x, \ldots, \alpha_n^x, \alpha_0^y, \ldots, \alpha_n^y)$ du morceau $p_i(t)$. Ces valeurs définissent ensuite toutes les formes de trajectoires permises par les contraintes du problème.

Si $M^{-1}(t)$ est appliqué sur l'espace des paramètres E, la fonction g_c est appliquée à chaque point $(\alpha_0^x, \ldots, \alpha_n^x, \alpha_0^y, \ldots, \alpha_n^y)$. Il est alors possible de comparer les valeurs possibles de la contrainte avec les valeurs projetées par g_c des paramètres envisagés dans E. L'intersection de ces deux espaces de valeurs fournit alors les valeurs projetées par g_c dans l'espace des paramètres qui respecte la contrainte considérée. Il suffit d'appliquer la transformation inverse pour ramener ces valeurs dans l'espace des paramètres.

La formulation de la construction de l'espace des paramètres devient alors :

$$E(T_{p_i}) = \{\bigcap M(t) \times ((M^{-1}(t) \times E) \cap G_c) \mid c \in C, t \in \{0, step, \ldots, T_{p_i}\}\} \tag{3.17}$$

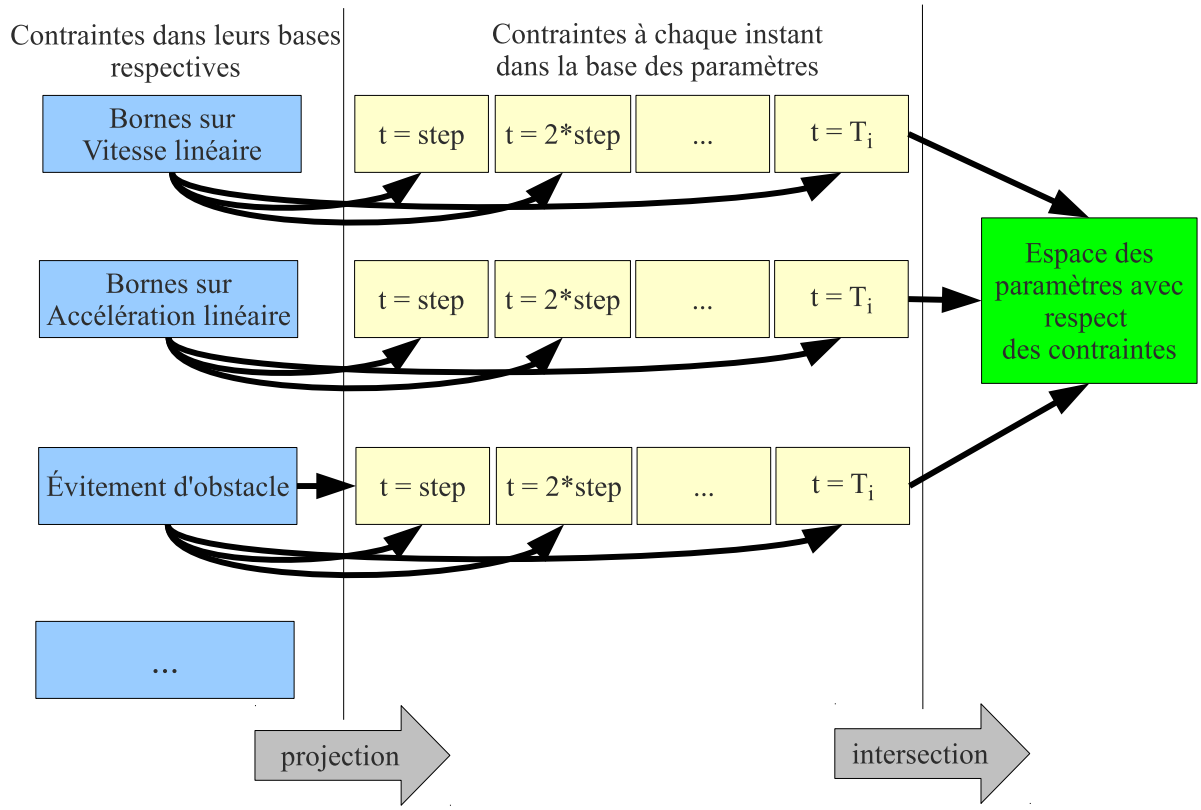

FIGURE 3.8 – Première méthode de construction de l'espace des paramètres : chaque contrainte est projetée dans la base des paramètres pour chaque pas de planification. L'intersection de l'ensemble ainsi obtenu avec l'espace des paramètres permet de retirer progressivement les n-uplets de paramètres qui ne respectent pas les contraintes du problème.

Cette solution est à privilégier en l'absence de transformation depuis l'espace des contraintes vers l'espace des solutions. C'est par exemple le cas des solutions discrètes exprimées dans la Section 3.6.

Quelle est la méthode à employer ?

Les deux méthodes présentées ci-dessus sont équivalentes. Suivant le type de trajectoire employé, l'une ou l'autre de ces solutions peut être avantageusement envisagée, en fonction de l'existence de fonctions de changement de base et de leur complexité. L'algorithme 3.3.1 de propagation reste de toute façon commun aux deux approches et seules les implémentations des transformations permettent de différencier les deux approches. Les opérations ensemblistes sont en effet directement gérées par le type de trajectoire employé et séparées de la contrainte elle-même.

3.3.2 Espace des paramètres et fenêtre dynamique

Dans le cas où une contrainte cinématique est imposée, il est tout à fait possible de la projeter sur l'espace des positions pour obtenir une estimation sur-évaluée de l'espace atteignable. Cette estimation est ensuite projetée dans l'espace des paramètres. Prenons l'exemple d'une entité

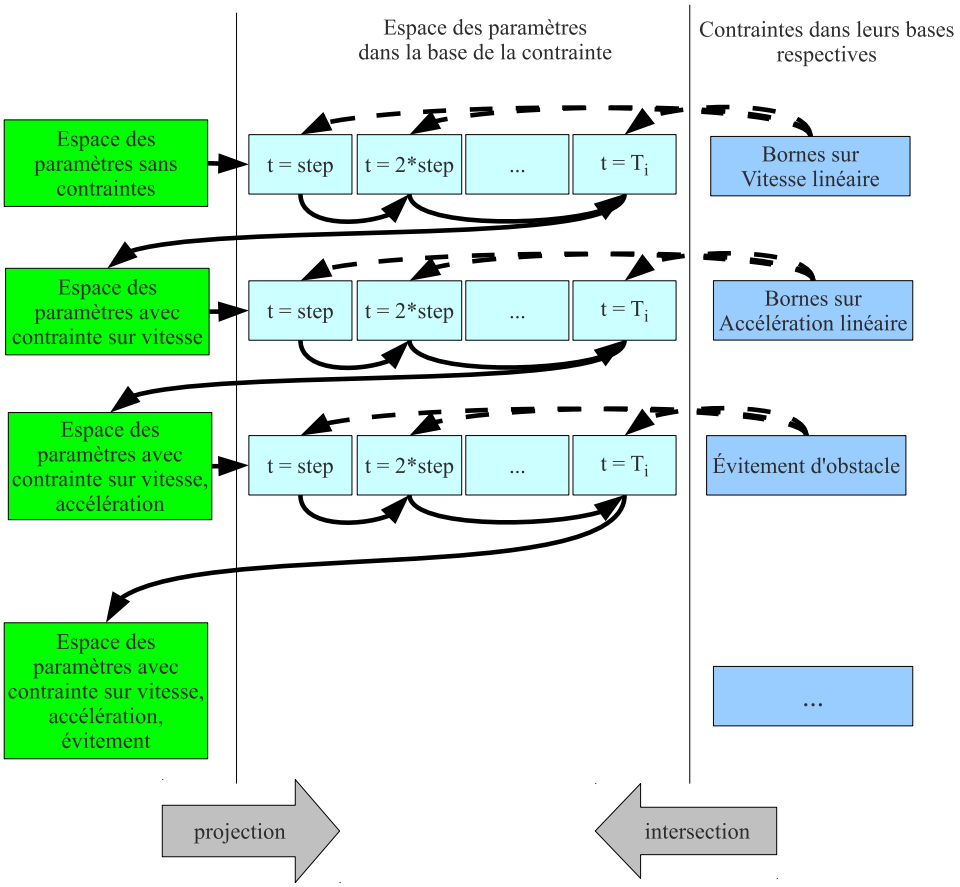

FIGURE 3.9 – Seconde méthode de construction de l'espace des paramètres : l'espace des paramètres est projeté dans la base de chaque contrainte pour chaque pas de planification. L'intersection de l'ensemble ainsi obtenu avec l'ensemble de valeurs admissibles de chaque contrainte permet de ne conserver dans l'espace des paramètres que les valeurs qui respectent les contraintes.

située en (x_0, y_0) dont le mouvement est limité entre autres par une contrainte de vitesse $c_{Speed}(t)$ bornée sur $[S^-, S^+]$, $G_{c_{Speed}}$ la surface associée et un horizon de planification T. L'homothétie $TA_{Tr}(TA_{Ho}(T, G_{c_{Speed}}), x_0, y_0) = Window$ permet d'obtenir une surface de points atteignables sur la durée T (voir l'Annexe 6.0.0.0 pour les opérateurs). L'intersection avec la surface de déplacement autorisée $A_{autorisée}$ permet de déduire la partie réellement navigable dans le problème local tout en réduisant la complexité de la résolution des contraintes. En effet, cette surface contient moins de courbes pour définir ses bordures ou au pire autant (cf. Annexe 6.0.0.0). Cette optimisation n'est pas sans rappeler l'approche de navigation par fenêtre dynamique proposée par [Fox *et al.*, 1997].

3.3.3 Choix des solutions

L'étape de propagation consiste à construire les successeurs d'un morceau de trajectoire noté $p_{kn}(t)$. Le point final de ce morceau d'horizon temporel T_{p_i} fait office de point initial pour la création des morceaux suivants. Les solutions sont alors choisies dans l'espace des paramètres $E(T_{p_i})$ en fonction d'un critère d'optimisation locale $\rho_{propagation}$. Ce critère est également

Algorithm 3.3.1 $propagation_{contraintes}()$

Require: C l'ensemble des contraintes
Require: T_{p_i} l'horizon de planification d'un morceau de trajectoire $p_i(t)$
Require: $step$ le pas de discrétisation
1: $E \leftarrow R^N$
2: $nb_{step} \leftarrow \frac{T_{p_i}}{steps}$
3: **for** $i = 1$ **TO** nb_{steps} **do**
4: $T \leftarrow step \times i$
5: **for** $c \in C$ **do**
6: $p_i.O_I(E, G_c, T)$ l'intersection de l'espace des paramètres avec les valeurs admissibles d'une contrainte à l'instant T
7: **end for**
8: **end for**

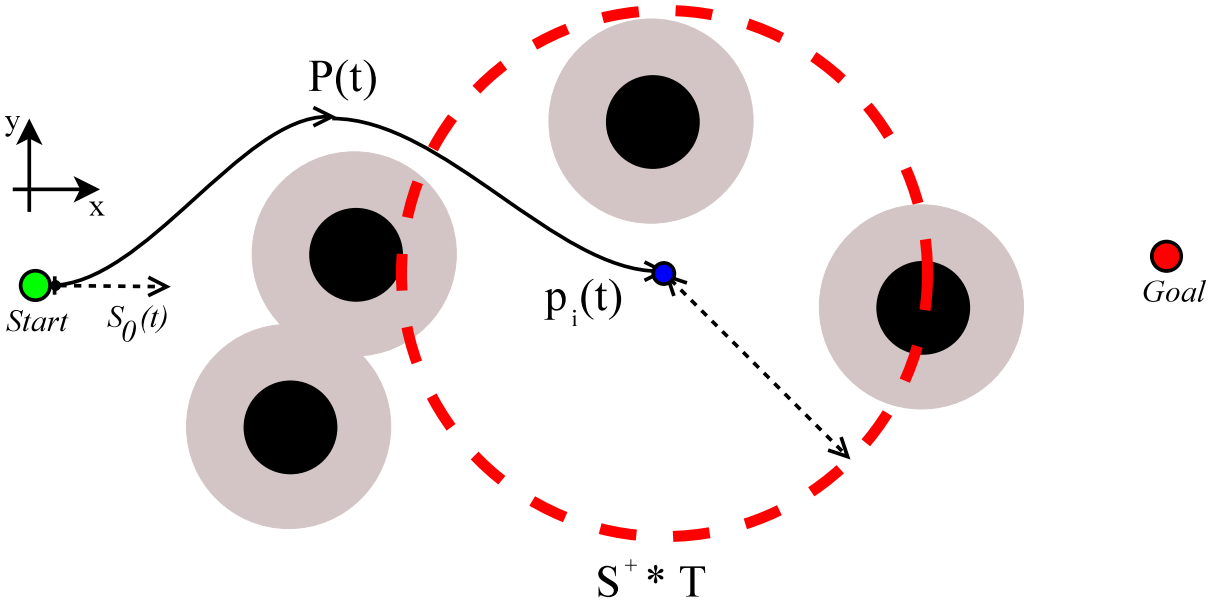

FIGURE 3.10 – La surface entourée par les tirets rouges délimite les points atteignables à partir de la borne maximale de la contrainte de vitesse S^+ pour le problème de planification local partant du point en bleu et sur un horizon de planification T.

noté $\rho_{propagation}(\alpha_0^x, \ldots, \alpha_n^x, \alpha_0^y, \ldots, \alpha_n^y, t)$ puisqu'il optimise les paramètres des morceaux de trajectoire suivants. Le gestionnaire d'objectifs est le mécanisme fournissant la fonction objectif ρ. La solution est optimale lorsqu'elle minimise le critère de recherche pour tout point pris dans l'espace des paramètres. Plusieurs manières d'obtenir des solutions sont établies.

Problème d'optimisation non-linéaire avec contraintes non-linéaires et variables continues

Dans cette méthode, la construction de l'espace des paramètres est l'étape préalable pour construire un programme non-linéaire sous contraintes non-linéaires. Les variables du problème sont continues. Ce sont les paramètres laissés libre du modèle de solution. L'idée est de

transformer l'espace des paramètres en un jeu de contraintes permettant de restreindre les valeurs des variables dans cet espace (qui dépend logiquement de la représentation de l'espace des paramètres). Le problème est initialisé avec un point appartenant à l'espace des paramètres, ce qui permet de résoudre une des critiques soulevées dans l'introduction de cette partie à propos de l'initialisation de ce type de problème pour la planification de trajectoire. Enfin, la recherche est effectuée selon un critère d'optimisation appliqué aux variables.

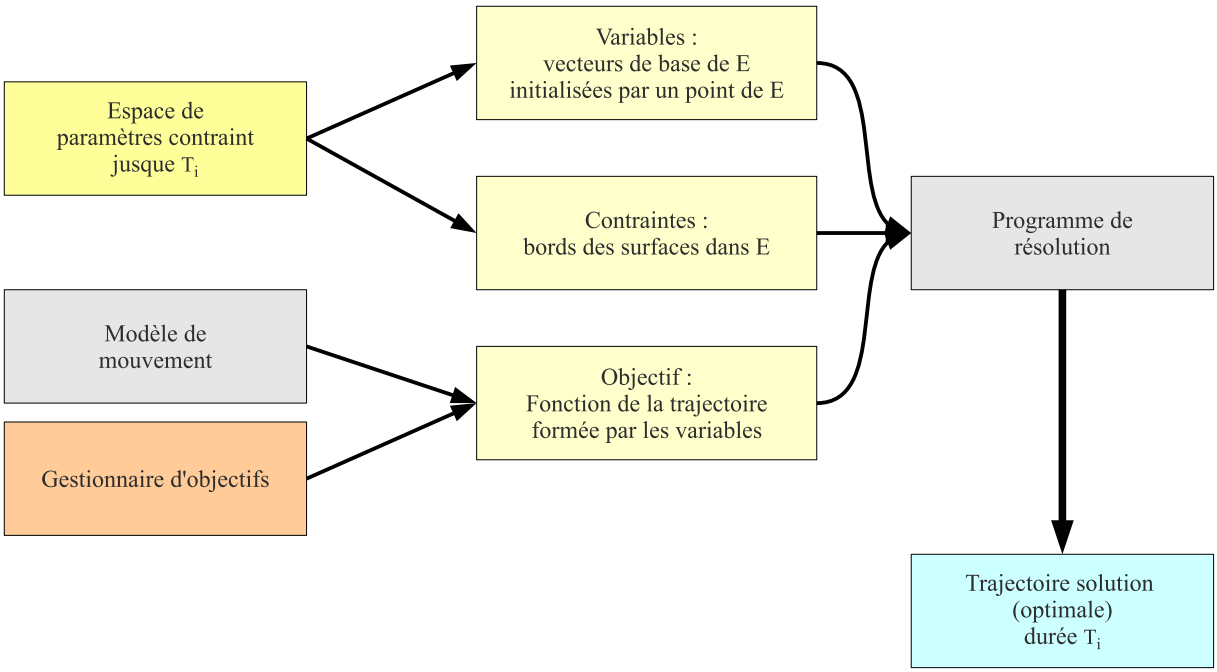

FIGURE 3.11 – Le problème de planification de trajectoire est fourni à un programme de résolution de problèmes d'optimisation non-linéaires à variables continues sous contraintes non-linéaires. Les variables sont les paramètres laissés libre de la solution. L'espace des paramètres est transformé en un jeu de contraintes permettant de borner les valeurs des variables dans cet espace. Le problème est initialisé avec un point appartenant à l'espace des paramètres. Enfin, la recherche est effectuée selon un critère d'optimisation appliqué aux variables.

Toutefois, cette démarche, illustrée par la Figure 3.11, laisse à un programme de résolution le soin d'établir la solution. Je ne maîtrise donc pas totalement le comportement du niveau de propagation. Par exemple, CFSQP utilise un bruit aléatoire pour sortir les variables des minima locaux. Cela génère un saut dans les valeurs des variables. Ainsi, une discrétisation implicite de l'espace des paramètres apparaît, discrétisation qu'il convient de repérer et correctement paramétrer. De plus, sauf cas particulier, il n'y a pas de garantie sur l'optimalité de la solution.

Problème d'optimisation non-linéaire à variables continues bornées

Dans cette méthode, la construction de l'espace des paramètres est l'étape préalable pour construire un ensemble de programmes non-linéaires à variables continues bornées. Les variables continues du problème sont les paramètres laissés libres du modèle de solution. L'idée est de paver l'espace des paramètres afin d'obtenir un ensemble de bornes sur ces variables (par exemple avec un Quadtree, voir Chapitre 2). Ensuite, la recherche est effectuée selon un critère d'optimisation appliqué aux variables.

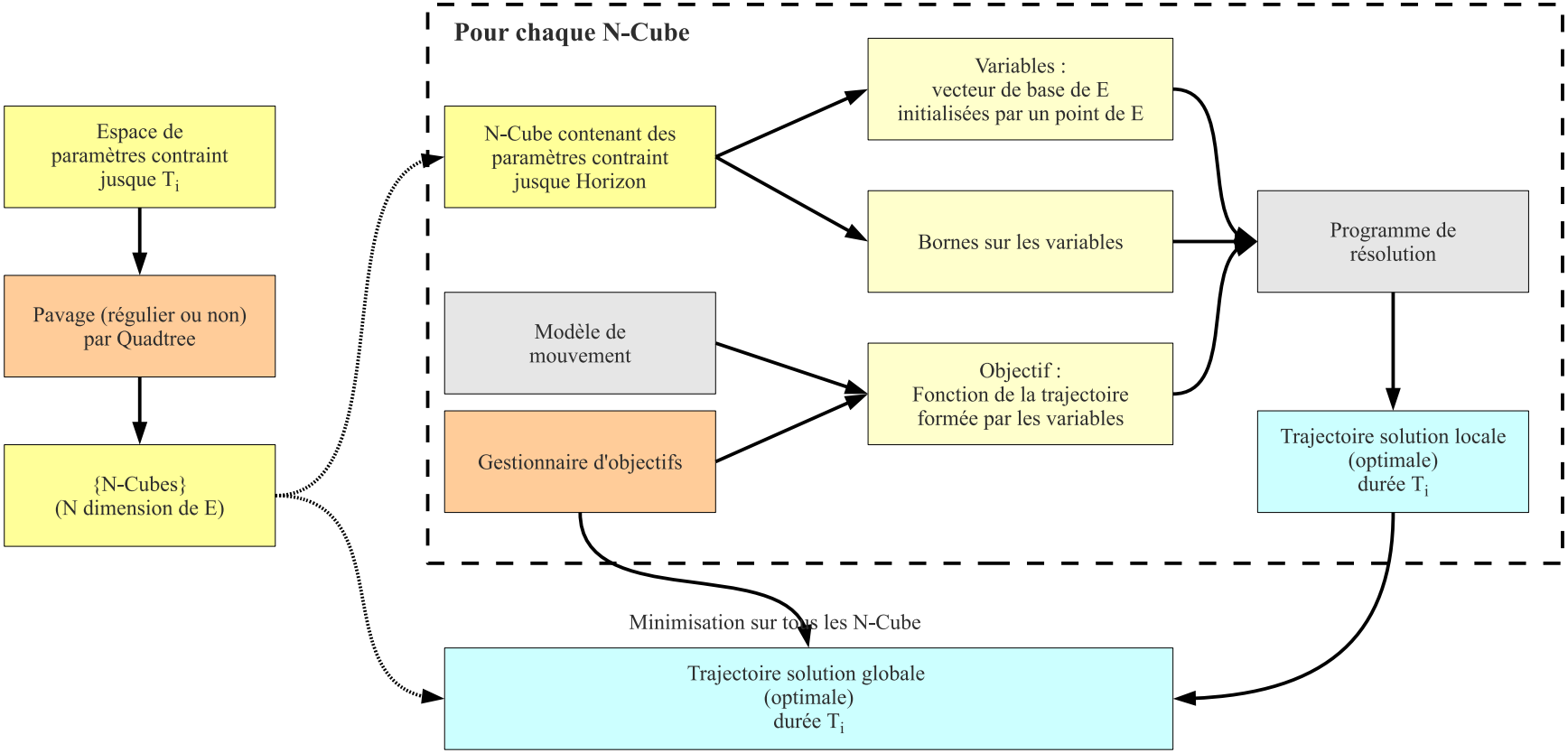

FIGURE 3.12 – Dans cette approche, l'espace des paramètres est d'abord pavé par un Quadtree. L'avantage de cette approche est que l'on peut utiliser un programme de résolution de problèmes d'optimisation non-linéaires à variable continues bornées. Les variables sont les paramètres laissés libres de la solution. Ces variables sont bornées par les n-cubes. De tels programme de résolution comme BLMVM peuvent fournir la solution optimale sur le pavé considéré. La contrainte est qu'il faut itérer cette méthode sur tous les pavés

Toutefois, l'intérêt de cette approche, illustrée par la Figure 3.12, réside dans le fait que les programmes de résolutions tel que BLMVM peuvent fournir une solution optimale. En revanche, la recherche est coûteuse car il faut tester tout les pavés. L'efficacité de l'approche dépend donc beaucoup de la méthode de pavage. De plus, le pavage peut négliger certaines portions de l'espace des paramètres.

Recherche directe de solution

Dans cette méthode, l'espace des paramètres est d'abord pavé puis un ensemble de points de référence est sélectionné. Ensuite, la recherche est effectuée selon un critère d'optimisation appliqué à la solution construite à partir de ces points de références.

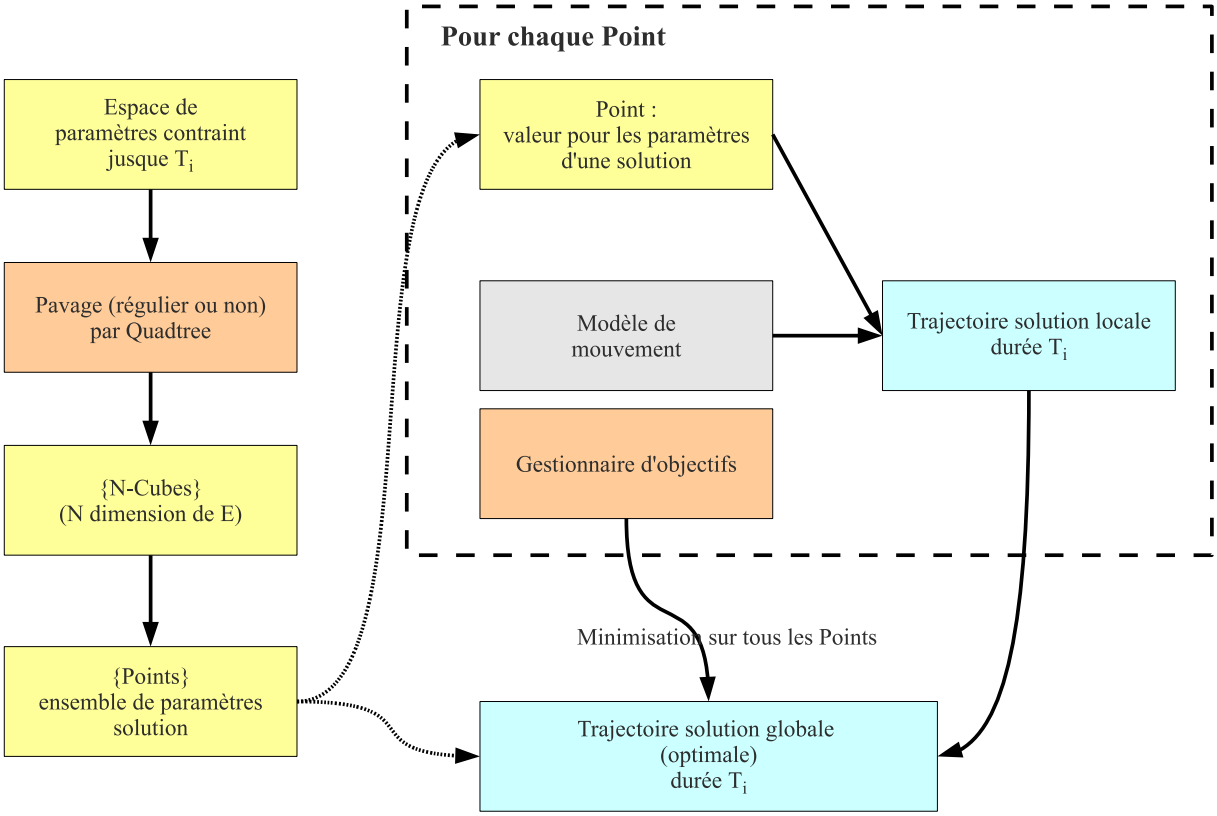

FIGURE 3.13 – Dans cette méthode de recherche de solution, l'espace des paramètres est d'abord pavé par un Quadtree. Puis de chaque pavé est tiré un ensemble de points. À partir de ces points sont générés des échantillons candidats sur lesquels est appliqué le critère de recherche afin de sélectionner la meilleure solution.

Si le pavage est plus précis sur la région de l'espace des paramètres où se trouve la solution optimale, alors cette approche, illustrée par la Figure 3.13, peut être bien plus rapide que les précédentes. Toutefois, il y a un risque de perte de précision qui dépend à la fois de la qualité du pavage, du repérage de la région où se trouve la solution optimale et du nombre de points de références pris dans chaque pavé.

Projection de l'espace des paramètres

Lorsque le critère de recherche ne concerne que le point final du morceau p_i, on peut projeter l'espace des paramètres E dans l'espace des positions par $M_{\rho_{propagation},p_i}(T_{p_i})$. Les points de référence sont alors les points finaux de toutes les solutions. Une recherche directe de la meilleure solution est possible. La meilleure solution est associée directement ou indirectement à des paramètres la générant. L'avantage de cette méthode est que toutes les solutions sont construites d'un seul coup, limitant le surcoût dû à la construction individuelle des solutions candidates.

3.3.4 Limites

La conception et l'utilisation ont soulevé de nombreuses limites. En premier lieu, les paragraphes précédents montrent que la recherche de solutions contient implicitement une discrétisation de l'espace des paramètres si cette discrétisation n'existe pas dès le départ dans le modèle de solution retenu.

Discrétisation sur le temps

E représente alors l'espace des paramètres pour l'horizon de planification T_{p_i} et le pas de planification *step* tel que les contraintes de C sont strictement respectées par $p_i(t)$ à chaque instant $t \in \{0, step, 2 \times step, \ldots, T_{p_i}\}$. Entre deux instants, le respect des contraintes n'est pas garanti : l'utilisateur devrait surévaluer les bornes appliquées aux contraintes afin de prévenir ces éventuelles erreurs.

Manipulation du modèle de trajectoire

Cette architecture requiert, pour être fonctionnelle, l'existence de changements de bases entre les espaces des contraintes mis en jeu et l'espace des paramètres. En particulier, pour facilement exprimer ces transformations, les modèles de trajectoire utilisés dans DKP présentent des paramètres linéairement différentiables. De plus, lorsque l'espace des paramètres est défini pour des dimensions élevées, il faut pouvoir représenter et manipuler un tel espace : on retrouve alors clairement les difficultés de manipulation de l'espace des configurations.

Continuité dans les contraintes

Une difficulté dans la gestion des contraintes provient des conditions de continuité et donc de la situation initiale. En effet, dans le cas particulier où un premier morceau p_{i-1} est planifié sur des bornes $[S^-_{p_{i-1}}, S^+_{p_{i-1}}]$ en vitesse linéaire et un morceau suivant p_i est planifié sur des bornes $[S^-_{p_i}, S^+_{p_i}]$ en vitesse linéaire avec $S^+_{p_i} < S^+_{p_{i-1}}$, il est alors possible que la vitesse linéaire à la fin du premier morceau soit supérieure à $S^+_{p_i}$. Dans ce cas, le morceau suivant ne peut satisfaire son propre jeu de contraintes. La solution utilisée est d'exprimer les contraintes seulement sur une partie de l'intervalle de planification afin de ne borner que l'état final par exemple. C'est ainsi que l'on peut faire varier la vitesse linéaire entre deux morceaux.

Problème de fenêtre dynamique

Dans la lignée du problème précédent, soit le cas particulier où un premier morceau p_{i-1} est planifié sur des bornes $[S^-_{p_{i-1}}, S^+_{p_{i-1}}]$ en vitesse linéaire et un morceau suivant p_i est planifié sur des bornes $[S^-_{p_i}, S^+_{p_i}]$ en vitesse linéaire avec $S^+_{p_i} < S^+_{p_{i-1}}$, si la vitesse linéaire à la fin du premier morceau est supérieure à $D^+_{p_i}$, alors la trajectoire peut parcourir une distance supérieure à celle

estimée à partir de la contrainte sur la vitesse. Pour corriger ce problème, il faut donc prendre en compte la vitesse finale du morceau $p_{p_{i-1}}$ pour estimer la fenêtre de points atteignables. Toutefois, si cette limite est couplée avec le problème précédent, selon le type de trajectoire, il faut alors tenir compte de la contrainte d'accélération linéaire (et des contraintes cinématiques dérivées supplémentaires) pour ne pas faire une mauvaise estimation de la fenêtre. Ce problème a été rencontré dans la Section 5.6.

3.4 Trajectoires et espace des paramètres

Les sections précédentes définissent un ensemble des trajectoires paramétrées manipulé à partir de sa représentation géométrique. Les contraintes qui s'appliquent sur ces trajectoires sont elles-même représentées et manipulées de manière géométrique. On donne ensuite les outils de base nécessaires à l'application des contraintes sur les trajectoires. De cette manière, je traduis les premières étapes de la planification de trajectoire comme un ensemble d'opérations ensemblistes basées sur ces transformations géométriques. Cette architecture ne résout pas de manière absolue tout type de problème de génération d'échantillons. En revanche, ce choix de conception offre de nombreuses propriétés qui peuvent être employées avantageusement dans une architecture de sélection/propagation permettant de construire des trajectoires complexes.

Mon objectif initial est de planifier des trajectoires pour des robots à deux roues indépendantes. Les robots de type unicyle possèdent des modèles cinématiques simples. Les modèles de trajectoires présentés dans ce chapitre le sont donc également. Je présente deux exemples de types de trajectoire qui ont été étudiés. Le premier est un modèle de trajectoire permettant d'exprimer un espace des paramètres continu. Il s'appuie sur les propriétés de platitude des robots à deux roues indépendantes pour fournir des solutions ;

Le second est un modèle de trajectoire permettant d'exprimer un espace des paramètres discret à partir d'un intégrateur et sur la base des courbes de Dubins/Reeds-Shepp. Ces solutions sont adaptées à des robots de type unicyle.

À partir des solutions quadratiques, je développe le processus complet de propagation pour un espace des paramètres à deux dimensions, utilisé par défaut dans DKP.

3.5 Solutions continues

3.5.1 Platitude d'un modèle

Les propriétés de platitude ont été développées par [Fliess *et al.*, 1995]. Elles ont servi à résoudre le cas du déplacement d'un véhicule avec des chariots [Lamiraux et Laumond, 1998].

D'après [Defoort *et al.*, 2007], un modèle est plat lorsqu'il existe une paramétrisation explicite de toutes les solutions. Soit le système :

$$\dot{q}_i = f(q_i, u_i) \tag{3.18}$$

où $q_i \in \mathbb{R}^n$ est l'état du système, $u_i \in \mathbb{R}^m$ est l'entrée de commande et f une application de $\mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$. Ce système est différentiellement plat s'il existe $z = [z_1, \ldots, z_m]$ (différentiellement indépendantes et ne dépendant que des entrées, des dérivées des entrées jusqu'à un ordre inférieur à n et de l'état), qui est appelé sortie plate, tel que l'état et la commande puissent être exprimés

en fonction de la sortie plate et de ses dérivées. Cela revient à dire qu'il existe trois fonctions :

$$\begin{aligned}\phi_0 &: \mathbb{R}^n \times \mathbb{R}^m \times \cdots \times \mathbb{R}^m \to \mathbb{R}^m \\ \phi_1 &: \mathbb{R}^m \times \cdots \times \mathbb{R}^m \to \mathbb{R}^n \\ \phi_2 &: \mathbb{R}^m \times \cdots \times \mathbb{R}^m \to \mathbb{R}^m\end{aligned} \tag{3.19}$$

telles que :

$$\begin{aligned}z &= \phi_0(q, u, \dot{u}, \ldots, u^{(l)}) \\ q &= \phi_1(z, \dot{z}, \ldots, z^{(l-1)}) \\ u &= \phi_2(z, \dot{z}, \ldots, z^{(l)})\end{aligned} \tag{3.20}$$

3.5.2 Solution pour robots à deux roues indépendantes

Expression

Je considère le déplacement d'un robot non-holonome à deux roues indépendantes. Ce robot se déplace sans glissement dans un plan aux coordonnées cartésiennes (x, y). Son modèle cinématique est alors :

$$\begin{aligned}\dot{x} &= v\cos(\theta) \\ \dot{y} &= v\sin(\theta) \\ \dot{\theta} &= u\end{aligned} \tag{3.21}$$

Ce modèle cinématique n'admet pas de solutions directes. Aussi, je m'appuie sur ses propriétés de platitudes pour exprimer d'autres types de solution.

Trajectoires possibles

D'après le modèle, les trois vecteurs (d'état, de commandes et la fonction qui relie commande et état suivant) sont :

$$\begin{aligned}q &= [x, y, \theta] \\ u &= [v, \omega] \\ f &= [v\cos\theta, v\sin\theta, \omega]\end{aligned} \tag{3.22}$$

En prenant comme sortie plate $z = [x, y]^T = \dot{u}$, il est possible d'écrire :

$$\begin{aligned}\theta &= arctan(\frac{\dot{y}}{\dot{x}}) \\ v &= \sqrt{\dot{x}^2 + \dot{y}^2} \\ \omega &= \frac{\dot{x}\ddot{y} - \ddot{x}\dot{y}}{\dot{x}^2 + \dot{y}^2}\end{aligned} \tag{3.23}$$

Dans ce cas :

$$\begin{aligned}z &= [x, y] = \phi_0(q) \\ u &= [\sqrt{\dot{x}^2 + \dot{y}^2}, \frac{\dot{x}\ddot{y} - \ddot{x}\dot{y}}{\dot{x}^2 + \dot{y}^2}] = \phi_1(z, \dot{z}, \ddot{z}) \\ q &= [x, y, arctan(\frac{\dot{y}}{\dot{x}})] = \phi_2(z, \dot{z})\end{aligned} \tag{3.24}$$

Le système est donc différentiellement plat.

Interprétation

Cela revient à ne pas considérer les robots à deux roues à travers un espace des configurations qui dépend de la variation des degrés de liberté du modèle cinématique. La sortie plate ne dépend que de l'état du robot. Toute trajectoire et ses dérivées peuvent donc servir à retrouver l'état du robot et ses commandes [Milam *et al.*, 2000]. Dans ce cas, le robot est considéré selon les déplacements qu'il peut faire, dépendants explicitement du paramétrage des solutions choisies.

L'autre avantage est que, contrairement aux approches telles que RRT dont les échantillons sont intégrés sur un seul pas de temps, cette approche utilise un horizon de planification bien plus grand. Soit $p_i(t)$ un morceau de trajectoire. Soit $P(t)$ la trajectoire complète qui contient les morceaux de trajectoire $p_0(t)$, $p_1(t)$,..., $p_i(t)$.

Solutions possibles : des morceaux quadratiques

Comme échantillons, j'utilise des morceaux de trajectoire $p_i(t)$ composés de polynômes du second degré, des morceaux quadratiques, avec $t \in [0, T_{p_i}]$:

$$p_i(t) = \begin{cases} x_{p_i}(t) = \alpha_0^x + \alpha_1^x t + \alpha_2^x t^2 \\ y_{p_i}(t) = \alpha_0^y + \alpha_1^y t + \alpha_2^y t^2 \end{cases} \tag{3.25}$$

Solutions possibles : des morceaux cubiques

Comme échantillons, il est également possible d'employer des morceaux de trajectoire $p_i(t)$ composés de polynômes du troisième degré, des morceaux cubiques, avec $t \in [0, T_{p_i}]$:

$$p_i(t) = \begin{cases} x_{p_i}(t) = \alpha_0^x + \alpha_1^x t + \alpha_2^x t^2 + \alpha_3^x t^3 \\ y_{p_i}(t) = \alpha_0^y + \alpha_1^y t + \alpha_2^y t^2 + \alpha_3^y t^3 \end{cases} \tag{3.26}$$

3.6 Solutions discrètes

Il est également possible de rester sur le modèle de fonctionnement des approches de type RRT. Comme indiqué dans l'introduction, mon approche est centrée sur l'exploitation de morceaux de trajectoire. Ainsi, je peux appliquer un intégrateur local pour créer un morceau de trajectoire complet jusque son horizon de planification.

3.6.1 Utilisation d'un intégrateur

Si le modèle de la solution le permet, alors l'état suivant q d'un robot peut être obtenu à partir des commandes $u = [v, \omega]$ et d'une situation initiale $q_0 = [x_0, y_0, \theta_0]$. Soit *step* le pas de discrétisation sur l'horizon de planification. Des intégrateurs du type Runge-Kutta sont largement utilisés dans les méthodes de planification de trajectoire par échantillons aléatoires : c'est le cas de RRT et d'autres approches de planification de trajectoire par échantillons vues dans le Chapitre 2.

Obtention des solutions

L'utilisation d'un intégrateur local peut s'appliquer à tout type de modèle localement intégrable pour générer des solutions locales. C'est exactement ce que font les courbes de

Dubins/Reeds-Shepp qui sont des solutions pour le déplacement de robot unicycle.

$$\begin{aligned} \dot{x} &= v \times \cos(\theta) \\ \dot{y} &= v \times \sin(\theta) \\ \dot{\theta} &= \omega \end{aligned} \tag{3.27}$$

Dans ce cas, le modèle est intégré sur une durée T. Soit l'angle θ_T à atteindre par rapport au vecteur initial. La vitesse angulaire requise vaut :

$$\omega = \frac{\theta_T}{T} \tag{3.28}$$

L'intégration du modèle cinématique permet de directement tenir compte des contraintes sur la vitesse linéaire (bornée sur S_- et S_+) et sur la vitesse angulaire (bornée sur ω_- et ω_+). J'intègre ce modèle avec un Runge-Kutta d'ordre 1 à vitesse constante sur des valeurs de $v \in [S_-, S_+]$ selon un pas de discrétisation sur la vitesse et à vitesse angulaire constante telle que $\theta \in [-\pi, +\pi]$ et $\omega \in [\omega_-, \omega_+]$ selon un pas de discrétisation de l'angle. On pose $nb\_steps = \frac{T}{step}$ et pour $n \in [0, nb\_steps]$, la position est :

$$p_i(n * step) = \begin{cases} x_{p_i}(n * step) = x_0 + \sum_{k=0}^{n-1} v \times step \times cos(\theta_0 + \omega \times k \times step) \\ y_{p_i}(n * step) = y_0 + \sum_{k=0}^{n-1} v \times step \times sin(\theta_0 + \omega \times k \times step) \end{cases} \tag{3.29}$$

L'espace des paramètres est donc un espace discret à deux dimensions dont la base est (v, θ) qui contient les valeurs permettant de générer les solutions selon la formule ci-dessus.

Vérification des contraintes

La méthode de construction de l'espace de solutions dans la base des contraintes est toute indiquée avec un tel espace discret. L'espace des paramètres E est projeté dans la base de la contrainte. L'application $M^{-1}(t)$ revient à appliquer les fonctions de contraintes g_c à partir d'une courbe paramétrique selon la formulation de la vitesse linéaire et de l'accélération linéaire. Il est alors possible de comparer les valeurs possibles de la contrainte avec les valeurs projetées par g_c des paramètres envisagés dans E. L'intersection de ces deux espaces de valeurs fournit alors les valeurs projetées par g_c dans l'espace des paramètres qui respectent la contrainte considérée. Il suffit d'appliquer la transformation inverse $M(t)$, c'est-à-dire revenir aux paramètres associés aux solutions, pour extruder les valeurs de l'espace des paramètres qui ne respectent pas les contraintes.

Il peut être fastidieux d'énumérer toutes les solutions associées à chaque jeu de paramètres. Cette inefficacité est à comparer à celle de la manipulation géométrique de l'espace des paramètres qui est développé dans la section suivante.

3.7 Échantillons à base de quadratiques dans DKP

Les solutions à base de quadratiques vont former la base des trajectoires paramétriques employées dans DKP. Les conditions de continuité établies entre morceaux de trajectoires limitent le nombre de paramètres à deux variables. Ainsi, l'espace des paramètres associé à ce type de trajectoire est représentable par des surfaces 2D, tout comme les contraintes dans leurs

bases respectives. Nous pouvons alors envisager l'application des contraintes comme un ensemble de transformations et d'opérations sur ces représentations, applicables grâce à la géométrie constructive de surface. Cette représentation des valeurs admises peut alors être partagée avec celle des contraintes. Nous utilisons dans cette section l'outil de géométrie constructive de surface qui sert à la représentation et la résolution du problème. Cet outil est développé dans l'Annexe 6.0.0.0.

DKP développe un arbre de trajectoires dans l'espace des paramètres. Cette section de la thèse présente les mécanismes qui vont servir à la génération des échantillons pour DKP pour créer un arbre d'exploration. Par homogénéité avec le Chapitre 4, les morceaux sont désormais notés avec un notation arbre : la racine de l'arbre est notée p_{k_0}. Les entiers $k_0, \ldots, k_n$ donnent le numéro du successeur. k_0 vaut toujours 0, étant la racine de l'arbre d'exploration. Un de ses fils est noté $p_{k_0 k_1}(t)$ parmi l'ensemble des K_1 successeurs désignés par $p_{k_0 K_1}(t)$. L'échantillon $p_{k_0 \ldots k_n}(t)$ désigne donc le $n-2$ fois arrière-petit-fils de la racine p_{k_0}.

3.7.1 Solution pour robots à deux roues indépendantes

On exploite la propriété de platitude expliquée dans la section précédente pour décider des solutions à la planification de trajectoire de robots à deux roues indépendantes. Cette approche utilise des morceaux de trajectoire $p_{k_0 \ldots k_n}(t)$ composés de polynômes du second degré, des morceaux quadratiques, avec $t \in [0, T_{k_0 \ldots k_n}]$:

$$p_{k_0 \ldots k_n}(t) = \begin{cases} x_{p_{k_0 \ldots k_n}}(t) = \alpha_0^x + \alpha_1^x t + \alpha_2^x t^2 \\ y_{p_{k_0 \ldots k_n}}(t) = \alpha_0^y + \alpha_1^y t + \alpha_2^y t^2 \end{cases} \tag{3.30}$$

Je détaille dans la suite les contraintes du problème à respecter : la continuité entre les échantillons, le respect des contraintes cinématiques et la commandabilité du robot avec les solutions.

Continuité

Le vecteur vitesse est noté :

$$\overrightarrow{S}_{p_{k_0 \ldots k_n}}(t) = \begin{cases} \dot{x}_{p_{k_0 \ldots k_n}}(t) = \alpha_1^x + 2 \times \alpha_2^x t \\ \dot{y}_{p_{k_0 \ldots k_n}}(t) = \alpha_1^y + 2 \times \alpha_2^y t \end{cases} \tag{3.31}$$

Le vecteur accélération est noté :

$$\overrightarrow{A}_{p_{k_0 \ldots k_n}}(t) = \begin{cases} \ddot{x}_{p_{k_0 \ldots k_n}}(t) = 2 \times \alpha_2^x \\ \ddot{y}_{p_{k_0 \ldots k_n}}(t) = 2 \times \alpha_2^y \end{cases} \tag{3.32}$$

La continuité en position et vitesse est assurée entre un morceau de trajectoire $p_{k_0 \ldots k_n}(t)$ et l'ensemble de ses successeurs $p_{k_0 \ldots k_n K_{n+1}}(t)$. Ainsi, les paramètres des degrés les plus faibles, $\alpha_0^x, \alpha_1^x, \alpha_0^y$ et α_1^y de $p_{k_0 \ldots k_n k_{n+1}}(t)$, avec $k_{n+1} \in K_{n+1}$, sont fixés par la position initiale :

$$p_{k_0 \ldots k_n}(T_{k_0 \ldots k_n}) = p_{k_0 \ldots k_n k_{n+1}}(0) \Rightarrow (\alpha_0^x = x_{k_0 \ldots k_n}(T_{k_0 \ldots k_n}), \alpha_0^y = y_{k_0 \ldots k_n}(T_{k_0 \ldots k_n})) \tag{3.33}$$

et par le vecteur vitesse initiale :

$$\overrightarrow{S}_{k_0 \ldots k_n}(T_{k_0 \ldots k_n}) = \overrightarrow{S}_{k_0 \ldots k_n k_{n+1}}(0) \Rightarrow (\alpha_1^x = \dot{x}_{k_0 \ldots k_n}(T_{k_0 \ldots k_n}), \alpha_1^y = \dot{y}_{k_0 \ldots k_n}(T_{k_0 \ldots k_n})) \tag{3.34}$$

Les paramètres restants (α_2^x, α_2^y) de $p_{k_0 \ldots k_n k_{n+1}}(t)$ déterminent donc la forme du morceau de trajectoire.

Représentation

L'ensemble des valeurs possibles des paramètres (α_2^x, α_2^y) définit un espace des paramètres E à deux dimensions. Celui-ci est représenté par une ou plusieurs surfaces qui délimitent ses valeurs possibles. Avant l'application de toute contrainte, je considère que toutes les valeurs de paramètres sont possibles, c'est-à-dire que $E = \mathbb{R}^2$. Dans les faits, $\mathbb{R}^2$ est approximé par un disque de rayon *Max*, *Max* étant la valeur maximale permise dans $\mathbb{R}$. La dimension de ce cercle est largement supérieure aux dimensions de E après application des contraintes.

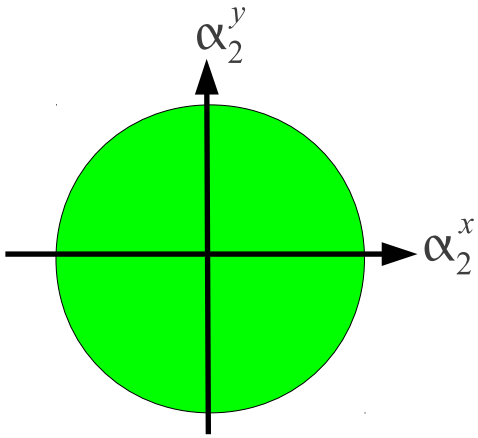

FIGURE 3.14 – Espace des paramètres de base $(\alpha_2^x; \alpha_2^y)$ avant application des contraintes.

3.7.2 Espace des paramètres à deux dimensions

Comme indiqué dans le chapitre précédent, les contraintes sont définies par une surface incluant les valeurs permises et reposant dans leurs bases respectives. Dans DKP, j'ai fait le choix d'utiliser des morceaux quadratiques de trajectoire dont les valeurs reposent dans l'espace des paramètres de base $(\alpha_2^x; \alpha_2^y)$. Cet espace est représenté par une surface 2D manipulable par géométrie constructive. Nous pouvons définir les transformations pour passer d'une base à l'autre : elles peuvent être vues comme l'opération de transformation affine $TA_M(A, M)$ par des matrices de changement de base. Ces changements de base sont précisés dans le reste de la section. Ma démarche s'apparente à de l'analyse par intervalle [Moore, 1966], mais en utilisant des surfaces pour représenter les ensembles manipulés.

Application des contraintes

Comme je souhaite travailler sur les formes autorisées des morceaux de trajectoire $p_{k_0 \ldots k_n}(t)$ selon les contraintes du problème, je définis les matrices de transformation affine permettant les changements de base entre l'espace des paramètres dans la base (α_2^x, α_2^y) et :

- la base de l'espace des accélérations $(\ddot{x}; \ddot{y})$;
- la base de l'espace des vitesses $(\dot{x}; \dot{y})$;
- la base de l'espace des positions $(x; y)$.

Je note $M_{c,p}(t)$ la matrice de transformation d'une contrainte $c(t)$ depuis sa base vers l'espace des paramètres et $M_{c,p}^{-1}(t)$ la matrice de transformation depuis l'espace des paramètres pour les morceaux de trajectoire du type p vers la base de la contrainte $c(t)$.

Espace des positions et changement de base

Reprenons la définition d'un morceau quadratique :

$$p_{k_0 \ldots k_n}(t) = \begin{cases} x_{p_{k_0 \ldots k_n}}(t) = \alpha_0^x + \alpha_1^x t + \alpha_2^x t^2 \\ y_{p_{k_0 \ldots k_n}}(t) = \alpha_0^y + \alpha_1^y t + \alpha_2^y t^2 \end{cases} \tag{3.35}$$

Les paramètres α_2^x et α_2^y sont indépendants. Nous pouvons exprimer la matrice de changement de base, notée $M_{Pos,p}^{-1}(t)$, depuis l'espace des paramètres vers l'espace des positions *Pos* :

$$M_{Pos,p}^{-1}(t) = \begin{pmatrix} t^2 & 0 & \alpha_0^x + \alpha_1^x t \\ 0 & t^2 & \alpha_0^y + \alpha_1^y t \\ 0 & 0 & 1 \end{pmatrix} \tag{3.36}$$

De même, les équations du morceau de quadratique peuvent se réécrire ainsi :

$$p_{k_0 \ldots k_n}(t) = \begin{cases} \alpha_2^x = \dfrac{(x_{p_{k_0 \ldots k_n}}(t) - \alpha_0^x - \alpha_1^x t)}{t^2} \\ \alpha_2^y = \dfrac{(y_{p_{k_0 \ldots k_n}}(t) - \alpha_0^y - \alpha_1^y t)}{t^2} \end{cases} \tag{3.37}$$

Ainsi, la matrice de changement de base, notée $M_{Pos,p}(t)$, depuis l'espace des positions *Pos* vers l'espace des paramètres est :

$$M_{Pos,p}(t) = \begin{pmatrix} \dfrac{1}{t^2} & 0 & \dfrac{(-\alpha_0^x - \alpha_1^x t)}{t^2} \\ 0 & \dfrac{1}{t^2} & \dfrac{(-\alpha_0^y - \alpha_1^y t)}{t^2} \\ 0 & 0 & 1 \end{pmatrix} \tag{3.38}$$

Espace des accélérations

Le vecteur accélération est noté :

$$\overrightarrow{A}_{p_{k_0 \ldots k_n}}(t) = \begin{cases} \ddot{x}_{p_{k_0 \ldots k_n}}(t) = 2 \times \alpha_2^x \\ \ddot{y}_{p_{k_0 \ldots k_n}}(t) = 2 \times \alpha_2^y \end{cases} \tag{3.39}$$

La matrice de changement de base depuis l'espace des paramètres vers l'espace des accélérations est déduite de l'équation du vecteur accélération. Elle est notée $M_{\overrightarrow{A},p}^{-1}(t)$:

$$M_{\overrightarrow{A},p}^{-1}(t) = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{3.40}$$

De même, les équations du vecteur accélération peuvent se réécrire ainsi :

$$p_{k_0 \ldots k_n}(t) = \begin{cases} \alpha_2^x = \dfrac{\ddot{x}_{p_{k_0 \ldots k_n}}(t)}{2} \\ \alpha_2^y = \dfrac{\ddot{y}_{p_{k_0 \ldots k_n}}(t)}{2} \end{cases} \tag{3.41}$$

Ainsi, la matrice de changement de base, notée $M_{\overrightarrow{A},p}(t)$, depuis l'espace des accélérations vers l'espace des paramètres est :

$$M_{\overrightarrow{A},p}(t) = \begin{pmatrix} \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{2} & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{3.42}$$

Notons que cette contrainte et les changements de base associés sont indépendants du temps.

Espace des vitesses

Le vecteur vitesse est noté :

$$\overrightarrow{S}_{p_{k_0\ldots k_n}}(t) = \begin{cases} \dot{x}_{p_{k_0\ldots k_n}}(t) = \alpha_1^x + 2 \times \alpha_2^x t \\ \dot{y}_{p_{k_0\ldots k_n}}(t) = \alpha_1^y + 2 \times \alpha_2^y t \end{cases} \tag{3.43}$$

La matrice de changement de base depuis l'espace des paramètres vers l'espace des vitesses est déduite de l'équation du vecteur vitesse. Elle est notée $M^{-1}_{\overrightarrow{S},p}(t)$:

$$M^{-1}_{\overrightarrow{S},p}(t) = \begin{pmatrix} 2 \times t & 0 & \alpha_1^x \\ 0 & 2 \times t & \alpha_1^y \\ 0 & 0 & 1 \end{pmatrix} \tag{3.44}$$

De même, les équations du vecteur vitesse peuvent se réécrire ainsi :

$$p_{k_0\ldots k_n}(t) = \begin{cases} \alpha_2^x = \dfrac{(\dot{x}_{p_{k_0\ldots k_n}}(t) - \alpha_1^x)}{(2 \times t)} \\ \alpha_2^y = \dfrac{(\dot{y}_{p_{k_0\ldots k_n}}(t) - \alpha_1^y)}{(2 \times t)} \end{cases} \tag{3.45}$$

Ainsi, la matrice de changement de base, notée $M_{\overrightarrow{S},p}(t)$, depuis l'espace des vitesses vers l'espace des paramètres est :

$$M_{\overrightarrow{S},p}(t) = \begin{pmatrix} \dfrac{1}{(2 \times t)} & 0 & \dfrac{-\alpha_1^x}{(2 \times t)} \\ 0 & \dfrac{1}{(2 \times t)} & \dfrac{-\alpha_1^y)}{(2 \times t)} \\ 0 & 0 & 1 \end{pmatrix} \tag{3.46}$$

Passage à l'échelle

La complexité de l'espace des paramètres à deux dimensions basé sur les ensembles semi-finis dépend du nombre de contraintes $card(C)$ et du nombre de pas de temps nécessaires à l'évaluation et la construction d'un morceau de trajectoire, donc du pas de temps *step* et de l'horizon de planification $T_{k_0\ldots k_n}$.

Rappelons que $nb\_steps = \frac{T_{k_0\ldots k_n}}{step}$ est le nombre de pas de temps auxquels sont vérifiés chaque contrainte. Le temps de calcul provient alors essentiellement de la difficulté à faire l'intersection de chacune des contraintes à chaque instant $k \times step$ avec $k \in 0, 1, \ldots, nb\_steps$.

Soit une contrainte de C représentée par une surface notée pour simplifier A, dont la bordure $B(A)$ est composé de $card(B(A))$ éléments quadratiques. Soient A_0 et A_1 deux surfaces représentant deux contraintes différentes. Dans le cas le plus défavorable, l'intersection de ces

deux surfaces A_0 et A_1 nécessite $card(B(A_0)) \times card(B(A_1))$ recherches d'intersection entre leurs bordures dans le but de construire la surface résultante. Celle-ci contient $card(B(A_0)) + card(B(A_1))$ éléments quadratiques dans le pire des cas.

La construction de l'espace des paramètres, limité par toutes les contraintes, requiert l'intersection de $card(C) \times nb\_steps$ surfaces. Toutes les contraintes sont définies à cet instant par la surface A. Le nombre total d'intersections nécessaires à la création de l'espace des paramètres est :

$$\sum_{i=0}^{card(C)\times nb\_steps} \{(\sum_{k=0}^{i-1} card(B(A_k))) \times card(B(A_i))\} \tag{3.47}$$

Ce nombre d'intersections est à peu près proportionnel au carré du nombre total d'évaluations nécessaire à la vérification de toutes les contraintes. Trois optimisations efficaces sont utilisées dans DKP pour réduire cette complexité. La première est d'arrêter l'évaluation des contraintes quand l'espace des paramètres est vide. La seconde est de simplifier les bordures par l'utilisation de segments de droite : les calculs d'intersection sont plus nombreux mais également plus simples. La dernière consiste à limiter l'évaluation des obstacles par une fenêtre dynamique comme cela est suggéré dans le chapitre précédent.

3.7.3 Recherche de solution

Dans ce cadre, il est tout à fait possible de réutiliser les méthodes d'optimisation présentées dans le chapitre précédent. Je montre leur mise en œuvre.

Recherche directe : un problème d'optimisation non-linéaire

L'espace des paramètres E résultant de l'application des contraintes est une surface semi-algébrique. Sa bordure est constituée de $card(B(E))$ éléments quadratiques qui permettent de vérifier si un point appartient à cette surface (voir Annexe 6.0.0.0). Ces équations constituent les contraintes non-linéaires d'un problème d'optimisation non-linéaire. Les variables continues sont les paramètres α_2^x, α_2^y qui définissent les morceaux de trajectoires.

Après détermination de l'espace des paramètres E, le problème de planification de trajectoire sous contraintes cinématiques est alors formulé de la manière suivante :

$$min(\rho(p_{k_0 \ldots k_n}(T_{k_0 \ldots k_n}))) \Leftrightarrow min(\rho(\alpha_2^x, \alpha_2^y)) \tag{3.48}$$

sous les contraintes pour tout $b(E)_k \in B(E)$:

$$b(E)_k(\alpha_2^x, \alpha_2^y) > 0 \tag{3.49}$$

Un programme de résolution tel que CFSQP peut résoudre ce type de problème. Les avantages de cette solution par rapport aux démarches classiques d'optimisation sont multiples. La résolution du problème peut être stoppée si l'espace des paramètres est vide, ce qui est facile à établir dans mon exemple. L'espace des paramètres est entièrement déterminé à l'avance. Une ou plusieurs initialisations sont sélectionnables selon la fonction à optimiser : il suffit de choisir des points appartenant à la surface ; La résolution du problème ne dépend plus du temps, dont l'ordre d'évaluation des pas de temps peut influencer les solutions obtenues par le programme de résolution (l'espace des paramètres dépend du temps). Toutefois, l'utilisation de mécanismes aléatoires pour parcourir l'espace des paramètres ne garantit pas que la solution obtenue soit la solution optimale. Par contre, une recherche peut être effectuée sur chaque sous-partie distincte de l'espace des paramètres pour au moins optimiser une solution sur la partie qui contient la solution optimale.

Discrétisation selon un Quadtree pour un problème à variables bornées

Dans la mesure où la résolution sous forme de problème d'optimisation non-linéaire peut s'avérer lourd, je propose de paver l'espace des paramètres E par des rectangles *rec* en partant du rectangle englobant l'espace des paramètres. Le Quadtree [Finkel et Bentley, 1974], vu dans la Section 2.2, est utilisé pour découper l'espace des paramètres. Il permet d'obtenir trois sortes de rectangles dont l'intersection avec E est :

- vide : $rec \cap E = \emptyset$;
- complète : $rec \cap E = rec$;
- incomplète : $rec \cap E \neq \emptyset \neq rec$.

Le critère d'arrêt de la récursivité est atteint lorsque le paramètre de profondeur *depth* est atteint. Soient *Rec* l'ensemble des rectangle *rec* obtenu par le Quadtree et Rec_{full} l'ensemble des rectangles ayant une intersection complète avec E. Un rectangle *rec* appartenant à Rec_{full} impose des bornes locales aux paramètres (α_2^x, α_2^y). Avec chacun de ces rectangles, nous pouvons alors définir un ensemble de problèmes d'optimisation quadratiques, dénommé $Q(E)$, avec des variables bornées par des limites issues de *rec* et une fonction objectif $\rho_{obj}(p_{k_0 \ldots k_n}, t)$ à minimiser, appliquée aux paramètres α_2^x et α_2^y du morceau de trajectoire $p_{k_0 \ldots k_n}$.

Les programmes utilisés pour résoudre ce type de problème, tel que BLMVM, sont plus simples et permettent de trouver la meilleure solution. Pour l'identifier, il faut répéter cette recherche parmi tous les rectangles *rec*. Si le Quadtree est itéré suffisamment de fois, alors, dans le cadre d'une résolution numérique, à défaut d'avoir la solution optimale exacte, la solution obtenue peut être considérée comme telle. Une optimisation efficace consiste à choisir des points remarquables sur les rectangles, par exemple les coins et le centre. La solution est alors obtenue parmi l'ensemble des points considérés.

Cas particuliers

Dans le cas de la minimisation de la distance entre le point final d'un échantillon $p_{k_0 \ldots k_n}(T_{k_0 \ldots k_n})$ et un point objectif *Goal*, il est aisé d'obtenir analytiquement les paramètres solutions α_2^x et α_2^y nécessaires à l'atteinte de l'objectif *Goal* : Soit (x_g, y_g) la position de l'objectif *Goal*.

$$\begin{aligned} \alpha_2^x &= \frac{(x_g - \alpha_0^x - \alpha_1^x T_{k_0 \ldots k_n})}{{T_{k_0 \ldots k_n}}^2} \\ \alpha_2^y &= \frac{(y_g - \alpha_0^y - \alpha_1^y T_{k_0 \ldots k_n})}{{T_{k_0 \ldots k_n}}^2} \end{aligned} \tag{3.50}$$

Si le point (α_2^x, α_2^y) appartient à E, alors il valide toutes les contraintes et le problème local est résolu. Ces paramètres solutions définissent la forme du morceau de trajectoire $p_{k_0 \ldots k_n}(t)$ depuis *Start* jusque *Goal* avec $t \in [0, T_{k_0 \ldots k_n}]$. Sinon, la solution optimale sur E se trouve à sa frontière. Pour améliorer les performances de DKP, le Quadtree est alors itéré uniquement sur les intersections incomplètes, dans le but d'obtenir un meilleur pavage de la bordure. La solution est alors obtenue avec l'une des méthodes précédemment citées. L'ensemble de la méthode est représenté sur la Figure 3.15.

3.7.4 Exemple

Soit la situation suivante d'un robot à deux roues indépendantes. Je choisis les solutions paramétriques à base de quadratiques comme échantillons de trajectoire. Le problème est initialisé avec les paramètres suivants :

- la position initiale est $Start = (0, 0)$;

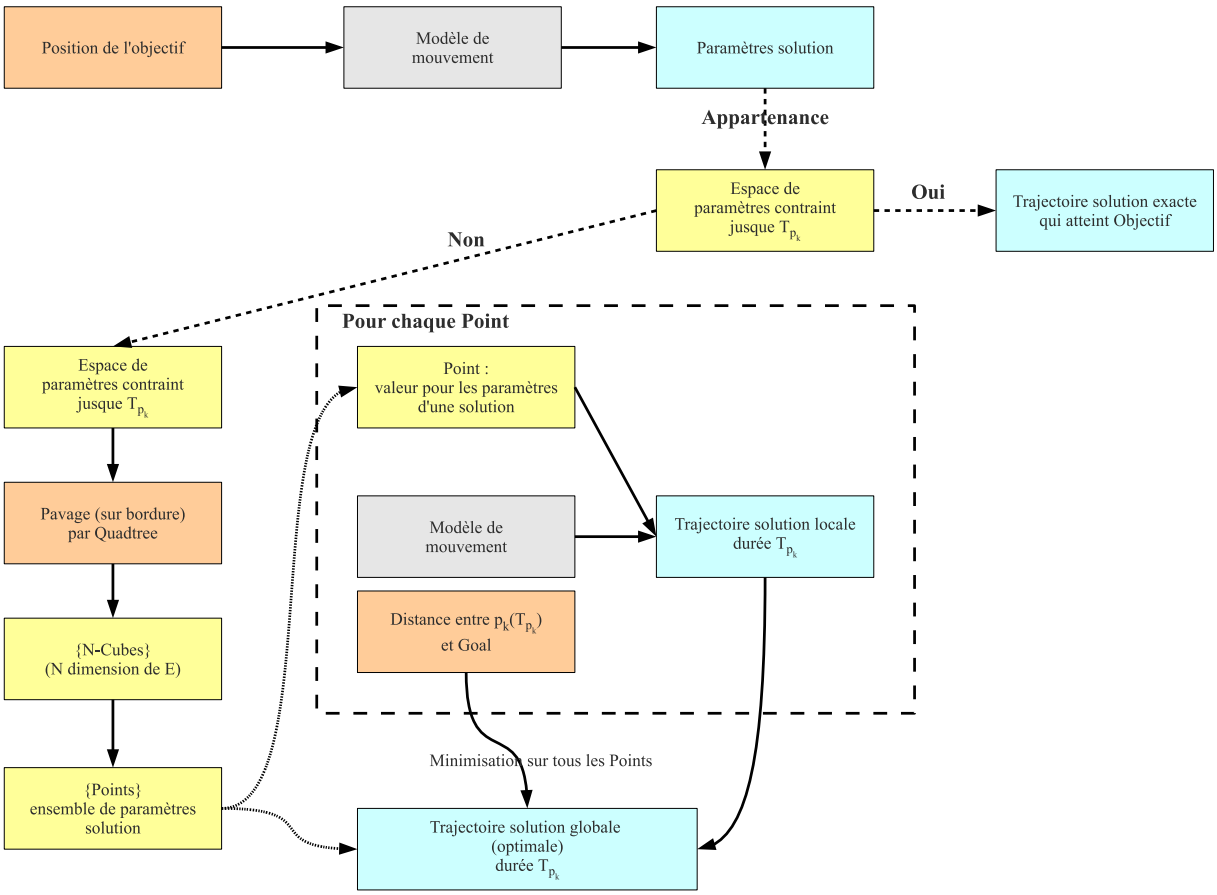

FIGURE 3.15 – Dans le cas particulier de la minimisation de distance entre l'objectif et la fin d'un échantillon, une solution directe est recherchée. Si elle appartient à l'espace des paramètres, alors le problème est résolu. Sinon, l'espace des paramètres est pavé par un Quadtree qui est itéré sur la bordure de l'espace des paramètres où se trouve la solution optimale. Puis, un ensemble de points est tiré de chaque pavé. À partir de ces points sont générés des échantillons candidats sur lesquels est appliqué le critère de recherche afin de sélectionner la meilleure solution.

- le vecteur vitesse initial est $\overrightarrow{S}(0) = (0.1, 0.2)$.
- une distance de sécurité autour du robot fixée à $r_{robot} = 0.5\text{m}$;
- les contraintes suivantes, considérées pour chaque pas de temps $step = 0.1s$ jusque l'horizon de planification $T = 10s$:
  - l'accélération linéaire bornée entre 0 et 1 m/s^2 ;
  - la vitesse linéaire bornée entre 0 et 1 m/s ;
  - un obstacle sous la forme d'un disque de rayon 1 centré en $Obs = (2, 0)$;
- un objectif $Goal = (4, 0)$.

Forme de l'espace des paramètres

La Figure 3.16 montre les contraintes de vitesse linéaire et vitesse angulaire projetées dans l'espace des paramètres puis évaluées sur tous les pas de temps du problème de planification de trajectoire.

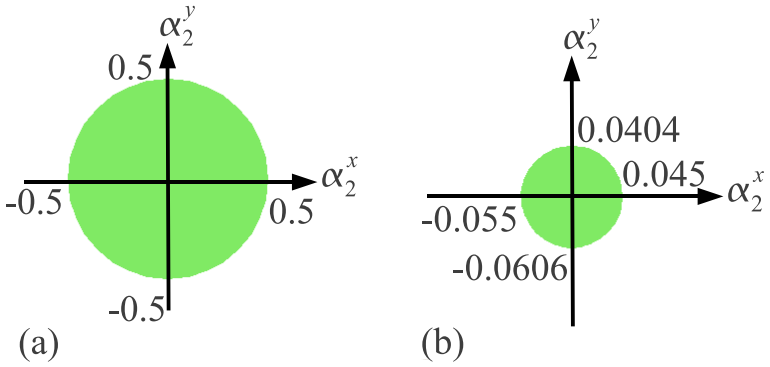

FIGURE 3.16 – Les contraintes (a) d'accélération linéaire et (b) de vitesse linéaire, projetées dans l'espace des paramètres après évaluation sur tous les pas de temps de la planification.

Contrainte sur l'accélération linéaire La matrice de changement de base depuis l'espace des accélérations vers l'espace des paramètres est :

$$M_{\overrightarrow{A},p}(t) = \begin{pmatrix} \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{2} & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{3.51}$$

Appliquée comme transformation affine sur une surface, elle la met à l'échelle $\frac{1}{2}$. La contrainte sur l'accélération linéaire est représentée par un disque de rayon 1 dans l'espace des accélérations : la contrainte ramène l'espace des paramètres à un disque de rayon $\frac{1}{2}$.

Contrainte sur la vitesse linéaire Ainsi, la matrice de changement de base depuis l'espace des vitesses vers l'espace des paramètres est :

$$M_{\overrightarrow{S},p}(t) = \begin{pmatrix} \frac{1}{(2\times t)} & 0 & \frac{-\alpha_1^x}{(2\times t)} \\ 0 & \frac{1}{(2\times t)} & \frac{-\alpha_1^y)}{(2\times t)} \\ 0 & 0 & 1 \end{pmatrix} \tag{3.52}$$

Appliquée comme transformation affine sur une surface, il y a translation de $(\frac{-\alpha_1^x}{(2\times t)}, \frac{-\alpha_1^y)}{(2\times t)})$ et une mise à l'échelle de $\frac{1}{2\times t}$. La contrainte sur la vitesse linéaire est représentée par un disque de rayon 1 dans l'espace des vitesses. Cette contrainte est projetée pour tout $t \in \{step = 0.1s, 2 \times step, \ldots, T = 10s\}$. Après intersection entre chacune des projections, il reste dans l'espace des paramètres un disque de rayon $\frac{1}{2\times 10}$ centré en $(\frac{-0.1}{(2\times 10)}, \frac{0.2}{(2\times 10)})$.

Contrainte d'évitement d'obstacle La Figure 3.17 présente en particulier l'espace des paramètres pour la contrainte d'évitement d'obstacle.

La matrice de changement de base depuis l'espace des positions vers l'espace des paramètres

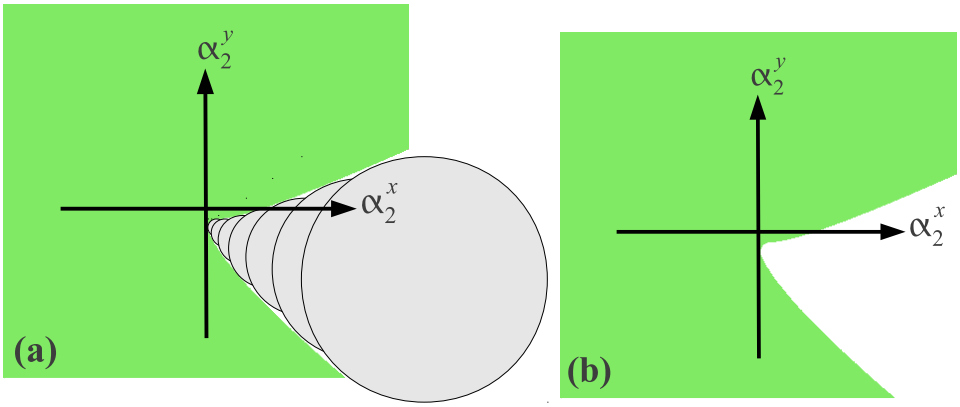

FIGURE 3.17 – La contrainte d'évitement d'obstacle projetée dans l'espace des paramètres. La figure (a) montre l'empreinte de l'obstacle qui extrude sur la figure (b) l'espace des paramètres pour chaque pas *step* jusque l'horizon de planification T.

est :

$$M_{Pos,p}(t) = \begin{pmatrix} \frac{1}{t^2} & 0 & \frac{(-\alpha_0^x - \alpha_1^x t)}{t^2} \\ 0 & \frac{1}{t^2} & \frac{(-\alpha_0^y - \alpha_1^y t)}{t^2} \\ 0 & 0 & 1 \end{pmatrix} \tag{3.53}$$

Appliquée comme transformation affine sur une surface, il y a une translation du disque en $(\frac{(-\alpha_0^x - \alpha_1^x t)}{t^2}, \frac{(-\alpha_0^y - \alpha_1^y t)}{t^2})$ et une mise à l'échelle de $\frac{1}{t^2}$. Appliqué au dernier instant de la planification T, le disque de rayon 1 qui représente l'obstacle devient un disque de rayon $\frac{1}{100}$ centré en $(\frac{1}{100}, \frac{-1}{100})$. À chaque pas de planification *step*, cette transformation fait donc apparaître un trou dans l'espace des paramètres. Par conséquent, une extrusion apparaît progressivement dans l'espace des paramètres à mesure que les pas de temps sont évalués jusqu'au dernier pas calculé ci-dessus.

Choix d'une solution

La Figure 3.18 montre en (a) l'intersection de toutes les contraintes développées ci-dessus dans l'espace des paramètres. En (b), le choix d'un point (α_2^x, α_2^y) décide de la forme d'un échantillon, initialisé selon les conditions initiales du problème. Cette forme respecte tout ou partie des contraintes selon que les paramètres sont choisis à l'intérieur ou à l'extérieur de l'espace des paramètres. Ainsi, p_0 et p_1, conséquences de paramètres pris sur le bord de l'extrusion de l'espace des paramètres provoquée par l'obstacle, sont des échantillons qui frôlent l'obstacle. p_2, conséquence de paramètres pris dans l'extrusion de l'espace des paramètres provoquée par l'obstacle, est un échantillon qui ne respecte pas la contrainte d'évitement d'obstacle.

Ensuite, une recherche de solution est effectuée dans l'espace des paramètres. Les méthodes proposées dans cette section reviennent à projeter l'espace des paramètres dans l'espace des positions pour $t = T$. Dans cette portion de l'environnement, un point final est recherché dans cet espace en minimisant la distance à *Goal*. Nous obtenons ainsi la trajectoire localement optimale comme le montre la Figure 3.19.

L'exemple donné montre une solution optimale localement, mais qui risque d'être gênante pour la suite si la vitesse est trop grande au point final : on n'aura pas forcément assez de temps

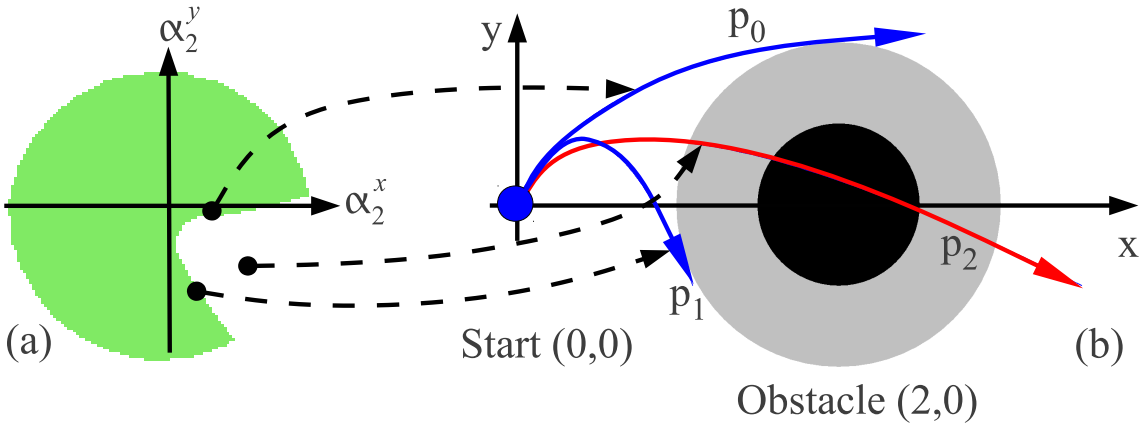

FIGURE 3.18 – La surface en vert (a) contient l'ensemble des paramètres (α_2^x, α_2^y) valides qui sont éligibles comme solutions, car ils respectent toutes les contraintes du problème. (b) montre l'impact du choix d'un point (α_2^x, α_2^y) sur la forme d'un échantillon. Le choix des paramètres (α_2^x, α_2^y) en dehors de l'espaces des paramètres conduit à la violation d'une ou plusieurs contraintes, ici la contrainte d'évitement d'obstacle.

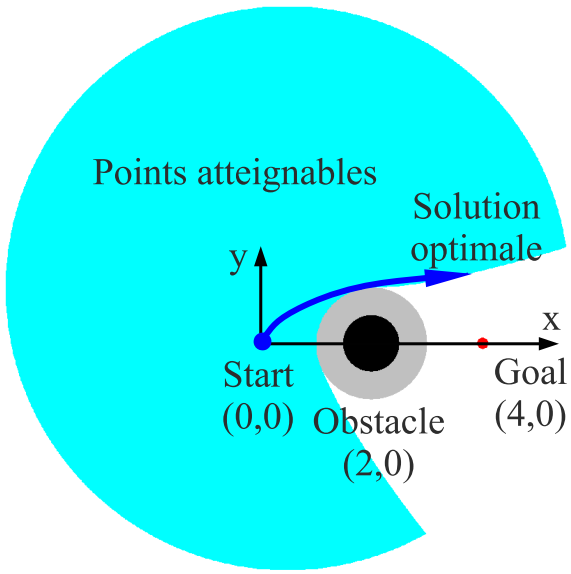

FIGURE 3.19 – La surface en cyan est construite à partir de tous les points finaux des trajectoires valides construites à partir de points de l'espace des paramètres. Cette figure est simplement obtenue en projetant l'espace des paramètres dans l'espace des positions pour $t = T$. Sur cette surface, il suffit de rechercher le point minimisant la distance à *Goal* pour obtenir la trajectoire localement optimale.

pour tourner vers l'objectif.

3.8 Analyse

Dans cet exemple, je vois donc que la simplicité des trajectoires permet d'envisager l'ensemble du processus de propagation comme un ensemble d'opérations ensemblistes et de transformations affines mettant en jeu l'espace des paramètres et les contraintes, vues comme des surfaces grâce à la géométrie constructive de trajectoire. Par rapport aux approches usuelles, je règle plusieurs problèmes. Je peux vérifier l'existence de solutions avant l'étape d'optimisation. En séparant construction de l'espace des paramètres et optimisation, je ne biaise pas la recherche par l'établissement d'un ordre de vérification des contraintes. L'initialisation du problème de planification nécessite que tout point choisi dans l'espace des paramètres respecte les contraintes du problème. La discrétisation qui apparaît sous la forme d'un bruit sur les variables lors de la recherche d'une solution par les programmes de résolution de problèmes d'optimisation non-linéaires est ici assumée et visible puisque cette discrétisation correspond tout simplement au Quadtree appliqué sur l'espace des paramètres : ainsi, cette démarche permet de contrôler la précision. On peut définir des nouvelles contraintes simplement en donnant l'espace des valeurs possibles dans la base de la contrainte : cette approche pour la propagation est extensible. Ces propriétés sont exploitées dans la couche de sélection présentée dans le chapitre suivant consacré à DKP.

3.9 Conclusion

L'approche de propagation présentée dans ce chapitre crée des morceaux de trajectoires sur un temps fixé. Le modèle des échantillons, des courbes à base de quadratiques, possède des paramètres fixés par continuité et deux paramètres laissés libres : ces derniers constituent la base de l'espace des paramètres. Cet espace est construit en faisant l'intersection des surfaces pour chaque contrainte projetées dans leurs bases respectives. Les points de cet espace fixent les paramètres libres des échantillons générant des morceaux de trajectoire qui respectent toutes les contraintes du problème. Une recherche de solution est effectuée dans l'espace des paramètres afin de déterminer le meilleur échantillon à utiliser.

Cette approche géométrique permet de contrôler l'étape de propagation en séparant la recherche de toutes les solutions locales respectant les contraintes et l'établissement d'une solution minimisant les critères de recherche.

À l'aide de ce mécanisme, j'envisage d'exploiter l'espace des paramètres dans la couche de sélection de DKP pour la création de diversité pour explorer l'espace des solutions et la séparation des heuristiques de recherche : le critère minimisé dans la propagation n'est pas nécessairement celui utilisé dans la sélection.

Enfin, je peux mettre en place dans DKP les processus de vérification de l'existence de solutions dans une situation donnée, de contrôle de la diversité par des mécanismes de filtrage et d'insertion de nouvelles contraintes et de nouveaux objectifs pour adapter les solutions à un contexte particulier.

Chapitre 4

Deterministic Kinodynamic Planning

Ce chapitre est consacré au niveau de sélection de l'approche cognitive pour la planification de trajectoire, introduit dans la Section 4.1. Il présente d'abord l'architecture de DKP dans la Section 4.2 puis des exemples dans la Section 4.3. Je montre ensuite comment je peux adapter le comportement de DKP dans la Section 4.4, comme ouverture vers les comportements de pilotage décrits dans le Chapitre 5.

Une étude expérimentale de DKP est proposée grâce à des séries de simulations dans la Section 4.6 puis des expérimentations sur de vrais robots dans la Section 4.7. Les qualités mais également des défauts de DKP sont traités dans la Section 4.8 dont une partie est discutée dans le Chapitre 5.

4.1 Introduction à DKP

DKP pour Deterministic Kinodynamic Planning emploie l'approche géométrique pour la planification locale de trajectoire, présentée dans le chapitre précédent, dans une approche plus générale de planification de trajectoire par échantillons. Ce planificateur s'intégrera ensuite dans l'approche cognitive pour la planification de trajectoire présentée dans le Chapitre 5. Il construit des trajectoires pour robots à deux roues indépendantes. DKP assure le respect des contraintes qui s'appliquent au modèle des robots à deux roues indépendantes que nous utilisons : continuité en position et vitesse linéaire, bornes sur la vitesse linéaire, sur l'accélération linéaire, sur la vitesse angulaire et respect de la commandabilité. Dans DKP, un arbre d'exploration est construit dans les parties atteignables de l'environnement. Il repose sur deux mécanismes d'exploration : une optimisation locale dans la couche de propagation et une optimisation globale dans la couche de sélection. DKP fournit de plus un comportement totalement déterministe : aucune source aléatoire ne sert aux processus d'exploration.

Espace des paramètres Pour rappel du chapitre 3, les trajectoires produites par DKP sont des courbes splines à base de quadratiques. On réutilise la manière de construire l'espace des paramètres présentée dans le chapitre précédent. Ces morceaux de trajectoire assurent les conditions de continuité en position et vitesse linéaire. Leur forme est donc définie par deux paramètres formant un espace dénommé espace des paramètres, ici en deux dimensions. Afin d'appliquer les contraintes s'appliquant sur les robots, cet espace des paramètres est réduit afin de ne conserver que les morceaux de trajectoires respectant le modèle du robot simulé. Afin de manipuler cet espace des paramètres, j'ai adopté une représentation géométrique des

ensembles étudiés : l'espace des paramètres est une surface qui rassemble l'ensemble des valeurs admissibles pour ce type de solution tout comme les contraintes sont représentées dans leurs bases respectives. On envisage toute la construction de l'espace des paramètres sur la base de transformations affines permettant l'application des contraintes. Ainsi, l'espace des paramètres associé à ce type de solutions est exactement décrit : après application des contraintes, il contient l'intégralité des solutions applicables dans la situation considérée.

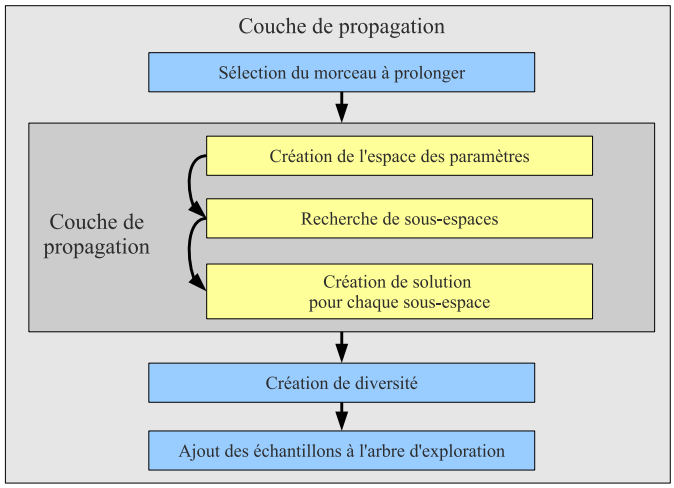

FIGURE 4.1 – Architecture de DKP, planificateur de trajectoires pour robots à deux roues indépendantes, basé sur une approche de planification de trajectoire par échantillon. Ce type d'architecture repose sur l'interaction d'une couche de propagation, qui génère des nouveaux échantillons, avec une couche de sélection, qui détermine les morceaux à prolonger placés dans un arbre d'exploration. L'originalité repose sur une nouvelle couche de propagation qui détermine un espace des paramètres représentant l'ensemble des solutions localement réalisables, avec le respect des contraintes posées sur le modèle du robot. Cette couche permet de plus de ne sélectionner que les parties de l'arbre réellement prolongeables

Architecture DKP est présenté au travers des différentes parties qui composent une approche par échantillonnage. Dans le niveau de propagation, la première étape consiste à créer un espace décrivant tous les échantillons localement faisables. Cette étape est une originalité par rapport aux approches par échantillonnage utilisant classiquement une propagation aléatoire. Ensuite, un ou plusieurs échantillons sont sélectionnés en fonction de la distance de la fin du morceau de trajectoire à l'objectif (fonction d'optimisation qui est employée par DKP). Enfin, une étape de sélection construit un arbre d'exploration en choisissant les échantillons à prolonger. Cette prolongation est effectuée par le niveau de propagation. L'ensemble des processus utilisés dans DKP sont déterministes.

L'espace des paramètres permet de déterminer l'existence de solutions aux contraintes posées avant la recherche effective d'une solution sur cet espace. Cette propriété permet donc de ne développer que les branches réellement prolongeables de l'arbre d'exploration et donc d'abandonner les parties inutiles.

Afin de générer l'arbre d'exploration, DKP est basé sur deux sources de diversité à l'issue de l'étape de propagation. Premièrement, la variation de l'horizon de planification des morceaux de trajectoire permet de générer de la diversité en créant des morceaux longs permettant de

franchir de grandes portions de l'environnement et des morceaux courts permettant d'effectuer des manœuvres délicates. Deuxièmement, la recherche de minima locaux dans l'espace des paramètres lorsque celui-ci présente plusieurs sous-ensembles distincts sont interprétés comme des alternatives de contournements à une même situation. Ces échantillons alimentent ensuite la couche de sélection, basée sur un algorithme de type A*.

DKP repose donc sur une optimisation locale lors de l'étape de propagation, usuellement la minimisation de la distance entre le point final d'un morceau de trajectoire et le point objectif puis une optimisation globale dans l'étape de sélection selon une heuristique ρ. DKP dispose de trois modes de fonctionnement :

- le mode **optimal**, basé sur la propagation non biaisée de l'heuristique servant au guidage de l'exploration ;
- le mode **glouton**, basé sur une propagation biaisée faisant tendre rapidement l'exploration vers l'objectif ;
- le mode **backtrack**, première démonstration d'un mode raisonné s'appuyant sur la possible absence de solution pour abandonner des zones explorées et les contourner par l'ajout de contraintes dites virtuelles.

L'ensemble de l'architecture est représenté par la Figure 4.1.

En spécialisant DKP aux robots à deux roues indépendantes, on peut affranchir DKP de toute source aléatoire : les morceaux de trajectoire sont générés de manière déterministe, les sources de diversité le sont également, tout comme le processus de sélection basé sur un algorithme de type A*. Grâce à cela, le comportement de DKP peut être entièrement maîtrisé, ce qui facilite son intégration dans une couche plus générale de cognition sur la planification de trajectoire (Figure 4.2). Cela sera développé dans le chapitre suivant.

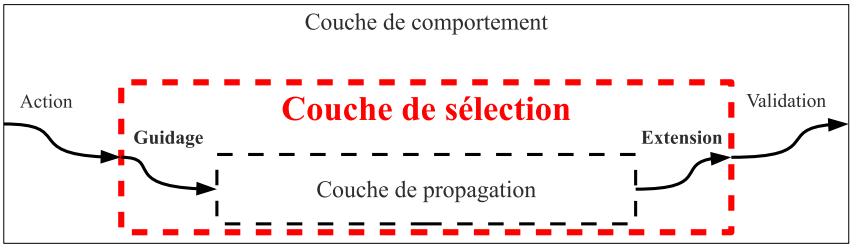

FIGURE 4.2 – Architecture retenue pour exprimer des comportements de pilotage pour la planification de trajectoire sous contraintes cinématiques. Ce chapitre se focalise sur le niveau de sélection et ses interactions avec la couche de propagation

Évaluation En plus de formaliser DKP en tant qu'architecture de sélection/propagation pour l'exploration de l'environnement par échantillonnage, on propose une évaluation par simulations et expérimentations réelles [Gaillard *et al.*, 2011]. Après la présentation de quelques exemples préliminaires, je fournis une évaluation de DKP par simulations : celle-ci a pour but d'établir les performances de DKP dans la recherche de solutions dans des environnements riches en obstacles. Les performances sont établies selon les trois modes de fonctionnement de DKP. Enfin, je présente des expérimentations réelles sur différents robots réels dans les locaux de l'ISEN. Ces expériences montrent la capacité de DKP à fournir des trajectoires qui respectent le modèle de fonctionnement des robots pour lesquelles ces solutions ont été établies.

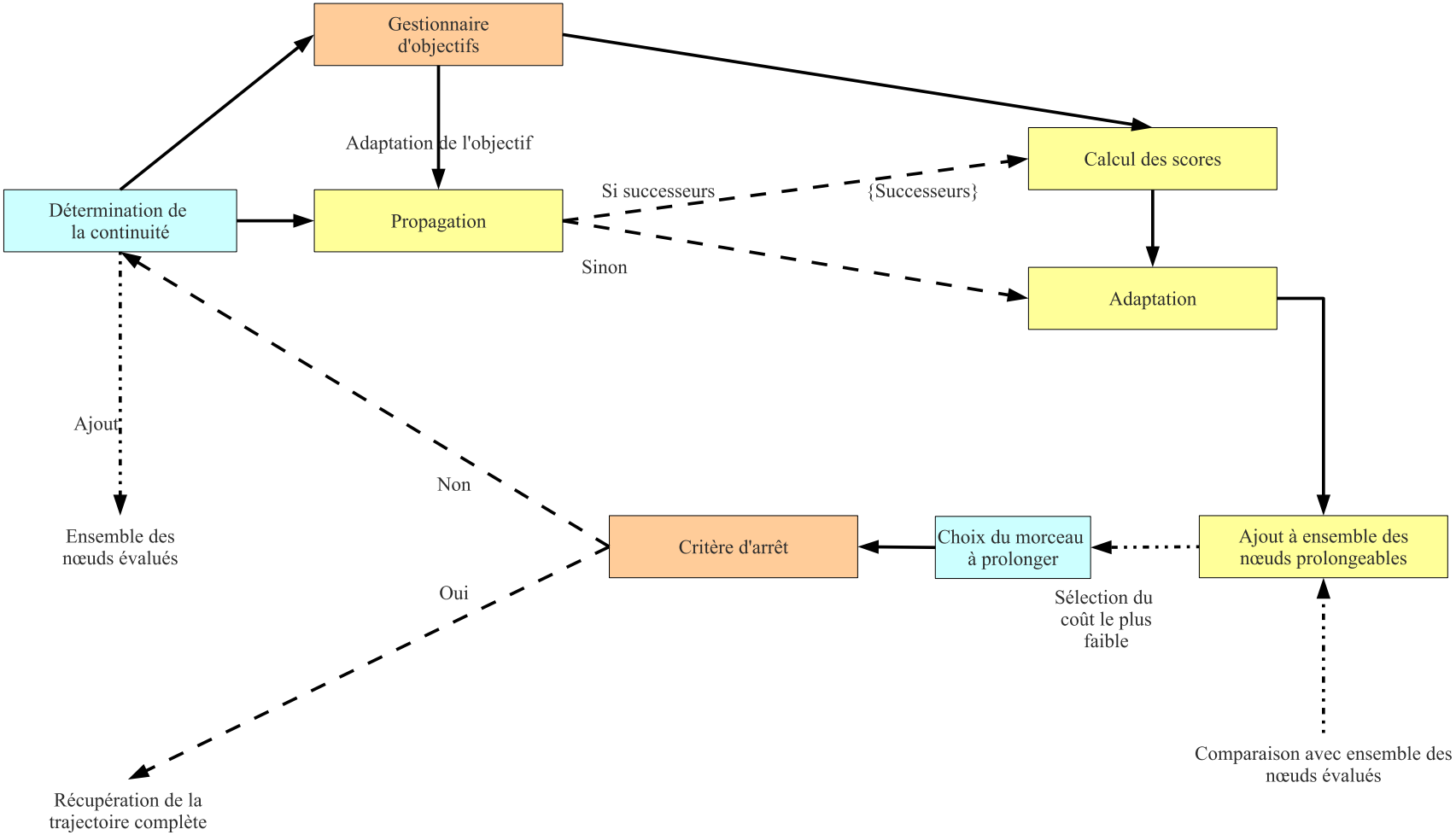

FIGURE 4.3 – L'architecture de sélection/propagation mise en œuvre dans DKP : un arbre d'exploration est développé grâce à la couche de propagation jusqu'à ce que le critère d'arrêt de la recherche soit atteint. La solution ainsi construite respecte les contraintes du problème grâce à la couche de propagation qui les vérifie dans son processus de génération des échantillons. L'arbre de DKP est développé grâce à un mécanisme de sélection basé sur A*

4.2 Mécanismes d'exploration

DKP est une approche de sélection/propagation. Son architecture est indiquée par le schéma de la Figure 4.3. Comme le montre la Figure 4.4, DKP produit des trajectoires splines sans qu'il y ait un nombre pré-établi de nœuds, contrairement à [Milam *et al.*, 2000].

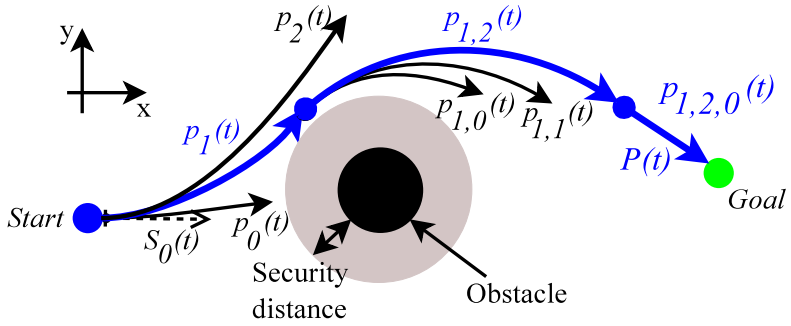

FIGURE 4.4 – DKP construit des trajectoires splines depuis *Start* jusque *Goal* avec des échantillons $p_{k_0 \ldots k_n}(t)$ organisés dans un arbre d'exploration. La distance de sécurité prend en compte l'empattement du robot

Les robots sont spécifiés selon leurs contraintes, telles que les limites sur la vitesse linéaire/angulaire ou l'évitement d'obstacles fixes ou mobiles (dont la forme et la trajectoire sont connues). Pour rappel, dans la couche de propagation, le robot est vu comme un point sur la trajectoire. Au préalable, chaque obstacle est grossi selon une distance de sécurité, basée à minima sur l'empattement du robot.

DKP construit un arbre d'exploration dont les branches sont des échantillons de trajectoires. Il recherche des trajectoires qui vont d'une position de départ *Start* jusqu'une position d'arrivée *Goal* dans les parties atteignables de l'environnement. Pour déterminer la branche à faire grandir, le processus de sélection est guidé à la manière d'un A* par un critère d'optimisation ρ, critère qui dépend du domaine simulé. Les morceaux sélectionnés sont poursuivis par la couche de propagation. Les nouveaux morceaux sont soumis aux conditions de continuité et aux contraintes cinématiques.

4.2.1 Implémentation

L'implémentation du niveau de sélection est donnée dans l'algorithme 4.2.1. Cette implémentation diffère de l'implémentation habituelle de A* car l'arbre d'exploration grandit au fur et à mesure de la planification selon son propre critère d'optimisation locale (géré par le niveau de propagation). Un nœud contient un échantillon de trajectoire $p_{k_0 \ldots k_n}(t)$ et sa durée $T_{k_0 \ldots k_n}$. Le voisinage d'un nœud est construit par la couche de propagation à partir du morceau $p_{k_0 \ldots k_n}(t)$. Le voisinage du morceau $p_{k_0 \ldots k_n}(t)$ est généré par le planificateur local de mouvement en se servant de l'état final $p_{k_0 \ldots k_n}(T_{k_0 \ldots k_n})$ comme point de départ des morceaux suivants $p_{k_0 \ldots k_n K_{n+1}}(t)$. Les nouveaux nœuds sont ajoutés à l'arbre d'exploration à chaque fois qu'un voisinage est généré.

La trajectoire finale $P(t)$ est construite en retrouvant les prédécesseurs du dernier morceau polynomial $p_{k_0 \ldots k_N}(t)$ qui satisfait le critère d'arrêt de la recherche. Par conséquent, la trajectoire est constituée de $N+1$ échantillons. La durée totale T de $P(t)$ est obtenue en sommant les durées de chacun des $N+1$ morceaux. Le parcours de $P(t)$ correspond aux évaluations successives de chaque échantillon selon leurs durées respectives. Soit $t \in [0; T_{k_0}] \cap \cdots \cap [T_{k_0 \ldots k_n}; T_{k_0 \ldots k_n k_{n+1}}] \cap$

$\cdots \cap [T_{k_0 \ldots k_{N-1}}; T_{k_0 \ldots k_{N-1} k_N}]$. De plus, la durée maximale correspond à $T_{k_0 \ldots k_{N-1} k_N} = T$. Si t appartient à $[T_{k_0 \ldots k_n}; T_{k_0 \ldots k_n k_{n+1}}]$, la valeur de la trajectoire $P(t)$ est la valeur du polynôme $k_0 \ldots k_n$ tel que :

$$P(t) = p_{k_0 \ldots k_n}(t - T_{k_0 \ldots k_{n-1}}) \tag{4.1}$$

4.2.2 Fonctions et scores transmis

Le processus de sélection utilisé par DKP s'appuie sur le fonctionnement de A*. DKP construit un arbre d'exploration dont les nœuds contiennent : un morceau de trajectoire $p_{k_0 \ldots k_n}(t)$, les contraintes, la situation initiale et le modèle de transformation des trajectoires utilisés pour produire les morceaux suivants et des liens vers les nœuds suivants et précédents. Tout comme l'algorithme A*, les scores que DKP propage aux morceaux de trajectoires suivants sont le résultat de la fonction d'évaluation $\rho = g + bias * h$: g est le coût réel cumulé, h est la partie heuristique et *bias* modifie cette heuristique et par conséquent le comportement du processus de sélection.

Les modes de fonctionnement de DKP

La valeur de *bias* modifie l'heuristique et le comportement du processus de sélection. On obtient alors les deux premiers modes d'exploration :

- le mode optimal lorsque *bias* vaut 1. La recherche est optimale sur l'arbre d'exploration de DKP. Ses performances sont également proportionnelles au degré de diversité accordée à DKP lors de la propagation ;
- le mode glouton lorsque *bias* est plus grand que 1. Celui-ci peut produire des solutions rapidement mais au prix d'une qualité inférieure au mode optimal. De plus, il peut rester bloqué dans des minima locaux, comme je le montre plus tard.

Les deux modes de fonctionnement peuvent être opposés :

- la vitesse de planification est au profit du mode glouton, bien qu'il soit très sensible aux minima locaux ;
- la qualité de la solution est le point fort du mode optimal, également sensible à la présence de minima locaux.

Il faut bien noter que l'optimalité du second mode ne se joue pas au sens de l'exploration de l'environnement. En effet, l'optimalité issue du fonctionnement de l'algorithme A* ne repose que sur le graphe de mouvement généré par le mécanisme de propagation. Or, cette propagation génère des morceaux optimisés par le critère d'optimisation locale. En réalité, DKP fonctionne de manière gloutonne, dans sa façon d'optimiser la distance à l'objectif. Être plus expansif dans l'exploration implique donc de générer des morceaux de trajectoire qui puissent aller dans d'autres directions que l'objectif : c'est le sens des tirages aléatoires de configurations à suivre dans des approches par échantillonnage comme RRT.

4.2.3 Sources de diversité

Diversité sur le temps

E représente l'espace des paramètres, calculé pour respecter les contraintes à tout instant *step* depuis $T_{k_0 \ldots k_n}$ jusque $T_{k_0 \ldots k_{n+1}}$. Par conséquent, E contient un espace des paramètres intermédiaires, noté E', pour tout instant *step* depuis $T_{k_0 \ldots k_{n+1}}$ jusque $T' < T_{k_0 \ldots k_n}$. Ainsi, la construction de l'espace des paramètres pour un horizon donné implique la construction d'espaces de paramètres pour des horizons intermédiaires. DKP peut donc produire des morceaux de

Algorithm 4.2.1 DKP(start,goal)

```
1: closed_set ← ∅
2: open_set ← ∅ samples to be selected
3: start.g_score ← 0
4: start.h_score ← distance(start, goal)
5: start.f_score ← start.h_score + start.g_score
6: open_set.add(start)
7: while open_set ≠ ∅ do
8:    evaluated_piece ← remove sample from open_set with lowest f_score
9:    if goal_reached(evaluated_piece) then
10:      return build overall trajectory
11:   end if
12:   closed_set.add(evaluated_piece)
13:   T ← evaluated_piece.total_time
14:   end ← evaluated_piece(T)
15:   neighbours ← propagation(end, goal, {C})
16:   filtered_neighbours ← filter(neighbours)
17:   for neighbour ∈ filtered_neighbours do
18:      if neighbour ∈ closed_set then
19:         break
20:      end if
21:      new_score ← evaluated_piece.g_score + distance(evaluated_piece, neighbour)
22:      if neighbour ∉ openset then
23:         open_set.add(neighbour)
24:         best_score ← true
25:      else if new_score < neighbour.g_score then
26:         best_score ← true
27:      else
28:         best_score ← false
29:      end if
30:      if best_score then
31:         neighbour.predecessor ← evaluated_piece
32:         neighbour.g_score ← new_score
33:         neighbour.h_score ←distance(neighbour, goal)
34:         neighbour.f_score ←neighbour.h_score + neighbour.g_score
35:      end if
36:   end for
37: end while
38: return start
```

trajectoires avec des durées intermédiaires, dont le seul surcoût est la recherche de solutions. L'horizon maximal de planification dans une propagation est noté T_{max}.

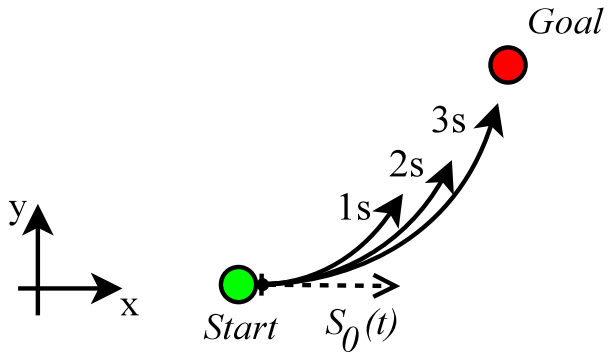

FIGURE 4.5 – Propagation déterministe avec des morceaux optimisés de trajectoires de différentes durées et qui respectent les contraintes cinématiques.

Diversité sur les solutions

L'espace des paramètres E permet de représenter toutes les échantillons quadratiques localement admissibles. C'est un ensemble de points représenté par $N(E)$ surfaces. Chacune représente un sous-espace des paramètres $E'_k \mid E'_k \subset E$ tel que $\bigcap_{k=1}^{N(E)} E'_k = E$. Ainsi, il est possible d'isoler des ensembles disjoints de solutions sur chacun des sous-espaces E'_k. Dans cette situation, la recherche de la solution optimale ne diffère pas des méthodes précédemment évoquées. Néanmoins, j'estime que la recherche d'une solution localement optimale sur E'_k est une bonne source de diversité : en effet, une disjonction de l'espace des paramètres permet de décrire des solutions pour des situations globalement symétriques. C'est en particulier le cas pour l'évitement d'un obstacle tel que décrit par l'exemple de la Section 4.3.

Filtrage sur les trajectoires

Contrairement aux planificateurs de mouvement classiques basés sur A*, les morceaux polynomiaux générés par DKP reposent sur un espace continu. Ainsi, deux nœuds ne coïncident que très rarement. Par conséquent, le nombre de nœuds à évaluer peut rapidement grossir, affectant l'efficacité de la recherche. À l'image de ce que fait [Branicky *et al.*, 2008] pour prévenir ce problème, les morceaux de trajectoire sont enregistrés avec un critère de discrétisation reposant sur : le point final du morceau de trajectoire, la direction du vecteur vitesse en ce point, la norme de ce vecteur et la longueur du morceau de trajectoire. Deux morceaux de trajectoires ne coïncident que s'ils partagent ces mêmes critères de discrétisation. Par la suite, les nouveaux morceaux de trajectoire issus des processus de propagation sont filtrés. L'ensemble résultant du filtrage ne contient que des morceaux qui ne coïncident pas avec les morceaux déjà référencés. Le nombre de morceaux générés par DKP est ainsi contrôlé et, par conséquent, les bornes sur le temps de calcul de l'algorithme le sont aussi. Ce filtrage s'insère dans la couche de sélection selon le schéma de la figure 4.6.

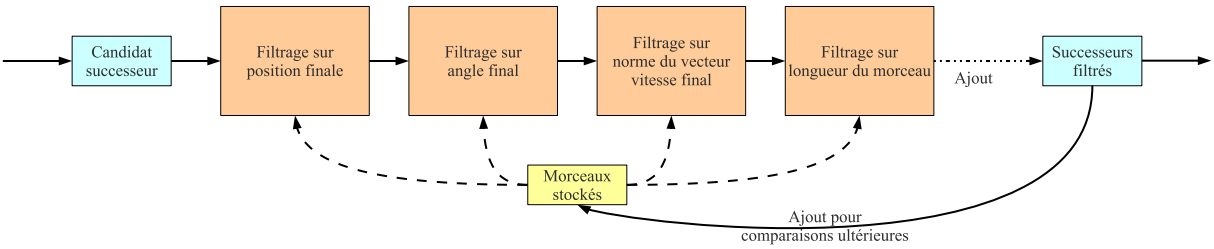

FIGURE 4.6 – Le filtrage s'insère entre la couche de propagation et l'ajout des morceaux à prolonger dans l'ensemble des ouverts afin de limiter la taille de l'arbre d'exploration et donc la complexité générale de DKP.

4.2.4 Exploiter l'espace des paramètres

DKP seul constitue une approche déterministe pour la planification de trajectoire de robots à deux roues. En prenant le contre-pied des approches par échantillonnage aléatoire, très généralistes, j'exploite les propriétés particulières de l'espace des paramètres pour explorer efficacement l'espace des solutions. Comme DKP repose sur une approche de sélection/propagation, l'efficacité du planificateur est basée sur ses sources de diversité et son guidage. Ici, on exploite une propriété de la construction de l'espace des paramètres pour générer de la diversité sur la durée des morceaux de base. Le surcoût lié à cette propriété est limité car la construction de l'espace des paramètres est partagée pour la recherche d'échantillons de trajectoire de chaque horizon. Cette diversité en temps permet de générer des mouvements adaptés aux zones explorées (Figure 4.7) de faible longueur là où les situations sont difficiles (environnement chargé d'obstacles, fortes contraintes sur les vitesses et accélération limitant la manœuvrabilité de l'entité), et de longueur plus importante lorsqu'il s'agit de traverser une portion de l'environnement ou amorcer le contournement d'un obstacle. De plus, comme on peut déterminer si l'espace des paramètres est vide ou non, l'arbre d'exploration n'est progressivement étendu que dans les parties de l'environnement comportant des solutions pour s'y propager. Toutefois, une telle diversité pourrait être décidée dynamiquement en fonction de l'environnement pour ne pas générer trop de morceaux inutiles : c'est l'objectif du backtrack décrit dans la Section 4.4.

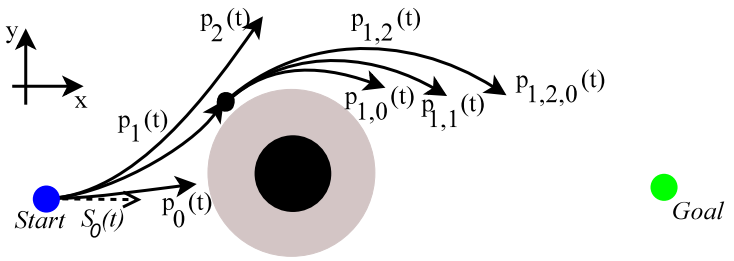

FIGURE 4.7 – La diversité de la durée des échantillons permet à l'arbre d'exploration de ne pas rester coincé dans les minima locaux. Les morceaux courts permettent d'avoir de la manœuvrabilité autour des obstacles alors que les morceaux longs permettent de franchir rapidement des zones larges sans obstacles.

Complexité

La complexité de DKP repose essentiellement sur le coût d'une propagation. La création de diversité n'est coûteuse que dans la mesure où il faut planifier des échantillons jusque T_{max} lors d'une propagation. L'implémentation du niveau de sélection peut bénéficier des améliorations classiques dans les approches par échantillonnage : tas de Fibonacci [Fredman et Tarjan, 1987] pour stocker les trajectoires pour la partie A* et Quadtree [Finkel et Bentley, 1974] pour stocker d'une manière spatiale les trajectoires dans le cadre du filtrage. La Section 4.6 propose une évaluation des performances de DKP en fonction du nombre d'obstacles, en utilisant ces techniques.

4.3 Mise en pratique

Reprenons l'exemple développé dans la Section 3.7 : la mise en situation d'un robot à deux roues indépendantes dans un environnement contenant un obstacle à franchir. On utilise les solutions paramétriques à base de quadratiques comme échantillons de trajectoire. Le problème est initialisé avec les paramètres suivants :

- la position initiale est $Start = (0, 0)$;
- le vecteur vitesse initial est $\overrightarrow{S}(0) = (0.1, 0.2)$;
- une distance de sécurité autour du robot fixée à $r_{robot} = 0.5\text{m}$;
- les contraintes suivantes, considérées pour chaque pas de temps $step = 0.1s$ jusque l'horizon de planification $T = 10s$:
  - l'accélération linéaire bornée entre 0 et 1 m/s^2 ;
  - la vitesse linéaire bornée entre 0 et 1 m/s ;
  - un obstacle sous la forme d'un disque de rayon 1m centré en $Obs = (2, 0)$.

DKP construit des échantillons selon le même principe que l'exemple développé dans la Section 3.7. Un échantillon de trajectoire $p_{k_0 \ldots k_n}(t)$ sélectionné par DKP est poursuivi avec un ensemble d'échantillons de trajectoire successeurs $p_{k_0 \ldots k_n K_{n+1}} = \{p_{k_0 \ldots k_n 0}, p_{k_0 \ldots k_n 1}, \ldots, p_{k_0 \ldots k_n k_{n+1}}\}$. Pour cela, le morceau $p_{k_0 \ldots k_n}$ définit les conditions de continuité en position et vitesse pour ses successeurs.

4.3.1 Solutions

Le processus de sélection crée un arbre d'exploration dans lequel chaque échantillon de trajectoire évite l'obstacle et satisfait les contraintes cinématiques. Si les échantillons peuvent être générés par la couche de propagation, leur durée varie de $T' = 0.5\text{s}$ à $T' = 10\text{s}$, toutes les 0.5 secondes.

Lorsque l'exemple de la Section 3.7 est fourni à DKP, on obtient une trajectoire solution notée $P(t)$ qui possède une durée totale de 6.5 secondes et une longueur de 4.61 mètres. Dans cet exemple, $P(t)$ est le résultat de la concaténation des 4 échantillons suivants :

$$\begin{cases} x(t) = 0.1t + 0.1141t^2 \\ y(t) = 0.2t + 0.0250t^2 \end{cases} \text{ pour } t \in [0; 3.5]s; \tag{4.2}$$

$$\begin{cases} x(t) = 1.7481 + 0.8989t + 0.0916t^2 \\ y(t) = 1.006 + 0.3750t - 0.4857t^2 \end{cases} \text{ pour } t \in [3.5; 4]s; \tag{4.3}$$

$$\begin{cases} x(t) = 2.2205 + 0.9906t - 0.1657t^2 \\ y(t) = 1.0723 - 0.1107t - 0.4485t^2 \end{cases} \text{ pour } t \in [4; 4.5]s; \tag{4.4}$$

$$\begin{cases} x(t) = 2.6744 + 0.8248t - 0.0810t^2 \\ y(t) = 0.9048 - 0.5592t + 0.0534t^2 \end{cases} \text{ pour } t \in [4.5; 6.5]s; \tag{4.5}$$

L'arbre d'exploration est illustré par la Figure 4.8, ainsi que la trajectoire solution.

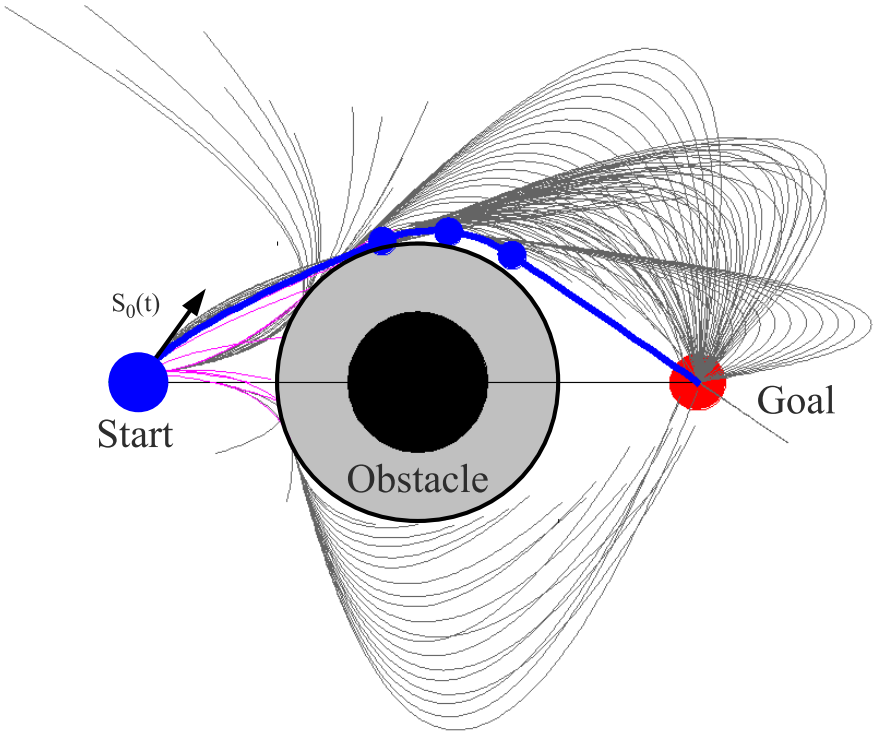

FIGURE 4.8 – Représentation de l'arbre d'exploration construit par DKP. Les échantillons de trajectoires apparaissent en gris foncé. Les échantillons sélectionnés par DKP pour être prolongés sont dessinés en magenta. En gras est tracée la première trajectoire qui satisfait le critère d'arrêt de la recherche de solution, c'est-à-dire rejoindre la région de l'environnement autour de *Goal*.

4.3.2 Respect du modèle des robots

L'utilisation de DKP garantit donc que la trajectoire planifiée pourra être suivie sous des contraintes cinématiques et dynamiques données. Il est toutefois important de noter que la qualité effective de ce suivi dépendra des capacités de localisation et d'asservissement.

La figure 4.9(c) illustre la meilleure trajectoire possible (en termes de longueur) avec des contraintes cinématiques fortes (vitesse linéaire minimale et vitesse angulaire maximale, contraintes classiques pour des drones). Cette solution ne peut être trouvée avec une approche discrétisée classique (Dijkstra) dont le résultat est illustré par la figure 4.9(a) réalisée avec Airplan [Soulignac, 2009].

La prise en compte du modèle du robot par DKP permet d'envisager l'inclusion de contraintes spécifiques à un contexte de déplacement, avec des contraintes liées à des capteurs (portion de l'environnement effectivement visible) ou à l'environnement (obstacles fixes ou mobiles, présence de pentes).

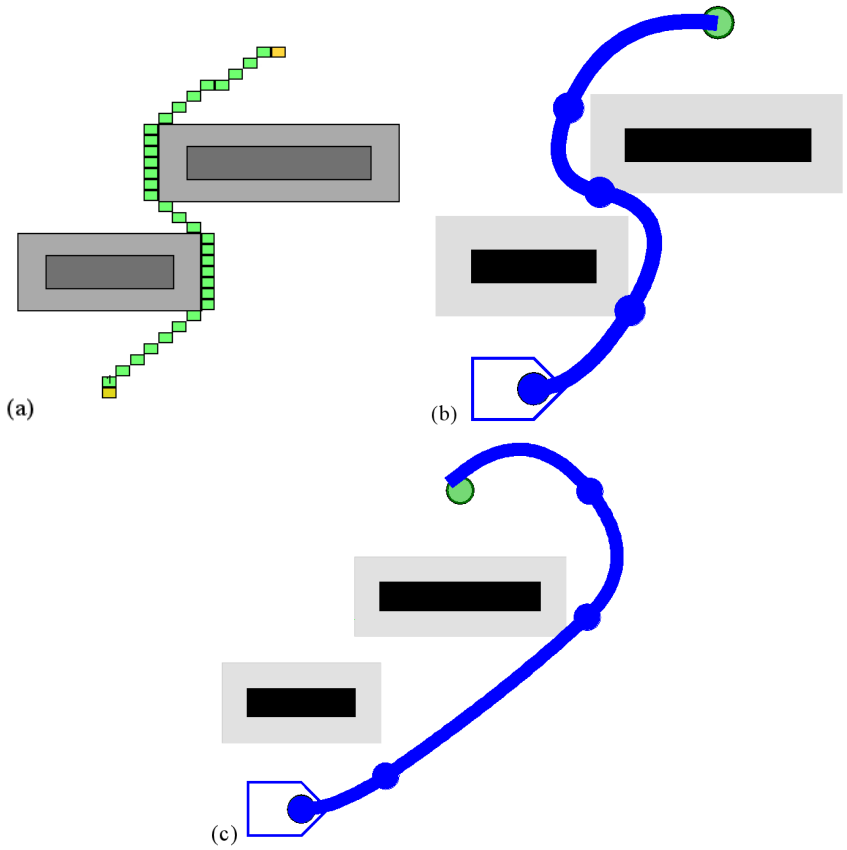

FIGURE 4.9 – Figure 4.9. Problème de planification de trajectoire (obstacles en gris). Le robot possède une borne inférieure sur sa vitesse linéaire et supérieure sur sa vitesse angulaire.(a) Solution trouvée par Dijkstra (grille50x50). Cette solution sera toujours fournie quel que soit le modèle du robot. Elle ne sera donc pas nécessairement réalisable. (b) Solution trouvée par DKP pour une vitesse angulaire maximale élevée et (c) pour une vitesse angulaire maximale faible.

4.3.3 Comparaisons

Le but de cet exemple est d'appliquer à RRT et DKP des environnements communs et d'en comparer les résultats. Il est tiré de [Gaillard *et al.*, 2010], dans lequel DKP en était à une version préliminaire ne gérant que des obstacles fixes de type disque. D'après la Section 3.7 consacrée à la mise à l'échelle de la propagation, ce type de représentation est très coûteuse. En effet, il ne faut pas moins de quatre courbes cubiques pour représenter le bord de chacun de ces obstacles. Le coût des intersections grandit fortement à mesure que des obstacles sont ajoutés dans l'environnement pour constituer le corridor. Heureusement, DKP gère dans la version actuelle des obstacles de formes bien plus variées et en particulier des polygones permettent de mieux représenter des corridors (par exemple dans la Section 5.5). Comme cela est montré dans l'Annexe 6.0.0.0, on peut décrire les obstacles par des splines (segments, quadratiques, cubiques), ce qui permet de facilement coller à toute forme d'obstacle.

Ces comparaisons se font sur le temps de calcul et la qualité de la solution (longueur et durée). Toutefois, DKP est guidé par nature par une heuristique, et non RRT, ce qui déséquilibre cette comparaison. Afin de rendre RRT plus proche de DKP, c'est-à-dire qu'ils

soient tous deux guidés par un objectif, RRT est utilisé avec une stratégie nommée « Goal bias » [Ferguson et Stentz, 2006] : avec une probabilité $1-p$, un échantillon aléatoire est utilisé et avec une probabilité p, le point *Goal* est choisi. Plus la valeur de p augmente, plus RRT se comporte de manière gloutonne. Je choisis la valeur habituellement utilisée dans la litérature $p = 0.2$.

Les solutions qualifiées de « trajectoires finales » correspondent à la première solution obtenue par chaque algorithme. Les temps de calcul sont obtenus sur un PC double cœur à 1.83 Ghz disposant 1 Go de RAM. Je donne deux exemples avec un environnement de type couloir (figure 4.10) et un environnement de type aléatoire (figure 4.11) selon le principe de la Section 4.6.

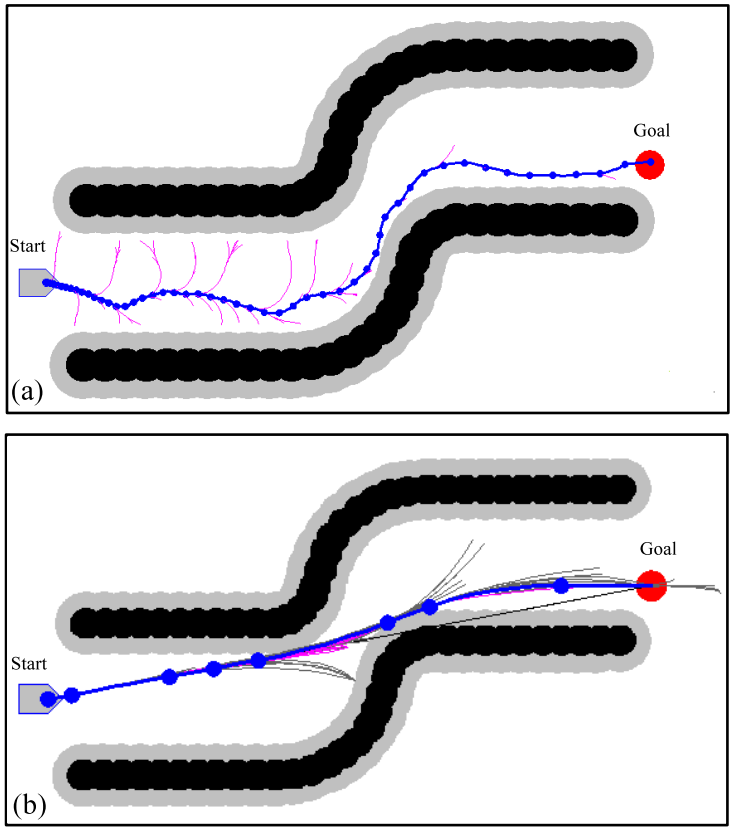

FIGURE 4.10 – Échantillons et trajectoires finales obtenues par (a) RRT et (b) DKP. Les trajectoires finales apparaissent en gras : (a) la longueur de la trajectoire est de 9.04m, sa durée est de 12.6s. Cette solution est obtenue par RRT en 67s. (b) la longueur de la trajectoire est de 8.19m, sa durée est de 3.2s. Cette solution est obtenue par DKP en 27.6s.

4.4 Contrôle de l'exploration

DKP construit un arbre d'exploration dans les parties atteignables de l'environnement. DKP fournit des garanties sur la faisabilité des trajectoires, la qualité des solutions et le contrôle du temps de calcul. Il a la propriété importante d'être déterministe. S'il est capable de produire une solution faisable, il la retourne. Deux exécutions de DKP sur le même problème vont produire très exactement les mêmes solutions. De plus, il est possible de modifier dans l'arbre d'exploration :

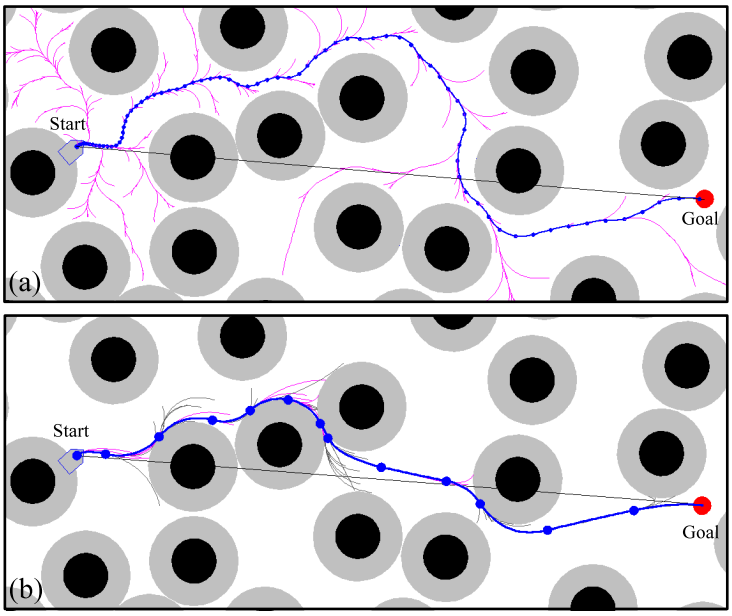

FIGURE 4.11 – Échantillons et trajectoires finales obtenues par (a) RRT et (b) DKP. Les trajectoires finales apparaissent en gras : (a) la longueur de la trajectoire est de 19.44m, sa durée est de 18.51s. Cette solution est obtenue par RRT en 168s. (b) la longueur de la trajectoire est de 16.41m, sa durée est de 18s. Cette solution est obtenue par DKP en 13.7s.

- les contraintes $c(t)$ à appliquer sur l'espace des paramètres ;
- la fonction d'optimisation dans le niveau de propagation $\rho_{obj}(p_{k_0 \ldots k_n}, t)$ pour choisir le meilleur échantillon $p_{k_0 \ldots k_n}$ sur l'espaces des paramètres ;
- les nœuds de l'arbre d'exploration à sélectionner, selon la fonction heuristique ρ pour le niveau de sélection, nœuds qui sont ensuite propagés.

De même, le processus de propagation de DKP offre de nombreuses possibilités de personnalisation :

- modification des bornes sur les contraintes usuelles ;
- ajout ou suppression de contraintes en cours de fonctionnement (par exemple pour n'exprimer une contrainte que sur certains pas de temps) ;
- en spécifiant de nouvelles contraintes en plus des contraintes usuelles de planification de trajectoire.

Exemples de contraintes particulières dans DKP

Cette maîtrise du comportement de DKP permet de faire face à des situations particulières dont voici un exemple. Nous voulons modéliser une faiblesse dans un moteur qui force un robot à deux roues à ne tourner que dans un seul sens. Pour cela, prenons le vecteur vitesse $\overrightarrow{S}_{k_0 \ldots k_n}(T_{k_0 \ldots k_n})$ final d'un morceau à prolonger. Pour bloquer le mouvement du robot dans un seul sens, il suffit de lui interdire un demi plan de l'espace des positions dont la section est orientée dans le sens du vecteur vitesse initial (valable uniquement en utilisant des solutions à base de quadratiques).

Une telle contrainte est mise en pratique dans l'exemple illustré par la Figure 4.12.

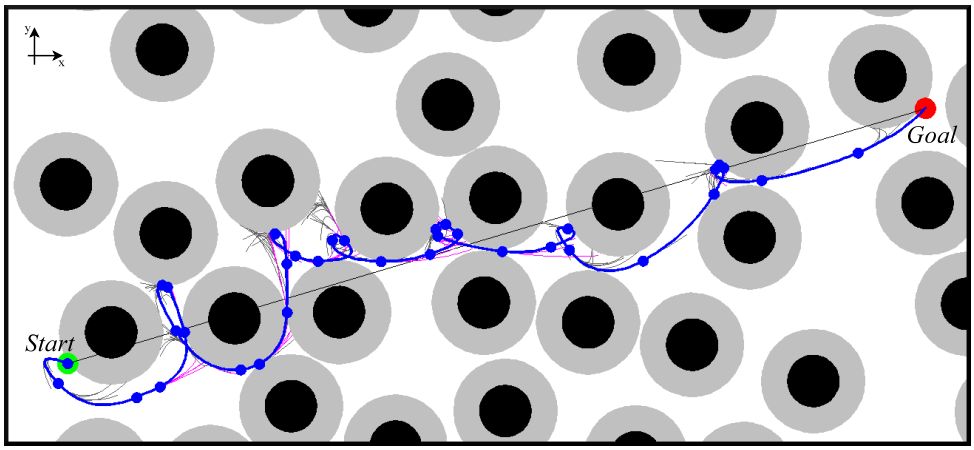

FIGURE 4.12 – Une trajectoire construite par DKP pour un robot à deux roues indépendantes contraint de tourner uniquement sur sa gauche à cause de la défaillance d'un moteur.

4.4.1 Mode backtrack

Les deux modes précédents offrent une exploration naïve de l'environnement : l'espace des paramètres n'offre pas d'information sur la propagation des morceaux autres que la possible création de ces solutions. Or, dans le cas où l'espace des paramètres suivant un morceau de trajectoire est vide, alors ce morceau ne peut être prolongé. Si ce cas de figure permet de limiter la diversité nécessaire à l'exploration, cette propriété peut être exploitée dans un mode dit de backtrack, revenant sur un morceau dont les fils ne peuvent être prolongés pour créer des alternatives grâce au placement d'obstacles dits virtuels. En effet, DKP souffre de son mécanisme d'exploration de l'environnement : il faut de la diversité pour générer des zones de contournement mais également de la gloutonnerie pour progresser rapidement vers l'objectif. Aussi, les différentes solutions proposées dans ce chapitre permettent d'améliorer le comportement général de DKP, tout particulièrement en limitant le nombre de morceaux nécessaires à l'exploration de l'environnement.

Définition

L'idée est d'adapter localement l'environnement exploré en ajoutant des obstacles virtuels là où la propagation échoue dans le but de modifier les parties atteignables par le niveau de propagation. Pour cela, deux ensembles de contraintes dans les nœuds de l'arbre d'exploration sont établis. Le premier contient les contraintes transmissibles TC_c, telles que les contraintes cinématiques et les obstacles à éviter. Le second contient des contraintes non-transmissibles NTC_c, qui contient les contraintes virtuelles. Un morceau de trajectoire $p_{k_0 \ldots k_{n+1}}$ ne récupère que les contraintes transmissibles provenant du morceau de trajectoire prédécesseur $p_{k_0 \ldots k_n}$. Les deux ensembles de contraintes sont évalués en même temps durant le processus de propagation.

Le processus de backtrack qui est proposé pour DKP est défini selon les étapes suivantes :

1. Si le processus de propagation (avec une situation initiale basée sur le morceau de trajectoire $p_{k_0 \ldots k_{n+1}}$ et les contraintes transmissibles TC_c) s'achève sur un espace des paramètres vide, ce morceau de trajectoire ne peut être poursuivi.
2. Ce morceau est alors à retirer de l'arbre d'exploration. On ajoute alors au morceau de trajectoire précédent $p_{k_0 \ldots k_n}$ une contrainte virtuelle d'évitement d'obstacle fixe $c_{vobstacle}$

(ajoutée à l'ensemble NTC), qui consiste en un disque centré sur la position de départ du morceau retiré $p_{k_0 \dots k_{n+1}}$ et dont le rayon dépend de la vitesse maximale et de la durée du morceau selon la formule :

$$r_{vobstacle, p_{k_0 \dots k_{n+1}}} = T_{k_0 \dots k_{n+1}} \times \frac{S_+}{size} \tag{4.6}$$

$T_{k_0 \dots k_{n+1}}$ est la durée de l'échantillon retiré.

3. De plus, un *count*, contenu dans les nœuds de l'arbre d'exploration, est incrémenté à chaque fois qu'un successeur du morceau de trajectoire $p_{k_0 \dots k_n}$ est effacé.
4. Lorsque ce *count* atteint une valeur *trigger*, le morceau de trajectoire $p_{k_0 \dots k_n}$ est également supprimé. Un obstacle virtuel plus grand d'un facteur *grow* est placé, dont le rayon vaut :

$$r_{vobstacle, p_{k_0 \dots k_n}} = grow \times r_{vobstacle, p_{k_0 \dots k_{n+1}}} \tag{4.7}$$

5. Une nouvelle propagation est entamée et ainsi de suite.

Le mode backtrack se place après la couche de propagation. Il agit à la fois sur l'arbre d'exploration et sur les ensembles de nœuds sélectionnables par DKP.

Comparaison avec le mode optimal

Avec le processus de backtrack illustré par la figure 4.13, DKP adopte un comportement différent des modes classiques de A*. En effet, là où le mode optimal va rester bloqué dans des minima locaux, le mode backtrack restructure l'environnement grâce aux obstacles virtuels. Ainsi, l'arbre d'exploration de l'environnement va progressivement anticiper les minima locaux qui empêchent sa propagation.

Problème de l'obstacle en U

Contradiction entre guidage et exploration L'exemple typique de minimum local est celui d'un obstacle en U, situation évoquée dans la Section 2.1. Soumis à ce problème, DKP en mode optimal ou glouton restera bloqué longtemps dans cette situation. En effet, il n'y a pas de remise en cause de l'arbre d'exploration déjà créé. Ainsi, à la manière d'un mode itératif, les nouveaux échantillons, guidés par la minimisation de la distance à l'objectif, sont tous générés vers le fond de l'obstacle en U. Les approches par échantillonnage utilisant des mécanismes aléatoires pour explorer l'environnement proposent des solutions dans cette situation car il y a une exploration complète de l'environnement à mesure que des échantillons sont générés. Ainsi, des comportements contradictoires apparaissent dans ce problème : le besoin de guidage pour obtenir une solution rapidement s'oppose au besoin d'explorer l'environnement en toutes circonstances.

Des obstacles virtuels pour remodeler l'environnement Comme le montre la Figure 4.14, le mode backtrack offre une solution en réinterprétant l'environnement : les obstacles virtuels vont progressivement boucher l'obstacle en U de manière à générer *in fine* un arbre d'exploration qui parcoure les côtés du U. Le problème est initialisé avec les paramètres suivants :

- la position initiale est $Start = (-80, 0)\text{cm}$;
- le vecteur vitesse initial est $\overrightarrow{S}(0) = (10, 0)\text{cm/s}$;
- une distance de sécurité autour du robot fixée à $r_{robot} = 20\text{cm}$;
- les contraintes suivantes :

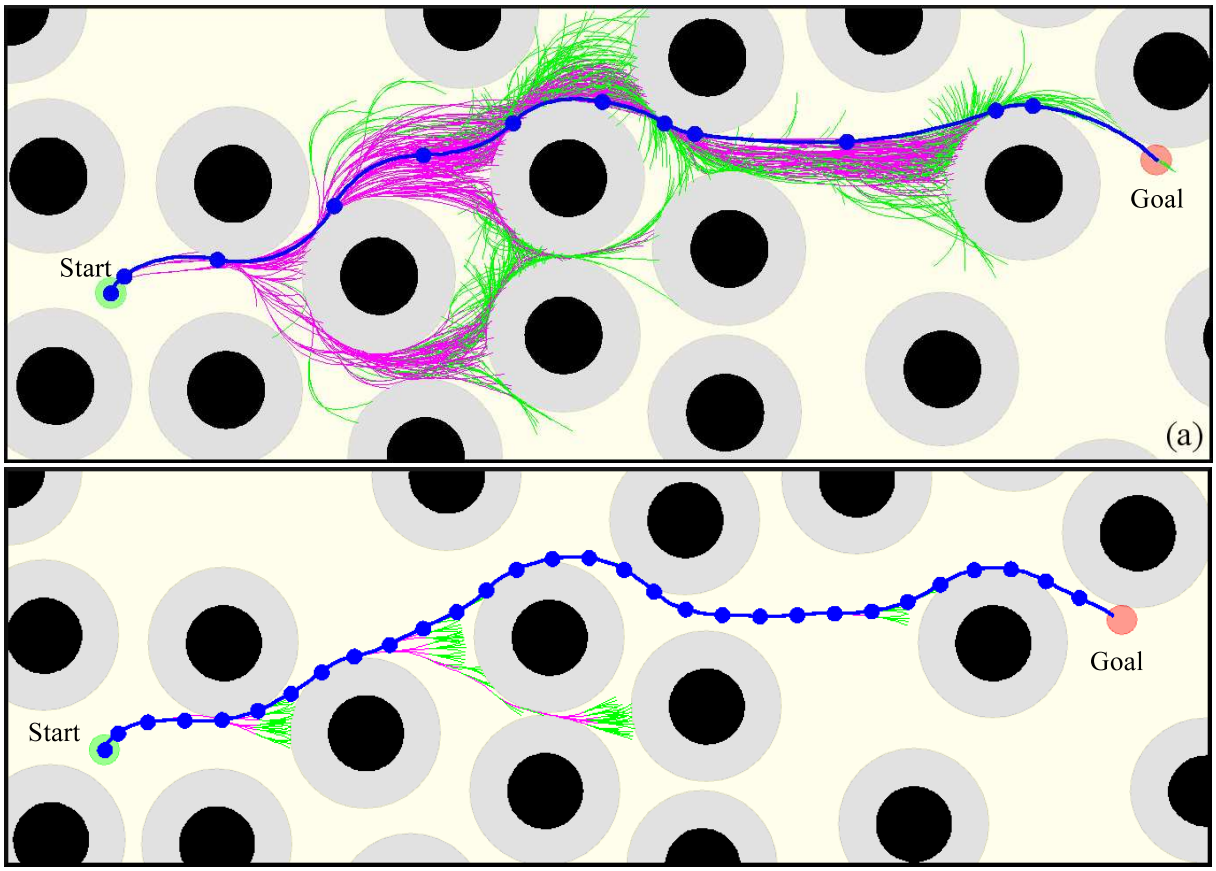

FIGURE 4.13 – Arbre d'exploration de trajectoire pour (a) une recherche en mode optimal et (b) une recherche en mode backtrack.

- l'accélération linéaire bornée entre 0 et 10cm/s^2 ;
- la vitesse linéaire bornée entre 10 et 20cm/s ;
- un obstacle en U.

Sur la Figure 4.14(a), les échantillons ont une durée de 0.5s. Aussi, très courts, il faut de très nombreux échantillons pour obtenir un début d'arbre d'exploration sortant de l'obstacle en U (l'exploration est arrêtée à plus de 130000 propagations !). À mesure que la durée des échantillons est augmentée, les obstacles virtuels sont proportionnellement plus grands car tenant compte de la durée de l'échantillon. Le backtrack se fait donc plus rapidement. Les Figures 4.14 (b) et (c) montrent que l'obstacle en U finit par être rempli d'obstacles virtuels et l'exploration démarre sur les côtés jusqu'à obtenir rapidement une solution. Enfin, la figure 4.14(d) est paramétrée de manière à ce que l'accélération linéaire soit désormais bornée entre 0 et 5cm/s^2. La diminution de manœuvrabilité qui en résulte a pour conséquence de forcer l'arbre d'exploration à rebrousser chemin pour achever le contournement de l'obstacle en U.

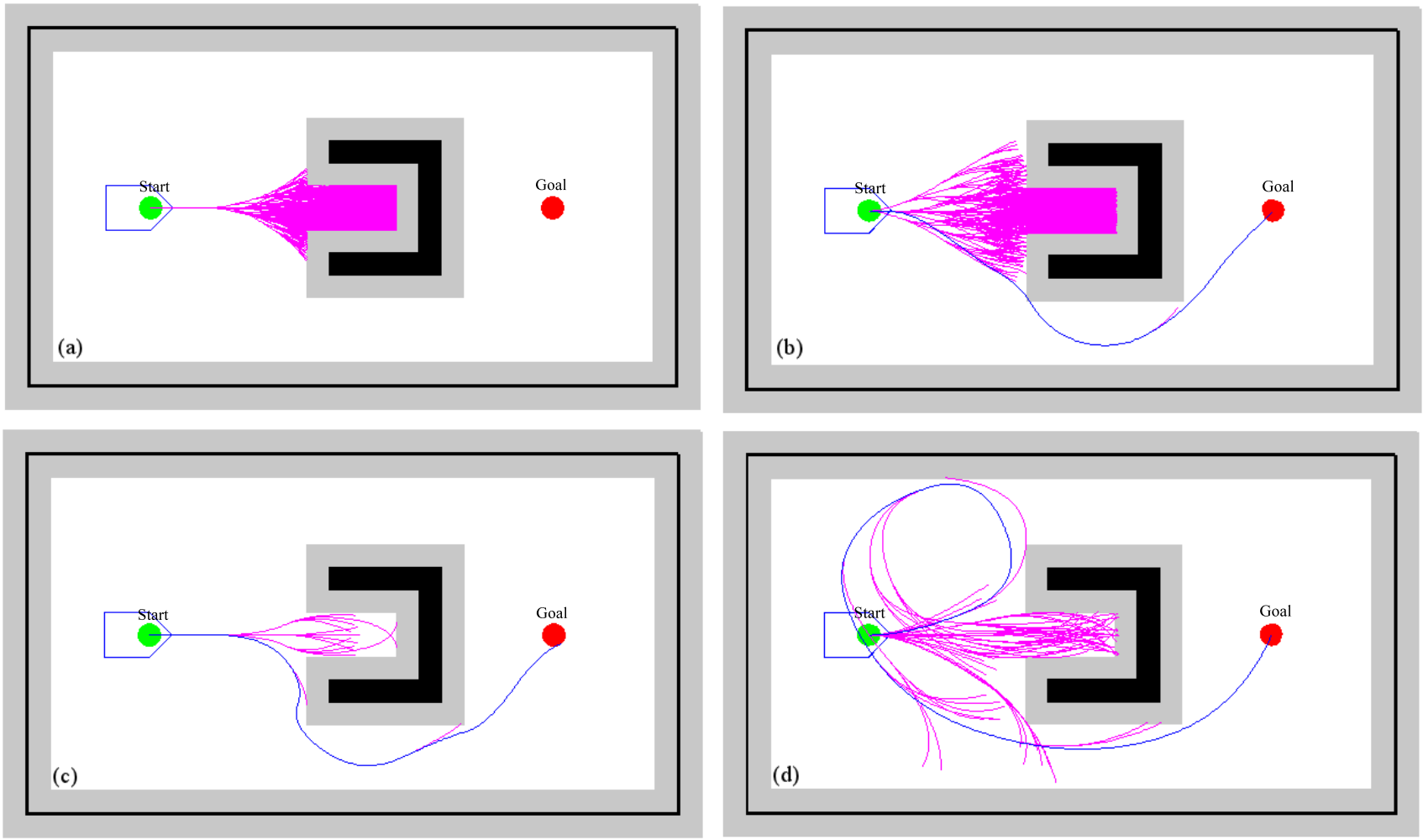

FIGURE 4.14 – DKP en mode backtrack sur le problème de l'obstacle en U montrant : (a) l'exploration interrompue avec des échantillons de 0.5s ; (b) l'arbre d'exploration et la solution avec des échantillons de 1s ; (c) l'arbre d'exploration et la solution avec des échantillons de 2s ; (d) l'arbre d'exploration et la solution avec des échantillons de 4s avec une manœuvrabilité limitée.

4.5 Étude expérimentale

DKP est un planificateur de trajectoire par échantillons qui est capable de générer des trajectoires splines. Basé sur une architecture de sélection/propagation, DKP crée un arbre d'exploration à partir des morceaux générés par la couche de propagation présentée dans le Chapitre 3. En particulier, DKP exploite l'espace des paramètres pour déterminer exactement l'ensemble des solutions locales qui respectent toutes les contraintes du problème. Avec ce processus de création des échantillons de trajectoire, il offre un bon équilibre sur la complexité des processus de propagation et de sélection, tout en restant totalement déterministe. De plus, la construction de l'espace des paramètres séparée de la recherche de solution et la création de l'arbre d'exploration permet de mettre en place des processus d'adaptation spécifiques à DKP, tel que le processus de backtrack.

Les sections suivantes de ce chapitre proposent une évaluation de l'architecture via l'utilisation de DKP et de ses variations : le mode optimal, le mode glouton et le mode backtrack.

La section suivante propose une étude des propriétés de DKP. Celle-ci se fait au travers d'une étude statistique mettant en avant les performances des différents modes de fonctionnement de DKP. On montre tout l'intérêt de l'utilisation de modes raisonnés pour un guidage efficace. Ces modes sont décrits dans le Chapitre 5.

La suite est consacrée aux différentes études expérimentales, montrant les capacités de DKP à résoudre des cas d'école. Ces expériences utilisent la plate-forme de robotique APM(Robot) développée en parallèle de la thèse au sein du département informatique de l'ISEN. Cette plate-forme sépare efficacement les concepts d'un programme lié à la robotique : communication entre les entités, gestion des données issues des capteurs et leur interprétation. Ainsi, cette plateforme permet de déployer efficacement DKP sur des robots aux caractéristiques et modes de communication variés dont je fais ici une description : Lego NXT, Miabot et iRobot Create.

Premièrement, on montre la capacité de DKP à retranscrire efficacement les caractéristiques cinématiques de ces robots fort différents. En effet, ces robots ne sont pas tous doués de la même capacité de manœuvre : leur empattement varie, la vitesse de leurs moteurs également... Nous verrons que DKP fournit des résultats pertinents pour chacun des robots utilisés.

Le second cas montre comment il est possible de retranscrire des pannes affectant la dynamique d'un robot. En effet, DKP permet de décrire et intégrer cette panne selon ses conséquences sur les caractéristiques cinématiques du robot. Cela est rendu possible en employant l'approche géométrique pour définir les contraintes. Ce chapitre montre donc l'implémentation d'une panne et la manière dont DKP gère ce cas de figure.

Enfin, ce chapitre propose un scénario de dépassement, impliquant plusieurs robots aux caractéristiques différentes.

4.6 Simulations : évaluation de la dynamique de DKP

Cette section présente une évaluation des performances de DKP dans ses différents modes de fonctionnement. L'environnement est volontairement généré de manière à ne pas contenir de minima locaux provoqués par des obstacles, ce qui n'empêche pas qu'il puisse y en avoir de par la cinématique du robot.

Contraintes

Pour ces simulations, nous produisons 100 séries d'environnements remplis d'obstacles. La taille de ces environnements est de 24 par 24 mètres. Ceux-ci contiennent de 0 jusque 100

obstacles par palier de 10. Ces obstacles sont des disques de rayon 1 mètre. Ils ne se recouvrent pas. Ce type d'environnements très chargés est intéressant à étudier car des approches classiques comme RRT sont rapidement mises en difficulté [Jaillet *et al.*, 2011].

Dans chacun de ces environnements, nous disposons aléatoirement un état de départ *Start*, défini par sa position et son vecteur vitesse (dans le respect des caractéristiques cinématiques du robot considéré). Un état d'arrivée *Goal* est également disposé aléatoirement. Ces états sont obligatoirement placés dans les zones libres de l'environnement, dans un carré de 20 par 20 mètres centré dans l'environnement. Ces deux états sont séparés d'une distance comprise entre 5 et 10 mètres.

En plus des contraintes d'évitement d'obstacles, nous définissons les contraintes cinématiques suivantes :

- la contrainte sur l'accélération linéaire la maintient entre 0 et $S^+ = 1$ m/s^2 ;
- la contrainte sur la vitesse linéaire la maintient entre 0 et 1 m/s.

Paramètres de DKP

La distance euclidienne entre le point final d'un morceau de trajectoire $p_{k_0 \ldots k_n}$ est utilisée comme partie heuristique de la fonction de guidage : $f = \rho_{\text{loc}} = d_{Goal, p_{k_0 \ldots k_n}(T_{k_0 \ldots k_n})}$.

La longueur du morceau de trajectoire $p_{k_0 \ldots k_n}$ est utilisée comme fonction de distance, noté *length*, entre les morceaux $p_{k_0 \ldots k_n}$ et le morceau précédent $p_{k_0 \ldots k_{n-1}}$ tel que $h = length$. Sur chaque environnement, en employant les contraintes définies au préalable, DKP fonctionne dans les modes de fonctionnement décrits par les 3 premières colonnes du tableau 4.1. Pour le mode backtrack, le rayon minimal des obstacles virtuels est défini à :

$$r_{vobstacle, p_{k_0 \ldots k_n}} = T_{k_0 \ldots k_{n+1}} \times \frac{S_+}{size} \tag{4.8}$$

avec $size = 10$. Le critère de déclenchement du backtrack est défini à $trigger = 4$.

Pour borner le temps des simulations, le nombre maximal de propagations est limité à 500 itérations. Le tableau 4.1 présente pour chaque mode la moyenne et l'écart-type du temps de calcul, la moyenne et l'écart-type de la durée de parcours des trajectoires solutions, le nombre d'explorations qui n'ont pas présenté de solution et le nombre d'expériences qui nécessitent plus de 500 propagations pour obtenir une éventuelle solution. Afin de comparer la qualité des solutions, on donne la moyenne des temps minimaux nécessaires pour atteindre en ligne droite l'objectif en partant de l'état de départ, lorsqu'une solution est obtenue par DKP. Ces temps de calculs ont été obtenus par un PC double cœur 2.53 GHz doté d'une architecture 64 bits avec 4 Go de RAM.

Résultats pour le mode optimal et le mode glouton

Les résultats dans le tableau 4.1 montrent que DKP requiert beaucoup de diversité en faisant varier la durée des morceaux pour gérer des environnements complexes : les modes glouton et optimal présentent tous deux des échecs lorsque DKP ne produit que des morceaux de trajectoires de durées 1 et 2 secondes. Avec beaucoup de diversité, via des trajectoires de 0.5, 1, 1.5 et 2 secondes, ces deux modes n'échouent que dans la situation typique où l'état de départ généré aléatoirement propose un vecteur vitesse initial faisant face directement à un obstacle.

Comme attendu de la part d'un algorithme de sélection basé sur A*, le mode optimal produit les meilleures solutions en terme de qualité, mais cela au prix de beaucoup de temps de calcul nécessaire à la construction de l'arbre d'exploration. Dans certaines situations, les 500

itérations de la propagation sont atteintes sans qu'une solution ne soit trouvée : l'extension de l'arbre d'exploration dans le mode optimal est très coûteux et cela se ressent d'autant plus que l'éloignement entre le point initial et le point objectif est important. Le mode glouton est alors bien plus rapide, au détriment de la qualité de la solution.

Résultats pour le mode backtrack

Le mode backtrack propose un bon équilibre entre la rapidité du mode glouton et la qualité des solutions obtenues dans le mode optimal. Avec des petits morceaux de trajectoire de 0.5 secondes, le mode backtrack fournit des solutions dont la qualité est proche du mode optimal sans dépenser tout le temps de calcul. Avec des morceaux de trajectoires plus grands, il permet d'obtenir des solutions rapidement mais la longueur de ces morceaux conduit à une diminution de la qualité, ce qui l'amène à des résultats proches du mode de fonctionnement du mode glouton. Pour améliorer les résultats de DKP, cette durée devrait être adaptée à la partie explorée de l'environnement, en favorisant les longs morceaux quand l'environnement est dégagé, tout en augmentant la finesse lorsque la zone explorée de l'environnement est remplie d'obstacles.

Temps de calcul

La Figure 4.15 met en avant les temps de calcul pour chaque ensemble d'obstacles pour chaque mode. L'évolution du temps de calcul est le même pour chaque mode. Lorsque l'environnement est presque rempli d'obstacles qui ne se recouvrent pas, l'espace libre entre les obstacles forme des couloirs qui guident naturellement la recherche. Par conséquent, les temps de calcul diminuent. Comme attendu, les performances en temps du mode backtrack se situent entre le mode optimal et le mode glouton.

	backtrack	**biais**	**durée des morceaux**	**Temps de calcul**	**Durée de la solution**	**Durée en ligne droite**	**DKP sans solutions**	**Simulations non terminées**	**Nombre de simulations**
1	sans	1	1 ;2	0.51 ± 0.12	9.72 ± 2.09	7.58 ± 1.41	170	0	1100
2	sans	1	0.5 ;1 ;1.5 ;2	1.54 ± 0.7	$\mathbf{8.79 \pm 1.62}$	7.46 ± 1.39	57	59	1100
3	sans	10	1 ;2	0.36 ± 0.07	12.84 ± 3.02	7.58 ± 1.41	170	0	1100
4	sans	10	0.5 ;1 ;1.5 ;2	$\mathbf{0.4 \pm 0.07}$	12.88 ± 2.87	7.58 ± 1.42	57	0	1100
5	avec	1	0.5	$\mathbf{0.5 \pm 0.19}$	$\mathbf{9.97 \pm 1.98}$	7.59 ± 1.42	61	0	1100
6	avec	1	1	0.37 ± 0.1	10.13 ± 2.17	7.59 ± 1.42	56	0	1100
7	avec	1	1.5	0.4 ± 0.15	14.08 ± 3.84	7.58 ± 1.42	62	0	1100
8	avec	1	2	0.42 ± 1.17	17.57 ± 7	7.57 ± 1.42	73	0	1100

TABLE 4.1 – Expériences sur DKP en mode optimal (1 et 2), glouton (3 et 4) et backtrack (5, 6, 7 et 8). L'adaptation employée par le mode backtrack fournit un bon équilibre entre la qualité de la solution inhérente au mode optimal et les bons temps de calcul du mode glouton. Les moyennes et écarts-types des solutions en ligne droite sont calculés pour les simulations donnant un résultat. Les durées et les temps de calcul sont donnés en secondes. Les meilleurs résultats sont mis en gras. Le nombre de planifications sans solutions dans le mode optimal avec des morceaux d'une durée 0.5s à 2s (ligne 4) correspond aux environnements pour lesquels le point de départ est situé à proximité d'un obstacle avec un vecteur vitesse initiale qui pointe vers cet obstacle. Le fait d'augmenter la durée des échantillons diminue la capacité d'anticipation du mode backtrack, ce qui explique le taux d'échec supérieur de la ligne 8.

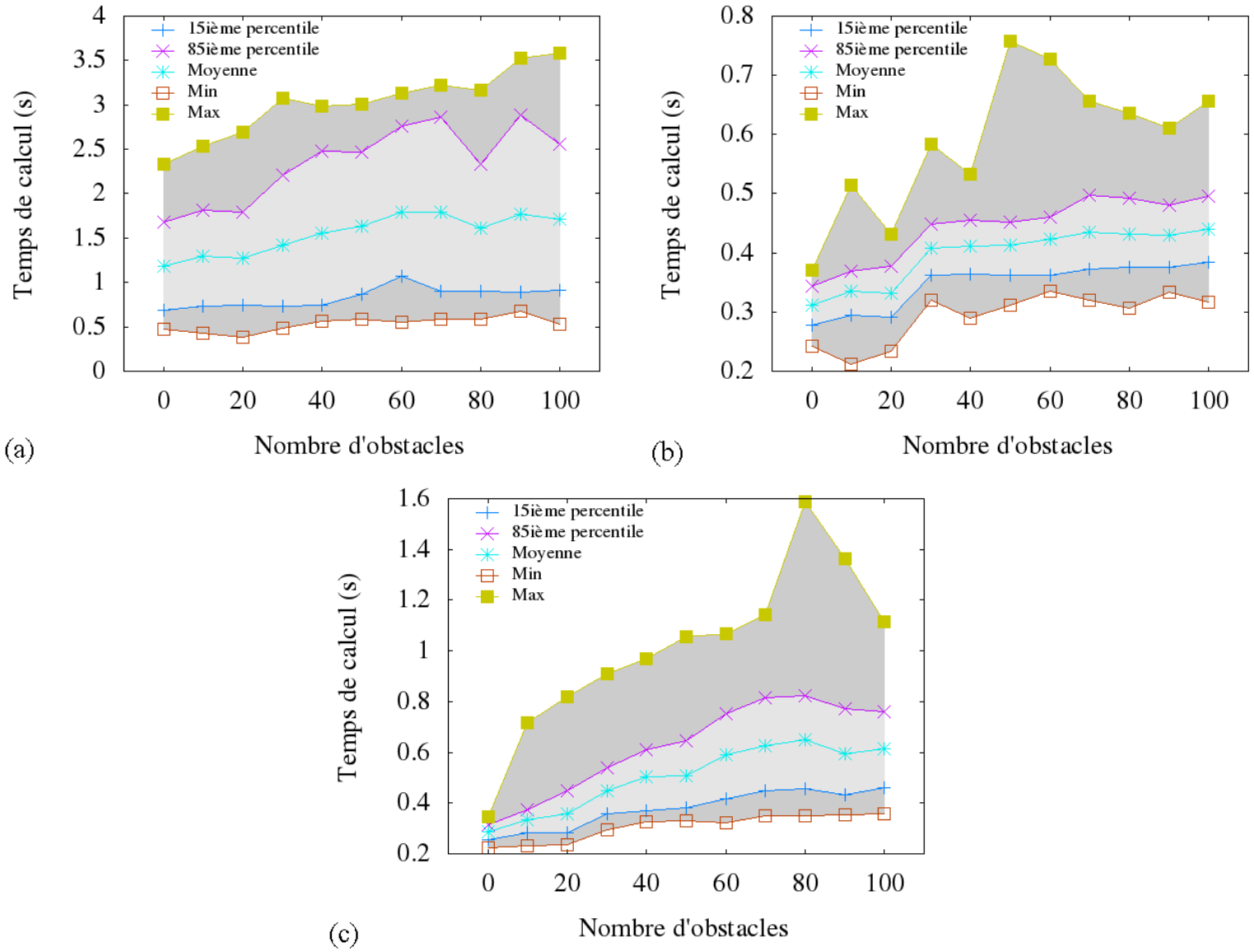

FIGURE 4.15 – Moyennes, minima, maxima, 15^{ieme} and 85^{ieme} percentiles des temps de calcul pour 1100 simulations de DKP dans le mode (a) optimal (avec les paramètres de la Table 4.1 ligne 2), (b) glouton (avec les paramètres de la Table 4.1 ligne 4) et (c) backtrack (avec les paramètres de la Table 4.1 ligne 5). Ces résultats sont donnés pour des environnements contenant [0,10,...,100] obstacles.

4.7 Expérimentations

L'environnement

Les expériences ont été menées dans le laboratoire d'informatique de l'ISEN. Il est équipé d'un plan de jeu de 2×1.6m. Des bandes noires forment un quadrillage de côté de 0.4 mètres. Une caméra $720p$ permet des prises de vues en 5/10/24 images/s du plan de jeu.

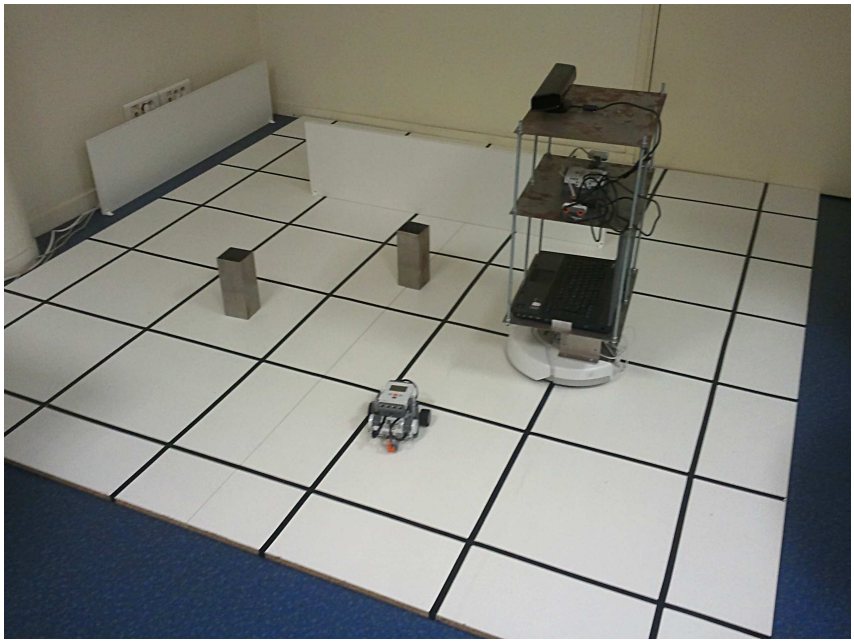

FIGURE 4.16 – Plan de jeu.

Les robots

Pour les expériences, plusieurs robots à deux roues indépendantes aux caractéristiques variées ont été employés :

Lego NXT Fournis sans modèle véritablement fixé, les Lego NXT permettent de créer plusieurs robots aux configurations différentes : par défaut, ils sont dotés de deux roues indépendantes d'un rayon de 14mm, reliés à deux moteurs dont la vitesse radiale maximale est de 900degrés/s, permettant une vitesse maximale d'environ 25cm/s. La stabilité est assurée par une roue folle. L'empattement est généralement contenu dans un rectangle de 25×15cm, l'entraxe étant fixé autour de 14cm selon les modèles. Le firmware Lejos permet d'exécuter notre plateforme APM(Robot), écrite en Java. Celle-ci se charge des communications et de l'application des commandes en vitesse linéaire/vitesse angulaire. Le positionnement est assuré par l'odométrie des moteurs. La faible durée des expériences évite de souffrir de leur importante dérive. Une mire au-dessus du robot fournit également un positionnement absolu des robots NXT, grâce à la caméra située au plafond.

iRobot Create La société iRobot fournit une version recherche de leur célèbre Roomba, le robot aspirateur autonome. Celui-ci, nommé Create, dispose des mêmes caractéristiques que son

FIGURE 4.17 – Exemple de robot à deux roues indépendantes Lego NXT.

homologue ménager sans la partie aspirante. En sus, il possède des entrées/sorties permettant l'ajout de nouveaux capteurs, ainsi que de nombreuses fixations permettant l'ajout d'un support à du matériel supplémentaire. C'est ainsi que notre iRobot Create accueille une plateforme supportant un PC portable. La communication est assurée par une liaison série vers le pc portable embarqué. Celui-ci permet au robot de jouer lui-même l'application. Le PC peut également faire relais par liaison Wifi avec un autre ordinateur disposant de la plateforme APM(Robot). Il faut noter qu'un robot de conception similaire est désormais commercialisé sous le nom de TurtleBot par Willow Garage, avec le support et un Kinect fournis d'emblée.

FIGURE 4.18 – iRobot Create surmonté d'une plateforme portant un ordinateur portable, des capteurs à ultra-son reliés à une brique NXT et un Kinect.

Miabot Bien que ce robot ne figure pas parmi les robots utilisés dans les expériences présentes ici, j'ai également expérimenté DKP sur des robots Miabots de Merlin Robotics. Ceux-ci sont des petits robots dénués de capteurs (en dehors d'une odométrie fort limitée). En revanche, leurs moteurs offrent une très bonne précision à faible vitesse (maximum 20cm/s à l'usage).

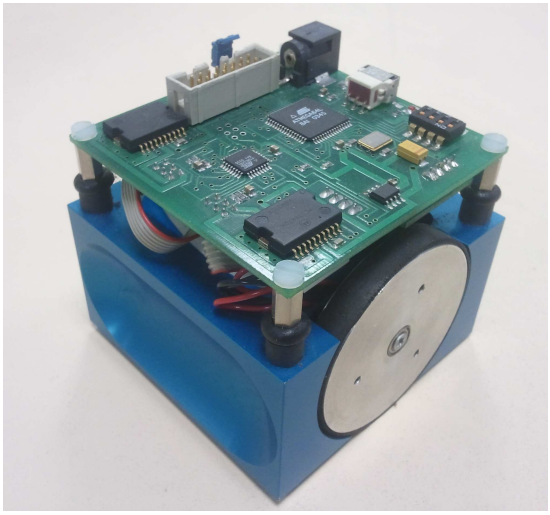

FIGURE 4.19 – Miabot, petit robot à deux roues produit par Merlin Robotics.

Utilisation

Les robots se localisent eux-même grâce à l'intégration des données des odomètres. Ces données sont ensuite envoyées directement à la plateforme APM(Robot) sur laquelle DKP planifie les mouvements du robot. La plateforme génère les commandes en utilisant un suivi de trajectoire spline pour robots à deux roues indépendantes, méthode décrite dans [Defoort *et al.*, 2007]. Ces commandes sont envoyées en retour aux robots.

4.7.1 APM(Robot) : plate forme pour le déploiement d'applications robotiques

DKP est déployé sur nos robots grâce à notre plateforme nommée APM(Robot) [Dinont *et al.*, 2010]. Le but est de transposer les raisonnements génériques du monde agent au sein d'applications robotiques. Ainsi, APM(Robot) fournit des processus génériques de communications entre les modules et/ou les robots. Le but est de faciliter l'implémentation, le test et le déploiement de nos expérimentations. Telle qu'utilisée dans cette thèse, APM(Robot) peut être apparentée à d'autres plateformes de simulation et de contrôle telles que l'architecture CHRIS [Lallée *et al.*, 2011], Robot Operating System [Quigley *et al.*, 2009] ou Player/Stage [Gerkey *et al.*, 2003]. APM(Robot) fournit un « data store » : un espace pour stocker les données auxquelles sont attachées des méta-données (type, source, timestamp, unité). Un des objectifs d'APM(Robot) avec ce data store est de mettre en place des applications robotiques modulaires auto-organisatrices, à la manière de [Ashley-rollman *et al.*, 2007].

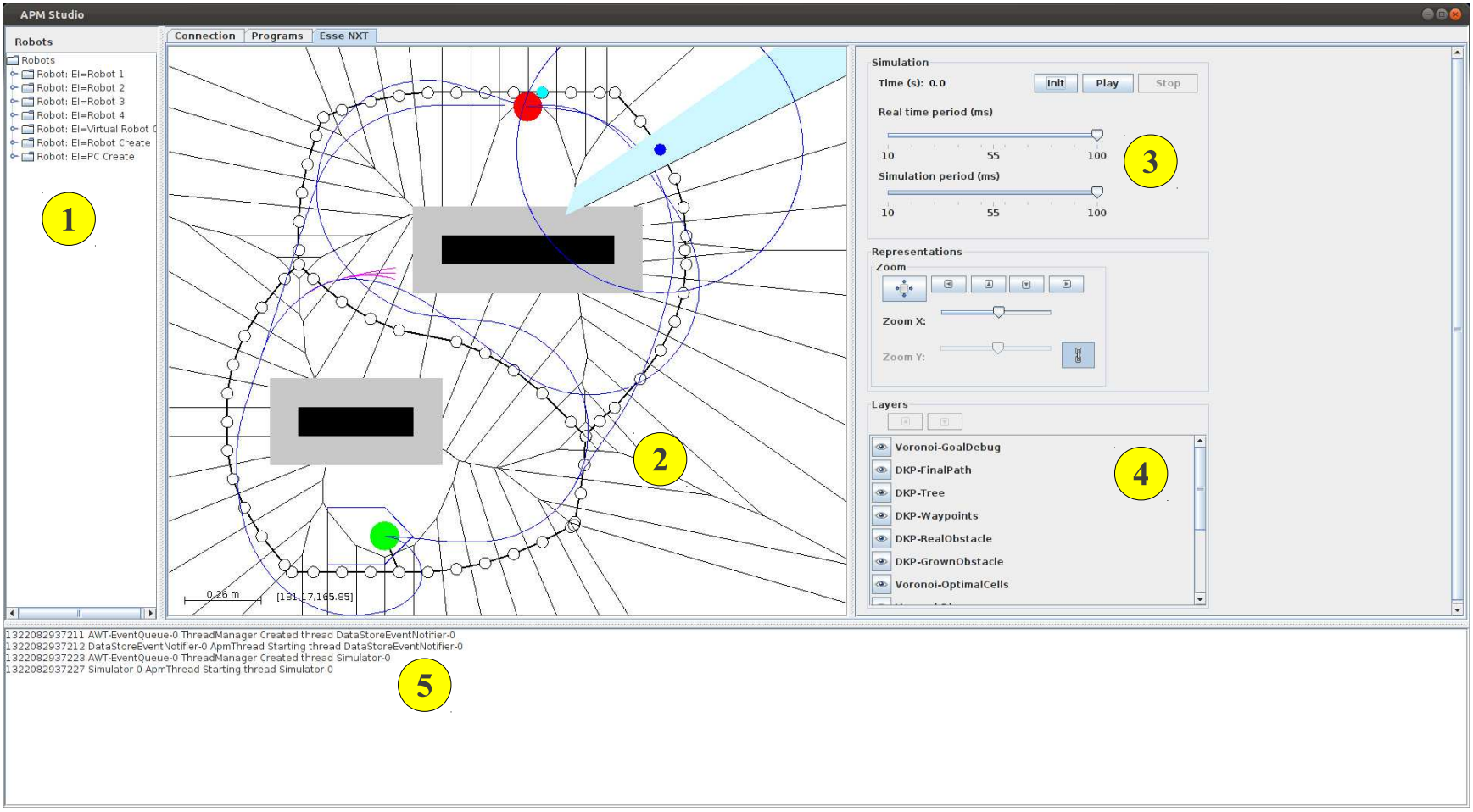

FIGURE 4.20 – Interface de APM(Robot) servant à la simulation et à l'exécution des trajectoires des robots. Nous voyons en : (1) les robots disponibles ; (2) la représentation de l'environnement ; (3) le contrôle du temps réel/simulé ; (4) les différentes couches d'information qui peuvent être affichées (Robot, trajectoire, obstacles réels, obstacles grossis) ; (5) le log d'APM(Robot), comprenant les messages qui transitent entre les entités du système (plateforme, robots contrôlés)

4.7.2 Environnement chargé d'obstacles

Cet exemple illustre la capacité de DKP à trouver rapidement des solutions dans des environnements chargés d'obstacles fixes, DKP planifie en employant le mode glouton. L'environnement contient des obstacles placés aléatoirement à la manière de la partie simulations. On obtient des environnements avec de nombreux passages sans issue, situation dans laquelle les approches par échantillonnage strictement aléatoires vont rester bloquées longtemps. DKP a été exécuté en mode glouton ($bias = 10$) avec les paramètres suivants :

- $T_{max} = 6$s, la durée maximale des échantillons ;
- $step = 0.5$s, le pas dans la durée des échantillons générés ;
- $S_- = 0$cm/s et $S_+ = 10$cm/s, les bornes sur la vitesse linéaire ;
- $A_- = 0$cm/s^2 et $A_+ = 10$cm/s^2, les bornes sur l'accélération linéaire.

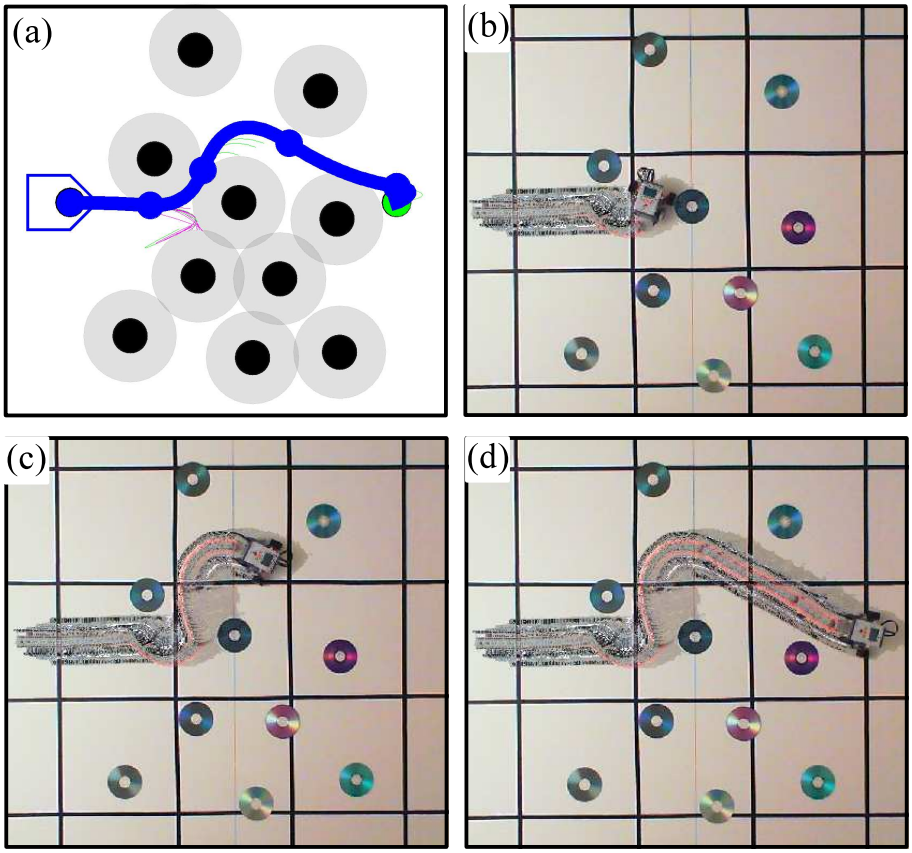

FIGURE 4.21 – Expérimentation dans un environnement chargé d'obstacles avec (a) trajectoire planifiée appliquée au robot de (b) jusque (d).

DKP trouve une solution avec un arbre de trajectoires pourtant limité (comparé à ceux de la figure 4.13, qui sont bien plus étendus). Même si la recherche est fortement biaisée vers l'objectif, la diversité en temps sur la durée des échantillons de trajectoire permet à DKP de s'échapper des minima locaux. Le résultat est illustré par la figure 4.21.

Le suivi de trajectoire [Defoort, 2007] compense efficacement l'imprécision des déplacements du robot. Toutefois, il est parfois nécessaire de faire un compromis entre la qualité du suivi en position et la qualité du suivi en angle. Nous avons fait le choix d'une forte précision en position :

ce compromis explique les rotations plus importantes que prévu dans nos expérimentations.

4.7.3 Planification en cas de panne

Cet exemple illustre la capacité de DKP à trouver des solutions malgré la présence d'une panne, modélisée sous la forme d'une restriction des points localement atteignables. Le mode optimal est utilisé. On contraint le robot à ne tourner que dans une seule direction (vers la gauche). L'environnement contient deux obstacles placés de manière à ce que le vecteur vitesse initial force le robot à tourner pour atteindre son objectif. DKP a été exécuté en mode glouton ($bias = 10$) avec les paramètres suivants :

- $T_{max} = 6s$, la durée maximale des échantillons ;
- $step = 0.5s$, le pas dans la durée des échantillons générés ;
- $S_- = 0\text{cm/s}$ et $S_+ = 10\text{cm/s}$, les bornes sur la vitesse linéaire ;
- $A_- = 0\text{cm/s}^2$ et $A_+ = 10\text{cm/s}^2$, les bornes sur l'accélération linéaire.

Le résultat est illustré par la figure 4.22.

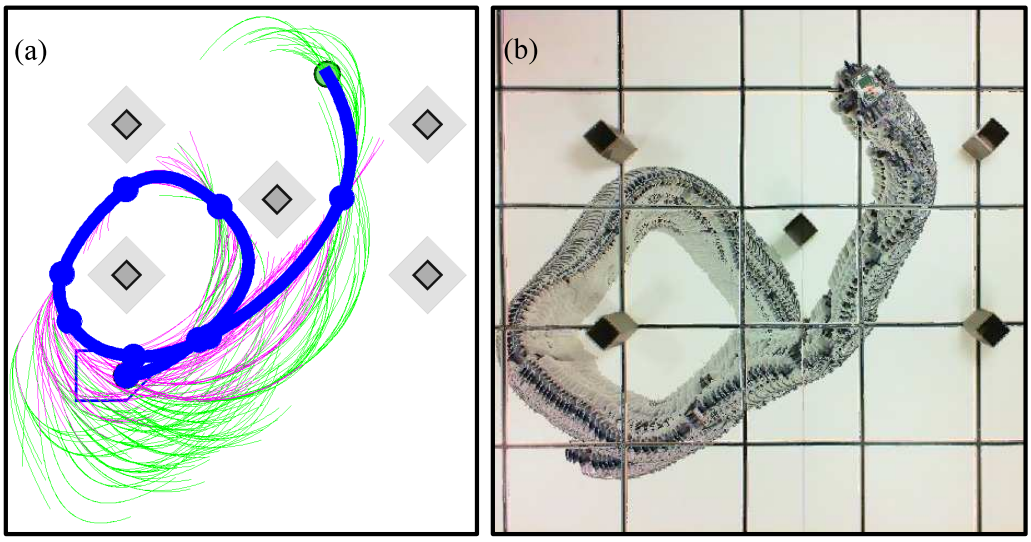

FIGURE 4.22 – Expérimentation d'une situation où le robot est confronté à une panne avec en (a) trajectoire planifiée appliquée au robot en (b).

4.7.4 Dépassement

L'un des objectifs de la thèse est de pouvoir décrire des comportements humains pour résoudre des situations de conduite usuelles. C'est pourquoi l'une des situations que nous allons détailler est le problème du dépassement. DKP ne dispose que d'un guidage simpliste. Cet exemple sera amélioré dans le chapitre suivant. Il est tout de même possible de confronter DKP à cette situation. De manière similaire à [Wilkie *et al.*, 2009], des données de haut niveau peuvent être intégrées à DKP pour anticiper les futurs états possibles des obstacles mobiles à éviter : c'est l'objet du module de raisonnement présenté dans le Chapitre 5.

Cette situation est illustrée par la Figure 4.23. Deux robots R_1 (en bas) et R_2 (au dessus) effectuent des déplacements en ligne à vitesse constante (respectivement 3cm/s et 4cm/s). Un troisième robot R_0 utilise DKP pour planifier une trajectoire minimisant le temps tout en évitant

les robots R_1 et R_2. La thèse n'est absolument pas focalisée sur l'estimation de trajectoire, bien que l'approche géométrique utilisée dans DKP pour la détermination de l'espace localement atteignable ouvre des pistes à explorer en la matière. Aussi, la trajectoire des obstacles est entièrement connue. DKP a été utilisé en mode backtrack ($bias = 10$) avec les paramètres suivants :

- $T_{max} = 6$s La durée maximale des échantillons ;
- $step = 0.5$s le pas dans la durée des échantillons générés ;
- $S_- = 0$cm/s et $S_+ = 20$cm/s les bornes sur la vitesse linéaire ;
- $A_- = 0$cm/s^2 et $A_+ = 10$cm/s^2 les bornes sur l'accélération linéaire.

Les paramètres supplémentaires, spécifiques au mode backtrack, sont :

- le rayon minimal des obstacles virtuels qui est fixé à $r_{vobstacle,p_{k_0 \dots k_n}} = T_{k_0 \dots k_{n+1}} \times \frac{S_+}{size}$ avec $size = 10$ et $T_{k_0 \dots k_{n+1}}$ la durée du morceau retiré ;
- la valeur de déclenchement du backtrack qui est fixée à $trigger = 4$.

Le résultat est illustré par la figure 4.23.

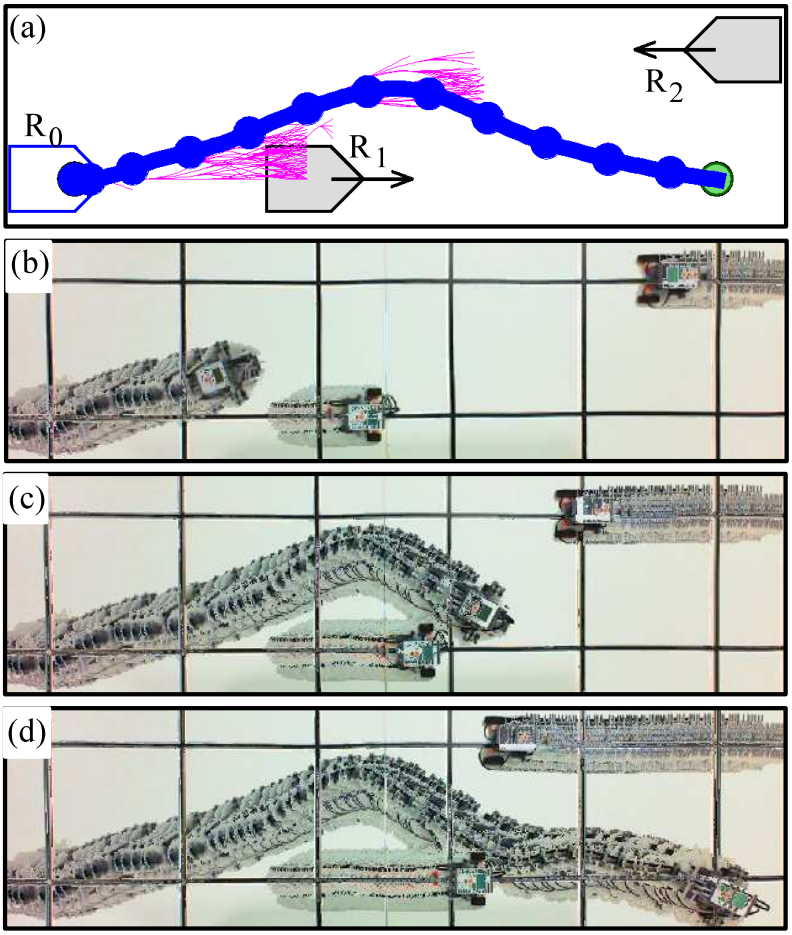

FIGURE 4.23 – Expérimentation d'un cas du dépassement d'un robot R_1 par R_0 en évitant R_2 avec (a) trajectoire planifiée appliquée au robot de (b) jusque (d).

4.8 Discussion critique de DKP

(Sur)guidage de DKP

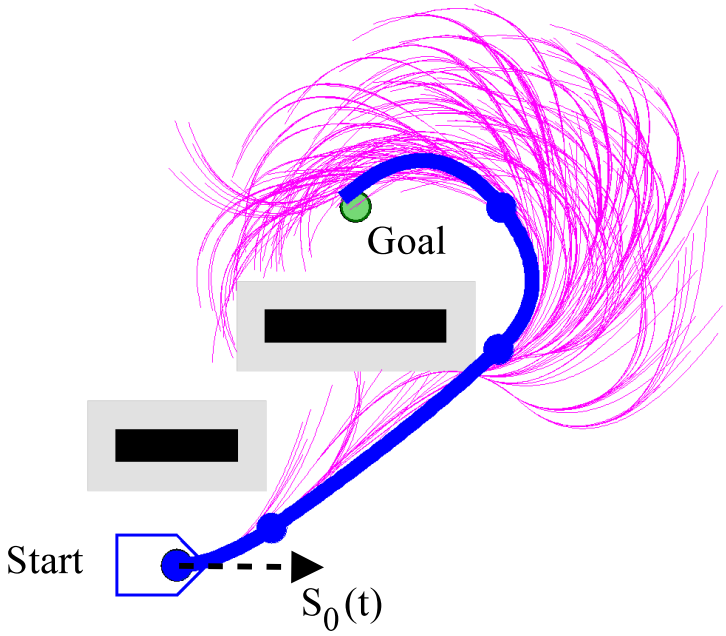

FIGURE 4.24 – Exemple d'un arbre DKP en magenta. La trajectoire sélectionnée par DKP depuis l'état *Start* jusque l'état *Goal* apparaît en bleu.

La Figure 4.24 donne un exemple de planification effectuée par DKP. L'environnement considéré, apparemment simple, n'est pourtant pas facile à manier pour un planificateur de trajectoire. En effet, selon la manœuvrabilité et l'empattement du robot, afin d'obtenir une trajectoire à la fois faisable et optimale, le robot pourrait choisir de passer les obstacles par la gauche, par la droite, voire de circuler entre les deux. Dans l'exemple précédent, DKP trouve une solution que l'on peut juger bonne. Toutefois, de nombreuses et inutiles branches échouent à franchir l'obstacle. Cette situation est provoquée par l'utilisation de la distance euclidienne séparant la fin d'un morceau de trajectoire à l'objectif comme critère de recherche. Ce guidage force le niveau de planification de trajectoire à générer des morceaux s'approchant au plus de l'objectif. C'est la diversité sur le temps qui permet de générer les trajectoires anticipant le franchissement des obstacles.

Influence de la situation initiale

Toujours sur la situation, nous pouvons remarquer l'influence du vecteur vitesse initiale $\overrightarrow{S}(0)$. Celui-ci biaise l'exploration de l'environnement vers la partie droite de l'environnement. Si aucune solution n'était admissible dans cette direction, DKP aurait passé beaucoup de temps à rechercher des solutions dans cette partie de l'environnement avant de générer des morceaux explorant la partie gauche. Ce problème n'est pas spécifique à DKP : il est difficile de trouver un bon compromis entre l'exploration de l'environnement et la recherche locale de la meilleure manière de contourner une difficulté. L'utilisation de processus aléatoires pour explorer de manière uniforme l'environnement trouve ici tout son sens. Toutefois, une couche cognitive

ajoutée au dessus de DKP me permet d'envisager ce problème sans l'utilisation de processus aléatoire : c'est l'objectif du chapitre suivant (et de la thèse dans son ensemble).

Difficulté de paramétrage

Comme vu ci-dessus, DKP apparaît parfois difficile à paramétrer dans certains cas : la manière de choisir les valeurs des paramètres, en particulier la diversité en temps des morceaux de trajectoires, dépend fortement de la situation rencontrée dans l'environnement exploré. Le mode backtrack résout partiellement ce problème en exploitant l'espace des paramètres pour déterminer si une trajectoire peut être poursuivie et dans le cas contraire, abandonner la branche explorée et reconstruire localement l'arbre de trajectoires. Le prochain chapitre explique comment je règle en partie ce problème.

4.9 Conclusion

DKP est une approche de planification de trajectoire par échantillonnage. Il est basé sur une architecture de sélection/propagation pour construire un arbre d'exploration dans les parties atteignables de l'environnement. Il produit des solutions pour des robots à deux roues indépendantes en utilisant les échantillons de trajectoire générés par la couche de propagation présentée dans le Chapitre 3. Le résultat produit, une trajectoire spline, permet de décrire des manœuvres complexes, même avec des échantillons à base de quadratiques uniquement. La couche de sélection s'appuie sur A^* pour décider des branches à développer. De plus, DKP utilise l'espace des paramètres construit à chaque propagation pour ne développer que les branches pouvant être prolongées en étant sûr de respecter l'ensemble des contraintes du problème. Enfin, l'espace des paramètres est avantageusement utilisé pour générer divers échantillons en se basant sur la variation de la durée des morceaux.

Trois modes d'exploration sont étudiés :

- le mode optimal propose une exploration exhaustive mais coûteuse de l'environnement, guidée par la minimisation de la distance à l'objectif et de la distance déjà parcourue ;
- le mode glouton se guide seulement en distance à l'objectif pour converger plus rapidement, au risque de faire des détours et de rester bloqué dans les minima locaux ;
- le mode backtrack modifie la structure de l'arbre d'exploration et de l'environnement grâce à des obstacles virtuels pour se détourner des minima locaux, principaux défauts des deux modes précédents.

Un point fort de DKP est le contrôle de chaque étape de la planification de trajectoire. En particulier, l'approche géométrique pour décrire et appliquer les contraintes des robots permet de facilement décrire un problème : il suffit de fournir des bornes sur les vitesses et accélération des robots, une description de leurs formes et des trajectoires pour l'évitement d'obstacle fixe ou mobile. Cela rend DKP facilement applicable à des problèmes variés dans lesquels des restrictions des mouvements doivent être spécifiés. Les expérimentations montrent tous ces points forts à travers 3 situations appliquées à différents robots à deux roues indépendantes. Dans chaque cas décrit, DKP rend précisément les capacités des robots tout en fournissant des solutions directement utilisables, afin de se reposer sur l'algorithme de suivi de trajectoire.

Vers une approche cognitive

Comme beaucoup d'approches pour la planification de trajectoire, DKP fait face à deux besoins contradictoires : le besoin d'exploration de l'environnement pour découvrir des solutions

et le besoin d'un guidage pour converger rapidement vers des solutions bonnes. Le mode backtrack offre un premier mécanisme d'adaptation du comportement du planificateur mais je peux aller plus loin : selon moi, l'expression du guidage peut être séparée de la planification de trajectoire pour intégrer une couche cognitive qui régule le comportement du planificateur de trajectoire. Pour mettre cela en pratique, nous pouvons tirer avantage de la description des contraintes dans DKP pour décrire les propriétés d'un comportement de pilotage. Ainsi, nous pourrons décrire de manière symbolique le comportement désiré dans telle ou telle partie d'un environnement. Selon la zone explorée par DKP, ce dernier peut alors faire grandir l'arbre d'exploration avec un ou plusieurs comportements de pilotage adaptés au problème. Si une solution existe, elle respecte de surcroît les contraintes de continuité et de cinématique nécessaire à la bonne application de cette solution. Ainsi, mon objectif dans le chapitre suivant est de décrire l'architecture et la mise en œuvre de comportements de pilotage permettant à un robot de raisonner sur ses capacités ou sur son environnement en cours de planification.

Nous avons vu dans le chapitre précédent qu'en utilisant une approche géométrique pour la prise en compte des contraintes sur l'espace des paramètres des solutions, je fournis des mécanismes de contrôle de toute la chaîne de planification de trajectoire, depuis la propagation jusqu'aux mécanismes de sélection. J'ai montré dans ce chapitre comment exploiter les contraintes pour décrire des comportements de déplacement et les vérifier. Je m'appuie sur les contraintes pour décrire à la fois les limites physiques du robot et des contraintes liées à l'adoption d'un comportement spécifique. Il est alors possible d'utiliser l'espace des paramètres s'il existe au moins des solutions admissibles localement.

Chapitre 5

Raisonnements par comportements de pilotage pour la planification de trajectoire

Face à la complexité croissante des applications robotiques, il est devenu nécessaire d'utiliser des raisonnements symboliques pour déterminer la suite d'actions à effectuer pour effectuer ses tâches. Ces situations impliquent souvent des interactions avec des humains : le robot doit alors avoir un comportement qui doit s'approcher de celui attendu par un humain pour pouvoir être plus compréhensible.

Architecture

Ce chapitre débute en présentant les besoins d'une architecture pour le guidage à haut niveau des planificateurs de trajectoire. Je pars du constat que les actuelles améliorations des planificateurs de trajectoires portent sur le délicat compromis entre plusieurs propriétés :

- besoin d'exploration de l'environnement pour trouver des solutions quelle que soit la situation et quel que soit le système ;
- besoin de performance pour obtenir rapidement des solutions dans le cadre des applications fonctionnant en temps réel ;
- besoin de prédictibilité dans le comportement du planificateur de trajectoire toujours dans le cadre du temps réel :
  - quantifier le temps nécessaire à la création d'échantillons de trajectoire ;
  - identifier les processus nécessitant des choix aléatoires ;
  - connaître le temps nécessaire pour trouver au moins une solution ;
  - déterminer le temps nécessaire pour améliorer une solution (à la manière de RRT*).
- besoin de qualité pour obtenir des trajectoires jugées bonnes voire optimales ;
- besoin d'adaptation à des informations venues du haut niveau.

Les approches les plus récentes (RRT, RRT*, AD*, RRT sur un espace de configuration hybride, DKP) proposent chacune à leur manière une solution pour l'une des propriétés présentées précédemment. Toutefois, ces propriétés sont codées dans le planificateur de trajectoire. À mon sens, cela devrait être le rôle d'un niveau de guidage de fournir l'abstraction permettant de décrire et fournir ces propriétés. Une tel processus sert mon besoin : décrire et introduire du savoir-faire humain en conduite dans le processus de planification de trajectoire.

Pour répondre à ce besoin, je présente une architecture à deux couches fortement couplées pour la planification de trajectoire de robots à deux roues indépendantes. Cette architecture peut être utilisée pour décrire des comportements de pilotage puis générer des trajectoires candidates. Ensuite, elles pourront être évaluées par une couche supérieure afin de choisir la trajectoire à suivre.

La première couche s'appuie sur un arbre TÆMS pour décrire l'objectif actuel du robot et sa décomposition en des comportements de pilotage alternatifs. La seconde couche s'appuie sur mon planificateur de trajectoire pour faire grandir un arbre de trajectoires splines. Ces trajectoires respectent les contraintes cinématiques du problème, telles que des limites sur les vitesses linéaires/angulaires ou l'évitement d'obstacle.

Ces deux couches interagissent entre elles, permettant aux deux arbres de grandir simultanément. D'abord, les nœuds de l'arbre TÆMS contiennent les paramètres de pilotage utilisés dans DKP pour générer ses branches. Puis, les points atteints par les nœuds de l'arbre DKP sont utilisés pour déclencher des événements qui génèrent de nouveaux sous-arbres dans l'arbre TÆMS. Le résultat est un arbre de trajectoires dont la faisabilité est garantie par l'utilisation de DKP qui apporte de plus des propriétés utiles : existence de solutions locales et déterminisme du processus d'exploration de l'espace des solutions. Les paramètres de DKP contrôlés par la couche cognitive sont détaillés dans la Section 5.2. Ensuite, je présente les interactions entre l'arbre de pilotage et l'arbre d'exploration dans la Section 5.3.

Exemples

On décrit deux exemples, présentés dans [Gaillard *et al.*, 2012], visant à illustrer des comportements de pilotage, ici des manières de résoudre une situation ou explorer des alternatives : la génération et l'évaluation de trajectoires dans un plan de circulation basé sur un diagramme de Voronoï et un comportement de dépassement dans un environnement de type route. Le premier exemple, développé dans la Section 5.4, montre comment je peux décrire et explorer des alternatives de contournement d'obstacles. Le second exemple, présenté dans la Section 5.5, se concentre sur le guidage de DKP sur un environnement structuré, constitué par les plans d'une partie de l'ISEN. Enfin, le dernier exemple, dans la Section 5.6, démontre l'expressivité des comportements de pilotage pour décrire la manière de résoudre une situation usuelle de conduite : le dépassement sur route à deux voies avec sens de circulation opposés.

A terme, je veux ajouter des traits humains liés à la conduite dans le mécanisme de planification de trajectoire. Ce type de comportements est recherché dans les jeux vidéo, à l'image de Gran Turismo 5 (Figure 5.1[7]). L'illustration montre la partie de création des pilotes et les différentes possibilités de personnalisation basées sur des compétences de pilotage (précision, anticipation des freinages...) mais également sur des caractéristiques physiques et psychologiques (résistance à l'effort, caractère qui aura une influence sur la prise de risque). Dans les jeux récents, cette personnalisation (des véhicules et des personnages) se fait souvent au travers d'une modification des propriétés physiques de l'entité en déplacement. Ce choix de conception permet l'application plus ou moins efficace des actions décidées par l'IA.

Avec mon modèle pour la planification de trajectoire, la couche cognitive est là pour décrire et évaluer des comportements de pilotage. Soumis à plusieurs hypothèses sur les caractéristiques de l'environnement, par exemple, la présence ou non d'un obstacle, les trajectoires possibles d'un obstacle mobile, je pense qu'il est possible de guider le processus de choix de la trajectoire à suivre en fonction d'un modèle de comportement décrivant le pilote. Par exemple, un pilote « agressif » choisira volontiers une trajectoire permettant un dépassement alors même que la présence d'un véhicule sur la voie de dépassement rend très probable le risque de collision.

FIGURE 5.1 – Image issue de Gran Turismo 5 qui montre l'écran de conception de pilotes selon des compétences de pilotage mais également des caractéristiques physiques et psychologiques.

5.1 Comportements de pilotage pour la planification de trajectoire

Mon objectif à terme est de fournir à des robots autonomes des comportements de pilotage qui soient semblables à notre manière de conduire au quotidien. Ces comportements doivent également respecter les contraintes physiques qui s'imposent aux robots utilisés. Je veux ainsi construire des applications robotiques sur la base de blocs de haut niveau décrivant des stratégies de navigation telles que *suivre* or *dépasser*. Ces stratégies peuvent être également couplées à des comportements tels que *conduite douce* ou *conduite agressive*. Il faut donc un langage permettant la description de la suite d'actions traduisant une stratégie de navigation et son évaluation. Cela nécessite enfin un niveau de vérification des contraintes liées aux déplacements.

5.1.1 Reprise du modèle de Reynolds

Une telle architecture a déjà été mise en place par [Reynolds, 1999] pour établir le comportement de personnages simulés autonomes et leurs déplacements. Dans ce papier, Reynolds propose une architecture à trois niveaux capable d'exprimer des comportements de pilotage : un niveau de sélection d'action décide des objectifs, fournis à un niveau de planification des mouvements. Ces mouvements sont ensuite fournis à une couche de locomotion qui se charge d'accomplir la série de mouvements. Je réutilise cette idée de base, ici transposée aux robots naviguant en environnement réel : en effet, les problématiques rencontrées et la manière de les résoudre sont similaires dans les deux cas. La principale idée retenue est d'imbriquer les couches cognitives et de planification de trajectoire. Toutefois, le modèle de Reynolds est initialement pensé pour des entités simulées. S'il est strictement appliqué, les mouvements nécessaires à la

réalisation des objectifs sont nécessairement faisables par la couche de locomotion. Dans un cadre simulé, il est relativement aisé de mêler la faisabilité physique d'une décision au processus de sélection d'action comme l'a montré le Chapitre 2 consacré à l'état de l'art. Cette hypothèse n'est plus vraie dans le cas de robots réels si le processus de sélection d'action ne vérifie pas la faisabilité des déplacements : les deux couches sont alors trop séparées. Ainsi, la couche de raisonnement de haut niveau, si elle ne représente pas toutes les contraintes physiques, pourrait décider des actions qui ne sont pas réalisables physiquement. Or, mêler ces deux éléments dans une même couche de raisonnement est une tâche particulièrement difficile.

5.1.2 Vers une approche cognitive de la planification de trajectoire

Pour réaliser mon architecture de description de stratégies de pilotage, il faut un planificateur de trajectoire gérant le modèle de mouvement des robots et générant les trajectoires candidates. DKP détient de mon point de vue les propriétés nécessaires pour être la couche basse d'un planificateur de mouvement à base de comportements de pilotage pour des robots à deux roues.

Les propriétés attendues d'un tel planificateur sont les suivantes. D'abord, Il doit produire des trajectoires respectant les contraintes cinématiques et les contraintes d'évitement d'obstacle. Ensuite, il doit transcrire les comportements de pilotage en paramètres utilisables par le planificateur de trajectoire sous-jacent. Ce dernier doit fournir des solutions satisfaisant un critère d'optimisation. Enfin, le planificateur de trajectoire sous-jacent doit être déterministe : la partie cognitive ne doit pas répéter la planification jusqu'à ce qu'une solution optimale puisse être trouvée si elle existe (dans ce cas, il faudrait au moins prouver l'inconsistance du problème avant de se lancer dans une recherche de solution).

DKP répond à ces besoins pour la couche basse. Pour la couche haute, bien que je n'exploite pas dans ce manuscrit toutes les caractéristiques de TÆMS, je décide de l'utiliser, les futurs travaux pourront bénéficier de toute son expressivité. L'ensemble est représenté par la Figure 5.2.

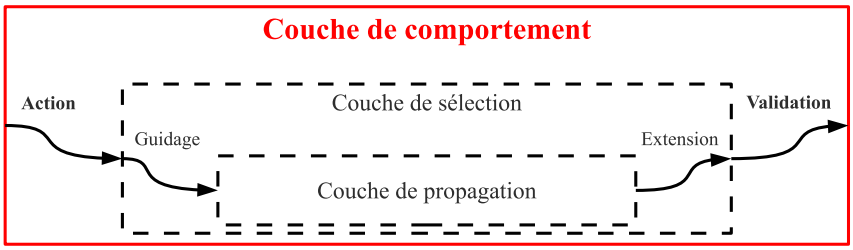

FIGURE 5.2 – Architecture retenue pour exprimer des comportements de pilotage dans le cadre de la planification de trajectoire sous contraintes. DKP assure la vérification de ces dernières tandis que les comportements de pilotage régulent le comportement de DKP.

5.1.3 Des arbres qui s'influencent

La solution utilise deux couches qui interagissent entre elles. Chaque couche fait grossir un arbre dont la construction est influencée par l'évolution de l'autre. L'*arbre de pilotage* est construit à partir de l'instanciation de sous-arbres HTN représentant des *comportements de pilotage*. L'*arbre de trajectoires* est dynamiquement étendu en appliquant sur les morceaux de trajectoires les *paramètres de pilotage* établis dans les comportements de pilotage.

Cette approche combine des interactions descendantes et ascendantes. Ces dernières fournissent une manière pour la couche de planification de trajectoire de transmettre des informations à la couche supérieure. Cela permet de décider quelles actions prendre en fonction des différentes alternatives possibles.

Fonctionnement

Je propose donc de créer un planificateur de trajectoire guidé par des comportements de pilotage. Il fonctionne selon les principes suivants. Deux arbres grossissent en parallèle : un premier arbre décrit les comportements de pilotage et le second contient les trajectoires. Ils font référence chacun aux niveaux correspondant du modèle de Reynolds [Reynolds, 1999]. L'arbre de comportements de pilotage contrôle la croissance de l'arbre de trajectoires selon son état. Pour cela, les comportements de pilotage s'appuient sur les propriétés des couches de sélection/propagation de DKP. L'arbre de trajectoires créé par DKP grandit si possible dans l'environnement et déclenche l'instanciation de nouveaux comportements dans l'arbre de comportements de pilotage en réaction à des situations rencontrées dans l'environnement. L'arbre de comportements de pilotage est le reflet de l'arbre de trajectoires : chaque comportement de pilotage correspond à un sous-arbre valide de l'arbre de trajectoires fourni par DKP, dans la mesure où le comportement de pilotage respecte la dynamique et la cinématique du robot. Finalement, l'arbre de comportements de pilotage est utilisé pour sélectionner une chaîne de comportements à suivre, décrite par le formalisme TÆMS.

5.1.4 Propriétés retenues

Cette architecture peut être vue comme un planificateur à base de roadmap « comportementale » dans lequel les points de passage sont remplacés par des comportements de pilotage situés dans l'environnement. DKP est ici utilisé pour traduire et vérifier la faisabilité de chaque nœud appartenant à l'arbre de comportements de pilotage en cours d'élaboration. Les propriétés suivantes facilitent la mise en place d'une telle architecture. Chaque comportement de pilotage est d'abord traduit en un ensemble de contraintes (en utilisant la représentation géométrique à la base du fonctionnement de DKP) et de paramètres internes à DKP. Le tout est ensuite transmis à DKP. L'arbre d'exploration de DKP est alors étendu tout en suivant les comportements de pilotage par héritage des paramètres sous-jacents. DKP garantit la continuité entre branches, permettant ainsi d'obtenir une continuité entre les comportements de pilotage à plus haut niveau. DKP génère des branches avec le respect des contraintes cinématiques pour chaque comportement de pilotage tant qu'il peut fournir une solution locale. Le comportement déterministe de DKP facilite l'évaluation des comportements de pilotage et la reproductibilité des résultats.

5.2 Contrôle de DKP par des paramètres de pilotage

Cette section identifie les paramètres de DKP permettant de mettre en place mon architecture de planification de trajectoire guidée par les comportements de pilotage. J'explique ensuite comment est établi un arbre de comportements de pilotage et ses interactions avec l'arbre de trajectoires.

5.2.1 Contraintes

Contraintes cinématiques usuelles

Dans DKP, toutes les contraintes de planification de trajectoire telles que les contraintes cinématiques ou les contraintes d'évitement d'obstacles partagent la même définition géométrique (voir la Section 3.7). Les contraintes cinématiques usuelles telles que les contraintes de vitesse ou accélération linéaire/angulaire sont définies en fournissant des bornes sur leurs équations respectives : DKP fournit alors la représentation géométrique associée dans l'espace des paramètres nommée $A_{cs,path_{cs}}(t)$ et qui peut être déplacée selon $path_{cs}(t)$ pour tenir compte des obstacles mobiles. Comme les positions sont dépendantes du temps, leur forme est translatée le long d'un chemin statique selon l'instant t au cours duquel la trajectoire est planifiée.

Autres contraintes

Pour les contraintes les moins usuelles, DKP requiert directement leur représentation géométrique complète $A_{cs,p}$ et leur trajectoire, le tout exprimé dans la base de définition de la contrainte. Si cette base n'est pas une de celles des contraintes usuelles (seules les bases pour la position, vitesse linéaire et accélération linéaire sont fournies par défaut dans DKP), DKP a également besoin des transformations T_{cs} permettant de passer de cette base à l'espace des paramètres et réciproquement. Finalement, une contrainte *cs* est formée par le couple $(A_{cs,p}(t), T_{cs})$. L'ensemble des contraintes s'appliquant sur un morceau de trajectoire est noté $CS = \{cs_k\}$.

5.2.2 Guidage pour la propagation

Un guidage pour le niveau de propagation est défini comme le tuple $(A_{Goal,path_{Goal}}(t), Goal)$ qui associe : un objectif local nommé *Goal*, une trajectoire $path_{Goal}(t)$ le long de laquelle la surface peut se déplacer au cours du temps et une partie finie de l'environnement $A_{Goal,path_{Goal}}(t)$ qui est associée à l'objectif local *Goal*. Cette surface est représentée selon l'approche par géométrie constructive de surface utilisée pour les contraintes.

Interprétation

Le niveau de propagation dans DKP étend localement un arbre d'exploration dans l'environnement sur lequel ce guidage pourra agir. Lorsque le point final du morceau de trajectoire $p_{k_0 \ldots k_n}(t)$ sélectionné par le niveau de sélection de DKP est contenu dans la surface $A_{Goal,path_{Goal}}(t)$, le morceau qui sera ensuite grossi minimisera la distance à l'objectif *Goal* associé à cette surface. Ce mécanisme est illustré par la Figure 5.3. Un ensemble de guidages pour le niveau de propagation est noté *PGuide*. A_{PGuide} est alors l'ensemble des surfaces qui sont associées à un objectif parmi l'ensemble de guidages par objectif. Il est également obligatoire qu'aucune surface de l'ensemble A_{PGuide} ne puisse en recouvrir une autre :

$$A_{Goal,path_{Goal},k}(t) \cap A_{Goal,path_{Goal},l}(t) = \emptyset \tag{5.1}$$

avec $k \neq l$ et k et $l \in A_{PGuide}$. Autrement, cela induirait des règles de choix qui ne pourraient que polluer le fonctionnement de DKP : si des objectifs multiples se présentent, la résolution de ce choix est du ressort d'une couche de haut niveau.

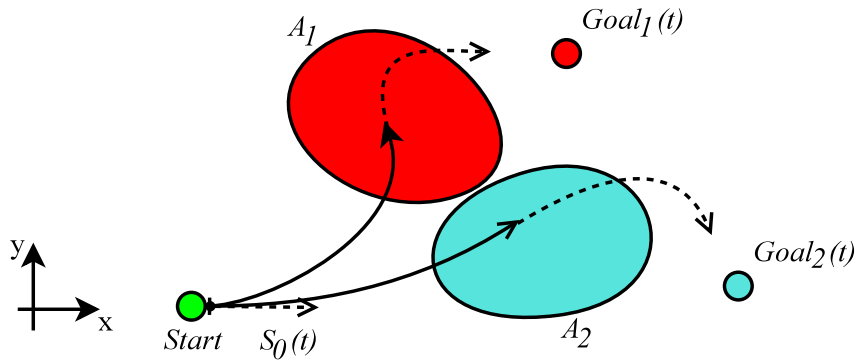

FIGURE 5.3 – Mécanisme de propagation permettant le guidage local de DKP : lorsqu'il propage des morceaux de trajectoire dans une partie A_k de l'environnement, les morceaux sont optimisés en direction de l'objectif associé à cette surface A_k.

5.2.3 Guidage pour la sélection

Selon la même définition que le guidage pour le niveau au niveau de propagation, le guidage pour le niveau de propagation est défini comme le tuple $(A_{Goal,path_{Goal}}(t), Goal)$ qui associe : une surface $A_{Goal,path_{Goal}}(t)$, toujours représentée selon l'approche par géométrie constructive de surface, pouvant être déplacée le long de la trajectoire $path_{Goal}(t)$ à un objectif $Goal$ associé à la surface $A_{Goal,path_{Goal}}(t)$.

Interprétation

Le niveau de propagation dans DKP étend un arbre d'exploration dans l'environnement sur lequel ce guidage pourra agir. Lorsque le point final du morceau de trajectoire $p_{k_0 \ldots k_n}(t)$ développé par le niveau de propagation de DKP est contenu dans la surface $A_{Goal,path_{Goal}}(t)$, les scores dans le niveau de sélection seront propagés selon l'objectif $Goal$ associé à cette surface. Autrement dit, ce mécanisme, illustré par la Figure 5.4, modifie localement l'heuristique de DKP. Un ensemble de guidages pour le niveau de sélection est noté $SGuide$. Ainsi, il est possible de différencier à l'usage les objectifs dans les couches de DKP. À terme, même si cette fonctionnalité n'est pas encore utilisée dans ce chapitre, je veux différencier la fonction à optimiser au niveau propagation de celle au niveau de sélection. L'idée est par exemple de générer des morceaux dans la propagation de manière à ce que les échantillons restent au milieu d'un couloir lorsqu'on planifie des trajectoires dans un bâtiment. Et c'est au niveau de sélection de s'assurer que la propagation tende finalement vers l'objectif principal (parcourir le couloir). À l'usage, seul un objectif global est défini à ce niveau pour conserver les bonnes propriétés de fonctionnement du niveau de sélection (qui fonctionne à la manière d'un A*).

5.2.4 Propriétés temporelles

Les propriétés temporelles qui définissent la génération de la diversité en morceaux de trajectoire dans le niveau de propagation sont :

- T_{min}, la durée minimale d'un échantillon ;
- T_{max}, la durée maximale d'un échantillon ;
- $Sample_{step}$, l'intervalle de temps qui permet de produire des morceaux dont les durées sont successivement $T_{min}, T_{min} + Sample_{step}, \ldots, T_{max}$.

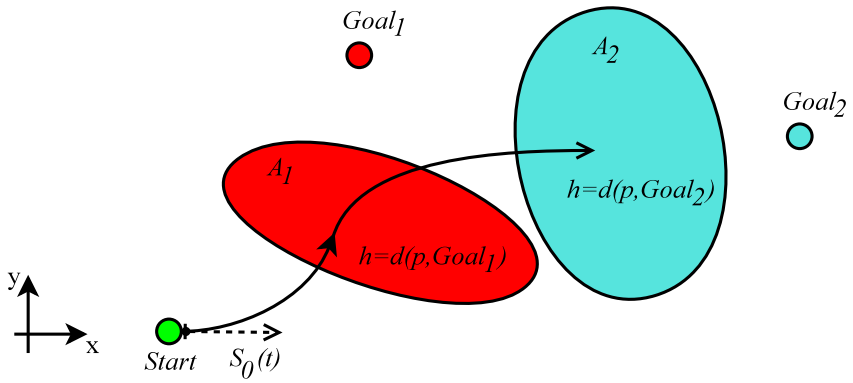

FIGURE 5.4 – Mécanisme de sélection permettant l'optimisation du choix des morceaux à prolonger : lorsque DKP propage des morceaux de trajectoire dans une partie A de l'environnement, les morceaux qui occupent cette surface propagent des scores qui sont fonction de la distance entre la fin du morceau et l'objectif associé à A.

Le pas de discrétisation *step* définit le pas de vérification des contraintes lors de la construction de l'espace des configurations dans le niveau de propagation. Ainsi, le tuple $T_{prop} = (T_{min}, T_{max}, Sample_{step}, step)$ décrit les propriétés temporelles héritées par les morceaux lors du grossissement de l'arbre d'exploration. Dans le mode backtrack utilisé dans les exemples de ce chapitre, un seul morceau est généré à chaque opération de propagation. Pour cela, on bloque les horizons des échantillons par les relations $Sample_{step} = 0$ et $T_{min} = T_{max}$. Le temps de discrétisation pour l'évaluation des contraintes est fixé à $step = \frac{T}{10}$ pour les exemples de ce chapitre (donc 10 instants de vérification du respect des contraintes lors de l'étape de propagation, quelle que soit la durée des morceaux générés).

5.2.5 Paramètres de pilotage

Notons $sp_{l_0 \ldots l_n}$ un ensemble de paramètres de pilotage. sp_{l_0} est la racine de l'arbre de comportements. L'ensemble de paramètres de pilotage $sp_{l_0 \ldots l_n}$ est le tuple $(Area_{sp_{l_0 \ldots l_n}}, CS_{sp_{l_0 \ldots l_n}}, PGuide_{sp_{l_0 \ldots l_n}}, SGuide_{sp_{l_0 \ldots l_n}}, T_{prop,sp_{l_0 \ldots l_n}})$ qui associe une surface $Area_{sp_{l_0 \ldots l_n}}$ (éventuellement translatée au cours du temps le long d'un chemin), qui est une partie de l'espace de travail $\mathbb{W}$ à des paramètres de pilotage :

- un ensemble de contraintes $CS_{sp_{l_0 \ldots l_n}}$;
- un ensemble de guidages pour le niveau de propagation $PGuide_{sp_{l_0 \ldots l_n}}$;
- un ensemble de guidages pour le niveau de sélection $SGuide_{sp_{l_0 \ldots l_n}}$;
- les propriétés temporelles $T_{prop,sp_{l_0 \ldots l_n}}$;

L'ensemble des associations de paramètres de pilotage avec des surfaces est noté SP. A_{SP} est l'ensemble des surfaces qui sont associées à des ensembles de paramètres de pilotage.

5.3 Des arbres à la croissance simultanée

L'arbre complet de comportements de pilotage est créé en utilisant une instanciation automatique de patrons d'arbre. Ces patrons constituent une librairie de comportements de pilotage spécifiques à l'application désirée. Pour décrire les comportements de pilotage, j'utilise le formalisme TÆMS [Horling *et al.*, 1999]. Deux types d'opérateurs sont employés pour

l'évaluation des comportements de pilotage : q_seq_sum est utilisé quand toutes les sous-tâches ont besoin d'être achevées dans l'ordre avant de pouvoir fournir à la super-tâche le niveau de qualité. Dans ce cas, la super-tâche va obtenir la qualité combinée de toutes ses sous-tâches. L'opérateur q_max fonctionne de manière équivalente à l'opérateur OU. En l'occurrence, la super-tâche obtient une qualité qui est le maximum de qualité de toutes les sous-tâches qui lui sont attachées. Avec ce formalisme, je peux rapidement mettre en place des politiques d'évaluation des trajectoires candidates. En particulier, si le critère d'évaluation est la distance parcourue, alors DKP fournit directement l'information grâce aux données diffusées dans l'arbre de trajectoires.

5.3.1 Une croissance commune de l'arbre de pilotage et de l'arbre d'exploration

Pour influencer le comportement exploratoire de DKP, je propose un système basé sur les surfaces pour appliquer les comportements de pilotage : chaque nœud de l'arbre TÆMS contient un ensemble d'associations de paramètres de pilotage avec des surfaces contenues dans A_{SP}. Concrètement, le but est de faire interagir l'arbre TÆMS avec l'arbre de trajectoires directement en cours de construction par DKP en agissant sur tous les paramètres déclarés précédemment et rassemblés dans les paramètres de pilotage $sp_{l_0 \dots l_n}$.

Diffusion de paramètres de pilotage

Lorsque le point final du morceau $p_{k_0 \dots k_n}(t)$ sélectionné dans l'arbre d'exploration par le niveau de sélection de DKP est contenu dans la surface $Area_{sp_{l_0 \dots l_n}}$, l'ensemble de paramètres de pilotage $sp_{l_0 \dots l_n}$ est appliqué pour propager les successeurs du morceau $p_{k_0 \dots k_n}(t)$. Cet ensemble de paramètres de pilotage $sp_{l_0 \dots l_n}$ surcharge alors le précédent ensemble de paramètres de pilotage $sp_{l_0 \dots l_{n-1} l_n}$ qui guidait jusque là la croissance de la partie de l'arbre d'exploration contenant le morceau $p_{k_0 \dots k_n}(t)$: la racine de ce sous-arbre étant le morceau ayant lui-même déclenché l'utilisation de l'ensemble de paramètres de pilotage $sp_{l_0 \dots l_{n-1} l_n}$.

Interprétation Cela signifie que les autre morceaux de l'arbres qui ne sont pas concernés par ce déclenchement vont continuer leur croissance avec leurs paramètres de pilotage (dont une partie avec l'ensemble de paramètres de pilotage $sp_{l_0 \dots l_{n-1} l_n}$). Quand l'ensemble de paramètres de pilotage $sp_{l_0 \dots l_n}$ est appliqué, les morceaux suivants qui possèdent parmi ses pères le morceau $p_{k_0 \dots k_n}(t)$ vont hériter de cet ensemble de paramètres de pilotage jusqu'à ce qu'un nouvel ensemble de paramètres de pilotage soit appliqué sur l'un des morceaux suivants et ainsi de suite. Ce processus est illustré par la Figure 5.5.

5.3.2 Croissance de l'arbre de pilotage

Quand une seule surface appartenant à A_{SP} peut être utilisée pour désigner l'ensemble successeur de paramètres de pilotage $sp_{l_0 \dots l_n}$, on crée un nouveau nœud. L'évaluation de ce nœud correspond au nœud q_seq_sum de TÆMS. Il est ajouté dans l'arbre de pilotage avec deux fils : le premier contient l'ensemble de paramètres de pilotage $sp_{l_0 \dots l_n}$ et le second contient l'ensemble de paramètres de pilotage $sp_{l_0 \dots l_n l_{n+1}}$. Pour simplifier le discours, ce nœud de pilotage est appelé plus simplement nœud q_seq_sum dans la suite du manuscrit. Si un seul ensemble de paramètres de pilotage $sp_{l_0 \dots l_n l_{n+1} l_{n+2}}$ associé à une surface $Area_{sp_{l_0 \dots l_n l_{n+1} l_{n+2}}}$ peut être désigné comme successeur de l'ensemble de paramètres de pilotage $sp_{l_0 \dots l_n l_{n+1}}$, l'ensemble de paramètres

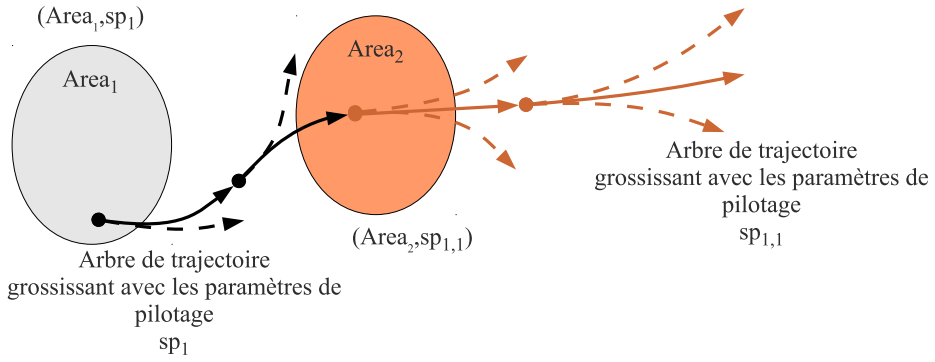

FIGURE 5.5 – Développement de l'arbre d'exploration. Lorsqu'un échantillon de trajectoire rencontre une surface à laquelle sont associées des nouveaux paramètres de pilotage, ceci sont diffusés dans les échantillons suivants.

de pilotage $sp_{l_0 \ldots l_n l_{n+1} l_{n+2}}$ est ajouté au nœud q_seq_sum et ainsi de suite. Ce processus est illustré par la Figure 5.6.

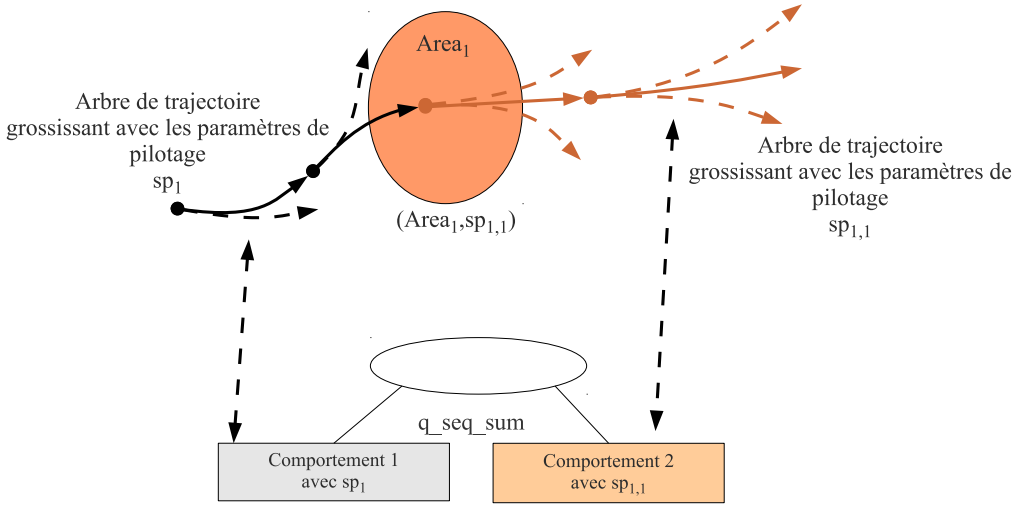

FIGURE 5.6 – Croissance d'une séquence dans l'arbre de pilotage lorsqu'un paramètre de pilotage est le seul successeur d'un paramètre de pilotage précédent.

Dans le cas où n ensembles de paramètres de pilotage peuvent être déclenchés comme successeurs de l'ensemble de paramètres de pilotage $sp_{l_0 \ldots l_n}$ associé à la surface $Area_{sp_{l_0 \ldots l_n}}$, on crée un nœud, dont l'évaluation correspond à un nœud q_max dans le langage TÆMS, ajouté dans l'arbre de pilotage avec n enfants, chacun contenant un des ensembles de paramètres de pilotage. Pour simplifier le discours, ce nœud de pilotage est appelé plus simplement nœud q_max dans la suite du manuscrit. Le nœud q_max est ajouté comme enfant du nœud q_seq_sum contenant l'ensemble de paramètres de pilotage $sp_{l_0 \ldots l_n}$ associé à une surface $Area_{sp_{l_0 \ldots l_n}}$.

Ce cas est illustré par les Figure 5.7 et Figure 5.8. L'évolution de l'arbre de pilotage est alors vue comme la construction d'une première partie d'un arbre de trajectoires associées au comportement $sp_{l_0 \ldots l_n}$ puis de n sous-arbres contenant des échantillons de trajectoire soumis à

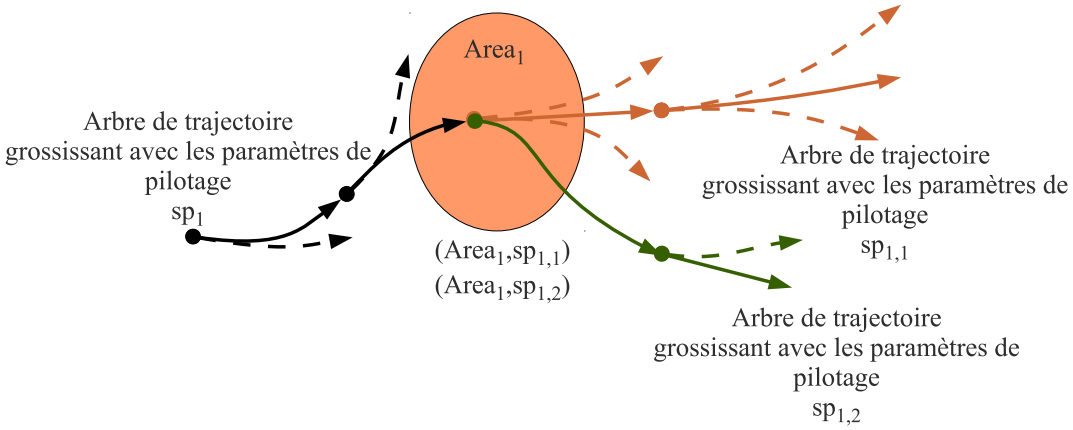

FIGURE 5.7 – Croissance d'une alternative dans l'arbre d'exploration lorsque plusieurs paramètres de pilotage sont associés à une même surface de déclenchement.

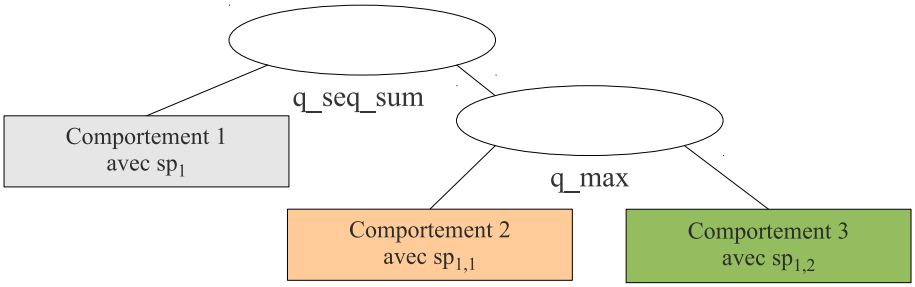

FIGURE 5.8 – Croissance d'une alternative dans l'arbre de pilotage lorsque plusieurs paramètres de pilotage sont associés à une même surface de déclenchement.

l'un des l_{n+1} comportements $sp_{l_0 \ldots l_n 1}, \ldots, sp_{l_0 \ldots l_n l_{n+1}}$.

5.3.3 Croissance de l'arbre de trajectoires

Lorsque le point final du morceau sélectionné $p_{k_0 \ldots k_n}$ est contenu dans la surface $Area_{sp_{l_0 \ldots l_n}}$, les morceaux de l'arbre d'exploration prolongés par la suite sont créés en appliquant les paramètres de pilotage contenus dans l'ensemble de paramètres de pilotage $sp_{l_0 \ldots l_n}$, c'est-à-dire qu'il respecte l'ensemble de contraintes $CS_{sp_{l_0 \ldots l_n},k}$.

Cela est effectué en appliquant à DKP un ensemble de guidages pour le niveau de propagation $PGuide_{sp_{l_0 \ldots l_n},k}$, un ensemble de guidages pour le niveau de sélection $SGuide_{sp_{l_0 \ldots l_n},k}$ et les propriétés temporelles $T_{prop,sp_{l_0 \ldots l_n}}$ induisant la diversité en morceaux générés par le niveau de propagation pour le niveau de sélection.

L'arbre d'exploration de DKP interagit fortement avec l'arbre de comportements de pilotage grâce au système de déclenchement par l'intermédiaire des surfaces explorées. Lorsque le point final du morceau sélectionné par le niveau de propagation est contenu dans une surface $Area_{sp_{l_0 \ldots l_n}}(t)$ associée à n paramètres de pilotage, n embranchements sont créés dans l'arbre d'exploration, chacun d'entre eux étant indépendants l'un de l'autre. On peut ainsi voir chacun de ces embranchements comme un nouveau sous-arbre d'exploration généré par DKP avec le morceau $p_{k_0 \ldots k_n}$ comme racine. Chacun de ces embranchements applique les paramètres de

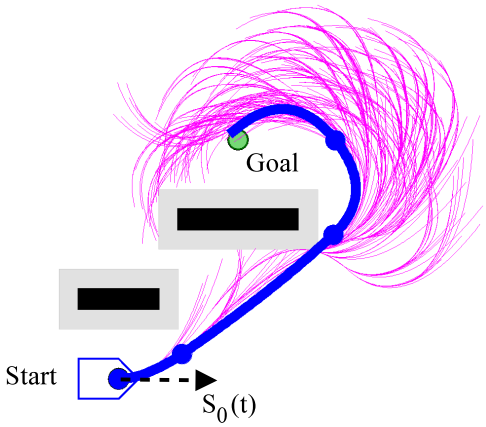

FIGURE 5.9 – Rappel de la trajectoire obtenue par DKP sur le problème de franchissement avec le respect de contraintes de vitesse et d'accélération linéaire.

pilotage fournis par les ensembles de paramètres de pilotage. Notons toutefois que certains morceaux peuvent se recouvrir dans l'espace de travail. Ce phénomène est naturellement fréquent à proximité d'un nœud $p_{k_0 \ldots k_n}$ commun à plusieurs embranchements.

5.4 Guidage par roadmap

Dans les sections précédentes, on montre comment on peut piloter DKP avec des comportements de pilotage. Le point principal est que DKP fournit un ensemble d'entrées qui permettent d'exprimer des comportements symboliques sous la forme de paramètres pour le contrôle de la planification de trajectoire. Par exemple, des bornes sur la vitesse linéaire dans un comportement sont ensuite traduites en contraintes dans le planificateur de trajectoire.

Le premier exemple reprend un environnement très simple en apparence (vu dans la Section 4.3 du Chapitre 4), composé de deux obstacles, mais pour lequel la recherche d'une solution optimale est plus difficile si les caractéristiques physiques des robots sont prises en compte. Il montre comment un guidage par des points de passage permet d'explorer efficacement des alternatives de cheminement dans un environnement. Les temps de calculs ont été obtenus dans ce chapitre par un PC double cœur 2.53 GHz doté d'une architecture 64 bits avec 4 Go de RAM.

5.4.1 Situation

Problème

Cet exemple utilise le même environnement que proposé dans la figure 5.9. Il s'agit d'un environnement comprenant 2 obstacles rectangulaires (0.4×0.1m et 0.6×0.1m) placés entre une situation initiale *Start* et un objectif *Goal*. Ce problème en l'apparence simple est rendu plus difficile par la présence de contraintes cinématiques : bornes sur les vitesses linéaire et angulaire. La solution la plus courte, le passage entre les deux obstacles, n'est pas forcément la plus efficace avec la prise en compte des contraintes sus-citées.

Roadmap et comportements de pilotage

À partir de ce cas, je vise à illustrer l'utilisation d'un arbre de comportements de pilotage pour le guidage d'un algorithme de planification de trajectoire basé sur une roadmap. Une roadmap permet de mettre en avant les connexions reliant différentes parties d'un environnement. Elle est fabriquée à partir de chemins pré-calculés permettant à l'entité planifiant ses mouvements d'explorer l'environnement tout en évitant les obstacles. Comme je l'explique tout au long de ce manuscrit, ces chemins ne sont là que pour guider l'exploration vers des possibilités de contournements. Par défaut, si le niveau qui fabrique ces chemins ne possède pas une représentation et un mécanisme de vérification de toutes les contraintes, ces chemins n'assurent pas par défaut le respect des contraintes. Ainsi, les comportements de pilotage permettent de décrire toutes les alternatives d'exploration. Ensuite, les interactions entre l'arbre de pilotage et l'arbre de comportements permettent de construire des solutions vérifiant les contraintes du problème tout en bénéficiant d'un guidage efficace. Dans le cas le plus défavorable, les solutions correspondant à des alternatives d'exploration de l'environnement dans des zones inatteignables seront abandonnées par l'arbre de DKP. Au final, il reste les solutions valides correspondant aux alternatives faisables. Il revient à la couche de raisonnement supérieure de déterminer s'il faut proposer des alternatives supplémentaires ; lancer une recherche brute ou considérer qu'il n'y a pas de solution.

5.4.2 Diagramme de Voronoï

Parmi toutes les approches existantes dans la littérature, j'ai opté pour une exploration à base de diagrammes de Voronoï, présentée dans le Chapitre 2 : ceux-ci ont la propriété de pouvoir maintenir les robots à distance des obstacles. En effet, les chemins composant la roadmap sont équidistants des frontières des obstacles et du bord de l'environnement exploré. L'environnement proposé dans la figure 5.9 est centré dans un carré de 2×2m. Cette distance maximisée aux obstacles permet donc de fournir naturellement une bonne roadmap servant au guidage du robot. De surcroît, cela permet également au planificateur de trajectoire de dévier du chemin pour aller dans l'espace libre.

5.4.3 Construction des roadmaps

Un exemple de roadmap généré à partir d'un diagramme de Voronoï est fourni en figure 5.11(a) (les lignes grasses en gris clair). Cette construction est effectuée en deux étapes illustrées par la Figure 5.10. Premièrement, les contours des obstacles et les limites de l'environnement exploré sont discrétisés afin d'obtenir un ensemble de points à partir desquelles le diagramme de Voronoï est généré. Deuxièmement, le diagramme de Voronoï est simplifié en supprimant les segments qui coupent un obstacle ou qui vont aux bords de l'environnement (celui-ci est fermé).

Détermination des points de passage Afin de guider le robot le long des points de passage, il faut retrouver le point de passage W le plus proche dans la roadmap qui puisse être atteint. De plus, il faut le retrouver depuis n'importe quelle position M qui pourrait être prise par les morceaux de trajectoire générés par DKP. On peut le faire efficacement en construisant ce qui est appelé un méta-diagramme de Voronoï : un diagramme de Voronoï avec des surfaces englobant les points de passage. Dans ce méta-diagramme, les points de passage sont alors vus comme des obstacles pour la génération du méta-diagramme. Cela permet d'obtenir un pavage complet de l'environnement. On obtient donc la décomposition en polygones observable sur la figure 5.11(b) (avec des lignes noires grasses). Dans ce méta-diagramme, chaque polygone P contient

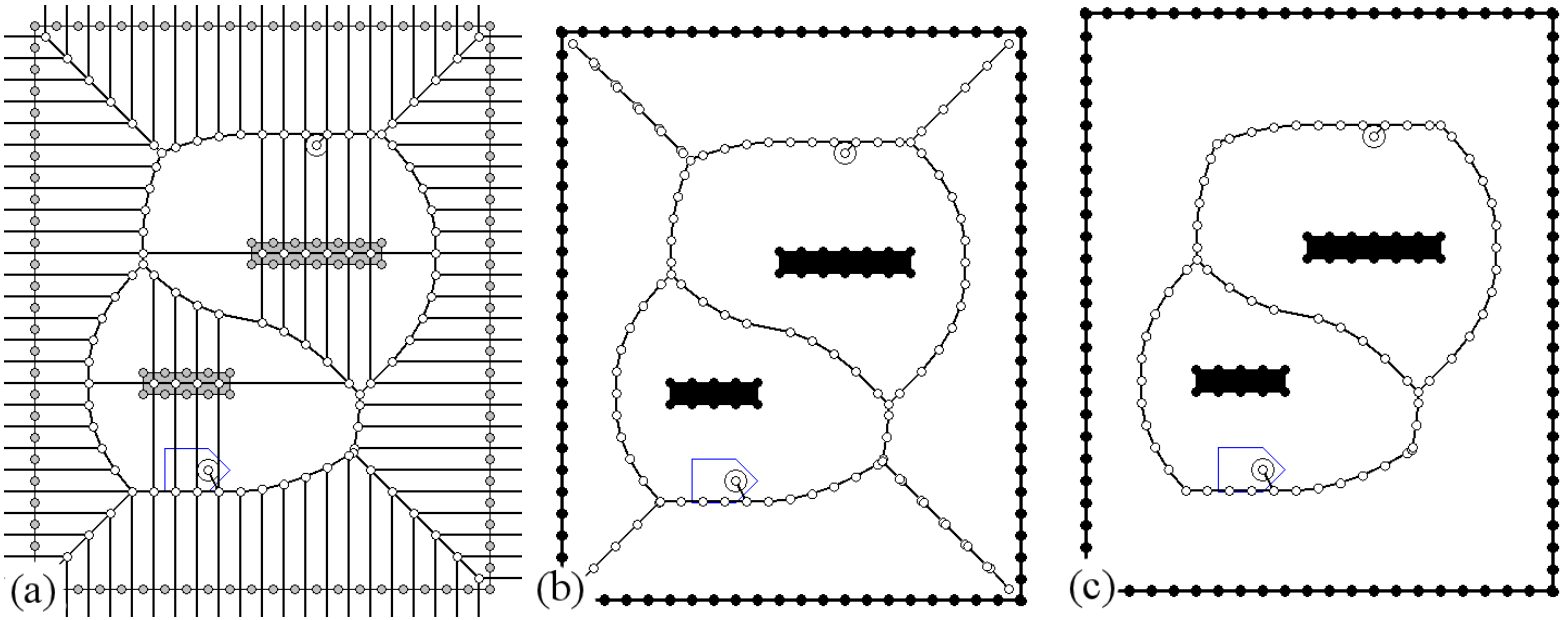

FIGURE 5.10 – Étapes de construction du diagramme de Voronoï sur l'environnement proposé dans cet exemple : (a) un diagramme de Voronoï est construit autour de la discrétisation des obstacles et des bords ; (b) les segments coupant un obstacle ou le bord sont retirés ; (c) les segments menant à une impasse sont retirés

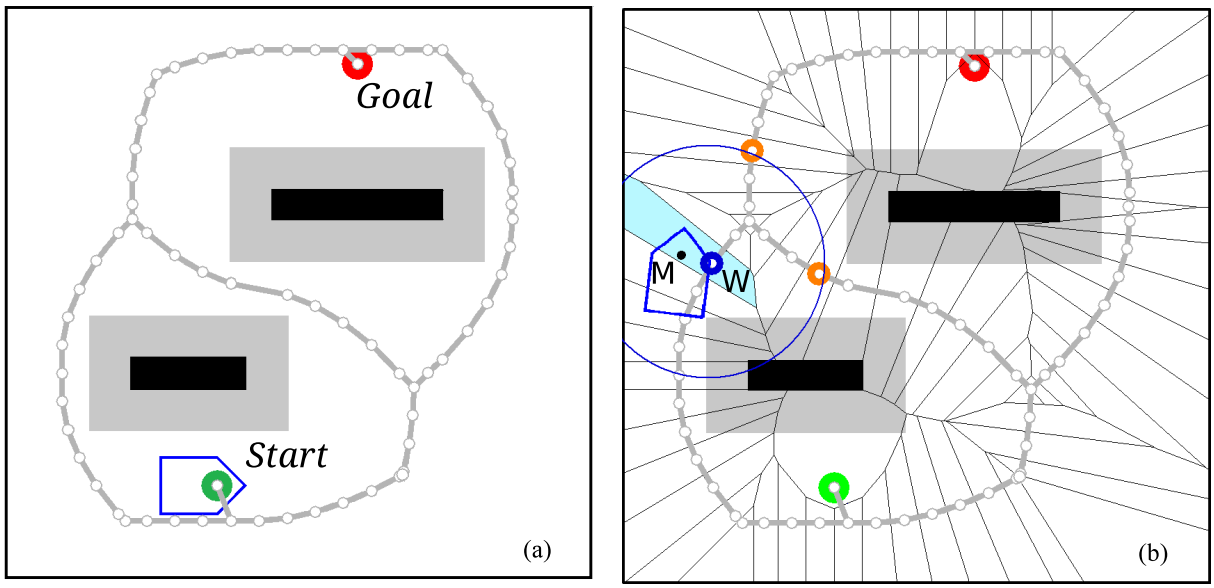

FIGURE 5.11 – (a) Une roadmap issue de la simplification du diagramme de Voronoï (les lignes grises épaisses). (b) Méta-diagramme de Voronoï (les lignes noires épaisses) associant une unique surface à chaque point de passage de la roadmap.

un unique point de passage W de la roadmap. La surface délimitée par le polygone P possède la propriété de contenir tous les points de l'environnement qui sont plus proches du point de passage W (contenu par P) que des autres points de passage. La Figure 5.11(b) montre que, depuis une position du robot notée M, il est possible retrouver le polygone conteneur (colorié en magenta) et le point de passage W auquel le polygone est associé (le point bleu).

5.4.4 Traduction de la roadmap en comportements de pilotage

En s'appuyant sur le graphe proposé précédemment, on peut construire un arbre de pilotage qui fait le pont entre le planificateur de trajectoire et une roadmap constituée de points de passage. L'arbre de pilotage contient toutes les stratégies possibles permettant le contournement des obstacles. Une stratégie de contournement peut être modélisée dans l'arbre de pilotage par un patron d'embranchement utilisant le formalisme TÆMS. Ce patron est illustré par la Figure 5.12 et fonctionne selon le principe suivant : le nœud q_seq_sum exprime le fait que la trajectoire n'a pas encore atteint l'objectif et doit être poursuivie. Le nœud N_0 correspond à la partie déjà achevée et donc pour laquelle on dispose d'une évaluation (longueur de la trajectoire jusque ce nœud, durée de cette trajectoire...). Le nœud q_max permet de déterminer le meilleur chemin alternatif (c'est-à-dire l'alternative offrant la meilleure qualité). Ce choix se fait entre les sous-arbres de pilotage N_1, ..., N_k parmi lesquels se trouve la meilleure stratégie de franchissement parmi les k disponibles.

Passage de la roadmap à un arbre d'objectifs

L'arbre de pilotage est constitué de plusieurs patrons d'embranchement. Il est progressivement construit grâce à une recherche en profondeur dans la roadmap, à partir du point de départ *Start*. Chaque fois qu'une zone de passage W de la roadmap est visitée, le

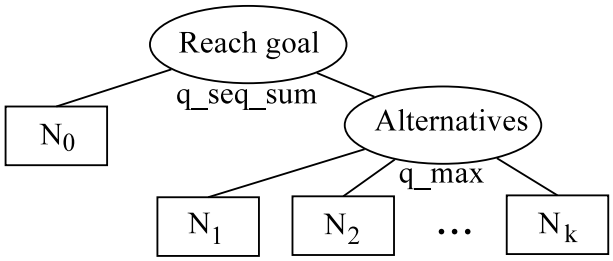

FIGURE 5.12 – Le patron d'arbre de pilotage servant au contournement des obstacles.

point de passage associé est projeté à une distance d sur la roadmap. Cette projection consiste à obtenir sur la roadmap les k points de passage W_i^p, $(i \in \{1, k\})$, situés au moins à une distance d de W.

Cette manière de procéder est illustrée par la Figure 5.13 où les points de passage projetés sont signalés par un point de couleur orange. Notons que si un point de passage projeté a déjà été visité, alors il est ignoré. Cela permet d'éviter les retours en arrière et les boucles. La figure permet d'observer cela : les points de passage visés par l'arbre d'exploration sont représentés avec la surface (coloriée en cyan) à laquelle ces points sont associés. Par exemple, pour le point de passage W de la Figure 5.13(a), il y a théoriquement deux points de passage possibles : W_1^p (au dessus de W) et W_2^p (en dessous de W), mais W_2^p n'est pas conservé dans ce cas.

Passage à l'arbre de pilotage

Lors de la création d'un ensemble de paramètres de pilotage SP_{N_i} associé au nœud de pilotage N_i, deux cas sont possibles selon les points de passage empruntables. Dans le premier cas, si le processus de projection amène un unique point de passage projeté ($k = 1$), la paire (P, W_1^p) est ajoutée à l'ensemble de guidages par objectif pour le niveau de propagation $PGuide_{sp,k}$. Dans le second cas, si plusieurs points de passage sont possibles ($k > 1$), un embranchement est détecté dans la roadmap : la surface de projection associée est identifiée comme membre de la surface de déclenchement d'embranchement $A_{sp,k}$. Cette surface de déclenchement d'embranchement est associée à chacun des k nouveaux ensembles de paramètres de pilotage $SP_{N_{i+k}}$. Dans ce cas de figure, un nouveau patron d'embranchement est ajouté à l'arbre de pilotage associant les nouveaux paramètres de pilotage ainsi créés aux nœuds TÆMS et générant ainsi des alternatives à évaluer.

La Figure 5.16 montre l'arbre de pilotage associé à l'environnement présenté dans la Figure 5.13. L'arbre de pilotage permet à l'arbre de trajectoires de la Figure 5.17 d'évaluer les quatre comportements de pilotage possibles, en planifiant les quatre trajectoires associées (dessinées en bleu). La distance d utilisée pour projeter les points de passage sur la roadmap impacte directement la douceur des trajectoires, permettant des virages doux là où la roadmap présente des coins pointus. Les trajectoires sont garanties comme étant faisables par le robot. En particulier, elles garantissent le respect des limites cinématiques en tout point de la trajectoire, ce que ne garantit pas initialement une roadmap basée sur un diagramme Voronoï. Une fois toutes les alternatives connues, la meilleure peut être choisie, en accord avec un critère dépendant de l'application, par exemple minimiser la courbure, la longueur, la durée ou la dépense d'énergie.

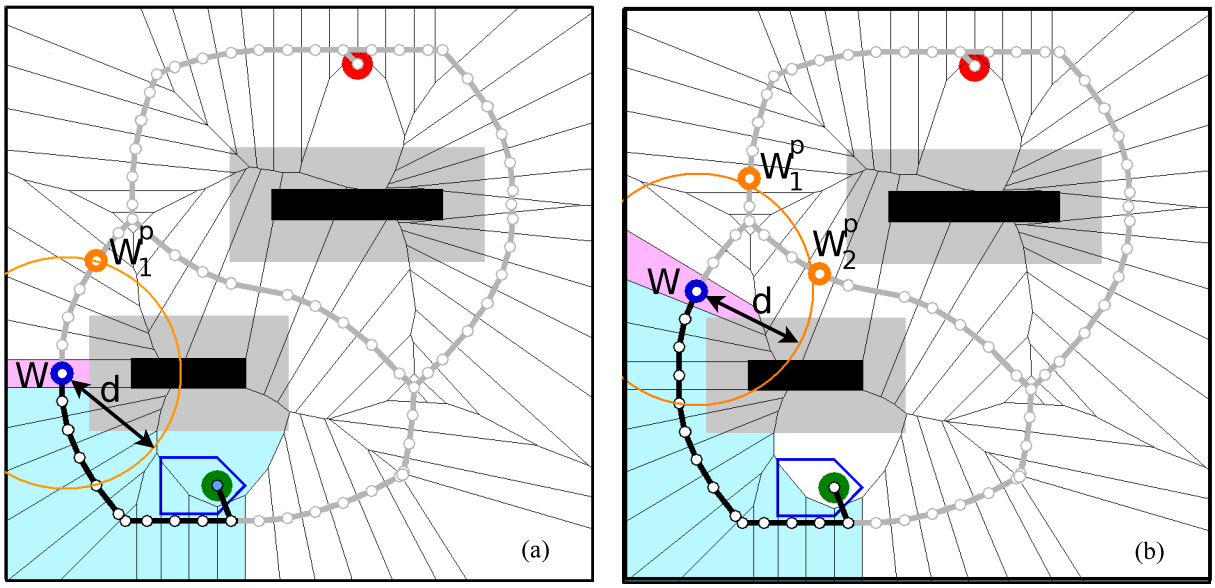

FIGURE 5.13 – Point de passage W de la roadmap choisi en fonction de la position du robot M et les points de passage projetés W_i^p à une distance d. (a) un point de passage projeté, sur une branche, avant un embranchement, (b) deux points de passage projetés, sur deux branches, après un embranchement.

5.4.5 Résultats

Paramètres

Pour cet exemple, le robot est contenu dans un rectangle de 0.2×0.3m. Les contraintes cinématiques prennent les valeurs suivantes : la vitesse linéaire est limitée aux valeurs de l'intervalle $[0.1, 0.2]$m/s et l'accélération linéaire est limitée aux valeurs de l'intervalle $[0, 0.1]$m^2/s. La durée des morceaux est limitée à 0.5s. La distance minimale entre le robot et les obstacles est fixée à 0.2m. Si on compare l'arbre de trajectoires issu d'une planification avec DKP utilisé seul de la Figure 5.9 avec l'une de la Figure 5.17, on peut voir que la forme des trajectoires est bien meilleure que celle fournies par DKP. L'arbre contient également bien moins de branches, ce qui réduit le temps de calcul, même avec une exploration plus profonde de l'environnement.

Solutions

Le temps complet de planification est de 1.25s. Les trajectoires obtenues ont les caractéristiques suivantes :

- le contournement par la droite fait une longueur de 2.45m et une durée de 13.0s, obtenu en 0.84s ;
- le contournement par la gauche fait une longueur de 3.04m et une durée de 17.0s, obtenu en 0.94s ;
- le passage droit par le centre fait une longueur de 3.27m et une durée de 17.0s, obtenu en 0.98s ;
- le passage gauche par le centre fait une longueur de 4.54m et une durée de 25.0s, obtenu en 1.25s.

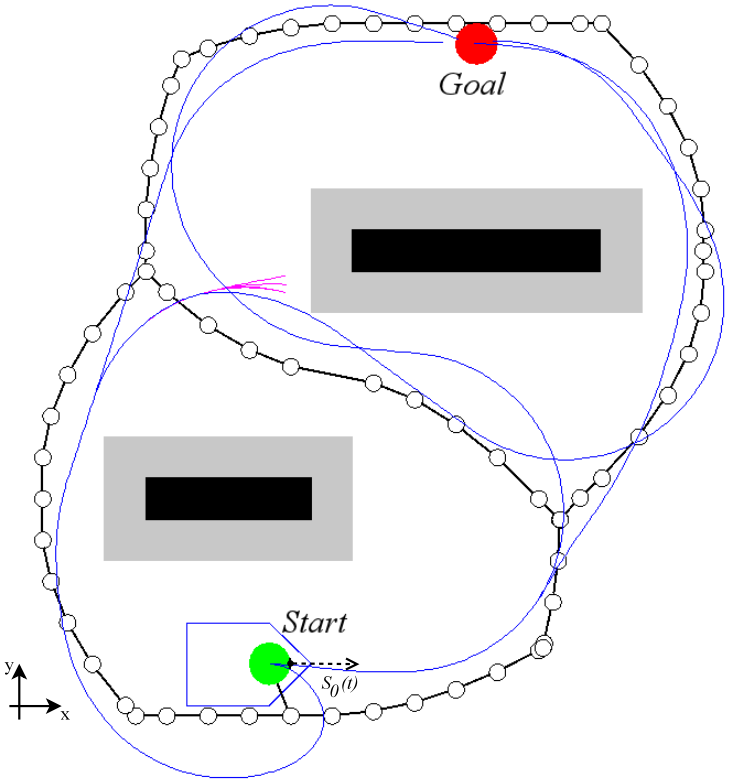

FIGURE 5.14 – Les quatre solutions obtenues avec le guidage par comportement de pilotage basé sur une roadmap.

Ces résultats sont à comparer à ceux de DKP dans une situation similaire en mode backtrack sans guidage (Figure 5.15) : le contournement par la droite fait certes une longueur de 2.23m et une durée de 12.0s mais ce résultat requiert 6s de calculs. De plus, la planification de trajectoire guidée par les comportements permet de bénéficier pleinement des architectures multi-processeurs, chaque comportement nécessitant une planification indépendante des autres.

5.4.6 Application à l'obstacle en U

Reprenons la situation proposée dans la Section 4.4 : un obstacle en U bouche la progression d'un arbre de pilotage lorsque la planification de trajectoire est guidée par la distance à l'objectif. Les approches de planification de trajectoire par échantillons explorent complètement l'environnement : il n'y a donc pas de minimum local dû à l'heuristique. Nous avons vu dans la Section 4.4 que le mode backtrack offre une solution en réinterprétant l'environnement : les obstacles virtuels bouchent progressivement le minimum local grâce à des obstacles virtuels.

La méthode de création de roadmap peut être employée pour explorer cet environnement. Prenon les paramètres suivants :

- la position initiale est $Start = (-80, 0)\text{cm}$;
- le vecteur vitesse initial est $\overrightarrow{S}(0) = (10, 0)\text{cm/s}$;
- une distance de sécurité autour du robot est fixée à $r_{robot} = 20\text{cm}$;
- nous posons les contraintes suivantes :

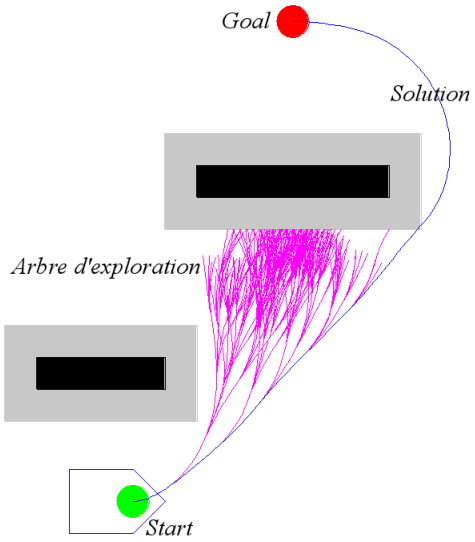

FIGURE 5.15 – Solution obtenue avec le mode backtrack de DKP sans guidage : DKP est bloqué par un minimum local qui l'empêche de trouver rapidement le franchissement de l'obstacle du haut.

- l'accélération linéaire bornée entre 0 et 10cm/s^2 ;
- la vitesse linéaire bornée entre 10 et 20cm/s ;
- un obstacle sous la forme d'un disque de rayon 1 centré en $Obs = (2, 0)$.

Le mode backtrack est utilisé avec le guidage par arbre de comportements.

Résultats

La détermination des points de passage génère des chemins dont l'un va au fond de l'obstacle en U (Figure 5.18(a)). Toutefois, après élimination des chemins ne menant pas au nœud final depuis le nœud de départ, seules restent les alternatives de contournement. Ainsi, deux solutions symétriques de contournement sont développées (Figure 5.18(b)). Bien entendu, cette solution n'est valable que s'il y a une connaissance complète de l'environnement.

5.4.7 Améliorations possibles

Le guidage par roadmap souffre d'un défaut mineur mais simple à corriger : le mode backtrack utilisé par DKP pose parfois problème pour converger vers l'objectif final. Dans ce cas particulier, la trajectoire a tendance à se « satelliser » autour de l'objectif, comme l'illustre la figure 5.19. C'est un défaut d'anticipation du freinage qui apparaît ici. Actuellement, mes solutions sont d'élargir la zone d'arrêt autour de l'objectif (dans le guidage de la roadmap, c'est un disque de rayon 0.1m centré sur l'objectif) et mieux, de faire basculer DKP en mode optimal à l'approche de l'objectif. Cette propriété supplémentaire de comportement de pilotage n'est pas exprimée afin de ne pas alourdir le formalisme présenté dans ce chapitre.

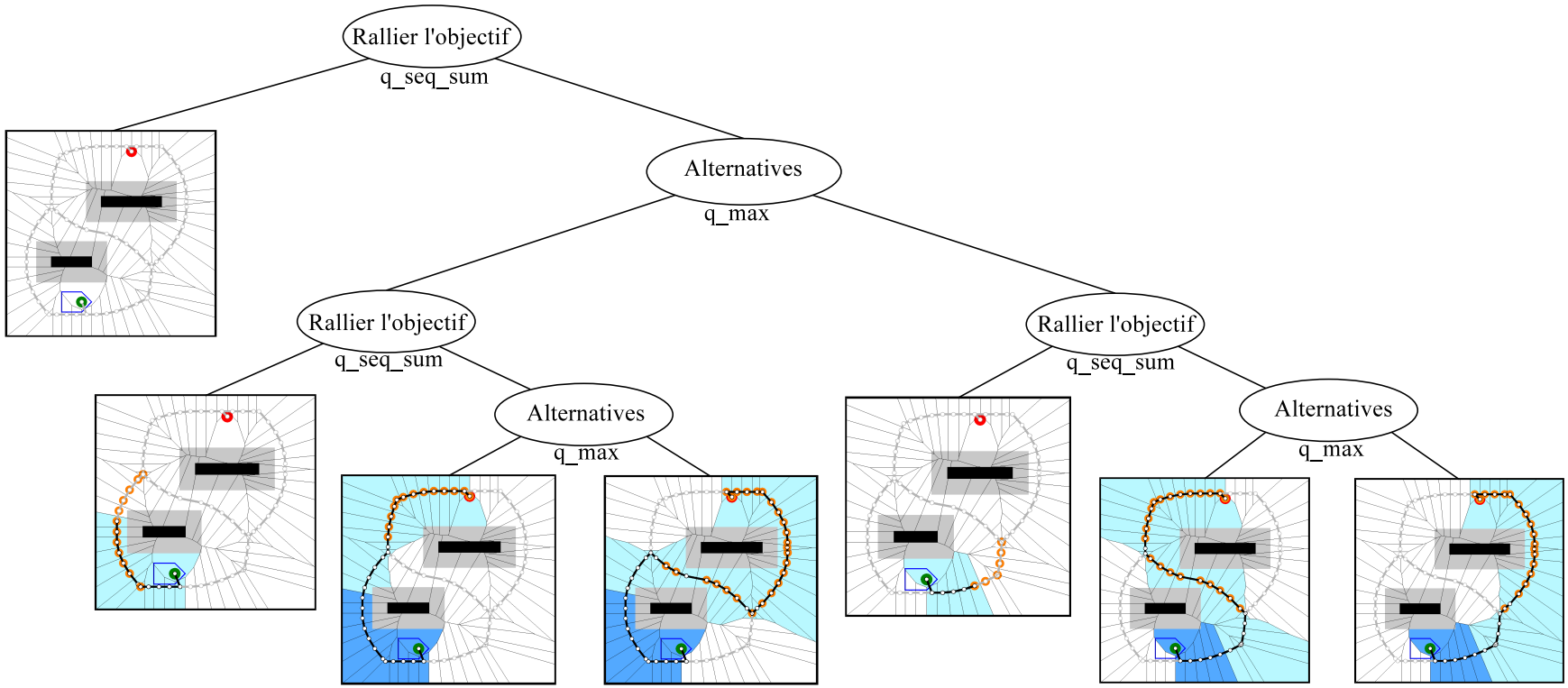

FIGURE 5.16 – L'arbre de pilotage pour le guidage par roadmap. Chaque feuille correspond à une surface, coloriée en cyan, dans laquelle il n'y a plus d'embranchements et où les objectifs intermédiaires sont dessinés en orange. Les polygones bleus foncés représentent les surfaces à ignorer car elles ont déjà été visitées dans les nœuds parents

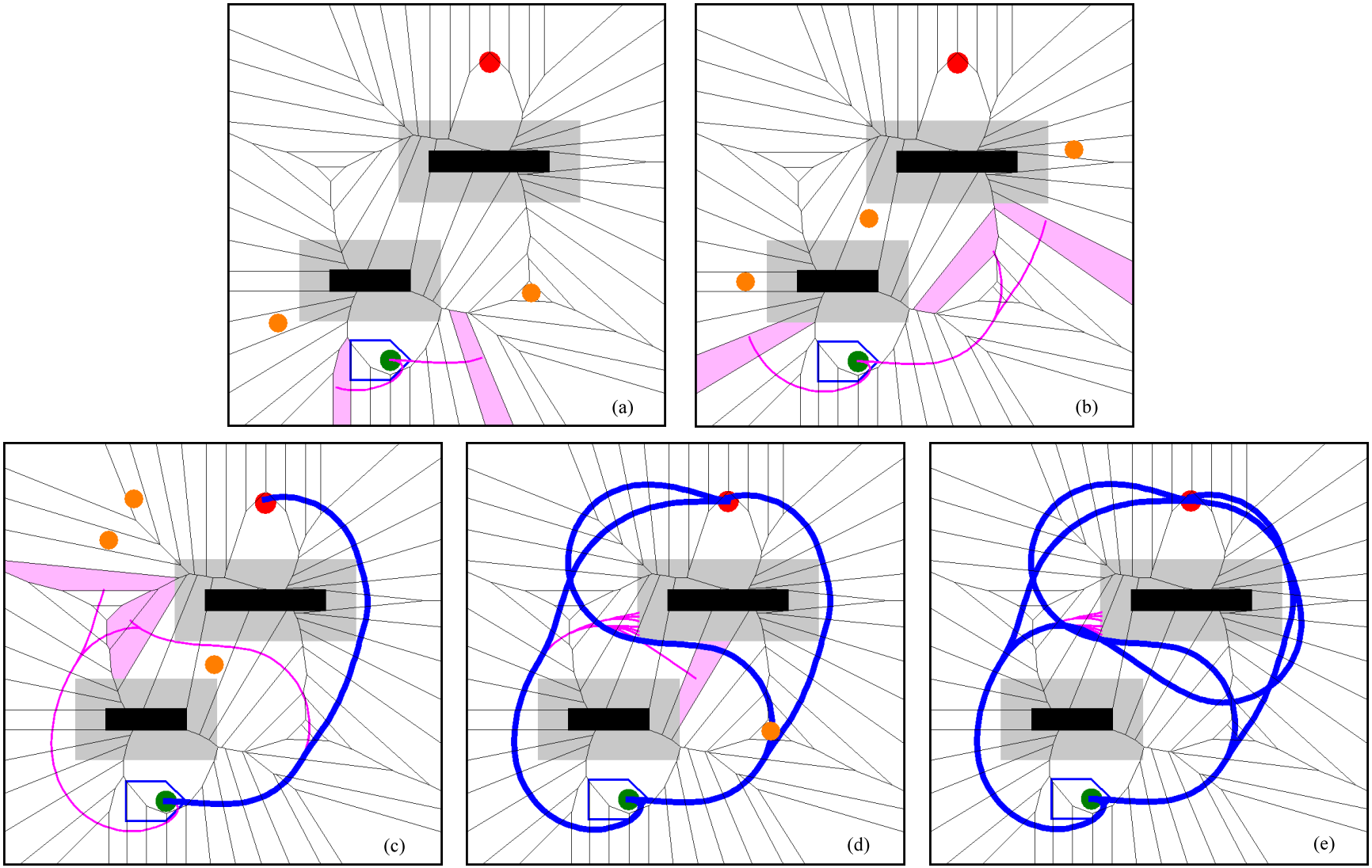

FIGURE 5.17 – L'arbre de trajectoires guidé par l'arbre de pilotage de la Figure 5.16. Les courbes bleues représentent les trajectoires alternatives qui atteignent l'objectif. Les courbes en magenta sont des échantillons de trajectoire en cours de construction ou abandonnées. Les polygones en rose clair correspondent aux surfaces où les trajectoires en construction poursuivent à cet instant leur progression. Les points oranges sont associés aux objectifs intermédiaires, fournis par l'arbre de trajectoires

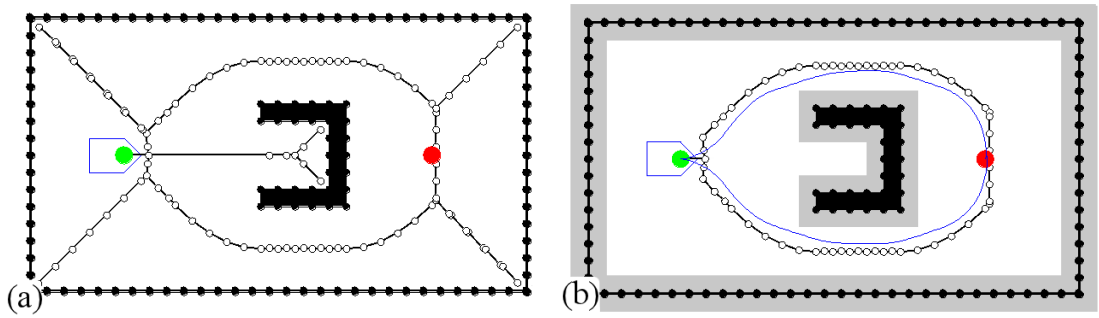

FIGURE 5.18 – Solutions obtenues en utilisant le mode backtrack de DKP et avec guidage sur un obstacle U. La roadmap est initialement constituée de trois parties : la première branche va vers le fond du U, les deux autres contournent l'obstacle. Toutefois, seules ces deux dernières servent au guidage et permettent le développement de deux solutions symétriques.

5.5 Exploration de l'ISEN

Sur le même principe que l'exemple précédent, il est tout à fait possible d'explorer des environnements plus complexes et plus structurés. C'est pourquoi j'ai envisagé de faire fonctionner mon planificateur de trajectoire sur une portion des plans de l'ISEN où se situe le laboratoire d'informatique (Figure 5.20).

Le guidage par distance à l'objectif de DKP a tendance à complètement se bloquer dans les minima locaux qui résultent des nombreux angles droits de l'environnement. Dans cette configuration, DKP seul est incapable d'effectuer le rebroussement nécessaire au passage de la porte du laboratoire. Un tel problème justifie pleinement l'ajout d'une planification de haut niveau que le guidage par roadmap peut offrir. Le couplage avec DKP permet de garantir le respect des contraintes le long de la trajectoire.

5.5.1 Recherche d'une route optimale

La création de la roadmap suit exactement les mêmes principes que l'exemple précédent. Premièrement, les contours des obstacles et les limites de l'environnement exploré sont discrétisés afin d'obtenir un ensemble de points à partir desquels est généré le diagramme de Voronoï. On considère comme obstacles toutes les parties grisées et les contours foncés présents sur la représentation du plan Figure 5.20. Deuxièmement, le diagramme de Voronoï est simplifié en supprimant les segments qui coupent un obstacle ou qui vont aux bords de l'environnement (celui-ci est fermé). Le résultat est représenté par la Figure 5.21.

A partir de ce diagramme, une recherche en profondeur est effectuée pour déterminer le chemin optimal en distance le long de la roadmap. Ce chemin optimal, qui emprunte le couloir en dessous du laboratoire, est affiché en bleu sur la Figure 5.21.

5.5.2 Guidage le long de la route

L'arbre de pilotage permettant le suivi de la route est constitué d'un unique ensemble de paramètres de pilotage *sp*. En utilisant le méta-diagramme généré à partir du diagramme de Voronoï, on retrouve chaque polygone P qui est associé à un unique point de passage W_1^p de la roadmap : il n'y a qu'un seul chemin généré. Chaque paire (P, W_1^p) est alors ajoutée à l'ensemble de guidages par objectif pour le niveau de propagation $PGuide_{sp,k}$. Cet ensemble est fourni à

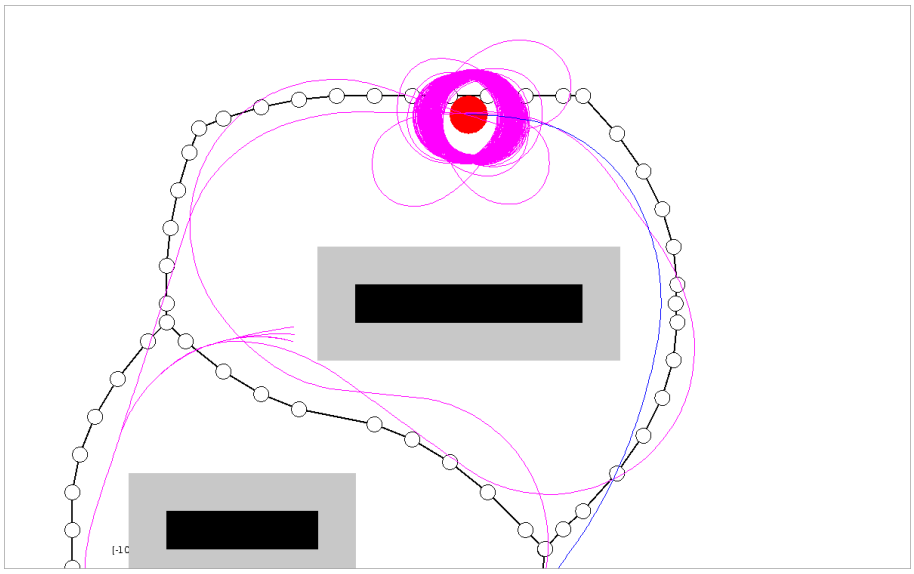

FIGURE 5.19 – Problèmes de convergence liés au surguidage du mode backtrack lors de l'atteinte d'un objectif, ici un disque de rayon 0.01m centré sur l'objectif.

DKP lors de la planification. Il serait possible de créer un nœud q_seq_sum qui contient n ensembles de paramètres de pilotage sp pour chaque paire (P, W_1^p).

5.5.3 Résultats

Pour cette simulation, le robot est contenu dans un rectangle de 0.2×0.3m. Les contraintes cinématiques prennent les valeurs suivantes : la vitesse linéaire est limitée aux valeurs de l'intervalle $[0.05, 0.2]$m/s et l'accélération linéaire est limitée aux valeurs de l'intervalle $[0, 0.1]$m^2/s. La durée des morceaux est limité à 0.5s. La distance minimale entre le robot et les obstacles est fixée à 0.3m. La distance d utilisée pour projeter les points de passage est fixée à 1m.

Le résultat, illustré par la Figure 5.22, est une trajectoire qui fait une longueur de 17.21m (la solution en ligne droite mesure 15.02m) pour une durée de 87.0s, obtenue en 1.95s. La distance pour projeter les points de passage impose un guidage fort le long du chemin.

Cette notion est importante car elle dicte l'influence du guidage sur DKP. Si la distance d utilisée pour projeter les points de passage est fixée à 2m, alors l'exploration de l'environnement repose plus sur les capacités de DKP. La Figure 5.23 montre une solution qui passe plus rapidement la porte du laboratoire mais de biais par rapport à l'ouverture.

Ce processus de planification permet d'obtenir rapidement une solution. S'il n'y en a aucune, il est toujours possible d'exprimer d'autres alternatives. Cette analyse en cours de planification de l'arbre de pilotage est une fonctionnalité que je souhaite développer à terme. L'une des qualités de l'approche est que la trajectoire, tout en suivant la route optimale en distance, respecte les caractéristiques cinématiques du robot. Enfin, DKP affine la route tracées par les points de passage : en particulier, les angles liés à la discrétisation de l'environnement pour générer le diagramme de Voronoï sont atténués par la projection des objectifs le long de la route pour guider DKP et anticiper les changements de direction.

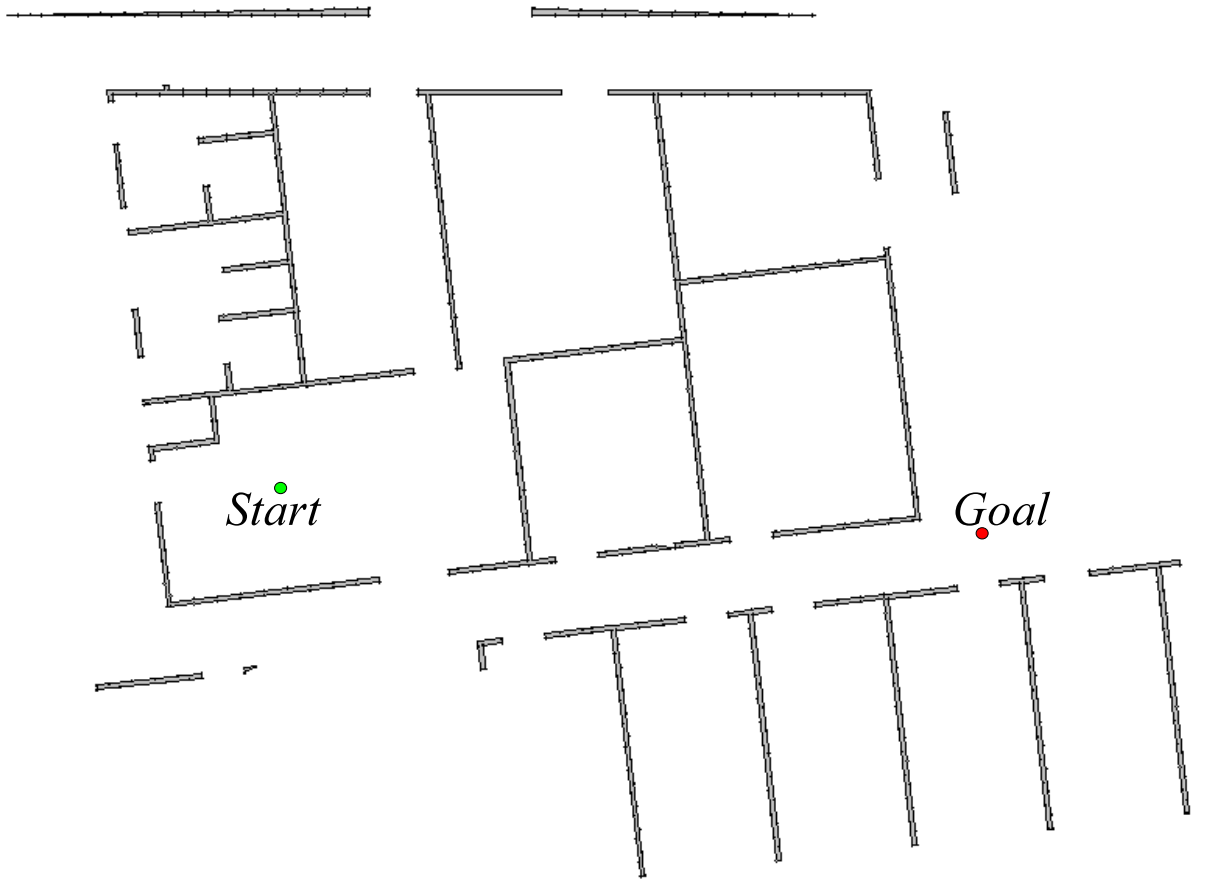

FIGURE 5.20 – Plans d'un niveau de l'ISEN. Le point *Start* est placé dans la pièce où se trouve notre laboratoire.

5.6 Dépassement

Cet exemple illustre la capacité de l'arbre de comportements de pilotage à retranscrire et évaluer des savoir-faire humains, ici un exemple de conduite illustré par le problème du dépassement.

5.6.1 Description

Pour cet exemple, je considère le scénario suivant, illustré par la Figure 5.27. Un robot R_0 se déplace aussi vite que possible sur un environnement de type route formé de deux voies sur lesquelles les sens de circulation sont opposés. R_0 débute le déplacement à la toute gauche de la voie de circulation du bas. Il souhaite se déplacer jusqu'à la fin de sa voie. Un robot R_1, plus lent, se déplace également sur la voie du bas à une vitesse constante S_1. Dans la voie du haut, un autre robot R_2 se déplace aussi à une vitesse constante $S_2 = S_1$, donc dans la direction opposée aux deux premiers robots R_0 et R_1. La route mesure 500m de long et 6m de largeur. Les dimensions des robots de la simulation sont de 4m pour la longueur et 1.7m pour la largeur (comme une petite voiture citadine). La distance de sécurité laissée entre les robots et les obstacles est fixée à 1.15m. Les robots R_1 et R_2 se déplacent à une vitesse de 110km/h ($\sim$ 30m/s) sur leurs voies

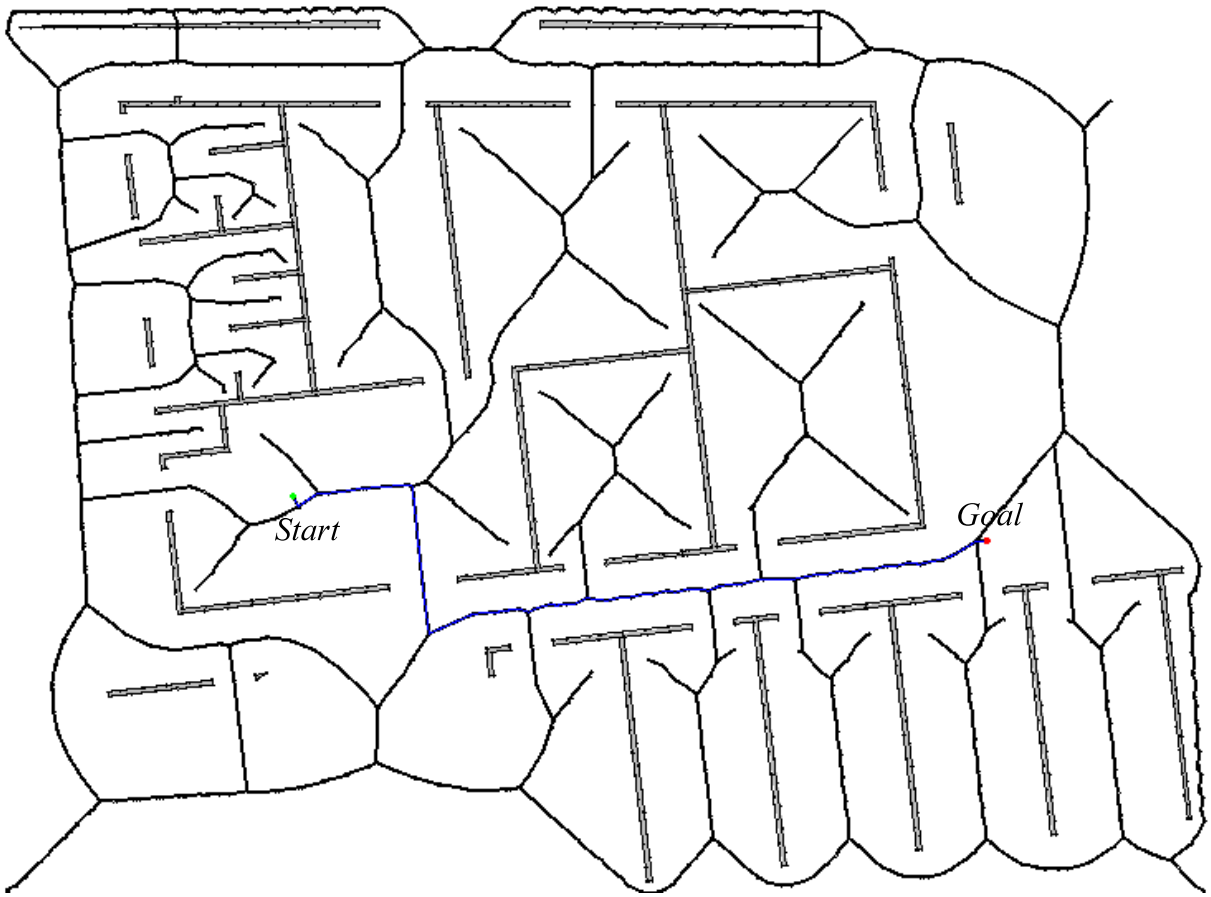

FIGURE 5.21 – Une roadmap issue de la simplification du diagramme de Voronoï (les lignes grises épaisses) sur les plans de l'ISEN. Le chemin optimal en distance sur la roadmap est représenté en bleu.

respectives. Le robot R_0 doit atteindre un objectif global $G_1(t)$ fixé à la fin de sa voie. Cet objectif fait office d'objectif principal défini pour le niveau de sélection *SGuide* de DKP.

5.6.2 Critique des planificateurs de trajectoire

Comme montré dans un environnement bien plus petit dans la Section 4.7, DKP peut gérer ce type de situation de dépassement. DKP obtient une solution pour ce cas de figure, illustré par la Figure 5.24 : le robot R_0 dépasse R_1 et revient ensuite sur sa voie de circulation tout en évitant le véhicule R_2. À mon sens, cette solution fournie par DKP seul, mais qui aurait pu être celle proposée par d'autres planificateurs de trajectoire, est un évitement d'obstacle par derrière : il n'y a là aucun raisonnement ou adaptation sur les contraintes cinématiques ou la durée des morceaux durant le processus de planification de trajectoire (mis à part le processus de backtrack mis en œuvre dans DKP). Il en résulte une trajectoire qui n'a rien de la forme d'un dépassement tel qu'un humain pourrait le faire. De plus, pour obtenir ce résultat avec DKP, les contraintes cinématiques ont des valeurs trop faibles pour être appliquées à un véritable dépassement sur route, bien que ces valeurs ne soient véritablement adaptées que pour la durée des morceaux et l'environnement considéré. De plus, même si les contraintes cinématiques autorisent un

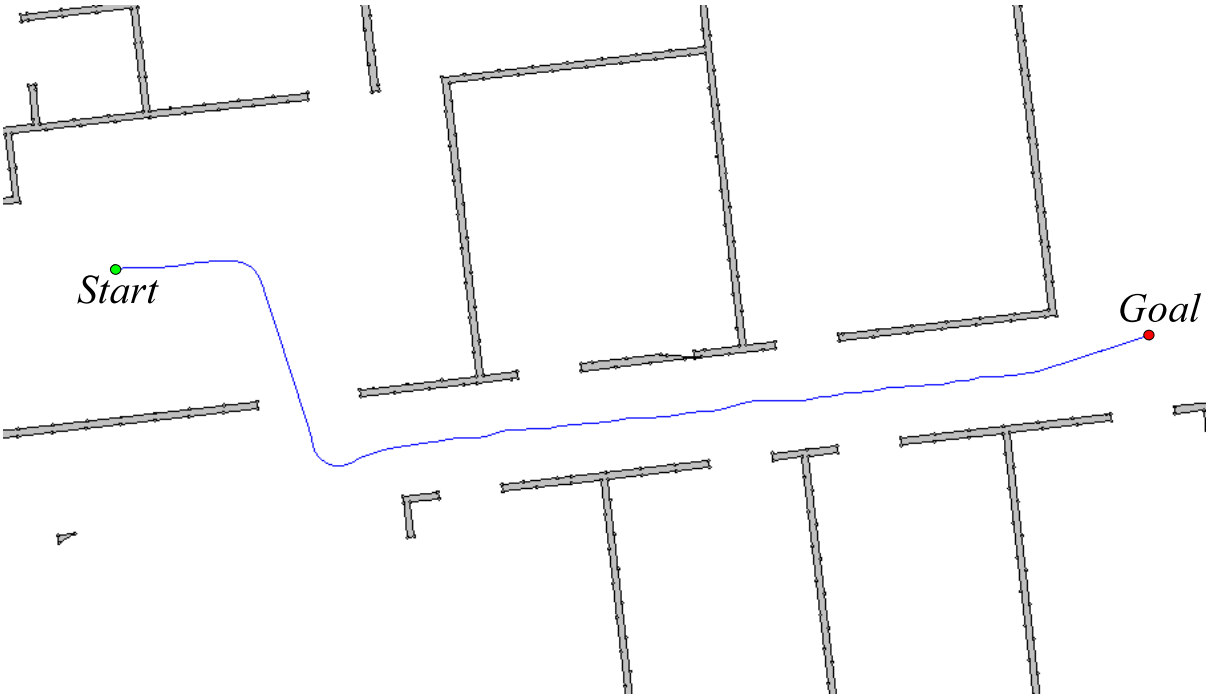

FIGURE 5.22 – La solution obtenue avec le guidage par comportement de pilotage le long d'une route optimale en distance. La distance d utilisée pour projeter les points de passage est fixée à 1m.

dépassement, si le robot R_2 démarre plus près de la zone de dépassement potentielle, DKP peut échouer à fournir une solution intuitive de type « ralentir pour suivre » : dans DKP, le processus de planification pousse à générer une trajectoire qui minimise le critère d'optimisation, ici la distance. DKP génère donc une solution qui tend toujours à dépasser si possible, avec le risque d'osciller derrière le véhicule à dépasser le temps que la voie de dépassement soit libérée par R_2.

5.6.3 Arbre de comportements de pilotage

En utilisant les comportements de pilotage, il est possible de gérer des situations avec des paramètres bien plus réalistes. Cela permet d'obtenir et d'évaluer à la fois les comportements « Suivre » et « Dépasser ». Tout d'abord, on décrit le comportement initial du robot R_0. Pour « Rallier l'objectif » $G_1(t)$, R_0 « Navigue » à une vitesse maximale de 130km/h : la contrainte de vitesse linéaire est fixée dans l'intervalle $[30; 36]$m/s, la durée des morceaux de trajectoire est fixée à 0.5s et l'objectif $G_1(t)$ est fourni comme objectif au niveau de sélection de DKP *SGuide*.

Lorsque R_0 rejoint R_1, c'est-à-dire entre dans la surface $A_1(t)$ derrière R_1, cela déclenche l'instanciation du comportement de pilotage « Suivre ou Dépasser ». Deux alternatives sont définies sous le nœud q_max puis ajoutées à l'arbre de trajectoires :

- R_0 ajuste sa vitesse à celle de R_1 et le « Suit » : la contrainte de vitesse linéaire est définie à l'intervalle $[27; 30]$m/s, la durée des morceaux de trajectoire est définie à 1s (suivre un robot circulant à vitesse constante ne nécessite pas de manœuvres très réactives) et l'objectif $G_1(t)$ est maintenu comme objectif du niveau de sélection de DKP ;
- R_0 « Dépasse » R_1 et ce comportement est factorisé sous un nœud q_seq_sum.

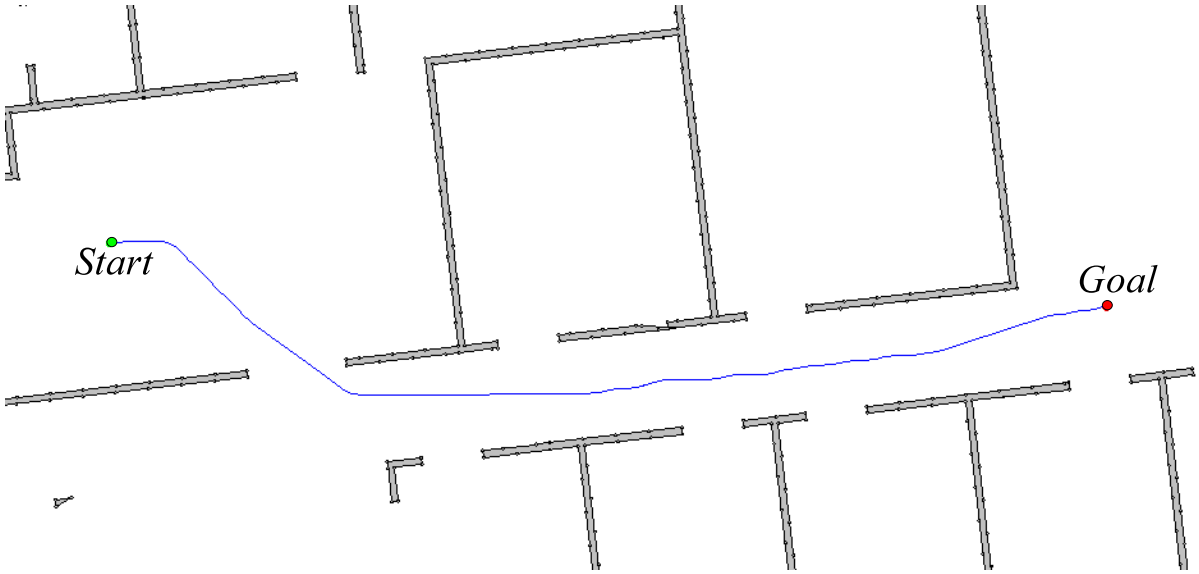

FIGURE 5.23 – La solution obtenue avec le guidage par comportement de pilotage le long d'une route optimale en distance. La distance d utilisée pour projeter les points de passage est fixée à 2m.

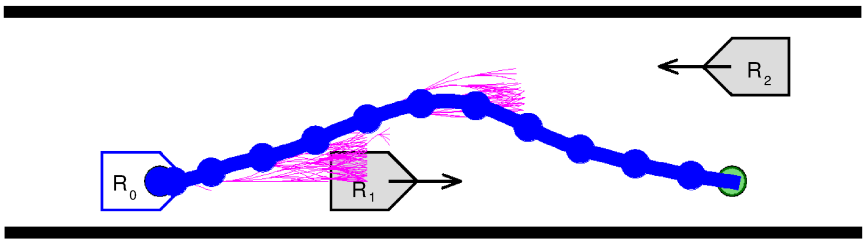

FIGURE 5.24 – Dans une situation de dépassement, la solution fournie par DKP seul peut être interprétée comme un évitement d'obstacle par derrière.

Le comportement de pilotage « Dépasser » utilise différents paramètres de pilotage :

1. R_0 accélère jusque 150km/h et « Va sur la voie du haut » : la contrainte de vitesse linéaire est définie sur l'intervalle $[30; 42]$m/s, la durée des morceaux est fixée à 0.1s (le dépassement requiert des manœuvres précises) et un objectif $G_2(t)$ est ajouté à *PGuide* pour forcer R_0 à tourner à gauche.
2. lorsque R_0 entre dans la surface $A_2(t)$ derrière R_1 sur la voie du haut, dans les mêmes conditions de vitesse, il « Reste sur la voie du haut » : un objectif $G_3(t)$ ajouté à *PGuide* est situé à la fin de la voie de gauche.
3. lorsque R_0 entre dans la surface $A_3(t)$ devant R_1 sur la voie du haut, dans les mêmes conditions de vitesse, il « va sur la voie du bas » : un objectif $G_4(t)$ ajouté à *PGuide* force R_0 à tourner vers la droite.
4. Lorsque R_0 entre dans la surface $A_4(t)$ devant R_1 sur la voie du bas, il retourne au comportement de pilotage « Naviguer ».

FIGURE 5.25 – L'arbre final de pilotage pour le dépassement, contenant les alternatives de comportement : « Suivre » ou « Dépasser »

La Figure 5.25 montre l'arbre de pilotage pour cet exemple. Les surfaces et les objectifs associés aux déplacements de R_1 sont illustrées dans la Figure 5.26[8].

5.6.4 Planification et résultats

La planification de trajectoire fonctionne comme précédemment décrit : l'arbre de pilotage applique les paramètres de pilotage aux surfaces de l'environnement décrites dans la Section 5.2. Si deux comportements de pilotage peuvent être suivis, deux branches de DKP peuvent s'étendre en parallèle, appliquant les différents paramètres de pilotage. A la fin, le planificateur retourne, si elle existe pour DKP, une solution associée à chaque comportement. Par exemple, dans la Figure 5.27(a), le comportement « Dépasser » n'est pas résolu tant que le robot R_2 se trouve sur la voie opposée au même instant : uniquement le comportement « Suivre » peut être appliqué comme trajectoire valide. Si R_2 se tient plus loin (Figure 5.27(b)), le comportement « Dépasser » peut être résolu et peut être appliqué comme trajectoire valide. La solution associée au comportement « Dépasser » dure 13s tandis que le comportement « Suivre » dure 15.5s.

Une fois de plus, les trajectoires obtenues sont meilleures que dans DKP seul. L'arbre de trajectoires est également bien moins complexe que dans la Figure 5.24, où énormément d'appels au backtrack sont nécessaires avant d'obtenir une trajectoire valide. Le temps de calcul est également grandement diminué.

5.7 Conclusion

Comme Reynolds l'a fait pour les personnages autonomes dans des environnement simulés, j'ai introduit les comportements de pilotage pour les robots tout en utilisant une approche cognitive plutôt qu'une approche strictement réactive. Ainsi, je présente une architecture dans laquelle deux couches intimement liées sont utilisées pour co-élaborer des trajectoires candidates. Celles-ci peuvent être évaluées par une couche cognitive qui sélectionnera la meilleure trajectoire à appliquer dans une situation particulière.

Pour le premier niveau, je m'appuis naturellement sur DKP comme planificateur de trajectoire, construit en ce sens. Celui-ci est capable de décrire et manipuler efficacement les contraintes cinématiques dans un monde réel continu rencontrées dans un problème de planification de trajectoire appliqué à des robots. Toutefois, cette couche n'est pas capable à elle seule de résoudre toute la complexité des tâches robotiques. C'est pourquoi j'ai ajouté une couche basée sur TÆMS dont le rôle est de faire une connexion entre la couche cognitive et la couche de planification de trajectoire. Dans cette couche, je suis capable de décrire les comportements de pilotage qui conditionnent à partir d'une interaction ascendante l'évolution de l'arbre de trajectoires fourni par DKP. L'interaction entre les deux couches est également descendante car les comportements de pilotage sont instanciés en réaction aux situations rencontrées durant la construction de l'arbre de trajectoires.

Les exemples précédents montrent comment cette architecture permet de rendre plus efficace l'exploration de l'environnement qu'avec DKP seul. J'ai présenté trois usages possibles de mon architecture, mais l'expressivité des paramètres de pilotage et du langage TÆMS est suffisamment élevée pour décrire d'autres comportements de pilotage basés sur des comportements humains. Les changements dynamiques d'accélération ou de vitesse linéaire ou angulaire ou encore du point objectif peuvent être utilisés pour décrire d'autre manœuvres de conduite ou des comportements de pilotage de type « calmes » ou « agressifs ».

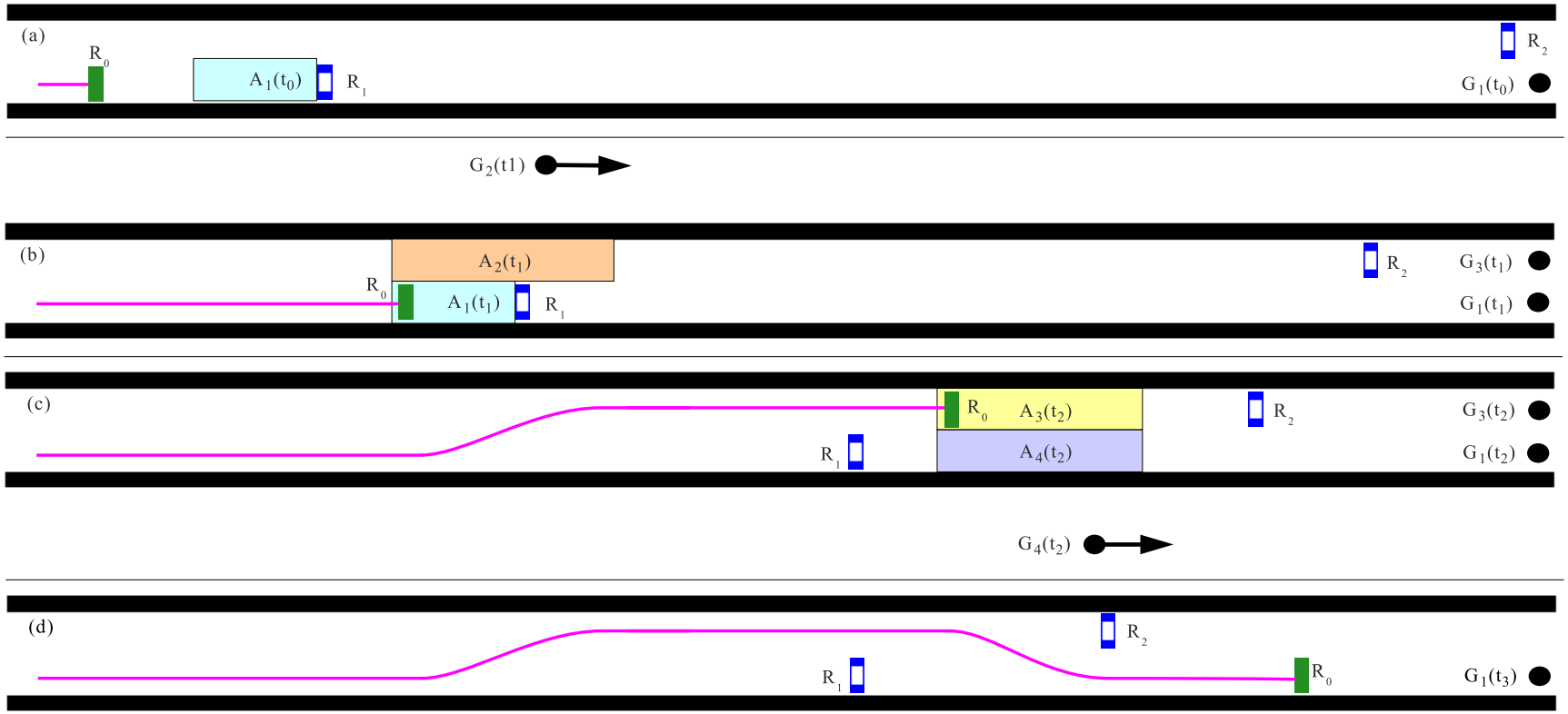

FIGURE 5.26 – Les surfaces de déclenchement de comportement et les objectifs représentés à plusieurs étapes du comportement de dépassement. (a) la situation initiale, (b) À t_1, lorsqu'une trajectoire candidate entre dans la surface $A1(t)$, (c) À t_2, quand une trajectoire candidate pénètre dans la surface $A3(t)$, (d) le dépassement est achevé. Le robot contrôlé est le rectangle vert et les véhicules à éviter sont les rectangles bleus

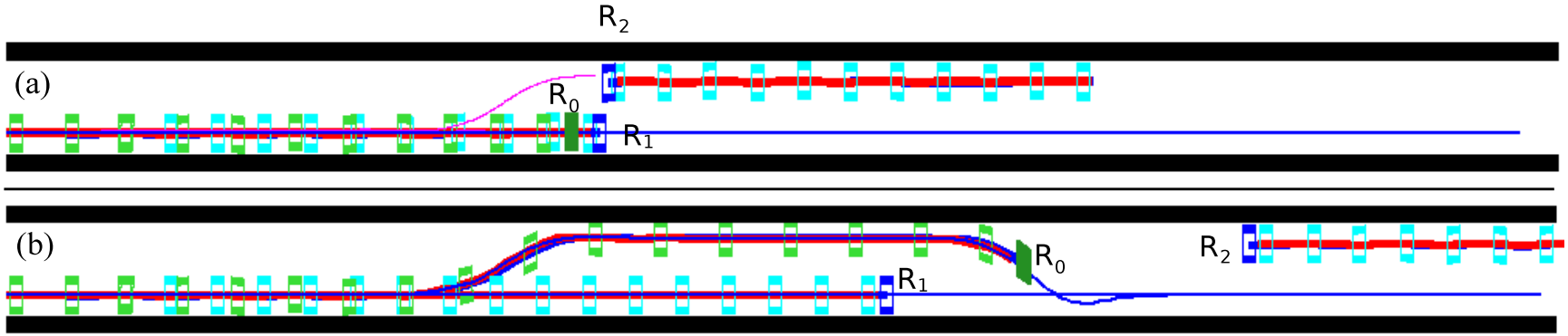

FIGURE 5.27 – Différentes situations de dépassement. (a) R_0, en vert, ne peut pas dépasser à cause de la présence de R_2. Il doit suivre le robot R_1 en utilisant la trajectoire en bleu. (b) R_0 peut suivre ou dépasser R_1 car le robot R_2 ne gêne pas la manœuvre : on obtient deux trajectoires alternatives en bleu. R_0 est dessiné sur celle qui dépasse

Chapitre 6

Conclusion et Perspectives

Contributions de la thèse

Dans cette thèse, je présente une approche cognitive pour la planification de trajectoire sous contraintes. Celle-ci permet aux planificateurs symboliques d'exprimer des actions qui sont ensuite traduites sous la forme de trajectoires qui respectent le modèle physique des robots. Cette technique est appliquée à des robots à deux roues indépendantes dans des environnements à deux dimensions en situation réelle. Pour mettre en œuvre cette approche, j'utilise plusieurs couches qui vont collaborer entre elles, chacune apportant une contribution. Ainsi, nous évitons les travers usuels des approches découplées. Cette approche est possible grâce aux contributions suivantes : l'espace des paramètres pour établir des solutions locales, DKP pour les propager dans un arbre d'exploration et les comportements de pilotage pour contrôler sa dynamique.

L'espace des paramètres Mon approche s'appuie sur une couche de propagation [Gaillard *et al.*, 2010] dont le but est de générer des échantillons de trajectoires dans le respect du modèle physique du robot. Pour le contrôle de robots à deux roues indépendantes, j'utilise des morceaux de trajectoire à base de quadratiques. Ceux-ci contiennent des paramètres en partie fixés par les conditions de continuité entre deux déplacements. Les deux paramètres laissés libres forment la base de l'espace des paramètres. J'utilise une approche géométrique pour représenter cet espace des paramètres. Celui-ci est construit en retirant les points créant des trajectoires qui ne respectent pas les contraintes du problème. Si l'espace des paramètres n'est pas vide à l'issue de l'étape précédente, une recherche est effectuée en minimisant un critère d'optimisation dans la surface restante.

DKP DKP est une approche pour la planification de trajectoire par échantillonnage, acronyme de Deterministic Kinodynamic Planning. Il s'appuie sur la couche de propagation précédente pour générer des échantillons de trajectoire qui vont constituer un arbre d'exploration dans les parties atteignables de l'environnement. La couche de sélection est inspirée de l'algorithme A* pour sélectionner les meilleures branches à prolonger. DKP utilise astucieusement l'espace des paramètres pour générer de la diversité en échantillons, nécessaire à l'exploration de l'environnement. De plus, il sert aussi à ne sélectionner que les parties réellement prolongeables de l'arbre d'exploration. Les solutions sont des trajectoires splines adaptées au contrôle de robots à deux roues indépendantes. DKP a la propriété d'être déterministe : ses résultats sont donc reproductibles. Dans [Gaillard *et al.*, 2011], je fournis une étude des différents modes de fonctionnement de DKP. En plus des modes optimaux et gloutons issus du fonctionnement de A*, je présente le mode backtrack qui améliore l'efficacité de DKP en modifiant à la fois l'arbre

de trajectoires et la structure de l'environnement en ajoutant des obstacles virtuels dans les minima locaux, ce qui permet de s'en extraire.

Les comportements de pilotage Je définis des comportements de pilotage à partir de paramètres de pilotage qui servent à contrôler DKP. Ceux-ci constituent la base de l'approche cognitive pour la planification de trajectoire [Gaillard *et al.*, 2012]. Les déplacements du robot sont exprimés sous la forme d'un arbre de comportements : chaque comportement agit sur DKP et donc sur la manière dont l'arbre d'exploration progresse dans l'environnement. Une action est reliée à une zone de l'environnement. Ainsi, dès qu'une branche de l'arbre d'exploration entre dans cette zone, l'arbre de pilotage est développé en conséquence et les nouveaux paramètres de pilotage sont appliqués sur l'arbre d'exploration. Le formalisme TÆMS permet d'évaluer les solutions ainsi construites. Au final, cette approche cognitive repose sur l'évolution conjointe d'un arbre de comportements de pilotage et d'un arbre d'exploration qui assure le respect des contraintes : ce couplage fort qui existe entre ces deux arbres est véritablement le point central de ma thèse. La construction de DKP a été faite pour que cela soit possible.

Applications

Dans l'architecture proposée, le modèle physique des robots et les contraintes sur l'environnement sont exprimés grâce aux contraintes suivantes :

- vitesse et accélération linéaire ;
- vitesse angulaire ;
- évitement d'obstacles fixes ou mobiles ;
- restriction à des zones de déplacement.

Grâce à l'approche géométrique pour construire l'espace des paramètres, ce planificateur peut être étendu à tout ensemble de contraintes. Il suffit d'exprimer la manière d'extruder l'espace des paramètres.

Une étude expérimentale, proposée dans [Gaillard *et al.*, 2011], montre comment DKP retranscrit fidèlement les capacités physiques de nos robots. Pour mettre en œuvre DKP, je m'appuie sur la plateforme de déploiement et d'exécution d'applications robotiques, nommée APM(Robot), qui a été développée par le département informatique de l'ISEN. DKP peut y déterminer les trajectoires dont sont extraites les commandes relayées ensuite à nos robots (voir la Figure 6.1).

Perspectives

Tout au long de ce manuscrit, j'ai mis en évidence quelques défauts de DKP que j'aimerai corriger par la suite. En particulier, DKP souffre du manque d'anticipation dans la construction de l'arbre d'exploration. Cela provoque des problèmes de freinage dont une conséquence visible est la satellisation de l'arbre d'exploration autour d'un objectif lorsque DKP est trop glouton. Il faudra donc certainement raisonner sur plusieurs échantillons pour déterminer à l'avance le bon comportement à adopter. D'autres politiques de recherche de solutions sur l'espace des paramètres pourraient également être exploitées.

De nombreuses améliorations de DKP peuvent être menées sur les mécanismes de planification de trajectoire employés par DKP. Tout d'abord, il faudrait étendre la construction de l'espace des paramètres à d'autres types de robots : par exemple des bras mécaniques. Ensuite, il serait bien de s'inspirer de RRT* pour améliorer l'optimalité des solutions construites par DKP.

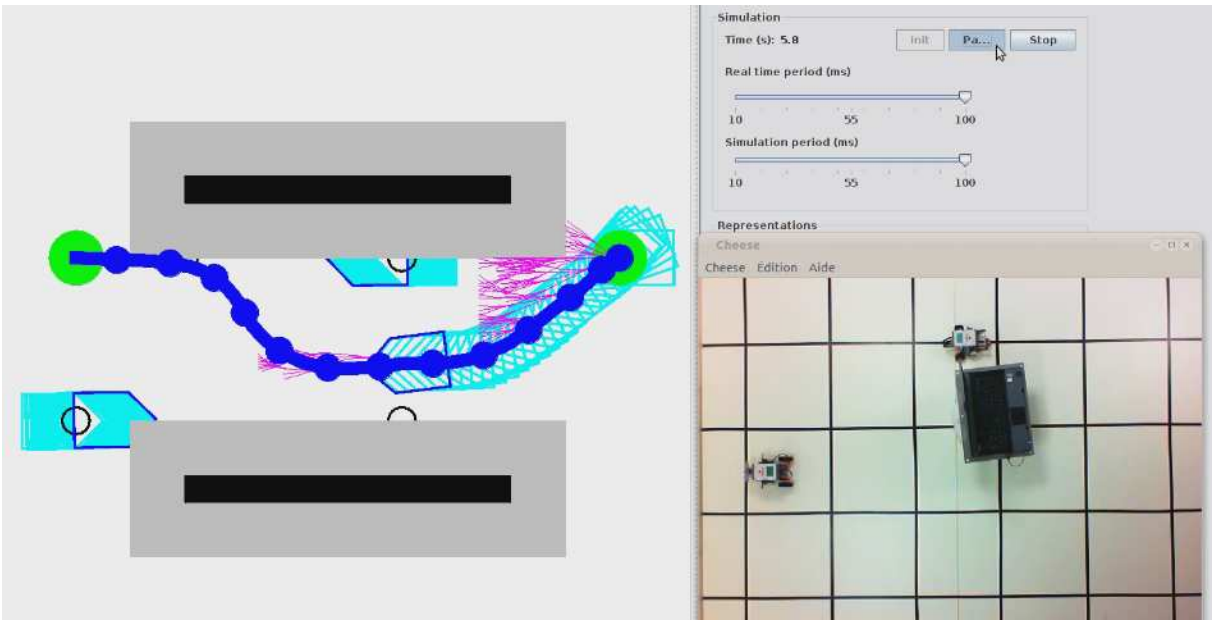

FIGURE 6.1 – Dépassement simulé par DKP puis appliqué à nos robots grâce à APM(Robot).

Deux autres améliorations de l'approche cognitive de planification de trajectoire sont également possibles. Premièrement, ce planificateur de trajectoire devra fonctionner de manière continue. En effet, pour l'instant, nous devons lancer des planifications successives et distinctes afin de faire fonctionner DKP dans un scénario sans fin. Pour cela, on pourrait faire grossir continuellement les deux arbres en réagissant à deux nouveaux événements, sélectionner les nœuds à exécuter et abandonner les nœuds déjà exécutés.

Enfin, je pourrai travailler sur une troisième couche de raisonnement sur les comportements de pilotage. Ils seraient instanciés par cette couche afin de résoudre des objectifs de plus haut niveau, avec la possibilité de les fusionner. Je pourrai ainsi générer et évaluer des comportements composés tels que *dépasser agressivement* or *suivre doucement*.

Chapitre 7

Annexe : Géométrie constructive de surface pour la planification locale de trajectoire

Cette annexe est consacrée au fonctionnement de la géométrie constructive de surface. J'utilise cet outil pour manipuler l'espace des paramètres et les autres ensembles de valeurs utilisés dans le niveau de propagation. Les transformations affines applicables à ces surfaces constituent la base des opérations ensemblistes que j'ai employées pour manipuler l'espace des paramètres.

7.1 Définition

Les ensembles semi-finis [Bierstone et Milman, 1988] sont des parties de $\mathbb{R}^n$. Un ensemble contient tous les $x \in \mathbb{R}^n : b(x) > 0$, où $b(x) = b(x_1, \ldots, x_n)$ est un polynôme, stable par intersection, union et complément. La succession de polynômes, qui peut être alors vue comme une courbe spline, décrit la bordure de l'ensemble considéré.

Ces ensembles semi-finis servent de base à la géométrie constructive de surface, par exemple employée de base par Java (package Java2D)[9] ou CGAL[10] (librairie C/C++) pour décrire des surfaces à deux dimensions. Le même concept peut être étendu à n'importe quelle dimension N, tant qu'il est possible de décrire la bordure des ensembles par des éléments de dimension $N-1$. C'est par exemple le cas de la géométrie constructive de solides, utilisée dans des logiciels et librairies aux applications variées telles que SolidWorks, Java3D ou encore GTKRadiant.

Dans le cas de la géométrie constructive de surface utilisée dans cette thèse, les contours des surfaces sont définis par une succession de polynômes d'ordre 3 maximum, d'après l'implémentation Java2D. Des règles de franchissement permettent de distinguer l'intérieur de l'extérieur de ces surfaces.

7.2 Opérations sur les surfaces

La géométrie constructive de surface permet d'effectuer les opération ensemblistes sur les surfaces, notées A. Il est ainsi possible de construire des surfaces à partir d'autres surfaces par les opérateurs suivants :

- union O_U des surfaces A_0 et A_1 ;
- soustraction O_S de la surface A_0 par la surface A_1 ;
- intersection O_I de la surface A_0 avec la surface A_1 ;

– ou exclusif O_{XOR} comme étant les parties non-communes aux surfaces A_0 et A_1.

D'autres opérations sont également possibles sur les surfaces :

– l'application de transformations affines TA sur la surface A :
  – par translation $TA_{Tr}(A, x_{dest}, y_{dest})$;
  – par rotation $TA_{Ro}(A, \theta, x_{centre}, y_{centre})$;
  – par homothétie $TA_{Ho}(A, facteur, x_{centre}, y_{centre})$;
  – selon la matrice de transformation M, notée $TA_M(A, M)$.
– l'appartenance d'une surface A' à A ;
– l'appartenance d'un point (x, y) à A.

7.3 Délimitation des surfaces

L'implémentation des surfaces que j'ai employée est celle du package Java2D. Dans cette librairie, les bordures sont des B-Splines, basés sur les polynômes de Bernstein. $B(n, m)$ est le polynôme de Bernstein de coefficient m et de degré n :

$$B(n, m) = C(n, m) * t^m * (1 - t)^{n-m} \tag{7.1}$$

avec les coefficients binomiaux :

$$C(n, m) = \frac{n!}{(m! * (n - m)!)} \tag{7.2}$$

Les bordures sont décrites à partir des polygones de contrôles de ces polynômes.

7.3.1 Segment

Un segment (illustré par la Figure 7.1) est défini par le point de départ DP et le point final EP selon la relation :

$$b(t) = B_{1,0} \times DP + B_{1,1} \times EP \text{ avec } 0 \leq t \leq 1 \tag{7.3}$$

FIGURE 7.1 – Un segment et son polygone de contrôle.

7.3.2 Quadratique

Une courbe quadratique (illustré par la Figure 7.2) est définie par le point de départ DP, le point de contrôle CP et le point final EP selon la relation :

$$b(t) = B_{2,0} \times DP + B_{2,1} \times CP + B_{2,2} \times EP \text{ avec } 0 \leq t \leq 1 \tag{7.4}$$

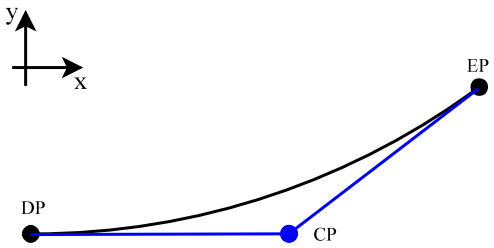

FIGURE 7.2 – Une quadratique et son polygone de contrôle.

7.3.3 Cubique

Une courbe cubique (illustré par la Figure 7.3) est définie par le point de départ DP, les points de contrôle $CP1$ et $CP2$ et le point final EP selon la relation :

$$b(t) = B_{3,0} \times DP + B_{3,1} \times CP1 + B_{3,2} \times CP2 + B_{3,3} \times EP \text{ avec } 0 \leq t \leq 1 \qquad (7.5)$$

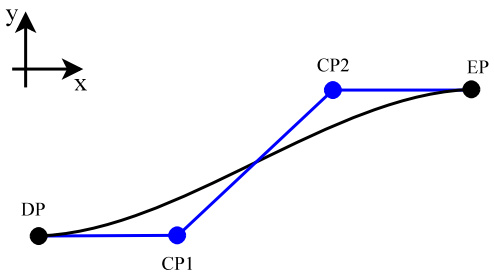

FIGURE 7.3 – Une cubique et son polygone de contrôle.

Ces expressions sont développées dans la Section 7.6.

7.4 Quelques figures de base : disque, rectangle

Je donne dans cette section quelques exemples de surfaces rencontrées fréquemment lors de la définition de contraintes cinématiques ou d'évitement. Les figures sont celles obtenues à partir de l'implémentation Java2D. Notons que les polygones de contrôle donnés ici sont ceux obtenus lors du parcours itératif de la bordure. Toutefois, ils ne sont pas exactement représentatifs des bordures véritablement utilisées, montrées dans la Section 7.6. En effet, dans certains cas, seule une partie de ces éléments de bordure sont nécessaires pour effectuer les calculs d'intersection que requiert la manipulation de ces surfaces.

7.4.1 Disque

Soit *Disk* une disque de rayon 1 centrée en $(0, 0)$. Il s'agit ici du cas d'une contrainte d'accélération dont la borne supérieure est fixée à 1m/s^2. Ce disque est délimité par 4 courbes

cubiques dont le polygone de contrôle est défini par :

$$\begin{aligned} &DP = (0,-1),\ CP1 = (-0.552,-1.0),\ CP2 = (-1.0,-0.552) \text{ et } EP = (-1.0,0.0) \\ &DP = (-1.0,0.0),\ CP1 = (-1.0,0.552),\ CP2 = (-0.552,1.0) \text{ et } EP = (0.0,1.0) \\ &DP = (0.0,1.0),\ CP1 = (0.552,1.0),\ CP2 = (1.0,0.552) \text{ et } EP = (1.0,0.0) \\ &DP = (1.0,0.0),\ CP1 = (1.0,-0.552),\ CP2 = (0.552,-1.0) \text{ et } EP = (0.0,-1.0) \end{aligned} \tag{7.6}$$

Le disque est représenté avec son polygone de contrôle par la Figure 7.4.

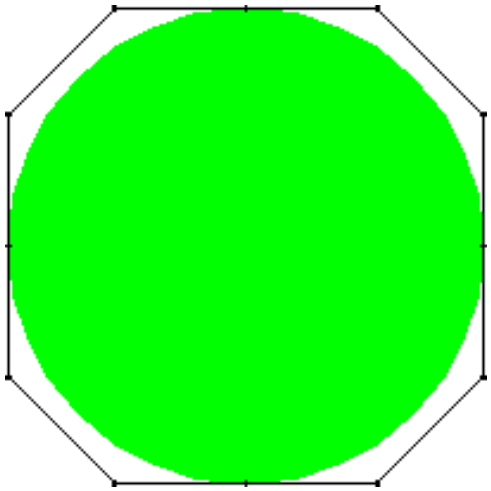

FIGURE 7.4 – Un disque et son polygone de contrôle.

7.4.2 Rectangle

Soit *Rectangle* un rectangle de longueur 2 et largeur 1, centré en $(0,0)$. Un tel rectangle est suffisant pour représenter les limites d'un robot de type voiture. Ce rectangle est formé par 4 segments :

$$\begin{aligned} &DP = (-1.0,-0.5) \text{ et } EP = (-1.0,0.5) \\ &DP = (-1.0,0.5) \text{ et } EP = (1.0,0.5) \\ &DP = (1.0,0.5) \text{ et } EP = (1.0,-0.5) \\ &DP = (1.0,-0.5) \text{ et } EP = (-1.0,-0.5) \end{aligned} \tag{7.7}$$

Le rectangle est représenté avec ses sommets par la Figure 7.5.

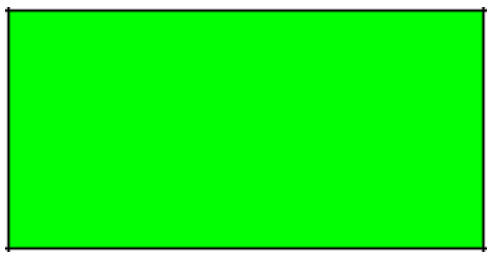

FIGURE 7.5 – Un rectangle et ses sommets.

7.5 Composition de surfaces

La géométrie constructive de surface permet d'effectuer des opérations ensemblistes sur les surfaces. Ces opérations permettent de construire des figures plus complexes, donc des contraintes plus complexes. Elles permettent également de délimiter progressivement des ensembles de valeurs permises, tels que montrés dans la Section 7.6.

Reprenons les deux figures précédentes :
- *Disk* une disque de rayon 1 centrée en $(0, 0)$;
- *Rectangle* un rectangle de longueur 2 et largeur 1, centrée en $(0, 0)$.

J'illustre les opérations ensemblistes possibles à partir de ces deux surfaces.

7.5.1 Union

L'union du disque *Disk* avec le rectangle *Rectangle* par l'opération $O_U(Disk, Rectangle)$ est délimitée par les morceaux :
- cubique dont les points de contrôle sont $DP = (0.000, -1.000)$, $CP1 = (-0.370, -1.000)$, $CP2 = (-0.693, -0.799)$ et $EP = (-0.866, -0.500)$;
- segment dont les points de contrôle sont $DP = (-0.866, -0.500)$ et $EP = (-1.000, -0.500)$;
- segment dont les points de contrôle sont $DP = (-1.000, -0.500)$ et $EP = (-1.000, 0.500)$;
- segment dont les points de contrôle sont $DP = (-1.000, 0.500)$ et $EP = (-0.866, 0.500)$;
- cubique dont les points de contrôle sont $DP = (-0.866, 0.500)$, $CP1 = (-0.693, 0.799)$, $CP2 = (-0.370, 1.000)$ et $EP = (0.000, 1.000)$;
- cubique dont les points de contrôle sont $DP = (0.000, 1.000)$, $CP1 = (0.370, 1.000)$, $CP2 = (0.693, 0.799)$ et $EP = (0.866, 0.500)$;
- segment dont les points de contrôle sont $DP = (0.866, 0.500)$ et $EP = (1.000, 0.500)$;
- segment dont les points de contrôle sont $DP = (1.000, 0.500)$ et $EP = (1.000, -0.500)$;
- segment dont les points de contrôle sont $DP = (1.000, -0.500)$ et $EP = (0.866, -0.500)$;
- cubique dont les points de contrôle sont $DP = (0.866, -0.500)$, $CP1 = (0.693, -0.799)$, $CP2 = (0.370, -1.000)$ et $EP = (0.000, -1.000)$.

Une partie des polygones de contrôle de chacune des figures de départ s'y trouvent, séparés par les points d'intersection entre chaque figure ($(-0.866, -0.500)$, $(0.866, -0.500)$, $(-0.866, 0.500)$ et $(0.866, 0.500)$). Le résultat de cette opération est représenté avec ses points de contrôle par la Figure 7.6.

7.5.2 Soustraction

La soustraction du disque *Disk* avec le rectangle *Rectangle* par l'opération $O_S(Disk, Rectangle)$ forme deux morceaux. La partie inférieure du disque restant est délimitée par les morceaux :
- cubique dont les points de contrôle sont $DP = (0.000, -1.000)$, $CP1 = (-0.370, -1.000)$, $CP2 = (-0.693, -0.799)$ et $EP = (-0.866, -0.500)$;
- segment dont les points de contrôle sont $DP = (-0.866, -0.500)$ et $EP = (0.866, -0.500)$;
- cubique dont les points de contrôle sont $DP = (0.866, -0.500)$, $CP1 = (0.693, -0.799)$, $CP2 = (0.370, -1.000)$ et $EP = (0.000, -1.000)$.

La partie supérieur du disque restant est délimitée par les morceaux :
- cubique dont les points de contrôle sont $DP = (-0.866, 0.500)$, $CP1 = (-0.693, 0.799)$, $CP2 = (-0.370, 1.000)$ et $EP = (0.000, 1.000)$;

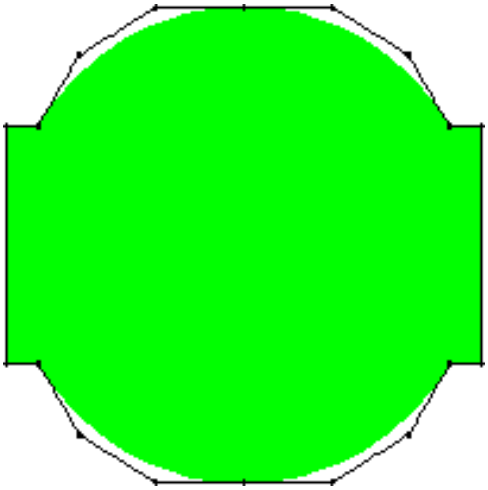

FIGURE 7.6 – L'union d'un disque avec un rectangle.

– cubique dont les points de contrôle sont $DP = (0.000, 1.000)$, $CP1 = (0.370, 1.000)$, $CP2 = (0.693, 0.799)$ et $EP = (0.866, 0.500)$;
– segment dont les points de contrôle sont $DP = (0.866, 0.500)$ et $EP = (-0.866, 0.500)$.

Le résultat de cette opération est représenté avec ses points de contrôle par la Figure 7.7. Il est important de noter que deux surfaces isolées sont également des surfaces.

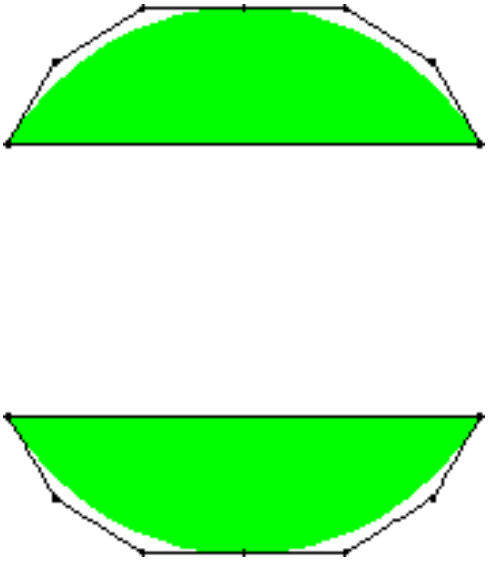

FIGURE 7.7 – La soustraction d'un disque par un rectangle.

7.5.3 Intersection

L'intersection du disque *Disk* avec le rectangle *Rectangle* par l'opération $O_I(Disk, Rectangle)$ est délimitée par les morceaux :

– segment dont les points de contrôle sont $DP = (-0.866, -0.500)$ et $EP =$

$(-0.866, -0.500)$;
- cubique dont les points de contrôle sont $DP = (-0.866, -0.500)$, $CP1 = (-0.951, -0.353)$, $CP2 = (-1.000, -0.182)$ et $EP = (-1.000, 0.000)$;
- cubique dont les points de contrôle sont $DP = (-1.000, 0.000)$, $CP1 = (-1.000, 0.182)$, $CP2 = (-0.951, 0.353)$ et $EP = (-0.866, 0.500)$;
- segment dont les points de contrôle sont $DP = (-0.866, 0.500)$ et $EP = (0.866, 0.500)$;
- cubique dont les points de contrôle sont $DP = (0.866, 0.500)$, $CP1 = (0.951, 0.353)$, $CP2 = (1.000, 0.182)$ et $EP = (1.000, 0.000)$;
- cubique dont les points de contrôle sont $DP = (1.000, 0.000)$, $CP1 = (1.000, -0.182)$, $CP2 = (0.951, -0.353)$ et $EP = (0.866, -0.500)$;
- segment dont les points de contrôle sont $DP = (0.866, -0.500)$ et $EP = (-0.866, -0.500)$.

Le résultat de cette opération est représenté avec ses points de contrôle par la Figure 7.8.

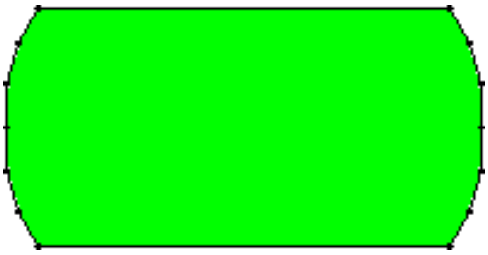

FIGURE 7.8 – L'intersection d'un disque par un rectangle.

7.5.4 Ou exclusif

Le Ou exclusif du disque *Disk* avec le rectangle *Rectangle* par l'opération $O_{XOR}(Disk, Rectangle)$ est plus complexe. Le résultat est composé de 6 surfaces. Comme attendu, on retrouve les parties inférieures et supérieures obtenues lors de l'opération de soustraction :
- cubique dont les points de contrôle sont $DP = (0.000, -1.000)$, $CP1 = (-0.370, -1.000)$, $CP2 = (-0.693, -0.799)$ et $EP = (-0.866, -0.500)$;
- segment dont les points de contrôle sont $DP = (-0.866, -0.500)$ et $EP = (0.866, -0.500)$;
- cubique dont les points de contrôle sont $DP = (0.866, -0.500)$, $CP1 = (0.693, -0.799)$, $CP2 = (0.370, -1.000)$ et $EP = (0.000, -1.000)$.
- cubique dont les points de contrôle sont $DP = (-0.866, 0.500)$, $CP1 = (-0.693, 0.799)$, $CP2 = (-0.370, 1.000)$ et $EP = (0.000, 1.000)$;
- cubique dont les points de contrôle sont $DP = (0.000, 1.000)$, $CP1 = (0.370, 1.000)$, $CP2 = (0.693, 0.799)$ et $EP = (0.866, 0.500)$;
- segment dont les points de contrôle sont $DP = (0.866, 0.500)$ et $EP = (-0.866, 0.500)$.

Les parties latérales sont chacune d'entre elles séparées en 2 morceaux. En effet, les points $(-1.000, 0.000)$ et $(1.000, 0.000)$ constituent la fine séparation entre les parties inférieures et supérieures. La partie de gauche est composée des surfaces suivantes :
- la partie inférieure :
  - segment dont les points de contrôle sont $DP = (-1.000, -0.500)$ et $EP = (-1.000, 0.000)$;
  - cubique dont les points de contrôle sont $DP = (-1.000, 0.000)$, $CP1 = (-1.000, -0.182)$, $CP2 = (-0.951, -0.353)$ et $EP = (-0.866, -0.500)$;

  - segment dont les points de contrôle sont $DP = (-0.866, -0.500)$ et $EP = (-1.000, -0.500)$.
- la partie supérieure :
  - segment dont les points de contrôle sont $DP = (-1.000, 0.000)$ et $EP = (-1.000, 0.500)$;
  - segment dont les points de contrôle sont $DP = (-1.000, 0.500)$ et $EP = (-0.866, 0.500)$;
  - cubique dont les points de contrôle sont $DP = (-0.866, 0.500)$, $CP1 = (-0.951, 0.353)$, $CP2 = (-1.000, 0.182)$ et $EP = (-1.000, 0.000)$.

La partie de droite est composée des surfaces suivantes :

- la partie inférieure :
  - cubique dont les points de contrôle sont $DP = (0.866, -0.500)$, $CP1 = (0.951, -0.353)$, $CP2 = (1.000, -0.182)$ et $EP = (1.000, 0.000)$;
  - segment dont les points de contrôle sont $DP = (1.000, 0.000)$ et $EP = (1.000, -0.500)$;
  - segment dont les points de contrôle sont $DP = (1.000, -0.500)$ et $EP = (0.866, -0.500)$.
- la partie supérieure :
  - cubique dont les points de contrôle sont $DP = (1.000, 0.000)$, $CP1 = (1.000, 0.182)$, $CP2 = (0.951, 0.353)$ et $EP = (0.866, 0.500)$;
  - segment dont les points de contrôle sont $DP = (0.866, 0.500)$ et $EP = (1.000, 0.500)$;
  - segment dont les points de contrôle sont $DP = (1.000, 0.500)$ et $EP = (1.000, 0.000)$;

Le résultat de cette opération est représenté avec ses points de contrôle par la Figure 7.9.

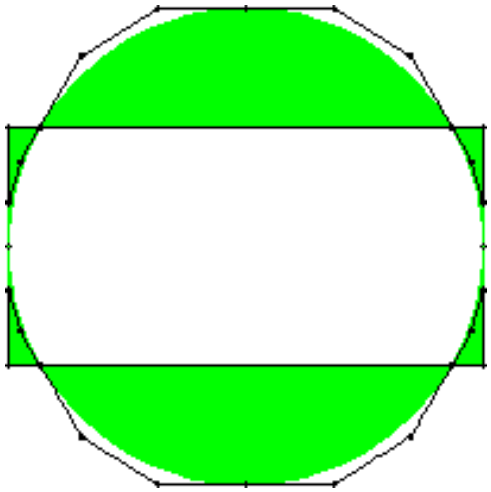

FIGURE 7.9 – L'opération Ou exclusif d'un disque avec un rectangle.

Dans ce cas de figure, on retrouve bien les résultats de l'union du disque *Disk* avec le rectangle *Rectangle* dont l'intersection de ces mêmes figures est soustraite.

7.6 Appartenance d'un point à une surface

Les bordures délimitent les ensembles de valeurs, rassemblées ainsi dans des surfaces fermées.

7.6.1 Équations de la bordure

Segment

Un segment est défini par le point de départ DP et le point final EP selon la courbe paramétrique suivante :

$$b(t) = \begin{cases} x_b(t) = x_{DP} + t \times (x_{EP} - x_{DP}) \\ y_b(t) = y_{DP} + t \times (y_{EP} - y_{DP}) \end{cases} \tag{7.8}$$

Quadratique

Une courbe quadratique est définie par le point de départ DP, le point de contrôle CP et le point final EP selon la courbe paramétrique suivante :

$$b(t) = \begin{cases} x_b(t) = x_{DP} + t \times (-2 \times x_{DP} + 2 \times x_{CP}) + t^2 \times (x_{DP} - 2 \times x_{CP} + x_{EP}) \\ y_b(t) = y_{DP} + t \times (-2 \times y_{DP} + 2 \times y_{CP}) + t^2 \times (y_{DP} - 2 \times y_{CP} + y_{EP}) \end{cases} \tag{7.9}$$

Cubique

Une courbe cubique est définie par le point de départ DP, les points de contrôle $CP1$ et $CP2$ et le point final EP selon la courbe paramétrique suivante :

$$b(t) = \begin{cases} x_b(t) = x_{DP} + t \times (-3 \times x_{DP} + 3 \times x_{CP1}) + t^2 \times (3 \times x_{DP} - 6 \times x_{CP1} \\ \qquad +3 * x_{CP2}) + t^3 \times (-x_{DP} + 3 \times x_{CP1} - 3 \times x_{CP2} + x_{EP}) \\ y_b(t) = y_{DP} + t \times (-3 \times y_{DP} + 3 \times y_{CP1}) + t^2 \times (3 \times y_{DP} - 6 \times y_{CP1} \\ \qquad +3 * y_{CP2}) + t^3 \times (-y_{DP} + 3 \times y_{CP1} - 3 \times y_{CP2} + y_{EP}) \end{cases} \tag{7.10}$$

7.6.2 Appartenance d'un point

L'appartenance d'un point à la surface A revient à vérifier s'il existe un nombre impair de morceaux qui respectent les règles suivantes :
pour tout $b \in A$,
pour tout $t \in [0, 1]$,

1. $y >= y_b(0)$ et $y < y_b(1)$
2. $x < Max(x_b(t))$ et,
3. $x < Min(x_b(t)$
4. ou, pour t' vérifiant $b_y(t') = y$, $x < x_b(t')$.

Une optimisation efficace est de vérifier au préalable que le point étudié fasse partie du rectangle englobant la surface A.

7.6.3 Exemple du rectangle

Retour sur la définition

Reprenons le rectangle *Rectangle* de longueur 2 et largeur 1, centré en $(0, 0)$. Ce rectangle est délimité par 2 segments à partir desquels on peut établir les équations paramétriques (remarquons que le nombre de segments utiles est inférieur au nombre de segments nécessaire à la construction du polygone de contrôle complet) :

– segment $DP = (-1.000, -0.500)$ et $EP = (-1.000, 0.500)$;
– $DP = (-1.0, -0.5)$ et $EP = (-1.0, 0.5)$ donne :

$$b_1(t) = \begin{cases} x_{b_1}(t) = 1 \times (-1.000) + t \times (2.000) \\ y_{b_1}(t) = 1 \times (0.500) + t \times (0.000) \end{cases} \tag{7.11}$$

– segment $DP = (1.000, 0.500)$ et $EP = (1.000, -0.500)$;

$$b_2(t) = \begin{cases} x_{b_2}(t) = 1 \times (1.000) + t \times (0.000) \\ y_{b_2}(t) = 1 \times (0.500) + t \times (-1.000) \end{cases} \tag{7.12}$$

Exemple

Dans le cas du rectangle, vérifions par exemple l'appartenance du point $(0, 0)$ à cette surface. Il faut alors que les inéquations soit vérifiées un nombre impair de fois :

1. pour b_1 :
   $y_{b_1}(0) = -0.5$, $y_{b_1}(t) = 0.5$
   $Max(x_{b_2}(t)) = -1$, $Min(x_{b_2}(t)) = -1$
   (1) est vrai, (2) échoue : le test échoue.
2. pour b_2 :
   $y_{b_2}(t) = -0.5$, $y_{b_2}(t)) = 0.5$
   $Max(x_{b_2}(t)) = 1$, $Min(x_{b_2}(t)) = 1$
   (1) est vrai, (2) est vrai, (3) est vrai ou (4) pour $t' = 0.5$, $y_{b_3}(t') = y$ et $x < x_{b_3}(t') = 1$ est vrai :
   le test est réussi.

7.6.4 Exemple du disque

Retour sur la définition

Reprenons le disque *Disk* de rayon 1 centrée en $(0, 0)$. Ce rectangle est délimité par 4 courbes quadratiques à partir desquelles on peut établir les équations paramétriques :

1. $DP = (0, -1)$, $CP1 = -0.552, -1.0$, $CP2 = -1.0, -0.552$ et $EP = (-1.0, 0.0)$ donne :

$$b_0(t) = \begin{cases} x_{b_0}(t) = 1 \times (0.000) + t \times (-1.657) + t^2 \times (0.314) + t^3 \times (0.343) \\ y_{b_0}(t) = 1 \times (-1.000) + t \times (0.000) + t^2 \times (1.343) + t^3 \times (-0.343) \end{cases} \tag{7.13}$$

2. $DP = (-1.0, 0.0)$, $CP1 = (-1.0, 0.552)$, $CP2 = (-0.552, 1.0)$ et $EP = (0.0, 1.0)$ donne :

$$b_1(t) = \begin{cases} x_{b_1}(t) = 1 \times (-1.000) + t \times (0.000) + t^2 \times (1.343) + t^3 \times (-0.343) \\ y_{b_1}(t) = 1 \times (0.000) + t \times (1.657) + t^2 \times (-0.314) + t^3 \times (-0.343) \end{cases} \tag{7.14}$$

3. $DP = (0.0, 1.0)$, $CP1 = (0.552, 1.0)$, $CP2 = (1.0, 0.552)$ et $EP = (1.0, 0.0)$ donne :

$$b_2(t) = \begin{cases} x_{b_2}(t) = 1 \times (0.000) + t \times (1.657) + t^2 \times (-0.314) + t^3 \times (-0.343) \\ y_{b_2}(t) = 1 \times (1.000) + t \times (0.000) + t^2 \times (-1.343) + t^3 \times (0.343) \end{cases} \tag{7.15}$$

4. $DP = (1.0, 0.0)$, $CP1 = (1.0, -0.552)$, $CP2 = (0.552, -1.0)$ et $EP = (0.0, -1.0)$ donne :

$$b_3(t) = \begin{cases} x_{b_3}(t) = 1 \times (1.000) + t \times (0.000) + t^2 \times (-1.343) + t^3 \times (0.343) \\ y_{b_3}(t) = 1 \times (0.000) + t \times (-1.657) + t^2 \times (0.314) + t^3 \times (0.343) \end{cases} \tag{7.16}$$

Exemple

Appliquons ce cas au disque pour déterminer si le point $(0, 0)$ lui appartient :

1. pour b_0 :
   $Max(y_{b_0}(t)) = 0$, $Min(y_{b_0}(t)) = -1$
   $Max(x_{b_0}(t)) = 0$, $Min(x_{b_0}(t)) = -1$
   (1) est vrai mais (2) est faux : le test échoue.
2. pour b_1 :
   $Max(y_{b_1}(t)) = 1$, $Min(y_{b_1}(t)) = 0$
   $Max(x_{b_1}(t)) = 0$, $Min(x_{b_1}(t)) = -1$
   (1) est faux : le test échoue.
3. pour b_2 :
   $Max(y_{b_2}(t)) = 1$, $Min(y_{b_2}(t)) = 0$
   $Max(x_{b_2}(t)) = 1$, $Min(x_{b_2}(t)) = 0$
   (1) est faux : le test échoue.
4. pour b_3 :
   $Max(y_{b_3}(t)) = 0$, $Min(y_{b_3}(t)) = -1$
   $Max(x_{b_3}(t)) = 1$, $Min(x_{b_3}(t)) = 0$
   (1) est vrai, (2) est vrai, (3) est faux ou (4) pour $t' = 0$, $y_{b_3}(t') = y$ et $x < x_{b_3}(t') = 1$ est vrai : le test est réussi.

7.6.5 Appartenance d'une surface à une autre surface

Montrer qu'une surface A_0 est incluse dans A_1 revient à chercher les intersections entre les bordures de A_0 et A_1 et à vérifier que les bordures de A_0 sont contenues dans A_1. Notons que dans ce cas, la complexité d'une opération d'intersection est $card(B_{A_0}) \times card(B_{A_1})$.

En s'appuyant sur les opérations ensemblistes utilisables sur les surfaces, ce test d'appartenance revient à montrer que $Op_I(A_0, A_1) = A_0$. Deux surfaces sont alors égales si elles partagent les mêmes polygones de contrôle.

7.6.6 Approximations

Les calculs d'intersection et d'appartenance d'un point sont soumis à des problèmes de convergences, provoqués par la recherche numérique des racines des polynômes. Ainsi, dans le cas de l'utilisation de courbes cubiques, il est difficile d'obtenir des calculs stables en temps. J'ai donc choisi d'approximer tous les courbes quadratiques et cubiques par une succession de segments (20 segments pour un disque).

7.7 Conclusion

La géométrie constructive de surface permet donc de représenter les surfaces et de les transformer de manière intuitive. L'intérêt de ce choix est également de repousser le plus tard possible les étapes de discrétisation qui peuvent intervenir dans la résolution d'un problème de planification de trajectoire. Avec cette approche, je fais le lien entre des données symboliques et des ensembles de valeurs continues, nécessaires à la construction de mon approche pour la propagation. En effet, on passe d'un objet symbolique à un continuum de valeurs manipulables dans un planificateur de trajectoire.

Table des figures

Liste des tableaux

Notes

[1] `http://www.universcience.fr/fr/bibliotheque-bsi/contenu/c/1248108361444/des-robots-pour-tout-des-robots-pour-tous/`

[2] Illustration provenant de `http://commons.wikimedia.org/wiki/File:Industrial_robots-transparent.gif`

[3] Illustration provenant de `http://commons.wikimedia.org/wiki/File:Robot-icon.png`

[4] Source : `http://commons.wikimedia.org/wiki/File:Voronoi_diagram.svg`

[5] Image provenant de la page `http://msl.cs.uiuc.edu/rrt/gallery_2dnorot.html`, sur laquelle on peut voir la solution telle qu'elle est parcourue

[6] Voir les vidéos éloquentes de Karaman concernant l'application de RRT au problème du déplacement d'un robot humanoïde `http://sertac.scripts.mit.edu/rrtstar/rrt-algorithm-on-the-pr2-robot/`

[7] Image provenant de `http://eu.gran-turismo.com/au/products/gt5/bspec/`

[8] Les Figures 5.26 et 5.27 sont mises à l'échelle d'un facteur ~ 8 en largeur.

[9] `http://download.oracle.com/javase/1.4.2/docs/api/java/awt/geom/PathIterator.html`

[10] `http://www.cgal.org/`

Bibliographie

[Agha-mohammadi *et al.*, 2011] Agha-mohammadi, A., Chakravorty, S. et Amato, N. (2011). Firm : Feedback controller-based information-state roadmap-a framework for motion planning under uncertainty. *In Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 4284–4291. IEEE.

[Akgun et Stilman, 2011] Akgun, B. et Stilman, M. (2011). Sampling heuristics for optimal motion planning in high dimensions. *In Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 2640–2645. IEEE.

[Ashley-rollman *et al.*, 2007] Ashley-rollman, M. P., Rosa, M. D., Srinivasa, S. S., Pillai, P., Goldstein, S. C. et Campbell, J. (2007). Declarative programming for modular robots.

[Benson et More, 2001] Benson, S. J. et More, J. J. (2001). A limited memory variable metric method in subspaces and bound constrained optimization problems. Rapport technique, in Subspaces and Bound Constrained Optimization Problems.

[Bierstone et Milman, 1988] Bierstone, E. et Milman, P. (1988). Semianalytic and subanalytic sets. *Publications Mathématiques de l'IHÈS*, 67(1):5–42.

[Bohlin et Kavraki, 2000] Bohlin, R. et Kavraki, L. (2000). Path planning using lazy prm. *In Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 1, pages 521–528. Ieee.

[Botea *et al.*, 2004] Botea, A., Müller, M. et Schaeffer, J. (2004). Near optimal hierarchical path-finding. *Journal of Game Development*, 1:7–28.

[Branicky *et al.*, 2006] Branicky, M., Curtiss, M., Levine, J. et Morgan, S. (2006). Sampling-based planning, control and verification of hybrid systems. *IEE Proceedings Control Theory and Applications*, 153(5):575.

[Branicky *et al.*, 2008] Branicky, M., Knepper, R. et Kuffner, J. (2008). Path and trajectory diversity : Theory and algorithms. *In ICRA'08*, pages 1359–1364.

[Brock, 1999] Brock, O. (1999). *Generating Robot Motion*. Thèse de doctorat, stanFoRd unIvERsIty.

[Brock, 2000] Brock, O. (2000). *Generating Robot Motion : The Integration of Planning and Execution*. Thèse de doctorat, Department of Computer Science, Stanford University, Stanford, CA, USA.

[Brock et Khatib, 1997] Brock, O. et Khatib, O. (1997). Elastic strips : Real-time path modification for mobile manipulation. *Citeseer*.

[Brock *et al.*, 2002] Brock, O., Khatib, O. et Viji, S. (2002). Task-consistent obstacle avoidance and motion behavior for mobile manipulation. *In Proceedings of IEEE International Conference on Robotics and Automation*, pages 388–393.

[Brooks, 1985] Brooks, R. A. (1985). A robust layered control system for a mobile robot. *MIT A.I. Lab Memo*, 1(864).

[Canny, 1988] Canny, J. (1988). *The complexity of robot motion planning*. The MIT Press.

[Choset, 2005] Choset, H. (2005). *Principles of robot motion : theory, algorithms, and implementation*. The MIT Press.

[Decker, 1996] Decker, K. (1996). TAEMS : A Framework for Environment Centered Analysis & Design of Coordination Mechanisms. *Foundations of Distributed Artificial Intelligence, Chapter 16*, pages 429–448.

[Decker et Lesser, 1995] Decker, K. et Lesser, V. (1995). Designing a Family of Coordination Algorithms. *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, pages 73–80.

[Defoort, 2007] Defoort, M. (2007). *Contributions à la planification et à la commande pour les robots mobiles coopératifs*. Thèse de doctorat, Thèse en Automatique et Informatique Industrielle, 22 octobre 2007. Encadrants : W. Perruquetti, T. Floquet et AM Kokosy.

[Defoort *et al.*, 2007] Defoort, M., Palos, J., Kokosy, A., Floquet, T., Perruquetti, W. et Boulinguez, D. (2007). Experimental motion planning and control for an autonomous nonholonomic mobile robot. *In ICRA'07*, pages 2221–2226.

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

[Dinont *et al.*, 2010] Dinont, C., Gaillard, F. et Soulignac, M. (2010). Apm(robot) : Towards a platform for meta-reasoning in robotic applications. *In Proceedings 5th National Conference on Control Architecture of Robots*.

[Donald *et al.*, 1993] Donald, B., Xavier, P., Canny, J. et Reif, J. (1993). Kinodynamic motion planning. *Journal of the ACM*, 40(5):1048–1066.

[Duc *et al.*, 2008] Duc, L., Sidhu, A. S. et Chaudhari, N. S. (2008). Hierarchical pathfinding and ai-based learning approach in strategy game design. *Int. J. Comput. Games Technol.*, 2008:1–11.

[Ferguson *et al.*, 2006] Ferguson, D., Kalra, N. et Stentz, A. (2006). Replanning with rrts. *In Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 1243–1248. IEEE.

[Ferguson et Stentz, 2006] Ferguson, D. et Stentz, A. (2006). Anytime rrts. *In International Conference on Intelligent Robots and Systems, Beijing, China*, pages 5369–5375.

[Finkel et Bentley, 1974] Finkel, R. A. et Bentley, J. L. (1974). Quad trees : a data structure for retrieval on composite keys. *Acta Informatica*, 4:1–9. 10.1007/BF00288933.

[Fliess *et al.*, 1995] Fliess, M., Lévine, J. et Rouchon, P. (1995). Flatness and defect of nonlinear systems : Introductory theory and examples. *International Journal of Control*, 61:1327–1361.

[Fourer *et al.*, 2002] Fourer, R., Kernighan, B. W. et Gay, D. M. (2002). *AMPL : A Modeling Language for Mathematical Programming*. Duxbury Press.

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

[Frazzoli *et al.*, 2002] Frazzoli, E., Dahleh, M. et Feron, E. (2002). Real-time motion planning for agile autonomous vehicles. *Journal of Guidance Control and Dynamics*, 25(1).

[Fredman et Tarjan, 1987] Fredman, M. L. et Tarjan, R. E. (1987). Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, 34:596–615.

[Gaillard *et al.*, 2010] Gaillard, F., Dinont, C., Soulignac, M. et Mathieu, P. (2010). Deterministic kinodynamic planning. *In Proceedings of the Eleventh AI*IA Symosium on Artificial Intelligence*, pages 54–61.

[Gaillard *et al.*, 2012] Gaillard, F., Dinont, C., Soulignac, M. et Mathieu, P. (2012). Towards cognitive steering behaviours for two-wheeled robots. *In ICAART'2012 proceedings.*

[Gaillard *et al.*, 2011] Gaillard, F., Soulignac, M., Dinont, C. et Mathieu, P. (2011). Deterministic kinodynamic planning with hardware demonstrations. *In Proceedings of the International Conference on Intelligent Robots and Systems (IROS).* IEEE/RS.

[Garvey et Lesser, 1993] Garvey, A. et Lesser, V. (1993). Design-to-time Real-Time Scheduling. *IEEE Transactions on Systems, Man and Cybernetics, Special Issue on Planning, Scheduling and Control*, 23(6):1491–1502.

[Gerkey *et al.*, 2003] Gerkey, B., Vaughan, R. et Howard, A. (2003). The player/stage project : Tools for multi-robot and distributed sensor systems. *In ICRA'03*, pages 317–323.

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

[Horling *et al.*, 1999] Horling, B., Lesser, V., Vincent, R., Wagner, T., Raja, A., Zhang, S., Decker, K. et Garvey, A. (1999). The TAEMS White Paper.

[Howard et Kelly, 2007] Howard, T. et Kelly, A. (2007). Optimal rough terrain trajectory generation for wheeled mobile robots. *IJRR*, 26(1):141–166.

[Hsu *et al.*, 1998] Hsu, D., Kavraki, L., Latombe, J., Motwani, R., Sorkin, S. *et al.* (1998). On finding narrow passages with probabilistic roadmap planners. *In Proc. Int. Workshop on Algorithmic Foundations of Robotics (WAFR)*, volume 19, pages 2577–2637. Citeseer.

[Hsu *et al.*, 2002] Hsu, D., Kindel, R., Latombe, J. et Rock, S. (2002). Randomized kinodynamic motion planning with moving obstacles. *The International Journal of Robotics Research*, 21(3):233.

[Jaesik et Eyal, 2009] Jaesik, C. et Eyal, A. (2009). Combining planning and motion planning. *2009 IEEE International Conference on Robotics and Automation.*

[Jaillet *et al.*, 2010] Jaillet, L., Corteands, J. et Simeandon, T. (2010). Sampling-based path planning on configuration-space costmaps. *Robotics, IEEE Transactions on*, 26(4):635 –646.

[Jaillet *et al.*, 2011] Jaillet, L., Hoffman, J., van den Berg, J., Abbeel, P., Porta, J. et Goldberg, K. (2011). Eg-rrt : Environment-guided random trees for kinodynamic motion planning with uncertainty and obstacles. *In Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 2646–2652. IEEE.

[Jones, 2004] Jones, J. L. (2004). Robot programming, a practical guide to behavior based robotics.

[Karaman et Frazzoli, 2009] Karaman, S. et Frazzoli, E. (2009). Sampling-based motion planning with deterministic μ-calculus specifications. *In Decision and Control, 2009 held jointly with the 2009 28th Chinese Control Conference. CDC/CCC 2009. Proceedings of the 48th IEEE Conference on*, pages 2222–2229. IEEE.

[Karaman et Frazzoli, 2010] Karaman, S. et Frazzoli, E. (2010). Incremental sampling-based algorithms for optimal motion planning. *Proceedings of Robotics : Science and Systems, Zaragoza, Spain*.

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

[Khatib, 1986] Khatib, O. (1986). Real-time obstacle avoidance for manipulators and mobile robots. *The international journal of robotics research*, 5(1):90.

[Kim et Kim, 2009] Kim, J. et Kim, D. (2009). Visibility graph path planning using a quad-tree. *Proceedings of the 9th POSTECH-KYUTECH Joint Workshop On Neuroinformatics*.

[Knepper et Mason, 2011] Knepper, R. et Mason, M. (2011). Improved hierarchical planner performance using local path equivalence. *In Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 3856–3861. IEEE.

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

[Krogh et Thorpe, 1986] Krogh, B. et Thorpe, C. (1986). Integrated path planning and dynamic steering control for autonomous vehicles. *In Robotics and Automation. Proceedings. 1986 IEEE International Conference on*, volume 3, pages 1664–1669. IEEE.

[Kuffner *et al.*, 2005] Kuffner, J., Nishiwaki, K., Kagami, S., Inaba, M. et Inoue, H. (2005). Motion planning for humanoid robots. *Robotics Research*, pages 365–374.

[Lacroix *et al.*, 2009] Lacroix, B., Mathieu, P. et Kemeny, A. (2009). Generating various and consistent behaviors in simulations. *In Proceedings of the 7th International conference on Practical Applications of Agents and Multi-Agents Systems (PAAMS'2009)*, volume 55 de *Practical Advances in Intelligent and soft computing*, pages 110–119. Springer.

[Lacroix *et al.*, 2007] Lacroix, B., Mathieu, P., Rouelle, V., Chaplier, J., Gallée, G. et Kemeny, A. (2007). Towards traffic generation with individual driver behavior model based vehicles. *In Proceedings of DSC-NA'07*, pages 144–154.

[Ladd et Kavraki, 2005] Ladd, A. et Kavraki, L. (2005). Fast tree-based exploration of state space for robots with dynamics. *Algorithmic Foundations of Robotics VI*, pages 297–312.

[Lallée *et al.*, 2011] Lallée, S., Pattacini, U., Boucher, J.-D., Lemaignan, S., Lenz, A., Melhuish, C., Natale, L., Skachek, S., Hamann, K., Steinwender, J., Sisbot, E. A., Metta, G., Alami, R., Warnier, M., Guitton, J., Warneken, F. et Dominey, P. F. (2011). Towards a platform-independent cooperative human-robot interaction system : Ii. perception, execution and imitation of goal directed actions. *In IROS*, pages 2895–2902.

[Lamiraux *et al.*, 2004] Lamiraux, F., Ferré, E. et Vallée, E. (2004). Kinodynamic motion planning : connecting exploration trees using trajectory optimization methods. *In ICRA'04. Proceedings*, volume 4, pages 3987–3992.

[Lamiraux et Laumond, 1998] Lamiraux, F. et Laumond, J. (1998). A practical approach to feedback control for a mobile robot with trailer. *In ICRA'98*, pages 3291–3296.

[Latombe, 1990] Latombe, J.-C. (1990). *Robot motion planning*. Springer Verlag.

[LaValle *et al.*, 2004] LaValle, S., Branicky, M. et Lindemann, S. (2004). On the relationship between classical grid search and probabilistic roadmaps. *The International Journal of Robotics Research*, 23(7-8):673–692.

[LaValle et Kuffner, 2001] LaValle, S. et Kuffner, J. (2001). Randomized kinodynamic planning. *IJRR*, 20:278–400.

[LaValle, 2006] LaValle, S. M. (2006). *Planning Algorithms*. Cambridge University Press, Cambridge, U.K. Available at http ://planning.cs.uiuc.edu/.

[LaValle et Konkimalla, 2001] LaValle, S. M. et Konkimalla, P. (2001). Algorithms for computing numerical optimal feedback motion strategies. *International Journal of Robotics Research*, 20(9):729–752.

[Lawrence *et al.*, 1994] Lawrence, C., Zhou, J. et Tits, A. (1994). User's guide for CFSQP version 2.5 : AC code for solving (large scale) constrained nonlinear (minimax) optimization problems, generating iterates satisfying all inequality constraints. *Institute for Systems Research TR*, 94:16r1.

[Lekavy et Návrat, 2007] Lekavy, M. et Návrat, P. (2007). *Expressivity of STRIPS-Like and HTN-Like Planning*, volume 4496/2007 de *Lecture Notes in Computer Science*, pages 121–130. Springer Berlin / Heidelberg, Berlin, Heidelberg.

[Lemaignan *et al.*, 2010] Lemaignan, S., Ros, R., Mösenlechner, L., Alami, R. et Beetz, M. (2010). Oro, a knowledge management platform for cognitive architectures in robotics. *In IROS*, pages 3548–3553.

[Likhachev et Ferguson, 2009] Likhachev, M. et Ferguson, D. (2009). Planning long dynamically feasible maneuvers for autonomous vehicles. *IJRR*, 28(8):933.

[Lozano-Perez *et al.*, 1987] Lozano-Perez, T., Jones, J., Mazer, E., O'Donnell, P., Grimson, W., Tournassoud, P. et Lanusse, A. (1987). Handey : A robot system that recognizes, plans, and manipulates. *In Robotics and Automation. Proceedings. 1987 IEEE International Conference on*, volume 4, pages 843–849. IEEE.

[Mainprice *et al.*, 2011] Mainprice, J., Sisbot, E. A., Jaillet, L., Cortés, J., Alami, R. et Siméon, T. (2011). Planning human-aware motions using a sampling-based costmap planner. *In ICRA*, pages 5012–5017.

[Marble et Bekris, 2011] Marble, J. et Bekris, K. (2011). Computing spanners of asymptotically optimal probabilistic roadmaps. *In Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 4292–4298. IEEE.

[McDermott *et al.*, 1998] McDermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., Weld, D. et Wilkins, D. (1998). Pddl-the planning domain definition language. *The AIPS-98 Planning Competition Comitee*.

[Merlet, 2001] Merlet, J.-P. (2001). An improved design algorithm based on interval analysis for spatial parallel manipulator with specified workspace. *In ICRA*, pages 1289–1294.

[Milam *et al.*, 2000] Milam, M., Mushambi, K. et Murray, R. (2000). A new computational approach to real-time trajectory generation for constrained mechanical systems. *In CDC'00*, pages 845–851.

[Moore, 1966] Moore, R. E. (1966). *Interval Analysis*. Prentice-Hall.

[Mori, 1970] Mori, M. (1970). Bukimi no tani [The uncanny valley]. *Energy*, 7(4):33–35.

[Nielsen, 2008] Nielsen, F. (2008). An interactive tour of voronoi diagrams on the gpu. *In ShaderX*^{6} *: Advanced Rendering Techniques*. Charles River Media (`http://www.charlesriver.com/`).

[Plaku *et al.*, 2005] Plaku, E., Bekris, K. E., Chen, B. Y., Ladd, A. M. et Kavraki, L. E. (2005). Sampling-based roadmap of trees for parallel motion planning. *TRO*, 21(4):597–608.

[Plaku *et al.*, 2007] Plaku, E., Kavraki, L. et Vardi, M. (2007). Discrete search leading continuous exploration for kinodynamic motion planning. *In Robotics : Science and Systems*, pages 326–333.

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

[Quinlan et Khatib, 1993] Quinlan, S. et Khatib, O. (1993). Elastic bands : Connecting path planning and control. *In In Proceedings of the International Conference on Robotics and Automation*, pages 802–807.

[Reif, 1987] Reif, J. (1987). Complexity of the generalized mover's problem. *In* Schwartz, J., Sharir, M. et Hopcroft, J., éditeurs : *Planning, Geometry, and Complexity of Robot Motion*, pages 267–281. Ablex Publishing Corporation.

[Reynolds, 1999] Reynolds, C. (1999). Steering behaviors for autonomous characters. *In Game Developers Conference. http ://www. red3d. com/cwr/steer/gdc99*. Citeseer.

[Soulignac, 2009] Soulignac, M. (2009). *Planification de trajectoire en présence de courants : application aux missions de drones*.

[Stentz, 1995] Stentz, A. (1995). The focussed d* algorithm for real-time replanning. *In In Proceedings of ICRA*, pages 1652–1659.

[Stilman et Kuffner, 2004] Stilman, M. et Kuffner, J. (2004). Navigation among movable obstacles : Real-time reasoning in complex environments. *In Proceedings of Humanoids'04*, pages 322 – 341.

[Takahashi et Schilling, 1989] Takahashi, O. et Schilling, R. (1989). Motion planning in a plane using generalized voronoi diagrams. *Robotics and Automation, IEEE Transactions on*, 5(2):143–150.

[Urmson *et al.*, 2006] Urmson, C., Ragusa, C., Ray, D., Anhalt, J., Bartz, D., Galatali, T., Gutierrez, A., Johnston, J., Harbaugh, S., Messner, W. *et al.* (2006). A robust approach to high-speed navigation for unrehearsed desert terrain. *Journal of Field Robotics*, 23(8):467–508.

[Wagner et Lesser, 2000] Wagner, T. et Lesser, V. (2000). Design-to-criteria scheduling : Real-time agent control. *In Infrastructure for Agents, Multi-Agent Systems, and Scalable Multi-Agent Systems, LNCS*, pages 89–96. Springer-Verlag.

[Wilkie *et al.*, 2009] Wilkie, D., van den Berg, J. et Manocha, D. (2009). Generalized velocity obstacles. *In IROS*, pages 5573–5578.