Planification de mouvements sans collision pour robots non holonomes

Stage de maîtrise d'informatique effectué au L.A.A.S. (Toulouse)
juillet-septembre 2000

Responsables : Florent Lamiraux et Jean-Paul Laumond

Élodie-Jane Sims.

M.M.F.A.I.

Chapitre 1

Introduction

1.1 Présentation générale

Mon stage s'est déroulé au Laboratoire d'Analyse et d'Architecture des Systèmes (L.A.A.S.-C.N.R.S.) dans le groupe Robotique et Intelligence Artificielle (R.I.A.).

1.1.1 La robotique

L'objectif d'une partie de la recherche en robotique est de réaliser des machines à grande autonomie, capables de travailler dans des environnements qui ne soient pas conçus pour elles. De telles machines doivent posséder des capacités de perception, de raisonnement, de déplacement et d'action sur le monde qui les entoure.

Notre travail s'inscrit dans le domaine de la planification de mouvements dont le but est de calculer des déplacements pour des robots.

1.1.2 La planification de mouvement

La problème de la planification de mouvement consiste à calculer automatiquement un chemin entre deux configurations d'un robot dans un environnement statique connu.

La position d'un robot dans son *espace de travail*, appelée *configuration*, peut-être représentée par un vecteur de paramètres $(q_1, ..., q_d)$. Par exemple, une configuration d'une voiture (voir fig. 1.1) peut être (x, y, θ, ϕ), x et y étant les coordonnées du milieu de l'essieu arrière, θ l'orientation de l'essieu arrière et ϕ l'angle des roues avant.

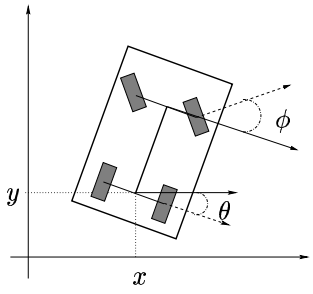

FIG. 1.1: *Configuration d'une voiture*

L'ensemble de ces positions est appelé *espace des configurations* $\mathcal{CS}$. Certaines parties de l'espace de travail du robot sont occupées par des obstacles. On appelle *espace des configurations libres* $\mathcal{CS}_{libre}$ l'ensemble des configurations pour lesquelles l'intersection entre le robot et les obstacles est vide.

Contraintes non holonomes En plus des contraintes physiques imposant au système de rester dans $\mathcal{CS}_{libre}$, des contraintes *non holonomes* s'appliquent aux vitesses du système. L'espace des vitesses accessibles a une dimension inférieure à celle de l'espace des configurations. Par exemple un robot est soumis à des contraintes de roulement sans glissement. Une configuration d'un robot a trois paramètres (x, y, θ) alors que sa vitesse n'en a que deux : une vitesse de rotation autour de son centre, et une vitesse linéaire dans la direction de ses roues (voir Fig. 1.2).

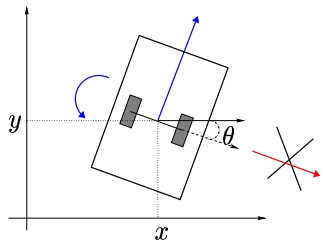

FIG. 1.2: *Non holonomie*

La planification de mouvement doit donc trouver un chemin dans $\mathcal{CS}_{libre}$ en respectant les contraintes non holonomes.

1.2 Le cadre du stage

1.2.1 Les systèmes

Nous nous intéressons au calcul des déplacements de trois systèmes particuliers. Les deux premiers sont constitués d'un robot mobile à deux roues tirant une remorque, mais dans un cas l'attache de la remorque est située au-dessus de l'axe des roues motrices (Fig. 1.7), alors que dans l'autre, elle est située derrière le véhicule tracteur (Fig. 1.6). Le troisième système est une voiture (Fig. 1.8).

Une configuration d'un robot à deux roues motrices peut être représentée par l'abscisse et l'ordonnée de son centre, et son orientation *(l'angle qu'il fait avec l'axe des abscisses)* $((x, y, \theta)$ Fig. 1.2).

Une configuration du système robot-remorque peut être représentée par les paramètres de configuration de la remorque et l'angle robot-remorque $((x, y, \theta, \phi)$ Fig. 1.6 et 1.7).

Comme on le voit sur le dessin, une voiture peut être modélisée par un système robot-remorque à attache centrée[1], l'essieu avant constitue le robot et l'essieu arrière avec la carcasse constitue la remorque (voir Fig 1.8)[2].

Ces systèmes en plus de contraintes de non glissement ont des contraintes d'*angle limite* : pour les robots à remorque l'angle robot-remorque a une borne que l'on peut appeler angle de braquage

1. le point d'attache de la remorque au robot est situé au milieu des deux roues du robot

maximal, de même les roues avant de la voiture ont un angle limite. Lors de la planification nous devons donc trouver des chemins respectant ces limites tout le long.

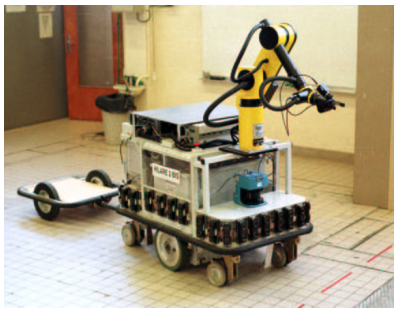

FIG. 1.3: *Hilare et sa remorque décentrée* (un robot du L.A.A.S.)

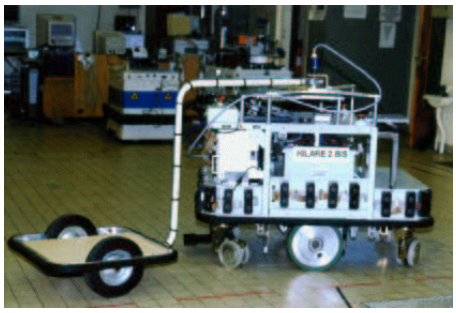

FIG. 1.4: *Hilare et sa remorque centrée* (un robot du L.A.A.S.)

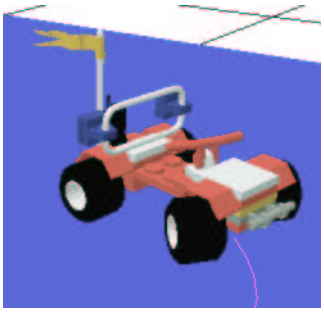

FIG. 1.5: *Lego-Car* (le modèle de voiture utilisé)

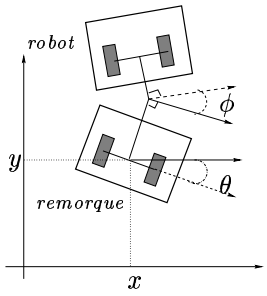

FIG. 1.6: *Robot-remorque attache décentrée*

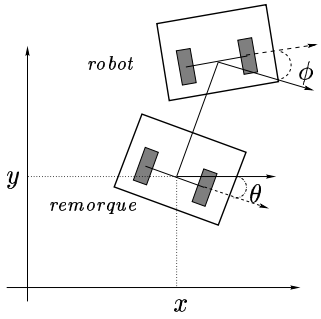

FIG. 1.7: *Robot-remorque attache centrée*

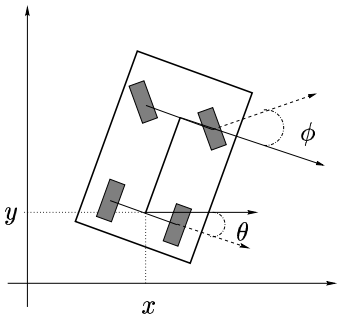

FIG. 1.8: *Voiture*

1.2.2 Move3D

Move3D *(présentée au Chap. 2)* est une plate-forme logicielle générique dédiée à la planification de mouvement dans des environnements tridimensionnels. Elle est basée sur un algorithme aléatoire *(qui assure s'il existe un chemin de le trouver – éventuellement en un temps infini)*.

Le principe :
- un planificateur local fournit un chemin local entre les deux configurations à relier
- un test de collision vérifie si le chemin est faisable
- si oui on a réussi à trouver un chemin
- si non on tire des configurations aléatoirement dans l'environnement que l'on essaie de relier de la même façon,

on crée ainsi un graphe et on s'arrête lorsque les configurations d'arrivée et de départ appartiennent à la même composante connexe.

L'algorithme de Move3D, si on lui fournit un planificateur local adapté à notre système, permet de trouver un chemin pour ce système.

Un *planificateur local* est une méthode qui construit un chemin cinématiquement faisable (i.e. qui respecte les contraintes non-holonomes) entre tout couple de configurations. Il ne tient pas compte des obstacles.

1.2.3 But du stage

Jusqu'à présent Move3D planifiait des mouvements 3D sans collision en suivant des courbes rectilignes ou circulaires. Pour les robots à remorques, ces mouvements ne permettaient pas de respecter toutes les contraintes *non holonomes*. Par ailleurs, des chemins non holonomes permettant de trouver des chemins effectuables par des robots à remorques avaient déjà été implémentés dans une plate-forme 2D au LAAS (voir Chap. 3).

Le but de mon stage était l'intégration de ces chemins *non holonomes* adaptés aux systèmes robot-remorque sous forme d'un planificateur local pour Move3D.

Chapitre 2

Move3D

Move3D[1] est une plate-forme logicielle générique dédiée à la planification de mouvement dans des environnements tridimensionnels. Elle peut être utilisée en CAO, en animation graphique et bien entendu en robotique. Vous pouvez voir ci-dessous quelques exemples d'utilisation de cette plate-forme.

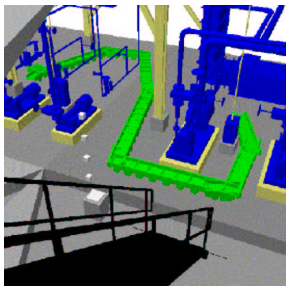

FIG. 2.1: *Mouvement d'un chariot dans une usine*

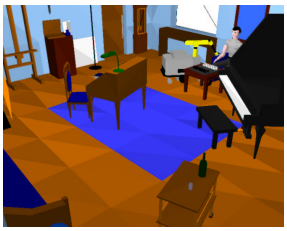

FIG. 2.2: *Mouvement d'un personnage dans un salon*

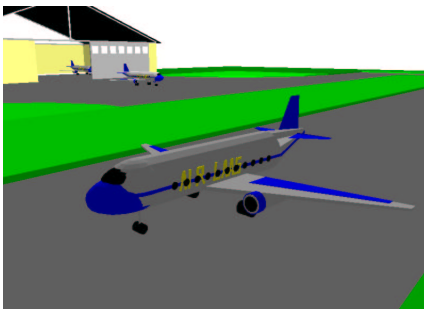

FIG. 2.3: *Planification multi-robot dans un aéroport*

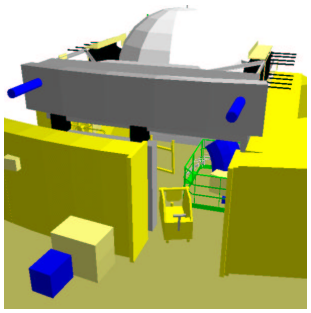

FIG. 2.4: *Centrale d'E.D.F.*

Comme il m'a été dit, Move3D est une "usine à gaz". Elle permet si on lui fournit un environnement, un système et une méthode locale qui lui est associée de planifier des mouvements sans collision.

L'algorithme principal est un algorithme aléatoire complet *(il assure s'il existe un chemin de le trouver - éventuellement en un temps infini!)*. Il repose sur l'élaboration d'un graphe (Fig. 2.5) des configurations dans $\mathcal{CS}_{libre}$ à l'aide de la méthode locale. Les noeuds du graphe sont des

1. `http://www.laas.fr/~nic/Move3D`

configurations sans collision tirées aléatoirement. On relie deux noeuds du graphe si le chemin local qui les joint est sans collision. L'algorithme se termine lorsque les configurations de départ et d'arrivée appartiennent à une même composante connexe du graphe.

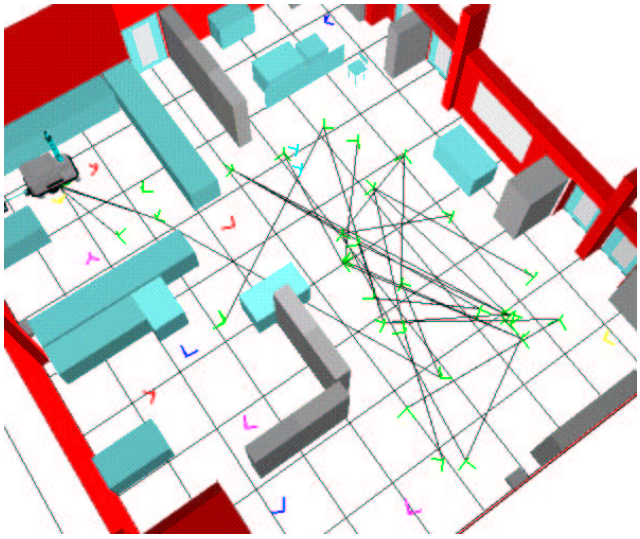

FIG. 2.5: *Exemple d'un graphe des configurations*

Ajoutées à cela, Move3D comporte différentes options permettant une optimisation du temps de création du graphe, du temps d'exploration (une fois les deux configurations reliées à une même composante du graphe il faut parcourir le graphe pour retrouver le chemin les reliant) ainsi qu'une réduction de la taille du graphe.

La principale option est basée sur un algorithme de visibilité. Il s'agit lors du tirage aléatoire des configurations de ne retenir que celles apportant une "information pertinente" au graphe déjà existant. Bien que le fonctionnement de Move3D soit intéressant, je ne vais pas approfondir cette description, mon stage n'ayant pas porté sur ces algorithmes.

Nous avons vu que les méthodes locales devaient vérifier des contraintes du système, en particulier les contraintes non holonomes. L'algorithme de Move3D impose également d'autres contraintes. En effet, nous avons dit qu'il était complet, pour cela la méthode locale doit rendre compte de la propriété de*commandabilité en espace petit* du système.

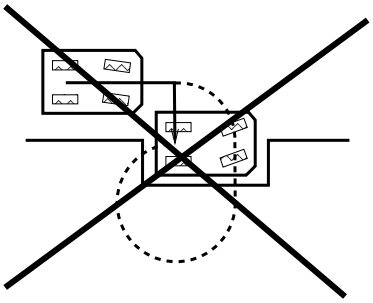

FIG. 2.6: *Pas commandabilité en espace petit*

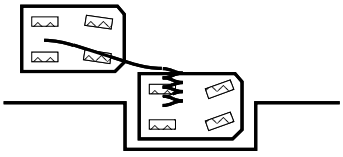

FIG. 2.7: *Commandabilité en espace petit*

Definition 1 *Un système est dit* localement commandable en espace petit *si pour tout voisinage U d'une configuration q, l'ensemble des configurations accessibles par un chemin inclus dans U est un voisinage de q.*

On dit qu'une méthode locale rend compte de la commandabilité en espace petit si plus deux configurations sont proches, plus le chemin fourni par la méthode locale est court. Cette propriété assure que si la solution traverse un passage arbitrairement étroit dans l'espace des configurations, l'algorithme parviendra à connecter des configurations tirées aléatoirement dans ce passage (avec probabilité convergente vers 1) (voir Fig. 2.6 et 2.7).

Notre travail a donc consisté à fournir à Move3D une méthode locale pour nos trois systèmes vérifiant toutes les contraintes énoncées ci-dessus.

Cette plate-forme logicielle a été développée dans le cadre de sa thèse par Carole Nissoux [3].

Chapitre 3

Une méthode locale basée sur la platitude différentielle

Nous voulons construire une méthode locale pour des systèmes robot-remorque, c'est à dire trouver un chemin reliant deux configurations données. Il n'existe pas de méthode générale permettant de calculer la commande à donner à un système non linéaire pour l'amener d'une configuration à une autre. Cela n'a rien d'étonnant dans la mesure où le problème inverse consistant à calculer la position finale d'un système soumis à une entrée connue et partant d'un point donné n'a pas non plus de solution générale. Pour certains types de systèmes (en l'occurence pour les robots à remorque) des solutions ont cependant été proposées.

3.1 Platitude

La méthode que nous avons utilisée repose sur la propriété de *platitude différentielle* du système robot-remorque.

Definition 2 *Un système $q \in \mathbb{R}^n$ est dit* plat *s'il existe une fonction $y = h(q)$, appelée sortie plate, à valeurs dans $\mathbb{R}^m$ et une fonction A telle que*

$$q = A(y_1, ..., y_1^{(\alpha_1)}, ..., y_m, ..., y_m^{(\alpha_m)})$$

avec $\{y_1, .., y_m\}$ différentiellement indépendants.

La propriété de platitude est telle que tout chemin de la sortie plate dans son espace, correspond à un chemin faisable du système dans son espace des configurations. La donnée de la valeur de la sortie plate et de ses dérivées permet de calculer l'état du système sans intégrer aucune équation différentielle.

Illustrons cette propriété dans le cas du robot-remorque à attache centrée (Fig. 3.1). Soit $y = (y_1, y_2)$ le milieu de l'axe des roues de la remorque. Si $y(s)$ désigne la courbe décrite par y, l'orientation du vecteur tangent à $y(s)$ donne l'orientation de la remorque :

$$\theta = \arctan \frac{\dot{y}_2}{\dot{y}_1}.$$

En dérivant une fois de plus cette expression, on peut montrer :

$$\phi = l \arctan \kappa$$

où κ est la courbure de $y(s)$ et l la longueur de l'attache de la remorque.

On peut donc reconstruire la trajectoire du système à partir de celle du centre de la remorque, y est donc la sortie plate.

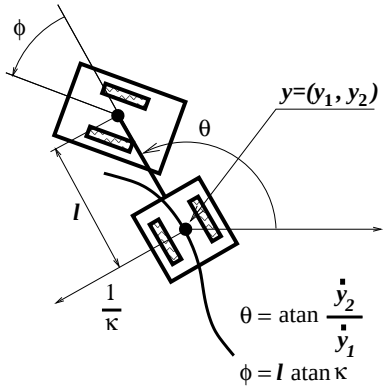

FIG. 3.1: *Sortie plate*

Dans le cas de l'attache décentrée, nous avons un résultat analogue mais la sortie plate n'est située en aucun point particulier de la remorque ou du robot, et sa position par rapport au système varie suivant l'angle que la remorque fait avec le robot *(donc tout le long du chemin en général).*

Pour un système plat, trouver un chemin entre deux configurations en respectant les contraintes non holonomes, revient donc à trouver une courbe avec des contraintes sur les dérivées aux extrémités

3.2 Méthode locale

Dans notre cas nous voulons construire une courbe reliant deux points ayant comme contraintes aux extrémités la tangente à la courbe et la courbure.

3.2.1 Courbes canoniques

Notre méthode repose sur la notion de courbe canonique associée à une configuration. La *courbe canonique* est l'unique courbe passant par cette configuration et gardant la courbure constante. Cette courbe correspond à celle que l'on obtiendrait si on soudait la remorque au robot et si l'on faisait avancer le robot. Il s'agit d'un cercle, ou d'une droite si la courbure est nulle (Fig. 3.2).

3.2.2 Combinaison convexe des courbes canoniques

On utilise alors une fonction de pondération α pour effectuer une combinaison convexe de ces deux courbes canoniques (Fig. 3.3). α définie et croissante de $[0,1]$ dans $[0,1]$ vérifiant :

$$\begin{array}{lll} \alpha(0) = 0 & \alpha'(0) = 0 & \alpha''(0) = 0 \\ \alpha(1) = 1 & \alpha'(1) = 0 & \alpha''(1) = 0 \end{array} \tag{3.1}$$

Appelons $\gamma_1(u)$ (resp. $\gamma_2(u)$) la courbe canonique passant par la configuration initiale lorsque $u = 0$ (resp. par la configuration finale quand $u = 1$) (Fig. 3.2).

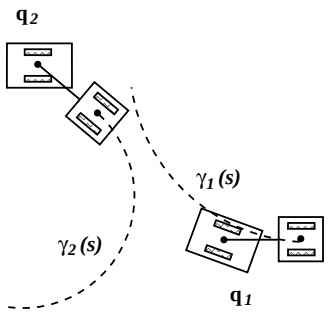

FIG. 3.2: *Courbes canoniques, cas attache centrée*

On définit la courbe suivante (paramétrée de 0 à 1) :

$$\gamma(u) = (1 - \alpha(u)).\gamma_1(u) + \alpha(u).\gamma_2(u)$$

On vérifie facilement que :

$$\begin{array}{ll} \gamma(0) = \gamma_1(0) & \gamma(1) = \gamma_2(1) \\ \gamma'(0) = \gamma_1'(0) & \gamma'(1) = \gamma_2'(1) \\ \gamma''(0) = \gamma_1''(0) & \gamma''(1) = \gamma_2''(1) \end{array} \tag{3.2}$$

Donc γ a la même tangente et courbure que γ_1 en 0 et que γ_2 en 1, ainsi lorsque la sortie plate parcourt γ entre $[0, 1]$, le système exécute un chemin entre q_1 et q_2.

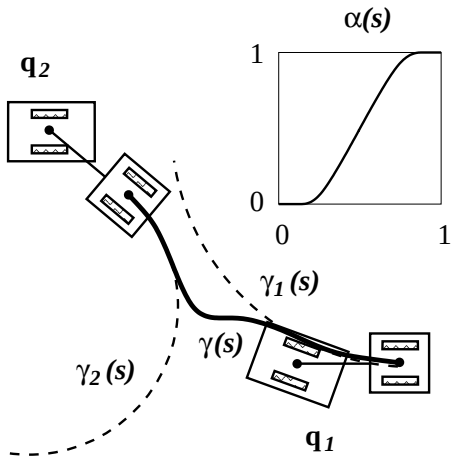

FIG. 3.3: *Interpolation des courbes canoniques*

3.2.3 Paramétrage des courbes canoniques

Le choix du paramétrage, bien qu'arbitraire, est guidé par la volonté que le système respecte la propriété de *commandabilité en espace petit* (Déf. 1).

L'idée consiste à rester autant que possible près des courbes canoniques. Par exemple si le point d'arrivée se trouve sur la courbe canonique du point de départ, la trajectoire engendrée sera exactement cette courbe canonique. Pour cela, on appelle v la distance sur la courbe canonique

du point de départ au projeté sur cette courbe du point d'arrivée (Fig. 3.4).

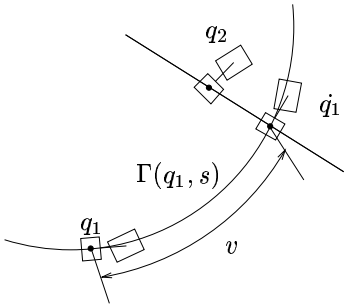

FIG. 3.4: *Calcul de v*

Si l'on appelle $\Gamma(q, s)$ la configuration sur la courbe canonique de q d'abscisse curviligne s, nous pouvons alors exprimer nos courbes canoniques par :

$$\gamma_1(u) = \Gamma(q_1, u.v)$$
$$\gamma_2(u) = \Gamma(q_2, (u-1).v)$$

On remarque que si $q_2 \in \Gamma(q_1, s)$ alors $\gamma_1(u) = \gamma_2(u) = \gamma(u)$.

Reste maintenant à trouver une fonction α aussi simple que possible mais vérifiant les conditions citées plus haut.

3.2.4 Choix de α

Naturellement le plus simple est de prendre un polynôme. Nous avons six contraintes (d'après les formules 3.1), nous devons donc prendre un polynôme de degré cinq : $\alpha(u) = a_0 + a_1.u + a_2.u^2 + a_3.u^3 + a_4.u^4 + a_5.u^5$, puis en résolvant le système (3.1) nous obtenons :

$$\alpha(u) = 10.u^3 - 15.u^4 - 6.u^5$$

Nous avons donc une trajectoire associée à un couple configuration de départ-configuration d'arrivée.

3.2.5 Points de rebroussement

Comme évoqué dans le chapitre 2, la méthode locale doit rendre compte de la commandabilité en temps petit du système pour assurer la convergence de la recherche. En d'autres termes, plus les configurations à joindre sont proches, plus le chemin produit par la méthode locale doit rester proche (dans l'espace des configurations) de ces configurations.

Examinons l'exemple décrit dans la figure 3.5. Les configurations initiale et finales sont côte-à-côte. La combinaison convexe entre ces configurations présente des variations angulaires du vecteur tangent supérieures à $\frac{\pi}{2}$, quel que soit l'écart latéral h. Donc notre méthode locale ne rend pas compte de la commandabilité en espace petit.

Pour résoudre ce problème, il faut ajouter un point de rebroussement. La méthode utilisée avant mon stage consistait à choisir une abscisse v sur la courbe canonique γ_2, puis à joindre q_1 et $\gamma_2(v)$ par une combinaison convexe et $\gamma_2(v)$ à q_2 en suivant la courbe canonique γ_2. v dépend

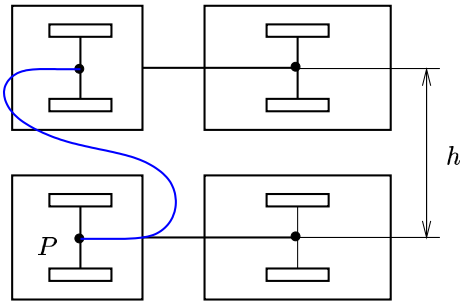

FIG. 3.5: *Problème du créneau*

de la distance entre q_1 et q_2. Il doit tendre vers 0 lorsque q_1 tend vers q_2, mais pas trop vite pour éviter de se retrouver dans des cas similaires à la figure 3.5.

Les développements relatifs au calcul de v sont exposés dans [1]. Nous ne donnons ici que les résultats.

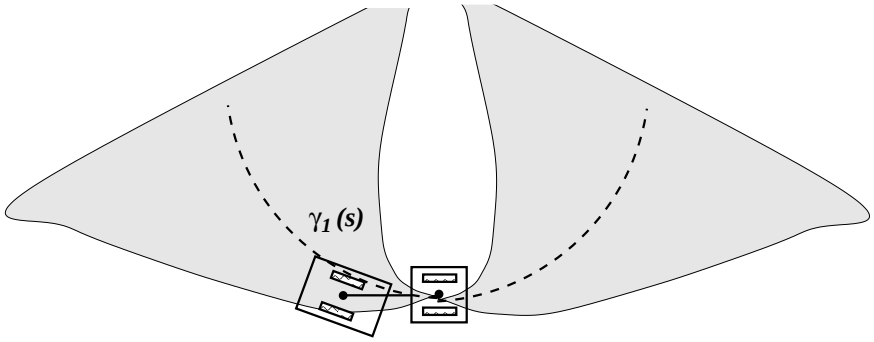

FIG. 3.6: *Cône d'accessibilité*

On définit pour chaque configuration q_1, une partie de $\mathcal{CS}$ appelée *cône d'accessibilité* relatif à q_1 (Fig. 3.6).

Definition 3 Cône d'accessibilité *relatif à* $q_1 = (x_1, y_1, \theta_1, \kappa_1)$*:*
Soit $q_2 = (x_2, y_2, \theta_2, \kappa_2)$ *et* $\dot{q_1} = (\dot{x_1}, \dot{y_1}, \dot{\theta_1}, \dot{\kappa_1})$ *la projection de* q_2 *sur* $\Gamma(q_1, s)$*,*
v *l'abscisse de* $\dot{q_1}$ *sur* $\Gamma(q_1, s)$ $(\dot{q_1} = \Gamma(q_1, v))$ *et* $\rho_2 = \sqrt{(x_2 - \dot{x_1})^2 + (y_2 - \dot{y_1})^2}$*,*

$$Cone(q_1) = \left\{ q_2 \in \mathcal{CS} / |v| \geq \frac{|\kappa_2 - \kappa_1|}{0,6.\Delta\kappa.\kappa_{max}}, |v| \geq \sqrt{\frac{|\theta_2 - \bar{\theta_1}|}{0,175.\Delta\kappa.\kappa_{max}}}, |v| \geq \sqrt[3]{\frac{\rho_2}{0,14.\Delta\kappa.\kappa_{max}}} \right\}$$

Choix du point de rebroussement Si q_2 est dans le cône d'accessibilité de q_1, la méthode locale retourne la combinaison convexe entre q_1 et q_2. Sinon, on prend le plus petit v tel que $\Gamma(q_2, v) \in Cone(q_1)$ et on joint q_1 à $\Gamma(q_2, v)$ par une combinaison convexe, puis $\Gamma(q_2, v)$ à q_2 en suivant la courbe canonique γ_2.

Dans le chapitre 4, j'expliquerai les améliorations que j'ai apportées à cette méthode.

Ces informations proviennent de la thèse de Florent Lamiraux [1].

Chapitre 4

Mon travail

Mon stage a donc consisté à implémenter les trajectoires locales décrites dans la partie précédente dans Move3D.

Lorsqu'on fournit à Move3D un chemin local, on doit lui fournir également une fonction primordiale appelée `stay_within_distance` qui retourne un déplacement élémentaire assurant la non-collision. Je présente d'abord l'intérêt de cette fonction puis les différentes options de calcul testées et celle retenue.

Je présente en deuxième section un autre calcul des points de rebroussement.

Move3D étant basé sur des algorithmes aléatoires, comme on l'a dit précédemment il trouve un chemin valide, s'il en existe un, mais ce n'est pas forcément le plus court. Il faut donc lui fournir une fonction d'optimisation que je présente dans la troisième sous-partie.

Enfin, dans la dernière sous-partie je reviens un instant sur la dérivée de courbure. Sa continuité, pour ces chemins, étant le principal bénéfice apporté par cette méthode à Move3D.

4.1 Test anti-collision

Ayant un chemin local, nous devons tester s'il est en collision. Cette opération ne peut pas être effectuée analytiquement sur l'ensemble du chemin. La stratégie de Move3D est de tester la non collision d'un nombre fini de configurations sur le chemin et en s'assurant que l'intervalle entre chaque point est également sans collision.

4.1.1 Description du test de collision dans Move3D

Étant donné un chemin du système, on veut savoir s'il est valide, c'est-à-dire si le système suivant ce chemin entrera ou non en collision avec son environnement.

Le test d'anti-collision est basé sur des appels répétés à un détecteur de collision statique, selon un pas de discrétisation du chemin non uniforme. La Figure 4.1 décrit ce mécanisme de validation de chemin. Étant donnés un chemin local $\gamma(u)$ entre deux configurations q_i et q_f, on place le système dans sa première configuration courante $q_c = \gamma(0)$ et on calcule à quelle distances `d_rob` et `d_rem` du robot et de la remorque se trouvent les obstacles les plus proches. Le planificateur local comporte une fonction appelée `stay_within_distance` fournissant un incrément de paramètre `du` du chemin assurant que `d_rob` et `d_rem` soient des majorants des distances parcourues par tous les points du robot et de la remorque respectivement. Cette fonction nous assure que tout l'intervalle $[u, u + du]$ est sans collision. On met le système dans la configuration $\gamma(u + du)$ et on répète la

même procédure. De cette manière, on parcourra γ avec un minimum d'appels au détecteur de collision statique.

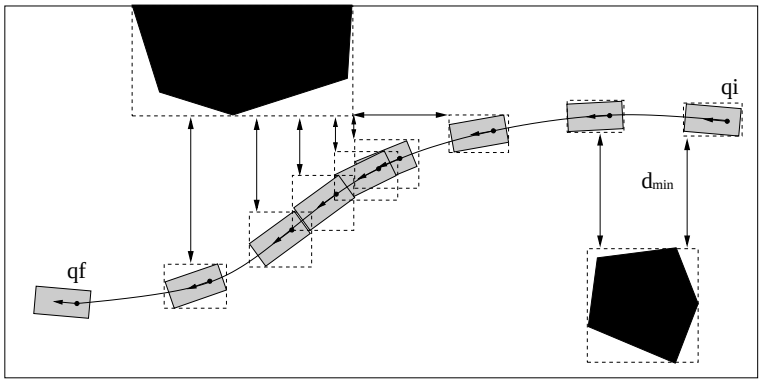

FIG. 4.1: *Validation d'un chemin dans Move3D.*

Il est clair que plus le système se rapprochera d'un obstacle lors du parcours de son chemin, plus le pas `du` sera petit. Si le chemin local comporte une collision, la procédure précédente converge vers la première configuration en collision. Pour éviter cette situation, on remplace `d_rob` et `d_rem` par `max(d_min, d_rob)` et `max(d_min, d_rem)` ou `d_min` est une distance choisie par l'utilisateur. Lorsque le système se rapproche de l'obstacle, les incréments sur le chemin local deviennent réguliers et une éventuelle collision peut être détectée (figure 4.2).

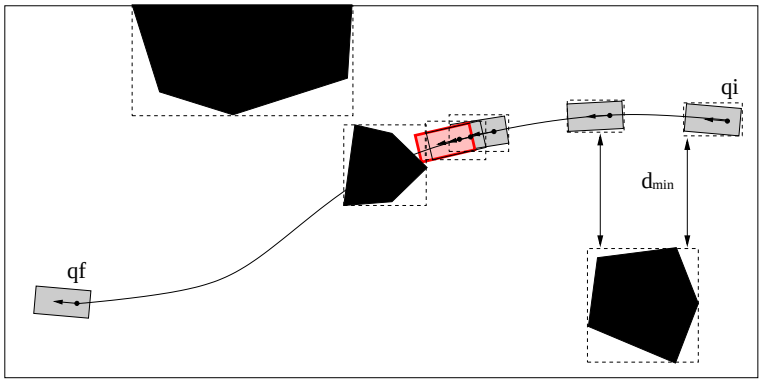

FIG. 4.2: *Validation d'un chemin dans Move3D.*

4.1.2 La fonction `stay_within_distance`

Comme décrite précédemment, la fonction `stay_within_distance` calcule, pour une configuration donnée sur un chemin local, un incrément `du` de paramètre assurant qu'aucun point du robot et de la remorque ne se déplace de plus d'une distance donnée en entrée.

Une première majoration globale. La première méthode implémentée consiste à assurer la sécurité à 100%. La méthode est de calculer analytiquement des majorations des vitesses linéaires et angulaires du centre du robot et de la remorque pour tout le chemin en majorant d'abord les normes des dérivées de la trajectoire de façon uniforme sur $[0, 1]$.

Les vitesses linéaires et angulaires du robot et de la remorque s'expriment sous la forme :

$$P(\gamma', \gamma'', \gamma''')/\|\gamma'\|^n$$

où P sont des fonctions polynômes. On majore ensuite les normes de $\gamma', \gamma'', \gamma'''$ et on minore celle de γ'. Puis on majore la vitesse de tous les points du robot par

$$\texttt{V\_rob} = V_{linéaire} + \emptyset_{rob} * V_{angulaire}$$

ainsi que celle de la remorque.

Soit :
- V_rob la vitesse maximale de tous les points du robot sur tout le chemin
- V_rem la vitesse maximale de tous les points de la remorque sur tout le chemin
- d_rob la distance du robot aux obstacles
- d_rem la distance de la remorque aux obstacles

On prend alors :

```
du = min (d_rob / V_rob, d_rem / V_rem)
```

.

En fait, cette première méthode est tellement sûre qu'elle rend des `du` excessivement petits (de l'ordre de 10^{-4} pour une trajectoire paramétrée entre 0 et 1) ce qui dans Move3D est sans intérêt et très long. Elle s'est avérée non efficace et en voici quelques explications.

Si l'on regarde les formules utilisées pour la majoration (voir 4.1.2), elles contiennent au dénominateur m le minimum global de la derivée de la trajectoire, et même au degré 5 pour la vitesse angulaire du robot. Comme m peut être très petit et même nul (par exemple dans le cas d'un robot a attache centrée, si le robot fait un angle de 90° avec la remorque il tourne donc en rond autour du centre de la remorque qui lui même reste sur place et donc la trajectoire de la sortie plate se résume à un point). Cette remarque nous a amenés par la suite à éliminer les trajectoires où $\|\gamma'\|_{min}$ est trop petit.

Maximisation contrôlée. La seconde méthode implémentée est la même que la précédente pour ce qui est du calcul du `du` et des vitesses maximales en fonction des normes des dérivées de la trajectoire mais utilise une méthode de maximisation plus fine de ces normes de dérivées des trajectoires. Pour la dérivée quatrième de la trajectoire on utilise la même méthode, puis pour les autres on utilise une fonction de maximisation à laquelle on peut donner la précision que l'on veut.
Le principe de cette maximisation est d'utiliser une procédure de dichotomie permettant de calculer un maximum. La formule utilisée pour maximiser une fonction f est (en connaissant M tel que $|f'(u)| < M$) :

$$\frac{f(u_{i+1}) + f(u_i)}{2} - M\frac{u_{i+1} - u_i}{2} \leq f(u) \leq \frac{f(u_{i+1}) + f(u_i)}{2} + M\frac{u_{i+1} - u_i}{2}$$

la marge d'erreur étant $\min(f_{max} - f(u_0), f_{max} - f(u_{end}))$ (voir Fig. 4.3).

La dichotomie consiste à couper l'intervalle $[0, 1]$ en plus petits intervalles tant que l'erreur est supérieure à celle désirée.
Cette méthode aurait été retenue si au départ nous avions une bonne appoximation de $\|\gamma''''\|_{max}$. N'obtenant pas par la méthode précédente un $\|\gamma''''\|_{max}$ précis, les marges d'erreur au début sont trop importantes et la fonction étant doublement récursive (c.à.d. si à un niveau la précision est supérieure à celle voulue on doit rappeler la fonction pour les deux sous intervalles), le temps de calcul des maxima est trop long, nous avons donc implémenté une troisième méthode.

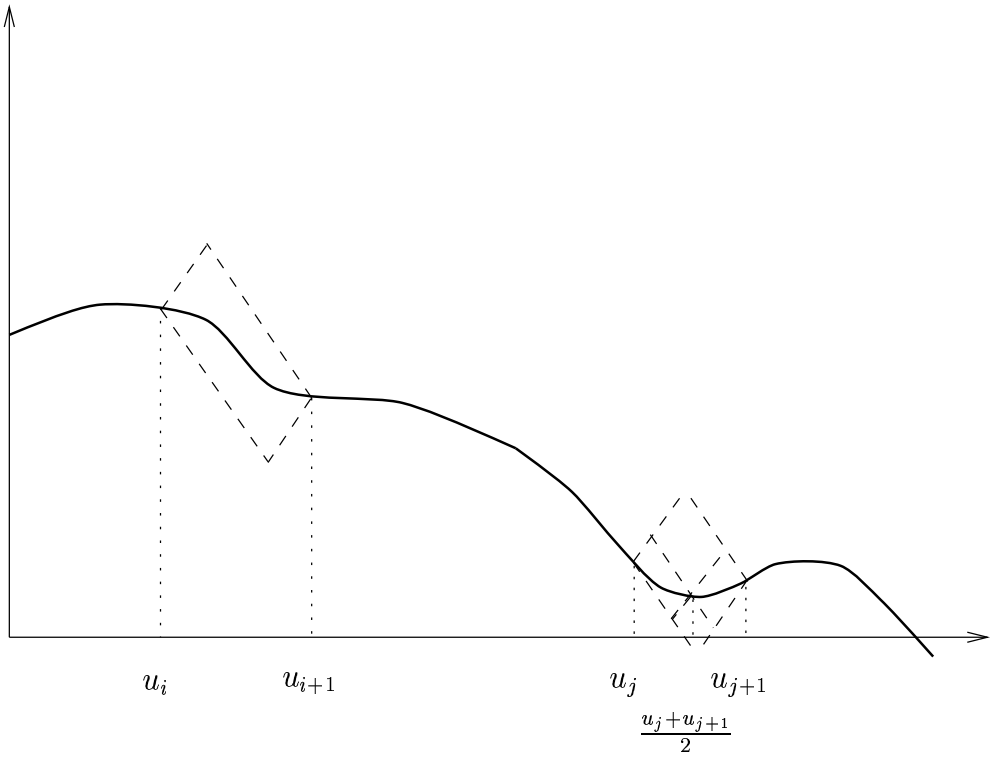

FIG. 4.3: *Détermination du maximum par la méthode*

FIG. 4.4: *Vitesse linéaire du centre de la remorque (γ') en fonction du paramètre le long du chemin*

FIG. 4.5: *Vitesse linéaire du centre du robot en fonction du paramètre le long du chemin*

Élimination de courbes et maximisation. La troisième méthode consistait à exprimer en un point les vitesses et accélérations exactes du centre du robot et de celui de la remorque, de calculer pour toute la trajectoire -par un découpage régulier de l'intervalle de paramètre- les maxima de ces accélérations. La distance parcourue par un point du robot (ou de la remorque) est majorée par le second membre des équations suivantes :

```
d_rob = V_rob.du_rob + V'_rob_max. du_rob.du_rob / 2
d_rem = V_rem.du_rem + V'_rem_max. du_rem.du_rem / 2
```

Ces équations donnent une valeur de `du_rob` et `du_rem`, on prend :

```
du = min(du_rob, du_rem)
```

[1]

Ici les calculs de `V'_rob_max` et `V'_rem_max` ne correspondent plus rigoureusement à des maxima de façon certaine. En effet, calculer le maximum d'une fonction sur un intervalle de façon numérique n'est possible qu'avec des fonctions régulières. Nous avons donc dû faire un tri des trajectoires.

Un exemple de graphe des vitesses et accélérations des centres du robot et de la remorque dans un cas critique est présenté en Fig. 4.4 et 4.5.

Au moment où toutes les vitesses et accélérations ont un pic, la vitesse linéaire de la remorque (qui dans ce cas est la dérivée de la trajectoire) est nulle. Un des premiers critères pour éliminer une courbe a donc été un $\|\gamma'\|$ trop petit. Les trajectoires que l'ont a également éliminées sont

1. `du_rob = (sqrt(V_rob.V_rob + 2.d_rob.V'_rob_max) - v_all_rob)/ V'_rob_max`

celles pour lesquelles l'angle de braquage robot-remorque est dépassé sur la courbe, lorsque la dérivée de la courbure est trop grande.
De là, on a supposé que si on retenait une courbe elle ne poserait pas de problèmes singuliers et on a calculé avec un pas de discrétisation régulier les accélérations maximales sur la courbe.

4.2 Le point de rebroussement

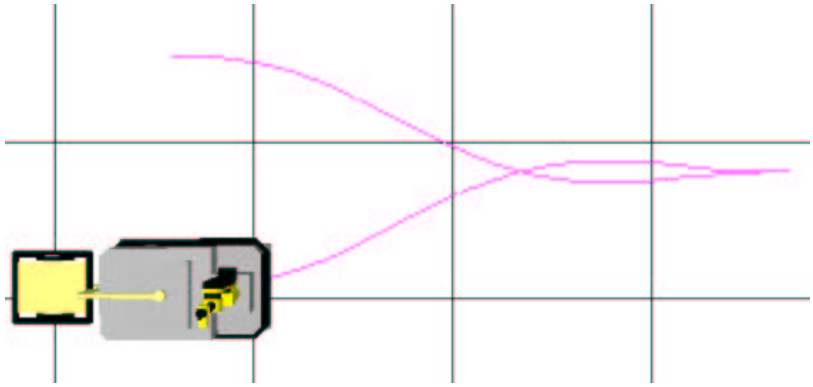

FIG. 4.6: *Exemple d'un point de rebroussement*

Dans le chapitre 3, j'ai défini ce qu'était le *cône d'accessibilité*. Il est à la base de notre méthode d'introduction de points de rebroussements.

Tout d'abord nous n'introduisons de points de rebroussements que si nécessaire, c'est à dire si la configuration d'arrivée n'est pas dans le cône d'accessibilité de la configuration de départ. L'implémentation du test en elle-même comprend aussi les tests cités plus haut, à savoir si les configurations ne sont pas proches et que le $\|\gamma'\|$ ou le $|v|$ sont trop petits *(*par rapport à la distance entre les deux configurations), si la dérivée de la courbure le long du chemin est trop grande ou si l'angle robot-remorque au cours du chemin dépasse l'angle de braquage (du système en question).

Maintenant le choix du point de rebroussement est intuitif (voir Fig 4.7).
Tout d'abord, on détermine un v d'après les formules (Def. 3). Puis on prend les quatre points à des distances v et $-v$ sur les courbes canoniques des configurations de départ et d'arrivée, cela nous fournit quatre configurations qui sont combinées (en faisant la moyenne de chaque composante) deux par deux en prenant les plus proches, cela nous donne alors deux configurations. On choisit ensuite la plus proche en moyenne des deux cônes d'accessibilité des configurations de départ et d'arrivée.

Cependant, nous ne sommes pas sûr que le point de rebroussement soit dans les deux cônes d'accessibilités, ni même dans un seul. Mais l'intersection de deux cônes d'accessibilité n'est pas calculable et cette méthode permet si l'on peut trouver un point de rebroussement d'en trouver un qui dans la mesure du possible relie les deux configurations de départ et d'arrivée par des chemins faisables.

L'ancienne méthode plaçait le point de rebroussement directement sur la courbe canonique d'une des configurations. La méthode proposée plus haut s'est avérée plus efficace, et permet de relier des configurations qui, avec l'ancienne méthode, impliquait des dépassements de l'angle limite robot-remorque.

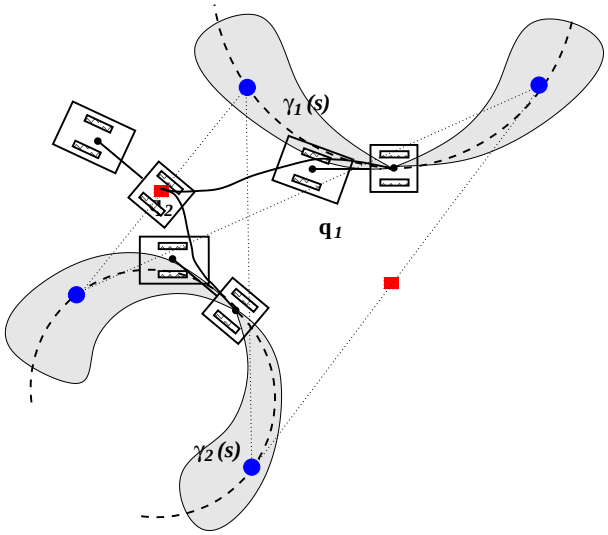

FIG. 4.7: *Calcul du point de rebroussement*

4.3 La fonction d'optimisation

4.3.1 Description de l'optimisation dans Move3D

Comme nous l'avons vu précédement Move3D est basé sur un algorithme probabiliste, donc lors de la création du graphe les deux configurations d'arrivée et de départ ne sont pas, en général, reliées par un chemin court (voir Fig 4.10). Pour pallier ce problème, après la création d'un chemin, une procédure d'optimisation est lancée sur ce chemin. Elle nécéssite une fonction calculant le coût d'un chemin, cette fonction est fournie par le planificateur. L'algorithme d'optimisation consiste à tirer aléatoirement deux configurations pour séparer le chemin en trois, à appeler le planificateur local sur ces trois sous-partie,puis grâce à la fonction de coût, on choisi s'il faut garder la partie de l'ancien chemin ou si le nouveau a un coût moindre (Fig. 4.8 et 4.9).

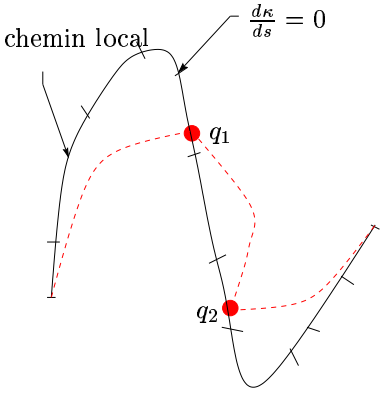

FIG. 4.8: *Un chemin (suite de chemins locaux) et l'optimisation éventuelle*

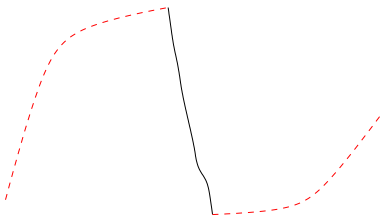

FIG. 4.9: *L'optimisation choisie*

Il faut donc fournir avec le planificateur local une fonction de coût du chemin local permettant à Move3D d'optimiser.

4.3.2 Le coût associé à nos trajectoires locales

La fonction de coût basique consiste à définir une longueur de trajectoire. Dans notre cas, nous l'avons un peu améliorée car pour une voiture le chemin le plus court n'est pas forcément le meilleur, une certaine quantité de manoeuvre doit être prise en compte. Dans le précédent planificateur, le coût était le nombre d'aller-retours. Le coût choisi cette fois est la somme d'une longueur de courbe ajoutée à quelque chose que nous appellons `phi'_total` (ϕ étant l'angle robot-remorque).
Ce `phi'_total` est l'intégrale *(discrète)* sur tout le chemin de la dérivée seconde de l'angle robot-remorque. Nous avons choisi ce terme car dans un premier temps le système avait tendance à chalouper ce qui donnait l'impression que le robot ne savait pas de quel côté aller, en particulier lors des passages critiques (passage de portes ou point d'arrivée et de départ). Nous avons donc introduit ce `phi'_total` dans le coût, il permet d'obtenir des trajectoires plus lisses (voir Fig 4.10, 4.11).

4.4 Continuité de la dérivée de courbure

Lors de l'exécution d'une trajectoire, les vitesses linéaires et angulaires de chacun des corps du système doivent être continues le long d'une trajectoire. Pour les deux systèmes robot-remorque, cette contrainte impose que la dérivée de la courbure de la sortie plate soit continue le long d'une succession de chemins locaux. Dans le cas contraire, lors de l'exécution, le système doit s'arrêter aux points de discontinuité de la dérivée de courbure. En dérivant l'équation $\phi = l.\arctan\kappa$, on obtient l'expression de la dérivée de ϕ

$$\frac{d\phi}{ds} = l.\frac{d\kappa}{ds}.\frac{1}{1+l^2\kappa^2}.$$

,la continuité de $d\phi/ds$ dépend donc de la continuité de $d\kappa/ds$. Cette continuité le long d'un chemin local est assurée mais le problème se pose lors du raccordement de plusieurs chemins locaux.

Afin d'assurer cette continuité, nous n'avons pas utilisé exactement la fonction α décrite précédement mais une fonction α avec des contraintes sur sa dérivée troisième aux extrémités :

$$\alpha'''(0) = a \quad \alpha'''(1) = b$$

a et b pouvant s'exprimer en fonction de $d\kappa/ds$ aux extrémités. Il nous faut alors un polynôme de degré sept, et en résolvant à nouveau le système nous obtenons :

$$\alpha_{a,b}(u) = \frac{a}{6}u^3 + (-\frac{2a}{3} - \frac{b}{6} + 35)u^4 + (a + \frac{b}{2} - 84)u^5 + (-\frac{2a}{3} - \frac{b}{2} + 70)u^6 + (\frac{a}{6} + \frac{b}{6} - 20)u^7$$

.

Calcul de a et b Nous avons introduit ces contraintes car nous voulions que la dérivée de courbure soit continue. Il ne reste donc qu'à exprimer a et b en fonction de la dérivée de courbure aux extrémités. Puis à prendre la dérivée de courbure souhaitée. Nous avons pris des dérivées de courbure nulles lors de la création de chemins locaux, la dérivée de courbure étant nulle sur les courbes canoniques, et nous avons calculé la dérivée de courbure sur un chemin lors de l'optimisation.

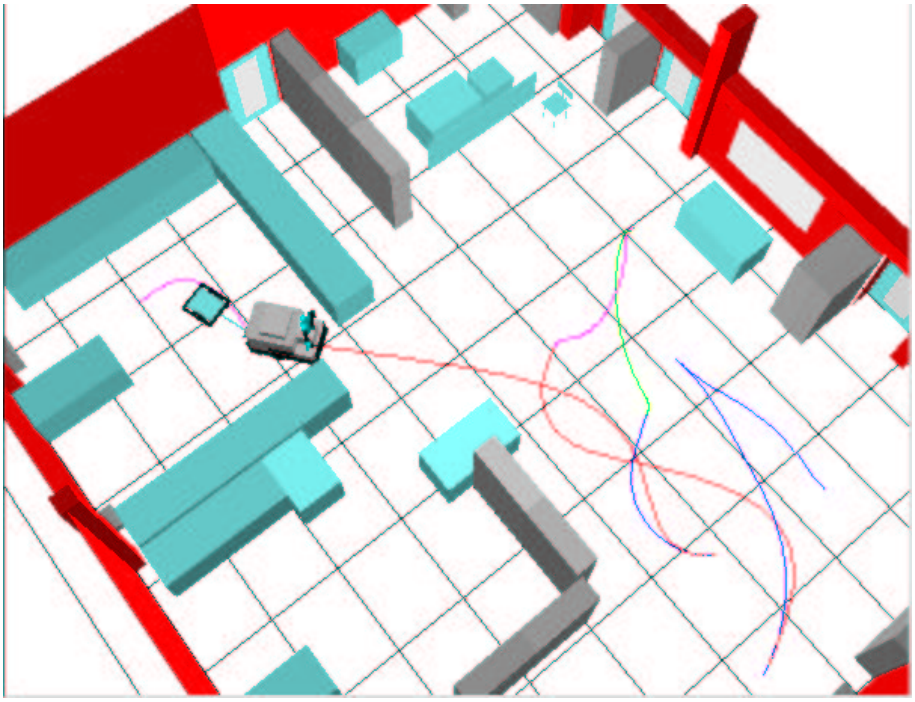

FIG. 4.10: *Exemple de chemin trouvé par Move3D à l'aide de notre méthode de chemin local avant optimisation*

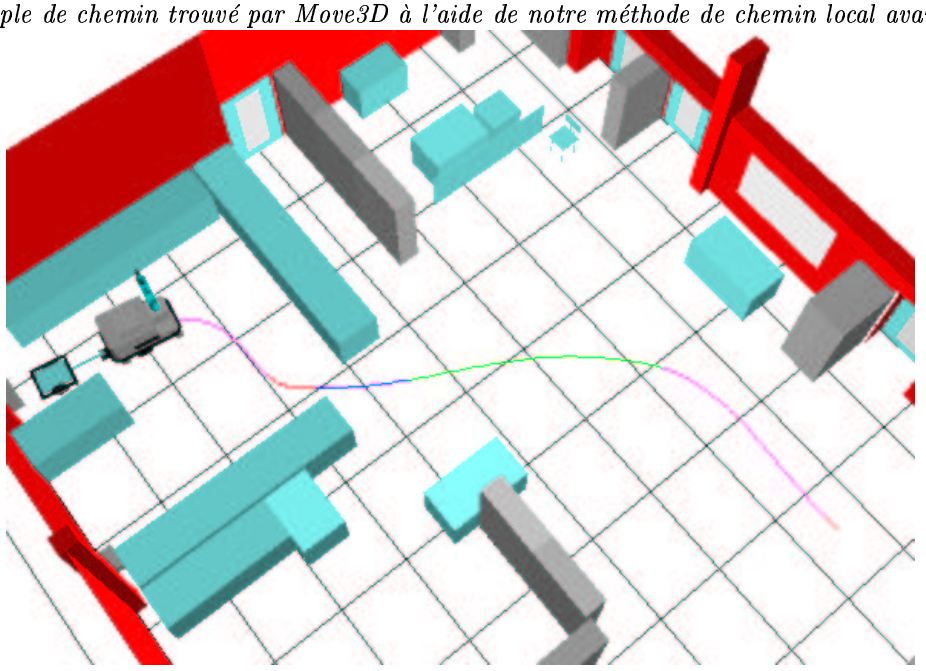

FIG. 4.11: *Exemple de chemin trouvé par Move3D à l'aide de notre méthode de chemin local après optimisation*

Ayant $\frac{ds}{du}(u) = \|\gamma'(u)\|$,
Nous avons :

$$\begin{aligned}
\kappa(u) &= \frac{\det(\gamma'(u), \gamma''(u))}{\|\gamma'(u)\|^3} \\
\frac{d\kappa}{du}(u) &= \frac{\det(\gamma'(u), \gamma'''(u))}{\|\gamma'(u)\|^3} - 3.\frac{\det(\gamma'(u), \gamma''(u))}{\|\gamma'(u)\|^5}.(\gamma'.\gamma'') \\
\frac{d\kappa}{ds}(u) &= \frac{\det(\gamma'(u), \gamma'''(u))}{\|\gamma'(u)\|^4} - 3.\frac{\det(\gamma'(u), \gamma''(u))}{\|\gamma'(u)\|^6}.(\gamma'.\gamma'')
\end{aligned}$$

Aux extrémités, nous avons (c.f. 3.2):

$$\begin{array}{ll}
\gamma(0) = \gamma_1(0) & \gamma(1) = \gamma_2(1) \\
\gamma'(0) = \gamma_1'(0) & \gamma'(1) = \gamma_2'(1) \\
\gamma''(0) = \gamma_1''(0) & \gamma''(1) = \gamma_2''(1)
\end{array}$$

γ_1 et γ_2 étant des cercles :

$$\begin{aligned}
\det(\gamma_1'(0), \gamma_1''(0)) &= 0 \\
\det(\gamma_2'(1), \gamma_2''(1)) &= 0
\end{aligned}$$

On peut simplifier ainsi l'expression de la dérivée de courbure aux extrémités par :

$$\begin{aligned}
\frac{d\kappa}{ds}(0) &= \frac{\det(\gamma_1'(0), \gamma_1'''(0))}{\|\gamma_1'(0)\|^4} \\
\frac{d\kappa}{ds}(1) &= \frac{\det(\gamma_2'(1), \gamma_2'''(1))}{\|\gamma_2'(1)\|^4}
\end{aligned}$$

Ainsi en remplaçant $\gamma_1'(0)$, $\gamma_1'''(0)$, $\gamma_2'(0)$ et $\gamma_2'''(0)$ par leurs expressions en fonction de α, ses dérivées et les paramètres de configuration aux extrémités, on obtient :

$$\begin{aligned}
\frac{d\kappa}{ds}(0) &= \frac{\alpha'''(0)}{v^3}.(\vec{u}(\theta(0)) \cdot (\gamma_2(0) - \gamma_1(0))) \\
\frac{d\kappa}{ds}(1) &= \frac{\alpha'''(1)}{v^3}.(\vec{u}(\theta(1)) \cdot (\gamma_2(v) - \gamma_1(v)))
\end{aligned}$$

En remplaçant, $\frac{d\kappa}{ds}(0)$ et $\frac{d\kappa}{ds}(1)$ par les valeurs souhaitées et en résolvant ces équations, on trouve $\alpha'''(0)$ et $\alpha'''(1)$ nous permettant d'assurer la continuité.

Chapitre 5

Conclusion

Jusqu'à présent la plate-forme Move3D ne permettait pas de planifier des mouvements pour des robots à remorque ou des voitures. Mon stage a donc permis, grâce à des chemins locaux déjà existants, de planifier des chemins pour ces robots. Le laboratoire disposant de réels robots à remorque, j'ai pu en voir un effectuer une trajectoire calculée par Move3D.

La planification s'est avérée efficace dans des espaces où le robot dispose de suffisamment de place pour manoeuvrer. En revanche, pour des espaces très contraints (ex: coude d'un couloir) plusieurs problèmes se posent. D'abord le temps de planification s'allonge considérablement. Ensuite, le robot doit être capable de se localiser avec une erreur inférieure à la distance de sa trajectoire aux obstacles. Le problème de la planification, bien qu'il semble simple au premier coup d'oeil, je me suis d'ailleurs demandé au début pourquoi les robots en étaient encore à chercher leur chemin, est en fait complexe. De plus les contraintes cinématiques des robots non holonomes réduisent l'ensemble des solutions.

Bibliographie

[1] F. Lamiraux. *Robots mobiles à remorque : de la planification de chemins à l'exécution de mouvements*. PhD thesis, INPT, LAAS-CNRS, Toulouse, France, September 1997.

[2] F. Lamiraux and J.-P. Laumond. Smooth motion planning for car-like vehicles. In *IAS-6*, pages 1005–1012, 2000.

[3] C. Nissoux. *Visibilité et méthodes probabilistes pour la planification de mouvements en robotique*. PhD thesis, U.P.S., LAAS-CNRS, Toulouse, France, 1999.

Table des matières