UNIVERSITÉ DE MONTRÉAL

PLANIFICATION DE CHEMIN POUR UN ROBOT MOBILE DANS UN ENVIRONNEMENT PARTIELLEMENT CONNU

ALEJANDRO AGUDELO
DÉPARTEMENT DE GÉNIE ÉLECTRIQUE
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(AUTOMATIQUE)
DÉCEMBRE 2007

Your file Votre référence
ISBN: 978-0-494-36897-8
Our file Notre référence
ISBN: 978-0-494-36897-8

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé:

PLANIFICATION DE CHEMIN POUR UN ROBOT MOBILE DANS UN ENVIRONNEMENT PARTIELLEMENT CONNU

présenté par: AGUDELO Alejandro

en vue de l'obtention du diplôme de: Maîtrise ès sciences appliquées

a été dûment accepté par le jury d'examen constitué de:

M. ROMANO De Santis, Ph.D., président

M. RICHARD Gourdeau, Ph.D., membre et directeur de recherche

M. ERIC Dupuis, Ph.D., membre

À ma cher épouse Viviana

REMERCIEMENTS

Je tiens à remercier toutes les personnes qui directement ou indirectement m'ont aidé au cours de la réalisation de ce projet de maîtrise. J'aimerais remercier tout d'abord mon directeur de recherche Monsieur Richard Gourdeau pour son encadrement, la motivation et la confiance qu'il m'a accordée, sans oublier le soutien financier. J'aimerais aussi remercier l'Agence Spatial Canadienne spécialement Monsieur Erick Dupuis et Monsieur Joseph Nsasi Bakambu pour l'opportunité qu'ils m'ont accordé de travailler dans un de ces projets.

Je désire aussi remercier Madame Suzanne Valade pour son aide inconditionnelle dans la correction de ce rapport, mes collègues du laboratoire pour le temps et les idées partagés, ainsi que les professeurs de la section d'automatisation spécialement Monsieur Romano De Santis pour son encouragement.

Finalement ma famille et mes amis, et particulièrement mon épouse Viviana pour son support et ses encouragements tout au long de cette expérience.

RÉSUMÉ

Dans le cadre de l'exploration planétaire et en collaboration avec l'agence spatiale canadienne, ce projet étudie le problème de la planification de trajectoire pour des robots mobiles. L'objectif est de déterminer un chemin libre d'obstacles pour un robot mobile en considérant sa géométrie afin de lui donner une plus grande autonomie lors de la planification de déplacements sur de longues distances dans un environnement partiellement connu.

À partir d'une première ébauche de trajectoire définie par un planificateur sur longues distances du robot et d'une représentation polygonale basée sur un maillage triangulaire de résolution variable de l'environnement, créée en fonction de l'information donnée par un scanner laser tridimensionnel placé sur le robot, une nouvelle trajectoire libre d'obstacles est générée en tenant compte de la géométrie du robot et éliminant dans la mesure du possible les dents de scie présentes dans la trajectoire originale. Un algorithme basé sur une fonction pondérée de différents coûts permet de définir cette trajectoire minimisant le risque de collision, la distance à parcourir et les forts changements d'orientation du robot.

Différents cas sont étudiés pour établir les performances de différentes approches.

ABSTRACT

This project studies the problem of motion planning for mobile robots. The aim is to define an obstacle-free path for a mobile robot which will consider its geometry with the goal to give it a greater autonomy in the planning of movements on long distances in a partially known environment. Starting from a first base path defined by the original long distance planner of the robot and a polygonal representation of the environment based on a 3D irregular triangular mesh, created from the information given by a three-dimensional laser scanner placed on the robot, a new obstacle free path is generated considering the robot geometry and eliminating as much as possible the saw tooth present in the base path. An algorithm based on a cost function makes possible to define this path, minimizing the collision risk, the distance to be traveled and the strong changes of orientation of the robot.

Different cases are studied in order to establish the performance of different approaches.

TABLE DES MATIÈRES

LISTE DES FIGURES

INTRODUCTION

L'intérêt scientifique croissant pour l'exploration planétaire fait appel à des robots mobiles exploratoires. Ces derniers doivent posséder une grande autonomie et une capacité élevée de décision dans le champ de la planification du mouvement. Ces qualités sont essentielles afin de diminuer ou éviter l'intervention humaine dans le processus d'exploration. Ceci est important étant donnés les délais de transmission et faibles bandes passantes entre les postes de contrôle terrestres et les sondes d'exploration.

Le but de ce projet est de rendre le robot capable d'explorer de plus grandes aires et de parcourir de plus longues distances sur des terrains peu connus, et ce, tout au long de sa mission. Cet objectif sera plus facilement accompli, si le robot est équipé d'un planificateur de chemin fiable et rapide, et qui minimise les risques engagés pour le robot.

Dans le cadre du projet d'exploration planétaire, l'Agence Spatiale Canadienne (ASC) est très active dans la recherche et le développement de nouveaux modèles de robots exploratoires. C'est dans l'amélioration de la tâche de la planification de chemins pour ces robots que ce travail de recherche trouve sa raison d'être.

Ce projet cherche donc à fournir une solution au problème de la planification de chemin tout en considérant la géométrie du robot et la topographie du terrain. L'objectif est de garantir la faisabilité du chemin tracé par le planificateur sur longues distances.

Ce mémoire est divisé en trois chapitres. Le premier présente la problématique reliée à cette recherche afin d'aborder par la suite, les solutions proposées dans le deuxième chapitre. Pour ce qui est du troisième chapitre, il porte sur l'analyse de chacune des solutions proposées par le biais d'exemples représentatifs pour chacune d'entre elles.

CHAPITRE 1

LA PROBLÉMATIQUE

Dans ce chapitre nous aborderons la question qui nous amène à la réalisation de ce projet de recherche. Il est divisé en quatre sections. Une première section sera dédiée à une brève mise en situation. Dans la deuxième, on établira la définition de plusieurs termes qui par la suite seront utilisés dans la formulation et la solution du problème. La troisième porte sur la définition du problème et ce qu'on cherche à retrouver dans la solution proposée. Pour ce qui est du dernier volet, on y énonce les critères d'évaluation avec lesquels sera jugée la solution donnée.

1.1 Mise en situation

À cause de l'intérêt croissant pour l'exploration planétaire, les robots mobiles sont appelés à avoir une très grande autonomie. Le besoin d'une intervention humaine dans les décisions de planification de chemin cause une énorme perte de temps et de ressources du robot. Le rôle d'un planificateur de chemins consiste donc à augmenter l'autonomie du robot en lui permettant de se déplacer d'une position initiale vers une position finale arbitraire tout en évitant les collisions avec les obstacles présents dans son environnement. Cette problématique a été dans l'intérêt de chercheurs dans différents domaines durant les dernières décennies.

Dans cette section, nous énoncerons d'abord le problème général de planification, par la suite nous présentons les grandes approches et méthodes de résolution proposées dans la littérature et enfin nous parlerons de certains travaux de recherche qui seront plus pertinents pour notre problème.

Avant d'aborder le problème de la planification du chemin il faut définir ce qui est un chemin. Un chemin pour le robot entre deux configurations q_{init} et q_{goal} est une fonction continue τ telle que $\tau : [0, 1] \rightarrow C$ avec $\tau(0) = q_{init}$ et $\tau(1) = q_{init}$ où C est l'espace de configurations du robot.

De plus, si toute configuration définie par le chemin τ entre l'intervalle $[0, 1]$ se trouve dans le même composant de C_{free}, il s'agit d'un chemin libre d'obstacles où C_{free} est l'espace de configuration qui n'est pas occupé par les obstacles (Latombe, 1991).

Comme nous l'avons déjà dit, le problème de la planification de trajectoire consiste à trouver cette fonction continue qui éventuellement permettra au robot de passer d'une position ou configuration initiale q_{init} à une autre finale q_{goal}. Cette fonction est définie sur l'espace des configurations C. Le problème se présente lorsque dans cet espace de configuration il y a certaines configurations qui sont interdites à cause des obstacles. Alors, l'espace de configuration C est l'union du sous-espace C_{obs} et du sous-espace C_{free}. Puisque nous voulons que le chemin soit libre d'obstacles, nous devrions effectuer notre recherche de la fonction τ sur le sous-espace C_{free}.

Dans cet ordre d'idées, le problème de planification de chemin consiste alors à définir le sous-espace C_{free}, pour continuer par la suite avec la recherche des configurations continues qui forment le chemin.

Comme solution à ce problème, il y a un nombre important d'approches qui se sont faites remarquer à travers le temps et qui ont inspiré plusieurs autres travaux de recherche. Nous allons donc donner un bref aperçu de chacune de ces approches que l'on peut retrouver de façon plus détaille dans (Latombe, 1991) et (LaValle, 2006).

1.1.1 Méthodes par décomposition cellulaire

Cette méthode consiste à partitionner l'espace de configuration libre en plusieurs régions adjacentes appelées cellules, par la suite un graphe de connectivité est défini entre les cellules afin de pouvoir tracer un chemin entre configurations qui appartiennent à différentes cellules adjacentes. Finalement pour trouver le chemin désiré il faudrait explorer le graphe de connectivité et déterminer le chemin qui relie la configuration initiale avec la configuration finale. Cet algorithme est un des algorithmes les plus utilisés dans la planification de chemins.

Cette méthode de planification est divisée en deux grandes catégories : décomposition cellulaire exacte et décomposition cellulaire approximative. Les méthodes de résolution dites exactes sont des méthodes où la décomposition effectuée se base sur un recouvrement exact de l'espace libre du robot, les limites des cellules correspondent à des changements dans les contraintes applicables à la cinématique du robot. Pour ce qui est des méthodes de résolution dites approximatives, elles font une division de l'espace de configuration libre avec des cellules élémentaires identiques et permettant l'adaptation de la taille de celles-ci à la géométrie des zones à recouvrir. Les méthodes de décomposition exacte sont caractérisées pour être plus difficiles à implémenter et plus gourmandes en ressources, pour cette raison les méthodes approximatives ont été préférées malgré l'incertitude dans la précision de la réponse.

Les méthodes de décomposition verticale du « quadtree » de décomposition verticale par triangulation, de décomposition cylindrique sont quelques méthodes de décomposition cellulaire utilisées.

1.1.2 Les méthodes Roadmap

Une deuxième méthode pour la résolution du problème de planification consiste à ramener la recherche du chemin à un espace de plus faible dimension que celle de l'espace admissible. Cette approche consiste à représenter la connectivité de l'espace libre du robot par un réseau de courbes unidimensionnelles pouvant être entièrement définies dans l'espace libre q_{goal} du robot, ce réseau de courbes est appelé *roadmap*.

Après avoir généré le *roadmap*, il faudra alors inclure dans le réseau les configurations initiale et finale, de cette façon, la planification de chemin sera réduite à trouver d'abord le chemin qui connecte la configuration initiale avec un nœuds du *roadmap*, déterminer un chemin entre la configuration finale et le *roadmap*, et finalement explorer le *roadmap* afin de trouver un chemin qui relie les deux nœuds connectés aux configurations initiale et finale.

Les méthodes du *roadmap* sont divisés en deux groupes : les méthodes déterministes et les méthodes probabilistes. La différence entre les deux groupes est la façon utilisée pour construire le *roadmap*.

Dans les approches déterministes, le *roadmap* est invariant pour un même espace de configurations. Quelques exemples d'approches qui utilisent cette méthode sont : l'approche par graphe de visibilité, l'approche par rétraction, la méthode *Freeway net* et la méthode *Silhouette*.

Dans le deuxième groupe des méthodes *roadmap*, nous avons les approches probabilistes classiques. Ces dernières utilisent des stratégies probabilistes pour construire le graphe. Ces méthodes sont rapides et permettent d'avoir de très bons résultats surtout dans des espaces complexes. Quelques exemples de ces approches sont : L'algorithme *Connect*, la méthode *Hsu*, l'algorithme *SBL*.

1.1.3 Méthode de champ du potentiel

Cette méthode consiste à représenter le robot comme une particule qui se déplace dans un champ de potentiel fictif. Les variations locales de ce champ de potentiel fourniront une idée de la structure de l'espace de configuration libre, dans la plus part des cas, l'espace libre sera obtenu par la composition d'un premier champ de potentiel attractif, qui a pour objectif d'attirer le robot vers la solution et d'un champ répulsif qui a pour but d'éloigner le robot des obstacles. Le chemin est alors calculé par un algorithme de descente du gradient du potentiel obtenu.

Cette méthode a connu du succès grâce à sa vitesse et à son applicabilité dans la planification de mouvements en temps réel. Mais elle présente l'inconvénient de rester bloquée dans des minimaux locaux. Ces minimums sont généralement liés à la géométrie et à la répartition des obstacles dans l'espace de travail du robot et surtout aux coefficients de pénalités qui leur sont associés pendant la construction du champ de potentiel. Pour éviter ce problème, plusieurs travaux ont visé deux solutions. La première est de définir une fonction de potentiel avec peu ou sans minimaux locaux, et la deuxième s'adresser à l'implantation d'algorithmes de recherche que puissent éviter ces minimaux locaux.

1.1.4 Méthodes de planification optimale et algorithmes de recherche de graphe

Les méthodes de planification pour recherche de graphes sont des méthodes qui effectuent une recherche systématique sur tous les états accessibles dans un graphe afin de trouver une solution optimale. Cette recherche doit tenir un registre des états déjà visités afin d'éviter une recherche cyclique sur les mêmes états. Normalement, ce type d'algorithmes est appliqué sur un graphe fini dont on pourrait garantir l'obtention ou pas une solution dans un temps fini. Afin d'utiliser ces méthodes pour des graphes infinis, il faudra définir les limites de la recherche afin de pouvoir attirer une conclusion dans

un temps donné. De plus, un coût peut être associé à la transition entre états. Nous allons réviser par la suite, les algorithmes de planification les plus répandus dans cette catégorie, lesquelles sont présentes en détail dans (LaValle, 2006)

1.1.4.1 La planification optimale

Dans le cas du problèmes de chemins optimaux il s'agit d'acheminer un objet entre deux états, de façon à optimiser un certain critère, tel que le temps, distance, ou l'énergie consommée. Afin de minimiser ou maximiser ce critère, une fonction de coût est définie et évaluée au long du chemin, elle est chargée de montrer le coût accumulé à chaque point du chemin. De plus, une décision d'arrêt devra être implémentée afin d'indiquer intuitivement quand sera le temps d'arrêter la recherche et fixer le coût total associé au chemin. Entre les approches de planification optimale, nous comptons avec la théorie de la programmation dynamique qui sera utilisée dans la solution de notre problème.

1.1.4.2 Algorithme de Dijkstra

L'algorithme de Dijkstra est connu pour résoudre le problème des chemins les plus courts entre deux sommets d'un graphe. Typiquement, la fonction de coût associée à l'arête qui lie un sommet à un autre, est la distance entre eux, pour cet algorithme le coût lié aux arêtes doit être positif ou nul. Cet algorithme, évalue le coût total pour aller d'un nœud initial à un nœud final, en choisissant le chemin avec coût le plus faible. Pour ce faire, il doit visiter tous les nœuds du graphe et calculer pour chacun le chemin avec le coût minimal qui le connecte avec le nœud initial.

La stratégie de recherche de cet algorithme consiste à rechercher tous les nœuds dans un rayon qui croît au tour de l'état initial en cherchant se rapprocher à l'état final.

Le principal problème de cet algorithme est qu'il visite tous les états possibles même s'ils s'éloignent de l'état final, cela implique une perte au niveau de temps de calcul.

1.1.4.3 L'algorithme A *

L'algorithme de recherche est une modification de l'algorithme de Dijkstra, qui essaye de réduire le nombre d'états explorés en incorporant une évaluation heuristique du coût pour atteindre l'état désiré à partir d'un état initial. Dans l'essence les deux algorithmes fonctionnent de la même façon, mais celui-ci va tenter de se rapprocher à l'état final en privilégiant les états intermédiaires qui représentent une possibilité de rapprochement direct à la destination. Alors dans cet algorithme nous profitons de l'information que nous avons à propos de l'état final afin d'orienter la recherche d'une façon directe vers l'état désiré.

1.1.4.4 La recherche vers l'arrière et bidirectionnelle.

Les deux derniers algorithmes sont partis d'un état initial en faisant une recherche extensive à travers le graphe afin de rejoindre l'état final avec le chemin le moins coûteux. Ces mêmes méthodes peuvent être appliquées dans le sens inverse, au lieu de commencer la recherche à partir de l'état initial, nous allons commencer la recherche à partir de l'état final en direction de l'état initial. Pour plusieurs problèmes de planification, cette façon de procéder permet d'atteindre l'objectif plus rapidement.

La recherche bidirectionnelle part de l'idée d'exécuter simultanément deux recherches, la première partira de l'état initial vers l'état final, et la deuxième partira de l'état final vers l'état initial. La recherche se termine avec succès quand les deux arbres de recherche se réunissent. Pour beaucoup de problèmes, la recherche bidirectionnelle peut nettement réduire la quantité d'exploration exigée.

1.1.5 Environnement partiellement connu

Afin de donner une plus grande autonomie, de nombreux chercheurs ont travaillé au développement de différents systèmes de vision, d'analyses d'images et de représentations du terrain afin de pouvoir effectuer une planification permettant au robot de se déplacer dans un terrain inconnu. L'utilisation d'une représentation en 3D par maillage triangulaire du terrain et d'objets est une des solutions à ce problème. Tel que présentée dans (Bakambu et al., 2006),(Gemme et al., 2005) et (Green et al., 2006), l'utilisation de ce type de représentation de l'environnement a été mise en place et testée sur des robots mobiles avec des résultats très convaincants.

D'un autre côté, nous avons le problème de planification de chemin sur un terrain non structuré. L'objectif d'obtenir un chemin en 3D continu et lisse, sur lequel une fonction de coût serait minimisée, nous a conduit à opter pour une représentation du chemin comme étant une fonction B-Spline. Tel que traité dans (Shiller and Chen, 1990), (Shiller, 2000) et (YAMAMOTO et al., 1998), la définition de ce type de chemin, nous permet de transformer le problème de planification en un problème d'optimisation de paramètres. L'utilisation de ce type de fonctions permet l'intégration de contraintes physiques du robot comme la vitesse, l'accélération et la courbure maximales lors de sa définition du problème.

Une autre difficulté est de faire de ce chemin, un chemin libre d'obstacles. Étant donné que notre robot est placé sur un terrain non structuré il faudra d'abord savoir reconnaître ce qu'est un obstacle et où il est placé. Il y a donc, dans la littérature, différentes approches qui cherchent à trouver une relation entre l'information du modèle du terrain et les obstacles afin de définir les zones traversables. Par exemple, l'analyse de pentes locales et de rugosité du terrain sont traités par (Seraji, 1999) et (Yokokohji et al., 2007). Dans (Page et al., 2006), la possibilité de définir un champ artificiel de potentiel sur le modèle 3D du terrain pourrait permettre par la suite d'utiliser la méthode de champs

potentiels pour trouver le chemin désiré. Cependant, la problématique des « minimum locaux » de potentiel peut créer des problèmes important de planification sur de longues distances.

Lorsque nous débutons la recherche du chemin à partir d'un chemin de base qui a été préalablement défini sur le terrain (Bakambu et al., 2006), nous voulons dans un première temps vérifier si ce chemin est libre d'obstacles. L'approche utilisée pour la détection d'obstacle évalue si la surface représentant le robot possède des points communs avec le terrain. L'implantation de cet algorithme de détection d'obstacle fera alors la tâche d'un détecteur de collision et des algorithmes d'optimisation et planification locale seront dédiés à la redéfinition de ce chemin initial.

1.2 Définition de termes

Dans cette section nous définirons un ensemble de termes qui seront utilisés tout au long de ce projet et qui nous permettront de traiter la solution du problème d'une façon plus simple. Les termes qui seront définis représentent l'environnement, le chemin ainsi que le modèle du robot utilisé dans notre approche.

1.2.1 L'environnement

L'environnement, aussi appelé le terrain, est représenté par un maillage triangulaire irrégulier 3D (ITM - *Irregular Triangular Mesh*). L'ITM consiste, à la base, à représenter une surface et les changements de sa topologie en utilisant des triangles de différente grandeur. Ceci permet donc de bien représenter la rugosité du terrain avec un ensemble de petits triangles et de réduire, par conséquent, la quantité d'information lorsqu'il s'agit d'une surface plate où sa représentation sera quelques triangles d'une aire majeure.

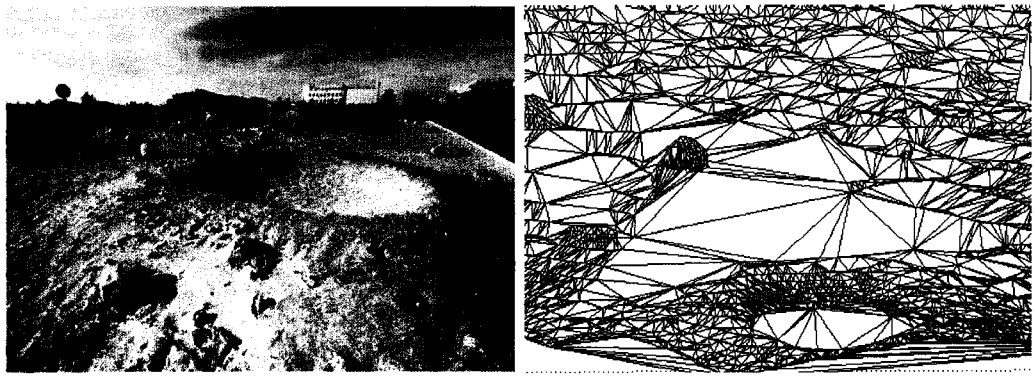
L'ITM est produit à partir d'un nuage des points fournis par un module de balayage au laser (LIDAR). Avant d'utiliser les données produites par le module de balayage, elles sont traitées par un algorithme de simplification et d'assemblage de cartes développé à l'Agence Spatiale Canadienne (ASC) (Bakambu et al., 2006). Les données des différents secteurs du terrain sont enregistrées et regroupées dans un même ensemble D puis lui est appliqué l'algorithme de triangulation de Delaunay disponible sous Matlab (Barber et al., 1996) pour visualiser et trouver la relation entre les points de l'ensemble D et le sommet des triangles.

L'algorithme prend comme paramètre d'entrée la matrice D et donne comme paramètre de sortie la matrice Tr . Alors, le terrain est défini comme l'ensemble $E = \{D, Tr\}$ où D est une matrice $n \times 3$ contenant les données 3D et Tr est une matrice $m \times 3$ qui est une description de la topologie du maillage comme triangulation de ces points (García, 1995).

$$D = \begin{bmatrix} x_1 & y_1 & z_1 \\ \vdots & \vdots & \vdots \\ x_n & y_n & z_n \end{bmatrix} Tr = \begin{bmatrix} s_{11} & s_{12} & s_{13} \\ \vdots & \vdots & \vdots \\ s_{m1} & s_{m2} & s_{m3} \end{bmatrix} \tag{1.1}$$

où x_i, y_i, z_i avec $i = 1, 2..., n$ sont les coordonnées cartésiennes d'un point du terrain, et s_{ij} avec $i = 1, 2..., m$ et $j = 1, 2, 3$ sont les sommets du triangle. Chaque s_{ij} est associé à un des points de la matrice D.

Les données du terrain ont été fournies par l'ASC comme résultats expérimentaux effectués dans « The Mars Yard » située au siège social de l'ASC à St. Hubert, Québec, Canada (voir la Fig.1.1).



(a) (b)

FIG. 1.1 : (a) « The Mars Yard » ; et, (b) sa représentation avec ITM.

1.2.2 Le chemin

Ce travail de recherche a pour but de fournir une solution au problème de la planification de chemin pour un robot mobile, mais tout au long du développement nous utiliserons divers types de chemins présentés dans cette section.

Le premier chemin d'intérêt est « le chemin de base » . Il est produit par le planificateur principal du robot qui trace un premier chemin sur de longues distances. Ce sera le point de départ pour le nouvel algorithme de définition du chemin.

L'algorithme du chemin le plus court de Dijkstra est employé par le planificateur principal du robot, suivi d'un algorithme de simplification pour éviter des points qui ne sont pas nécessaires dans le chemin (Bakambu et al., 2006). La figure 1.2 montre un exemple de cette sorte de chemin. À gauche nous avons le chemin avant la simplification et à droite le chemin après la simplification.

Le deuxième est « le chemin initial » qui est représenté par une fonction B-Spline cubique $\pi(\lambda) : \Re \rightarrow \Re^3$ avec $\lambda \in [0, 1]$ comme paramètre. Il est créé en faisant une interpolation par spline de la liste de points qui définissent le chemin de base. L'objectif

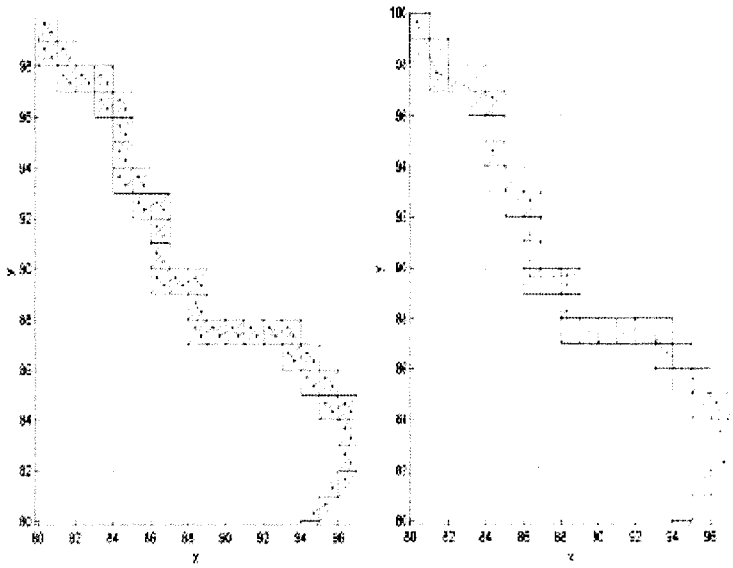


FIG. 1.2 Chemin de base

est d'éliminer les fortes irrégularités du chemin de base et de produire un chemin plus lisse et éventuellement plus facile à suivre par le robot.

Dans certains cas, des points additionnels pourront être ajoutés à l'ensemble de données du chemin de base afin de diminuer l'erreur d'interpolation de la composante z du chemin initial.

Le troisième est « le chemin local » qui est généré par une de nos solutions. Ce type de chemin est chargé de remplacer un sous-ensemble du chemin initial dans le but de contourner ou d'éviter les obstacles.

Le quatrième et dernier type est « le chemin final » défini lui aussi par fonction B-spline $\pi_F(\lambda) : \Re \rightarrow \Re^3$ avec $\lambda \in [0, 1]$ comme paramètre, et il sera la solution à notre problème

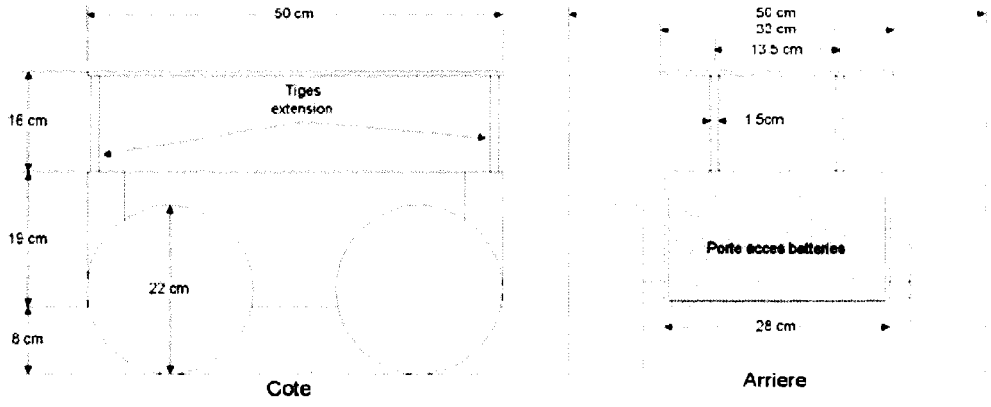


FIG. 1.3 Le Robot.

1.2.3 Le robot

Il s'agit d'un robot mobile à commande différentielle pour lequel on développera notre modèle. La figure 1.3 nous donne un bon aperçu de la géométrie et des dimensions du robot en question.

Nous considérons le robot comme une surface plane de forme carrée, ayant comme limites les dimensions du robot. Partant de cette considération, il faudra donc calculer l'équation du plan de la base du robot, et pour ce faire nous devons évaluer d'abord les positions des roues du robot PWs. Après avoir fait l'estimé de la position des quatre roues, il s'agit par la suite de trouver le plan qui s'ajuste le mieux aux quatre points. L'étape suivante consiste en un déplacement du plan dans la direction de l'axe Z d'une distance h égale à la distance entre la terre et la base du robot. On a opté pour utiliser l'analyse de composantes principales (PCA) (Ellis, 2002; The MathWorks, 2005) qui nous donne l'équation $aX + bY + cZ + d = 0$ décrivant la base du robot. La figure 1.4 montre le plan résultant après avoir fait le PCA aux quatre roues du robot.

Dans le paragraphe précédent, on a utilisé PWs comme la base de tous les calculs pour trouver la représentation du robot. Il nous faut donc décrire les étapes pour calculer

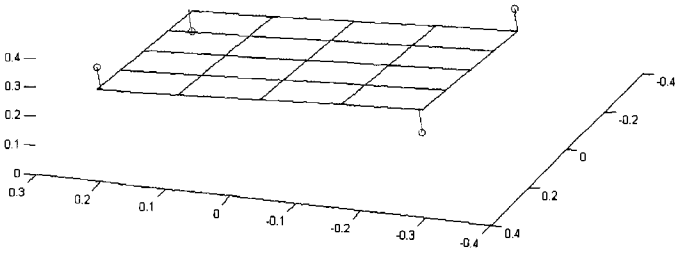


FIG. 1.4 Plan qui contient le robot, PCA.

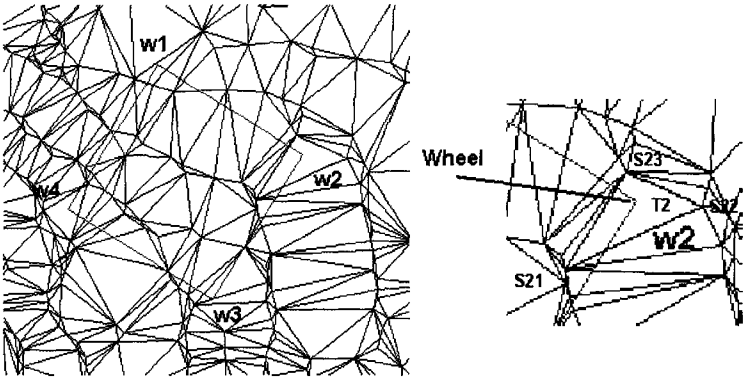


FIG. 1.5 Représentation du robot placé sur le terrain

PWs. La figure 1.5 montre dans la partie gauche, le robot sur le terrain et les triangles d'intérêts où les roues sont placées. Pour ce qui est de la partie droite, un zoom du secteur d'une roue du robot est illustré, indiquant les variables utilisés dans les développements.

- D'abord, calculons l'estimé de la position du centre du robot $PR = [PR_x PR_y PR_y]$ pour un point du chemin. Il suffit d'évaluer $PR = \pi(\lambda_i)$ pour $\lambda_i \in [0, 1]$.
- En suite, l'orientation pour le point $\pi(\lambda_i)$ avec l'équation suivante

$$\theta = \arctan(\pi'(\lambda_i)_y / \pi'(\lambda_i)_x). \tag{1.2}$$

- Définissons la matrice Ws

$$Ws = \begin{bmatrix} l/2 & l/2 & -l/2 & -l/2 \\ w/2 & -w/2 & -w/2 & w/2 \\ h & h & h & h \\ 1 & 1 & 1 & 1 \end{bmatrix} \tag{1.3}$$

Où l et w sont respectivement les mesures de longueur et de largeur du robot. La matrice Ws représente la position des roues relative au centre de la base du robot.

- Pour connaître la position des roues relative au repère de base PWs, on défini une matrice de transformation homogène comme suit.

$$TrMX = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & PR_x \\ \sin\theta & \cos\theta & 0 & PR_y \\ 0 & 0 & 1 & PR_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.4}$$

Donc, PWs va être calculé :

$$PWs = [TrMX][Ws] = \begin{bmatrix} PW1_x & PW2_x & PW3_x & PW4_x \\ PW1_y & PW2_y & PW3_y & PW4_y \\ PW1_z & PW2_z & PW3_z & PW4_z \\ 1 & 1 & 1 & 1 \end{bmatrix} \tag{1.5}$$

- Même si on a un premier estimé de PWs, due à l'irrégularité et/ou à l'inclination du terrain, les roues ne seront pas à la même altitude, donc nous actualisons la composante Z pour chacune des roues en analysant l'information que nous avons du terrain. La procédure pour actualiser ces valeurs est la suivante.
  - D'abord, avec les valeurs trouvées pour la position de chaque roue en x et y, nous sommes capable de trouver sur quel triangle T_i, la roue est placée. Chaque triangle T_i est cherché dans la matrice TR et les données de la position de ses sommets $PST_{ij} \in \Re^3$ sont recueillis de la matrice D. Les indices $i = 1, ..., 4$ et $j = 1, 2, 3$

sont utilisés respectivement, pour se référer à un triangle en particulier et à chacun de ses sommets.

- Pour chaque triangle, on calcule l'équation du plan $N \cdot (r - PST_{i1}) = 0$ qui le contient, où

$$\begin{aligned} r &= [x, y, z] \\ p &= PST_{i2} - PST_{i1} \\ q &= PST_{i3} - PST_{i1} \\ N &= \frac{p \times q}{\| p \times q \|} \end{aligned}$$

- Finalement la composante z de la position des roues PWs_z est calculée avec l'équation

$$PWi_z = (-N_x PWi_x - N_y PWi_y + (N \cdot PST_{i1}))/N_z \tag{1.6}$$

Pour obtenir l'estimé de la position des roues (Équation 1.5), nous avons calculé l'orientation du robot à partir du vecteur tangent au chemin pour un point donné (Équation 1.2). Les développements précédents sont basés sur l'hypothèse que les angles de roulis et de tangage (roll et pitch) doivent être petits. Étant donné que nous cherchons à amener le robot vers un chemin le plus sécuritaire et le moins coûteux possible, les régions du terrain qui présentent une inclinaison trop grande par rapport aux inclinaisons maximales permises par le robot sont évitées. Par conséquent, nous pourrions dire que l'approche utilisée pour obtenir l'estimé de la position des roues nous donne une bonne idée de la position du robot sur le terrain.

1.2.4 L'obstacle

D'une manière générale, un obstacle est une zone qui est inaccessible par le robot. Une classification selon le nature des obstacles a été faite afin de réduire la quantité

d'opérations nécessaires pour identifier les obstacles. Par la suite, nous présenterons les classes d'obstacle. De plus, la procédure d'identification se rattachant à chacune des classes sera décrite à la section 2.2.

Obstacle dû à la rugosité. Région avec rugosité excessive, dans cette catégorie d'obstacle on inclut autant les grands obstacles qui ne permettent pas le passage du robot que les petites roches qui pourraient toucher sa base et le coincer.

Obstacle dû à la pente. Secteurs avec une pente raide où le robot pourrait glisser produisant, par le fait même, des erreurs dans les systèmes de navigation ou par le risque de capotage.

Obstacle dû à la méconnaissance. Secteur dont l'information est incertaine, partiellement connue ou inconnue.

1.3 Définition du problème

Dans la section 1.2, nous avons défini la plupart des termes les plus importants qui seront utilisés au long de notre ouvrage et qui nous permettront de mieux comprendre le but de ce travail de recherche.

Comme l'indique le titre de ce projet, nous sommes à la recherche d'un chemin qui conduit le robot d'un point de départ quelconque à un point d'arrivée choisi. De plus, nous sommes dans un environnement qu'on ne connaît pas bien et qui n'est pas structuré. En effet, nous partons de l'information fournie par des capteurs sur une partie du terrain, que l'on actualise au fur et à mesure des déplacements effectués par le robot. Cependant, nous ne savons pas où sont les roches, les collines ou les trous qui pourront mettre fin à notre aspiration d'arriver au point choisi.

Abordons maintenant la concrétisation de la problématique qui est à l'origine de ce pro-

jet. Un chemin de base à travers le terrain est produit par le planificateur sur longues distances du robot. Ce chemin est produit avec certains critères d'optimisation (Bakambu et al., 2006), mais le robot est modélisé pour cet algorithme comme un point. La conséquence de cette simplification est que le chemin de base deviens donc, un chemin semi-libre, ce que signifie que le robot pourrait peut-être rencontrer un obstacle (Latombe, 1991). Quelques répercussions de cette assertion sont que le chemin de base permet au robot de passer à proximité ou entre des obstacles, augmentant ainsi le risque de collision, de dérapage, ou dans le pire des scénarios, guide le robot vers une collision inévitable. Notre objectif est donc de valider et/ou redéfinir ce chemin pour le rendre libre d'obstacles et plus sécuritaire pour le robot. Notre algorithme devra donc prendre en considération la géométrie du robot.

Pourquoi parle-t-on de valider un chemin ? Malgré les inconvénients mentionnés ci-dessus, le chemin de base est un bon estimé du chemin devant être suivi par le robot. De plus, il s'avère comme un des chemins les plus courts, et donc privilégié par certaines topographies du terrain. Il y a ainsi une bonne probabilité de détenir un chemin avec la totalité ou une grande partie de sa longueur libre d'obstacles. Si le chemin initial ne présente aucun risque de collision après avoir été analysé par notre algorithme, le chemin résultant va donc être identique au chemin initial, à moins de vouloir minimiser d'autres caractéristiques qui n'ont pas été prises en compte lors de sa génération. Dans le cas où des obstacles sont présents, le chemin initial sera redéfini localement. Par la suite le chemin résultant aura donc, une partie du chemin initial et de nouveaux morceaux de chemin définis localement.

Une deuxième partie du problème est liée à la présence de fortes irrégularités dans le chemin de base, qui sont dues à la manière dont il a été construit. En conséquence, la solution proposée devrait les éliminer afin de donner un chemin lisse et plus facile à suivre.

1.4 Critères

Pour tenter de solutionner le problème de la planification de chemin pour ce cas spécifique, nous avons tenté plusieurs approches. Nous n'avons donc retenu que celles qui présentaient une solution convenable selon notre objectif. Dans cette section, nous donnerons quelques critères employés pour évaluer la performance des solutions proposées dans le chapitre 2.

Critère 1. L'algorithme doit donner comme réponse un chemin libre d'obstacle qui minimise le risque de collision.

Critère 2. L'algorithme doit éliminer les fortes irrégularités du chemin de base et éviter les changements abrupts de l'orientation du robot, afin de nous fournir un chemin final suffisamment lisse et facile à suivre par le robot.

Critère 3. La rapidité de calcul de l'algorithme est prise en considération. La solution doit être produite dans le temps assez court pour être compatible avec une mission d'exploration.

CHAPITRE 2

RÉSOLUTION DU PROBLÈME

Ce chapitre portera sur les solutions proposées pour le problème décrit à la section 1.3. Il comprendra un ensemble d'idées retenues après plusieurs mois de développement qui seront présentées en deux parties. La première partie appelée le modèle générique, aborde la solution d'une manière générale, sans entrer trop en détail dans la façon dont l'algorithme évalue certaines fonctions. Puis une deuxième section portera sur les particularités des solutions où on élabore plus en détail la façon dont les calculs ont été réalisés.

2.1 Le modèle générique

Dans cette section, nous présenterons une description générale de la solution donnée au problème proposé. Nous avons étudié trois algorithmes basés sur la théorie de commande optimale. Plus exactement, le problème d'optimisation de chemin qui cherche à trouver le chemin le long duquel une fonction de coût est minimal (Shiller and Chen, 1990).

Dans les deux premiers algorithmes, nous avons un chemin initial $\pi_I(\lambda)$ et une fonction de coût à minimiser $f(\lambda)$ de la forme de l'équation 2.1. En premier lieu, nous évaluons la fonction $f(\lambda)$ pour le chemin $\pi_I(\lambda)$, puis nous modifions le chemin à chaque itération et recalculons la valeur de la fonction $f(\lambda)$ pour le nouveau chemin, on répète la même procédure jusqu'à trouver le chemin qui minimise la fonction objective $f(\lambda)$. En théorie, trouver le chemin optimal semble un problème facile. Il suffit d'essayer tous les chemins possibles, et de comparer les valeurs calculées pour la fonction objectif, puis choisir le meilleur. Cependant, en pratique, il n'est pas évident de le faire parce que le temps pour trouver la solution est aussi un facteur déterminant.

La troisième approche est basée sur la méthode de résolution de la programmation dynamique. La première étape est de faire emprunter le chemin initial par le robot afin d'identifier les endroits où il y a des obstacles. S'il y en a un, une aire doit être définie sur la carte du terrain et devra comporter le début et la fin de l'obstacle. Dans cette aire, nous définissons une grille où chacun de ses nœuds a cinq directions possibles de mouvement. C'est-à-dire, pour chaque nœuds de la grille, on établit, si possible, les chemins droits qui uniront celui-ci avec ses voisins. Pour chacun de ces chemins, un coût relatif à sa faisabilité y est associé, Lorsqu'on a une connectivité entre les nœuds de la grille, nous faisons une recherche pour trouver le chemin avec le coût minimal pour aller d'un point à un autre. Ce chemin serait donc le chemin local qui permettrait de contourner l'obstacle.

$$f(\lambda) = \int_0^1 (\alpha f_1(\lambda) + \beta f_2(\lambda) + \ldots \rho f_n(\lambda)) d\lambda \tag{2.1}$$

L'équation 2.1 représente la fonction de coût $f(\lambda)$ qui se compose de plusieurs sous-fonctions. Ici α, β et ρ sont les différents facteurs de poids associés à chaque sous-fonctions $f_i(\lambda)$, selon leur importance. En voici une description.

Longueur. C'est la longueur totale du chemin, la distance totale à parcourir par le robot.

$$f_1(\lambda) = \left\| \frac{\partial \pi(\lambda)}{\partial \lambda} \right\| \tag{2.2}$$

Courbure. C'est la courbure associée à chaque point du chemin. Elle est calculée pour chaque λ_i avec l'équation qui suit.

$$f_2(\lambda) = \frac{\| \pi'(\lambda) \times \pi''(\lambda) \|}{\| \pi'(\lambda)^3 \|} \tag{2.3}$$

Angles. Cette fonction de pénalité a une valeur différente de zéro après qu'un seuil pour les angles de roulis et de tangage du robot soit atteint. Le roulis γ et le tangage ψ représentent la rotation du robot autour de l'axe X et Y respectivement.

$$f_3(\lambda) = f_\psi(\lambda) + f_\gamma(\lambda) \tag{2.4}$$

Où f_ψ et f_γ sont des fonctions de la forme :

$$f(x) = \begin{cases} 0 & \text{if } |x| \leq a \\ (|x| - a)^2 & \text{if } x > a \end{cases} \tag{2.5}$$

Le seuil est la valeur associée à a, lorsque x est la valeur d'un des angles ψ ou γ.

Obstacle. La fonction a comme but d'évaluer si un point spécifique du chemin ou du terrain est un obstacle pour le robot. La fonction est donnée par l'équation :

$$f_4(\lambda) = \begin{cases} 0 & \text{Si } \forall s_{ij}, PST_{ij}z < zmax_{ij} \\ \sum_{i=1}^{n}(PST_{ij}z - zmax_{ij})^2 & \text{Si } \exists s_{ij}, PST_{ij}z \geq zmax_{ij} \end{cases} \tag{2.6}$$

Dans l'équation 2.6, s_{ij} sont les sommets des triangles qui sont sous le robot où sa coordonnée en Z est représentée par $PST_{ij}z$. Finalement, $zmax_{ij}$ est le seuil calculé indépendamment pour chaque $PST_{ij}z$.

Dans la section 2.2, la détection d'obstacles sera traitée plus en détail.

Dans le premier algorithme utilisé, il s'agit de résoudre un problème d'optimisation sans contraintes, la fonction $f(\lambda)$ contient l'ensemble de sous-fonctions ci-dessus. Les fonctions $f_1(\lambda)$ et $f_2(\lambda)$ auront pour objectif de réduire la longueur du chemin et d'augmen-

ter le rayon de ses courbes, tandis que $f_3(\lambda)$ et $f_4(\lambda)$ s'efforcent de maintenir le chemin éloigné des obstacles.

Évidement, les facteurs de poids utilisés pour $f_3(\lambda)$ et $f_4(\lambda)$ sont beaucoup plus élevés que pour les fonctions $f_1(\lambda)$ et $f_2(\lambda)$. L'effet d'assigner un facteur de poids plus élevé est de donner une plus grande importance à maintenir le chemin libre d'obstacles plutôt que d'obtenir le chemin le plus court.

Toujours avec l'objectif de trouver un chemin libre d'obstacles, le chemin sera complètement faisable si la valeur de $f_3(\lambda)$ et $f_4(\lambda)$ est toujours zéro. A partir de cette idée, la deuxième approche est alors envisagée, et il s'agit de traiter le problème plutôt comme un problème d'optimisation avec contraintes.

Comme nous venons de le mentionner, le deuxième algorithme envisage d'utiliser l'optimisation avec des contraintes pour donner la solution souhaitée. Maintenant, la fonction de coût $f(\lambda)$ inclut seulement $f_1(\lambda)$ et $f_2(\lambda)$ et les contraintes seront les fonctions $f_3(\lambda)$ et $f_4(\lambda)$ qui devront être zéro pour le chemin final. Avec ce deuxième raisonnement nous devenons plus exigeants, étant donné qu'une solution n'est possible que si la valeur de $f_3(\lambda)$ et de $f_4(\lambda)$ est égal à zéro. De plus, les facteurs de poids ne jouerons plus un rôle aussi important que pour la première approche.

Dans le troisième algorithme, lorsqu'il s'agit des segments de chemin droits entre les nœuds de la grille, la courbure n'est plus au rendez-vous. Ainsi, la fonction de coût sera composée par la somme de $f_1(\lambda)$, de $f_3(\lambda)$ et de$f_4(\lambda)$. Comme dans le premier algorithme, $f_3(\lambda)$ et $f_4(\lambda)$ auront un facteur de poids plus élevé. Malgré les bons résultats de cette méthode avec $f(\lambda)$ tel que défini précédemment, nous avons opté pour assigner une valeur très élevée à $f(\lambda)$, si $f_3(\lambda)$ ou $f_4(\lambda)$ est différent de zéro, afin de réduire le temps de calcul et ainsi éliminer toute connectivité où elle est interdite.

2.2 Mise en œuvre

Dans la section précédente, la solution du problème a été abordée d'un manière grossière, dans le but d'avoir une vision générale des différentes solutions envisagées, mais sans s'attarder dans les particularités ou dans les détails de calcul. Cette section est divisée en deux parties. Une première partie porte sur un ensemble des procédés mis en commun pour les solutions proposées, puis une deuxième partie qui est un complément de la section 2.1 et présentera d'une façon plus approfondie des procédés particuliers aux différentes solutions.

2.2.1 Lissage du chemin

Dans la section 1.2.2, nous avons parlé du chemin de base, qui se veut un ensemble de données qui contient les coordonnées des points à traverser par le robot. Comme nous pouvons constater à la figure 1.2 chacun de ces points ou positions à être atteints par le robot, sont liés entre eux par des segments de ligne droite. Ceci fait en sorte que notre chemin initial présente une irrégularité élevée qui n'est pas désirée.

Pour faire face à ce problème, on a redéfini le chemin de base, en appliquant aux ensembles de points, un algorithme d'interpolation, et comme mentionné dans la section 1.2.2, la fonction résultant de cette interpolation sera appelée chemin initial. Sa représentation mathématique est donné par une fonction B-Spline cubique $\pi(\lambda)_I : \Re \rightarrow \Re^3$ avec $\lambda \in [0, 1]$.

Maintenant, le chemin initial est le point de départ de nos méthodes de solutions. À la figure 2.1, nous présentons un exemple du processus de lissage du chemin. Nous pouvons nous rendre compte que le chemin initial, même s'il ne passe pas sur tous les points qui font partie du chemin de base, nous fourni une bonne approximation et demeure près

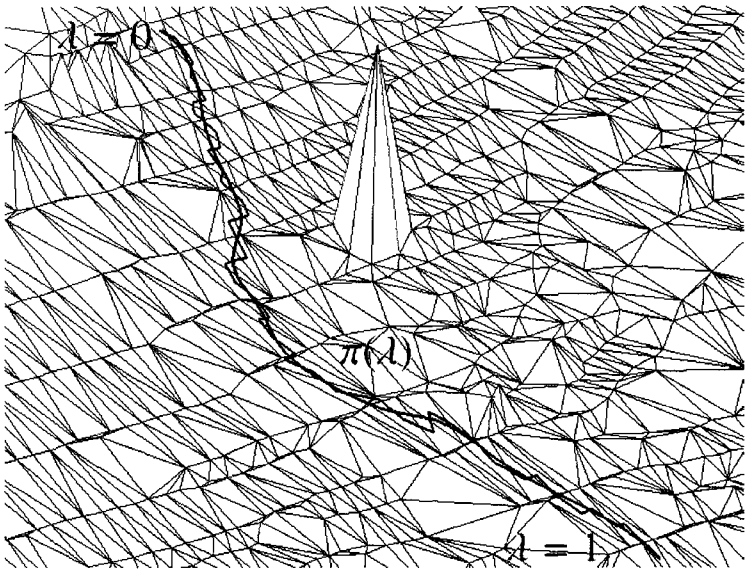


FIG. 2.1 Processus de lissage. Chemin de base(noir) et chemin initial(rouge)

du chemin initialement défini. En réduisant les changements abruptes de direction, nous obtenons un deuxième chemin moins accidenté et plus facile à suivre compte tenu de la cinématique du robot mobile.

2.2.2 La détection d'obstacles

À la section 1.2.4, nous avons classifié les obstacles en trois catégories dont tous représentent des secteurs interdits pour le robot où le chemin final ne pourra pas être défini. Mais nous n'avons pas abordé comment ces obstacles sont identifiés sur le terrain.

La description du procédé sera de nouveau divisée également en trois parties (comme à la section 1.2.4). La première sera donc, l'identification d'obstacles dû à l'aspérité du terrain, la deuxième, sera l'identification d'obstacles dû à l'inclinaison et la dernière portera sur l'identification d'obstacles dû à la méconnaissance du terrain.

Obstacle dû à la rugosité. L'approche utilisée pour la détection de ce genre d'obstacle

est une comparaison géométrique entre le terrain et l'aire occupée par le robot. Dans la section 1.2.3, nous avions développé une façon de représenter le robot comme une surface plane. Ainsi, nous pouvions regarder si pour l'aire définie par cette surface plane il y aurait des points d'intersection avec le terrain, puis dans l'affirmative cette configuration pour le robot sera interdite et indiquée comme obstacle.

Ci-dessous, le pseudo algorithme présente l'identification des obstacles due à la rugosité du terrain.

Pseudo algorithme obstacle-rugosité

paramètres d'entre :

- Information du terrain :
  - Matrice D
  - Matrice Tr

  Voir section 1.2.1 pour plus d'information.
- Information du robot :
  - Position du robot $PR = [x, y, z]$
  - Orientation du robot θ
  - Équation du plan qui décrit le robot $aX + bY + cZ + d = 0$

  Voir section 1.2.3.

Pour chaque PR et θ à vérifier.

Faire .

1. Chercher dans la matrice D les coordonnées correspondantes des points s_{ij} qui se trouvent sous le robot, où chaque s_{ij} est le sommet des triangles qui décrivent le terrain.
2. Pour chaque point s_{ij}, nous trouvons la valeur maximale $zmax_i$ qui pourrait avoir dans sa coordonnée z, pour ne pas être considéré comme un obstacle. Pour calculer $zmax_i$, nous dégageons la variable Z de l'équation du plan $aX + bY + cZ + d = 0$, le dernier pas est de remplacer dans l'équation

résultant $zmax_i = Z = (-aX - bY - d)/c$ les valeurs de $X = s_{ij}x$ et $Y = s_{ij}y$.

3. Si $\forall s_{ij}, s_{ij}z < zmax_i$, alors la configuration du robot PR, θ ne pose pas de problème et dans un premier temps et sera considéré comme libre (no-obstacle). Mais si $\exists$ un s_{ij} pour qui $s_{ij}z \geq zmax_i$ donc cette configuration est marquée comme interdite lorsqu'un obstacle est présent et une valeur de pénalité est calculée pour elle selon l'équation 2.6.
4. Si la dernière étape donne comme résultat une configuration permise pour le robot, afin de garantir qu'il s'agit vraiment d'une aire libre d'obstacles, nous analysons les points voisins du robot dans la recherche d'une possibilité d'interférence latérale. Pour ce faire, nous suivons les étapes suivantes :
   - chercher dans la matrice Tr tous les triangles pour qui les points s_{ij} sont un de ses sommets ;
   - pour chaque triangle, chercher les sommets $\hat{s_{ij}}$ qui sont à l'extérieur de l'aire occupée par le robot ;
   - déterminer l'équation de la ligne $L_p = s_{ij} + t(\overrightarrow{\hat{s_{ij}}s_{ij}})$ qui lie le point s_{ij} avec $\hat{s_{ij}}$;
   - vérifier pour chaque ligne s'il n'y a pas d'intersection avec le plan qui représente le robot. S'il $\exists$ une L_p, laquelle intersecte le plan du robot, donc la configuration du robot sera interdite, de plus, une valeur de pénalité sera calculée pour elle.

La figure 2.2 représente les différentes variables utilisées dans l'identification d'obstacles dus à la rugosité du terrain.

La figure 2.3 montre un exemple de l'algorithme développé ci-dessus, où nous pouvons voir le robot représenté comme une surface plane qui heurte un obstacle. Les points noirs sont les points nécessaires pour l'analyse de la présence d'obstacle, les deux points rouges montrent l'intersection entre le robot et l'obstacle.

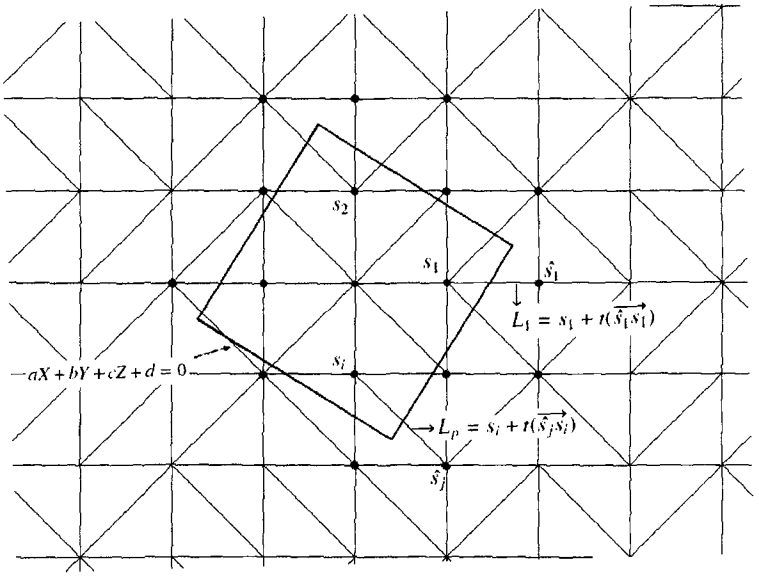


FIG. 2.2 Éléments servants à la détection d'obstacles

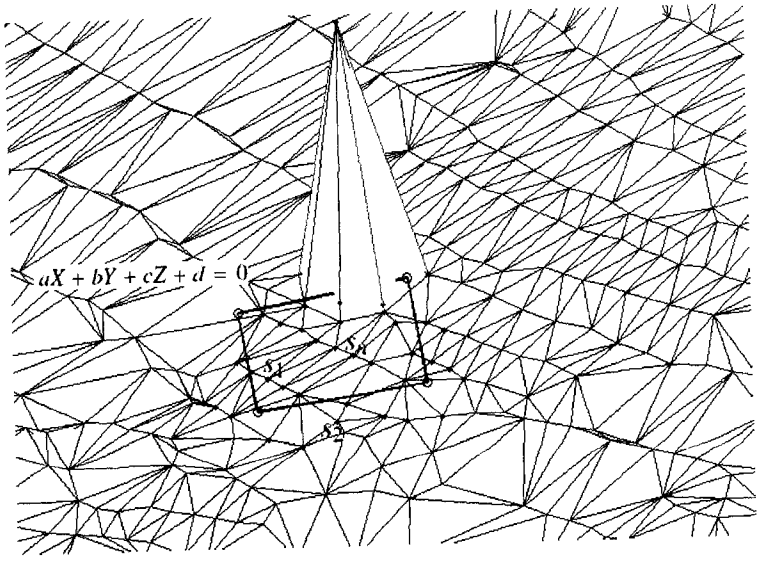


FIG. 2.3 détection d'obstacle

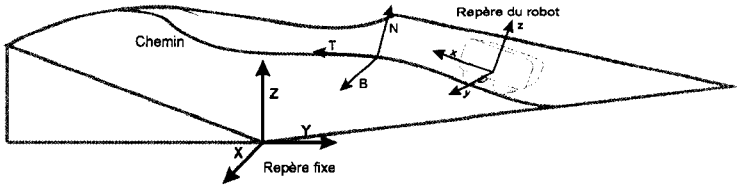


FIG. 2.4 Référentiel du robot

Obstacle dû à la pente. Pour des obstacles liés à une inclination élevée, nous avons appliqué la fonction de pénalité donnée par l'équation 2.4. La valeur estimée des angles de roulis et de tangage est calculée à partir de la matrice de rotation définie entre un repère de référence fixe et un repère attaché au robot.

Le repère du robot est attaché à son centre géométrique (voir la figure 2.4), où l'axe de X est définis du centre du robot se dirigeant vers l'avant de lui et perpendiculaire à l'axe de rotation des roues avant. L'axe Z est le vecteur normal du plan qui représente la base du robot mais en direction du haut en s'éloignant du sol. L'axe Y est celui qui est perpendiculaire aux deux derniers axes et qui complète le repère du robot selon la règle de la main droite. Les vecteurs unitaires n, o et a définissent le repère du robot exprimé dans le repère fixe.

Lorsqu'on connaît n, o et a, la valeur numérique des angles de roulis γ et de tangage ψ est calculée avec l'aide des équations suivantes : (De Santis, 2005)

$$
\begin{aligned}
Rot(\theta,\psi,\gamma) &= \begin{pmatrix} C\theta C\psi & C\theta S\psi S\gamma - S\theta C\gamma & C\theta S\psi C\gamma + S\theta S\gamma \\ S\theta C\psi & S\theta S\psi S\gamma + C\theta C\gamma & S\theta S\psi S\gamma - C\theta S\gamma \\ S\psi & C\psi S\gamma & C\psi C\gamma \end{pmatrix} \\
Rot(\theta,\psi,\gamma) &= \begin{pmatrix} n_x & o_x & a_x \\ n_y & o_y & a_y \\ n_z & o_z & a_z \end{pmatrix} \\
\psi &= \arctan 2\left(-n_z, \sqrt{n_x^2 + n_y^2}\right) \in (-\pi/2, \pi/2) \qquad (2.7)
\end{aligned}
$$

$$\gamma = \arctan 2\left(\frac{o_z}{\cos(\psi)}, \frac{a_z}{\cos(\psi)}\right) \tag{2.8}$$

Obstacle dû à la méconnaissance. Le dernier type d'obstacle correspond aux secteurs où l'information de la topographie du terrain n'est pas disponible. Il est le plus facile à identifier en vertu de la façon dont le robot est modélisé, c'est-à-dire, lorsque le chemin essai de pénétrer dans un secteur inconnu. Notre algorithme ne peut donc pas calculer l'équation du plan qui représente le robot parce qu'il n'a pas l'information de l'estimé de toutes les positions des roues. À ce moment là, il s'avère impossible de continuer l'exécution des algorithmes de détection d'obstacles antérieurement présentés, donc les fonction $f_3(\lambda)$ et $f_4(\lambda)$ seront donc remplacées par une valeur élevée de pénalité, faisant du chemin une solution non optimale.

Maintenant que nous avons décrit les procédures en commun pour l'ensemble de solutions, il est temps de parler un peu plus des particularités de solutions proposées dans la section 2.1.

Pour les deux premiers algorithmes de solution, lorsqu'il s'agit de résoudre le problème de la planification du chemin en le regardant comme un problème d'optimisation, nous cherchons à avoir pour chaque itération, un chemin différent, en partant du chemin initial $\pi_I(\lambda)$.

Alors, la façon de procéder est la suivante :

1. Calculer le point initial $PI = [\pi_I(0)_x, \pi_I(0)_y]$ et le point final $PF = [\pi_I(1)_x, \pi_I(1)_y]$ du chemin.
2. Définir un vecteur λCP de dimension n, où chaque $\lambda CP_i \in (0, 1)$ et sera défini en ordre croissant.

3. Évaluer la valeur de $\pi_I(\lambda)$ pour chaque λCP_i et créer avec ces valeurs une matrice de points de contrôle CP.

$$CP = \begin{pmatrix} CP1_x & CP1_y \\ CP2_x & CP2_y \\ \vdots & \vdots \\ CPn_x & CPn_y \end{pmatrix} = \begin{pmatrix} \pi_I(\lambda CP_1)_x & \pi_I(\lambda CP_1)_y \\ \pi_I(\lambda CP_2)_x & \pi_I(\lambda CP_2)_y \\ \vdots & \vdots \\ \pi_I(\lambda CP_n)_x & \pi_I(\lambda CP_n)_y \end{pmatrix} \tag{2.9}$$

4. Évaluer la fonction de coût $f(\lambda)$ le long du chemin $\pi_I(\lambda)$.
5. Modifier les points de contrôle afin de générer un nouveau chemin.

$$CP = \begin{pmatrix} CPi_x \pm \delta CP_x & CPi_y \pm \delta CP_y \end{pmatrix} \tag{2.10}$$

6. Générer un nouveau chemin $\pi_v(\lambda)$ qui traverse les nouveaux points de contrôle et qui commence et fini respectivement en PI et PF.
7. Évaluer la fonction de coût $f(\lambda)$ le long du nouveau chemin $\pi_v(\lambda)$.
8. Effectuer les étapes 5, 6 et 7 jusqu'à trouver le chemin pour lequel $f(\lambda)$ est au minimum.

La figure 2.5 montre d'une façon graphique la procédure décrite ci-dessus. La partie supérieure de la figure est un exemple du processus d'optimisation, les chemins en mauve sont seulement quelques chemins représentatifs pour illustrer la variation du chemin.

Pour effectuer le processus d'optimisation, nous avons appliqué des méthodes d'optimisation d'ordre zéro, c'est-à-dire, qu'ils ne requièrent pas de calcul du gradient (Lagarias et al., 1998; Kirk, 2004). Nous avons utilisé l'« optimization toolbox » de Matlab pour effectuer le processus d'optimisation sans contraintes, tandis que pour le processus d'optimisation avec contraintes, une adaptation du code a été requise (D'Errico, 2006).

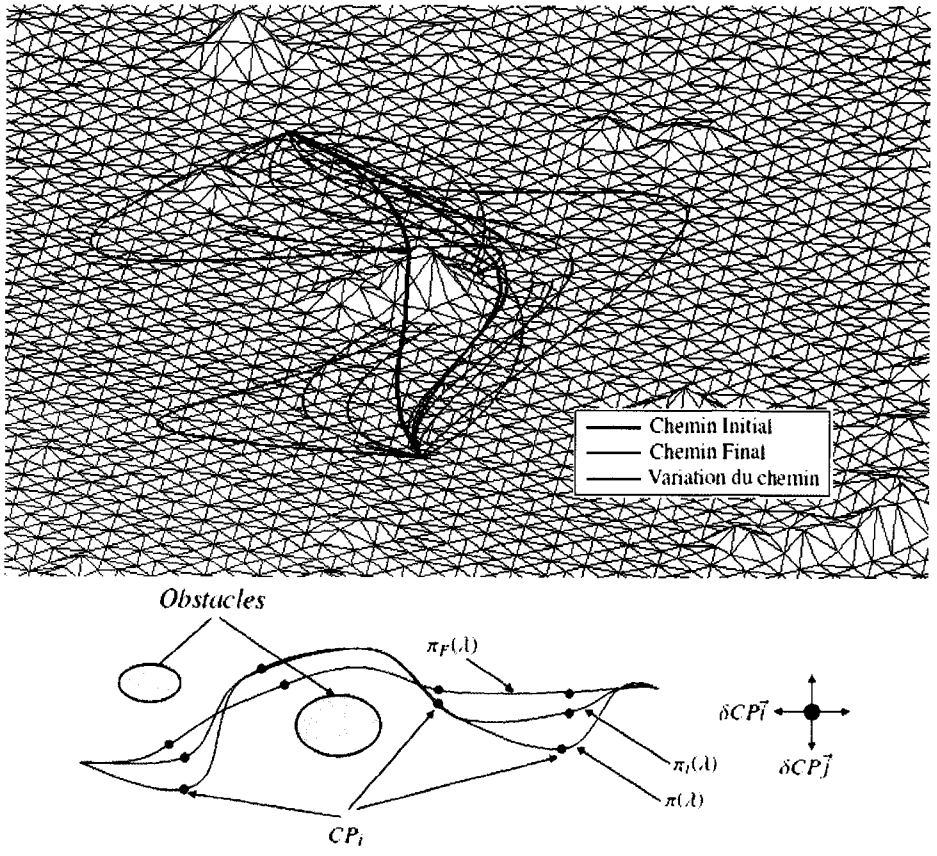


FIG. 2.5 Processus d'optimisation

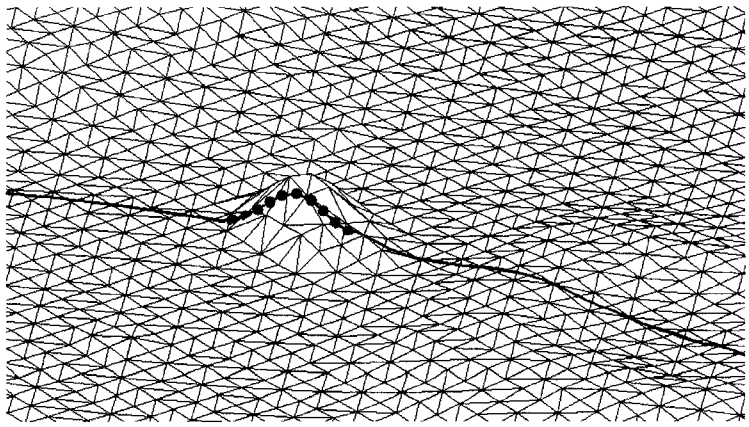


FIG. 2.6 Obstacle

Notre troisième approche est basée sur l'idée que le chemin initial présente seulement des problèmes dans certaines zones. Donc une solution de planification locale de chemin est proposée, en appliquant un algorithme d'optimisation de programmation dynamique.

Dans cet ordre d'idée, il faudra d'abord identifier les zones ou morceaux du chemin où il y a des obstacles pour pouvoir effectuer la planification locale et redéfinition du chemin. Ci-dessous seront exposées les étapes nécessaires pour faire la planification locale.

1. Pour chaque $\lambda_i \in [0, 1]$, déterminer si la configuration du robot, donnée par le chemin initial $\pi_I(\lambda_i)$, est valide ou libre d'obstacles. Dans chaque configuration où un obstacle est détecté, celle-ci sera marquée comme interdite (voir la figure 2.6).
2. Identifier le début et la fin des obstacles, en analysant les zones marquées dans l'étape précédente. Chaque $\pi_I(\lambda_i)$ marquée comme interdite pour qui $\pi_I(\lambda_{i-1})$ est valide, sera donc le début d'un obstacle. ainsi que chaque $\pi_I(\lambda_i)$ marquée comme interdite pour qui $\pi_I(\lambda_{i+1})$ est valide, sera donc la fin d'un obstacle.
3. Déterminer la proximité entre obstacles. Mesurer la distance euclidienne entre la fin et le début de deux obstacles consécutifs, si la distance est plus petite que deux fois la longueur du robot, les deux obstacles seront considérés comme étant un

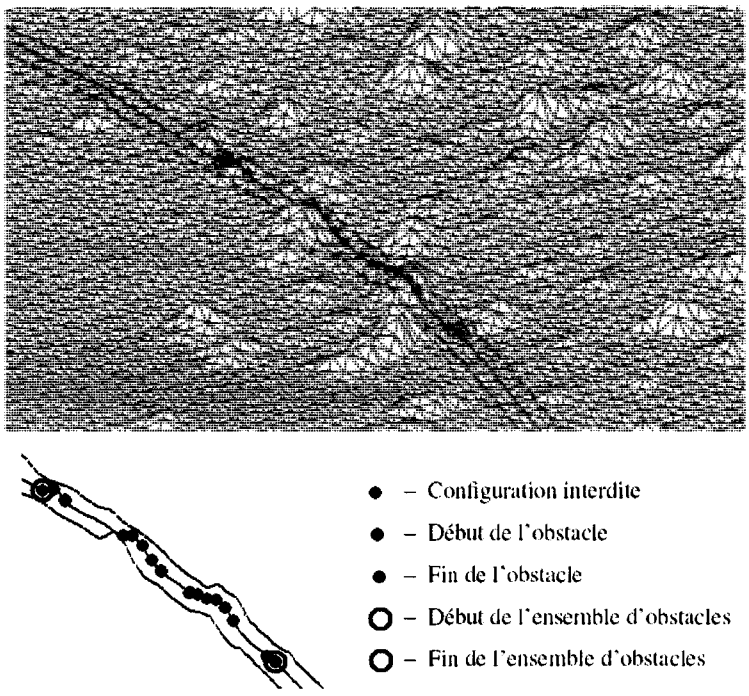


FIG. 2.7 Ensemble d'obstacles

seul (voir la figure 2.7).

4. Pour chaque obstacle.

   (a) Définir le point PI et le point PF. Ils seront respectivement, le point initial et le point final du chemin local cherché. Ces points appartiennent au chemin initial $\pi_I(\lambda)$, et ils se trouvent respectivement à une distance égale à la longueur du robot du point initial et final de l'obstacle.

   (b) Isoler de la carte, une aire autour de l'obstacle qui comporte PI et PF.

   (c) Définir la distance entre les nœuds de la grille Δg comme suit.

$$\Delta g = \frac{\sqrt{(PI - PF)^2}}{ceil((\sqrt{(PI - PF)^2})/l)} \tag{2.11}$$

   (d) Définir une grille entre les points PI et PF.

   (e) Calculer le coût associé à chaque segment de droite qui lie les nœuds de la

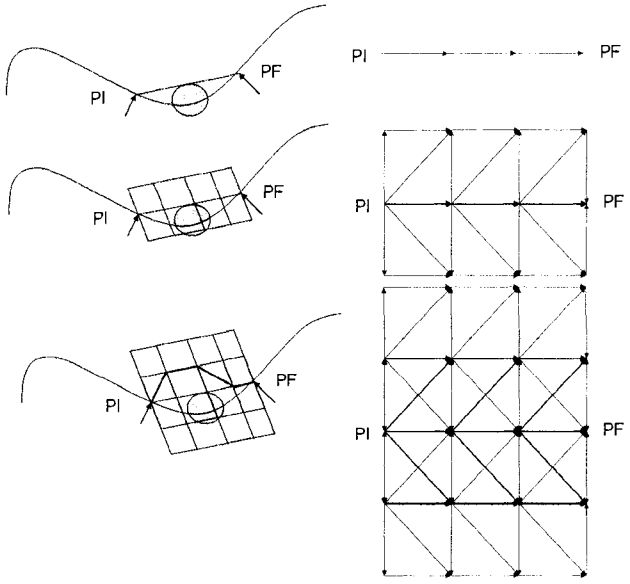


FIG. 2.8 Planification local de chemin

grille. Chaque nœuds est lié par segments de droite a cinq nœuds voisins.

(f) Chercher le chemin le moins coûteux entre les points PI et PF.

Avec le but de réduire le temps de calcul, la grille employé par la recherche du chemin local, est croissante. On augmente la quantité de rangées à chaque itération jusqu'à ce qu'elle puisse nous fournir le chemin local qui contourne l'obstacle.

La figure 2.8 illustre la procédure précédemment décrite et montre comment la grille est redéfinie pour chaque itération. La partie droite de la figure montre en rouge, les segments de droite définies pour chaque itération afin d'augmenter les probabilités de trouver un chemin admissible.

Après avoir trouvé les chemins locaux nécessaires pour contourner tous les obstacles identifiés à la étape 1. Il suffit maintenant de redéfinir le chemin final comme la juxtaposition des morceaux de chemin initial n'ayant pas été affectés par la planification locale et des nouveaux chemins locaux évitant les obstacles.

CHAPITRE 3

RÉSULTATS

Les chapitres précédents ont été consacrés à présenter la définition du problème et des différentes solutions proposées afin de le résoudre. Cependant, nous n'avons jamais parlé des performances des solutions proposées. Nous allons aborder pourquoi, dans le chapitre 2, nous avons présenté trois approches plutôt qu'une seule.

Ce chapitre est divisé en trois parties. La première porte sur l'évaluation de chacune des solutions énoncée dans le chapitre 2. Par la suite, la deuxième section présentera une série d'exemples où sera illustré comment les différentes solutions performent. Pour ce qui est de la dernière partie de ce chapitre, des possibilités de recherches futures et d'améliorations à effectuer sur les solutions sont proposées.

3.1 Analyse

Dans cette section, nous allons discuter de la performance de chacune des solutions données à notre problème. Les critères utilisés pour les évaluer ont été présentés à la section 1.4.

Mais pourquoi avoir présenté au chapitre 2 trois solutions différentes plutôt que de choisir parmi elles, celle qui représente la meilleure performance en oubliant les autres ? La réponse à cette question est simple et est basée sur l'idée de montrer l'évolution du travail de recherche et de documenter les raisons pour lesquelles nous n'étions pas satisfaits de la première approche, nous obligeant ainsi à migrer vers d'autres solutions.

3.1.1 Algorithme optimisation sans contraintes

Malgré que cette approche fournit une solution au problème dans la plupart des cas, elle a obtenu la pire performance basée sur les trois critères d'évaluation.

Le premier critère évalue la fiabilité de la méthode, puisque l'on cherche à avoir la certitude que notre chemin final est libre d'obstacles. La méthode qui utilise l'optimisation sans contraintes présente le problème qu'elle ne donne pas toujours un chemin libre d'obstacles.

Une des causes de la non-réussite de cet algorithme est la présence de minimums locaux lors de la recherche du chemin, c'est-à-dire, l'algorithme d'optimisation qui part du chemin initial et qui s'arrête lorsqu'il atteint un minimum dans son voisinage. Jusqu'ici, il n'y a rien de surprenant, mais lorsque nous analysons le chemin résultant, nous nous apercevons qu'il traverse un obstacle. En général, il s'agit d'un obstacle de grande magnitude. La raison attribuée à ce fait est que le chemin initial traverse cet obstacle. De plus notre algorithme en essayant de le contourner augmente la valeur de la fonction $f(\lambda)$ faisant en sorte que chaque nouveau chemin soit rejeté.

Ce phénomène est plus fréquent, comme nous l'avons dit, lorsque le chemin initial pénètre un obstacle assez grand. Donc, lorsque la variation effectuée au chemin initial n'est pas suffisamment grande pour faire dévier celui-ci de l'obstacle, on a augmenté la longueur total du chemin et aussi la courbure, mesurées respectivement avec $f_1(\lambda)$ et $f_2(\lambda)$, faisant que l'algorithme d'optimisation ne continue pas dans cette direction de recherche. De plus, si lorsque l'on cherche dans la direction contraire, nous rencontrons le même phénomène, l'algorithme ne cherche donc plus à faire sortir le chemin de l'obstacle, en nous donnant une chemin avec obstacle mais localement optimal selon les autres critères.

Pour les obstacles d'une dimension mineure, le problème est moins fréquent lorsqu'une variation du chemin initial dans le voisinage est capable d'envoyer ce dernier hors de la position où l'obstacle se trouve.

Pour ce qui est du deuxième critère d'évaluation, nous pouvons dire que pour cette solution, l'algorithme est capable de nous fournir un chemin qui n'est plus accidenté et où les changements d'orientation ont été aussi minimisés. Il faut dire que le chemin résultant est le plus court dans le voisinage du chemin initial.

Par contre, l'évaluation selon le troisième critère n'est pas du tout favorable. Le temps de calcul est élevé et il est directement proportionnel à la longueur de la trajectoire ainsi que la quantité de points de contrôle choisis. Dans la section 3.2, des exemples de solutions seront présentés avec le temps de calcul requis pour une trajectoire choisie.

Les bons résultats de cette approche dépendent de la capacité de l'algorithme d'optimisation de faire sortir du chemin initial les aires occupées par des obstacles. Une fois que l'algorithme rencontre un chemin libre d'obstacles, il est capable de minimiser la longueur et la courbure tout en évitant les obstacles. Pour augmenter la chance de réussite dans la recherche d'un tel chemin, nous pouvons d'abord effectuer la procédure de détection d'obstacles sur le chemin, utilisé pour la solution de la programmation dynamique. Lorsque les positions des obstacles sur le chemin sont connues, nous pouvons définir un point de contrôle du chemin au milieu de chaque obstacle. De cette façon, nous modifions directement la partie du chemin où il y a des obstacles. Le problème avec cette dernière idée, est que le temps de calcul est accru.

L'utilisation de cet algorithme de solution présente quelques inconvénients. Le premier inconvénient est relié aux facteurs de poids assignés à chacune des fonctions qui forment $f(\lambda)$. Ces facteurs sont-ils difficiles à calibrer ? Ils dépendent de chaque chemin en particulier. À titre d'exemple, imaginons un chemin très long, par conséquent la valeur de

$f_1(\lambda)$ (longueur) sera élevée, et ainsi lorsque les variables utilisées pour $f_3(\lambda)$ et $f_4(\lambda)$ seront proches des seuils établis pour chacune d'elles ; malgré le poids élevé assigné à ces fonctions, sa valeur commencera à jouer un rôle important dans l'algorithme d'optimisation, occasionnant ainsi un risque de collision plus grand auquel le robot sera soumis.

Le deuxième inconvénient est relié à la quantité de points de contrôle. Plus grande est la quantité de points de contrôle définis pour le chemin, plus grande est la capacité de définir de nouveaux chemins plus précis pour la même aire de recherche. Cependant, nous augmentons considérablement le temps de calcul, lorsqu'il s'agit de la variation de plus de points. Ainsi, nous avons pu nous apercevoir, que plus le nombre de points de contrôle choisis est élevé, moins grande est l'influence de la variation d'un de ces points sur le chemin au complet.

Étant donné les problèmes de cette approche, surtout dans la fiabilité, nous avons décidé de migrer vers une autre approche. Celle-ci comporte l'utilisation d'un algorithme d'optimisation avec contraintes.

3.1.2 Algorithme avec optimisation avec contraintes

Basée toujours sur l'idée de l'utilisation d'algorithmes d'optimisation, en consultant la littérature sur de problèmes similaires, l'ouvrage de Shiller et Chen (Shiller and Chen, 1990) à retenu notre attention. Ces derniers y présentent aussi un chemin par B-Spline, et utilisent, afin de trouver le chemin optimal, un algorithme d'optimisation dont la fonction de coût possède une partie à minimiser, telle que la distance, et un facteur de pénalité dû à la quantité de points du chemin qui pénètre dans de zones interdites.

À partir de cette idée, nous avons décidé de traiter le problème comme un problème d'optimisation avec contraintes. De cette manière, la difficulté due à l'assignation des facteurs de poids disparaissait, lorsque la solution est valide si les contraintes sont zéro.

Suivant ces modifications nous avons obtenu un algorithme qui selon le premier critère est meilleur que le précédent. Lorsque $f_3(\lambda)$ et $f_4(\lambda)$ doivent être zéro pour obtenir le chemin désiré. Maintenant que nous avons une solution à l'aide de cet algorithme, le chemin résultant est complètement faisable.

Pour le deuxième critère d'évaluation, cette approche à une performance similaire à celle de la première approche puisque le chemin résultant est lisse et localement le plus court.

Pour ce qui est du troisième critère, nous n'avons pas pu améliorer du tout le temps de calcul. Pour un chemin de plusieurs mètres, si un chemin libre d'obstacle est trouvé, le temps de calcul est inacceptable. Malgré plusieurs essais de simplification, nous n'avons pas réussi à rendre l'algorithme aussi rapide que souhaité.

Même si, nous avons réglé le problème de la fiabilité de notre algorithme, il reste quelques inconvénients qui n'ont pas été réglés, comme celui du choix de points de contrôle. Aussi, lorsque le chemin est plus long, la probabilité de ne pas converger vers une solution augmente.

3.1.3 Algorithme avec la programmation dynamique

Comme nous en avons déjà parlé à la section 2.2, cette approche est basée sur l'idée que le chemin initial a été créé en faisant une recherche, sur la carte du terrain, avec certains critères d'optimisation. Ainsi, l'unique problème est d'avoir modélisé le robot comme étant un point, ce qui engendre la possibilité de rencontrer des obstacles sur son chemin.

Donc, dans l'approche de planification locale avec l'aide d'un algorithme de programmation dynamique, nous partons de l'hypothèse que notre chemin initial est optimal en longueur. Ainsi, dans le cas où celui-ci soit complètement faisable en tenant compte de la géométrie du robot, il sera donc, le chemin final à suivre.

Avec cette dernière façon de voir le problème, l'idée de la validation du chemin, plutôt que la recherche du chemin, nous paraît simplifier la tâche de la planification. De cette manière, il est possible d'utiliser un pourcentage considérable ou la totalité du chemin initial dans la création du chemin final, tout en économisant les ressources du robot et du temps de calcul.

Maintenant en regardant la performance de cette approche, nous pouvons dire que, pour la plupart des cas, une solution est trouvée et le chemin final résultant est complètement libre d'obstacles. Nous sommes conservateurs lorsqu'on dit que pour la plupart des cas une solution est atteinte, et la raison est que lorsque nous faisons passer le chemin à travers d'un obstacle de dimension spectaculaire, l'algorithme atteint la grandeur maximale de la grille de recherche sans trouver une solution. Dans ce cas là, il faudrait donc prendre une décision, soit d'agrandir la grille afin de poursuivre la recherche, ou d'annuler la tâche et changer complètement la direction de recherche.

En ce qui concerne le deuxième critère, cet algorithme a une performance inférieure aux deux approches présentées précédemment. Les irrégularités du chemin de base ayant été éliminées lors de la définition du chemin initial. Cependant, de nouvelles irrégularités pourraient donc êtres introduites dues à une planification locale, puisque l'algorithme permet que le chemin comporte des virages jusqu'à 90 degrés afin d'éviter un obstacle. Ces irrégularités sont dues aussi au fait que le chemin local est défini par un ensemble de segments de droites. Pour minimiser l'impact de la planification locale, nous avons décidé d'effectuer une dernière interpolation, dans le but d'obtenir comme chemin final une fonction continue et lisse aussi représentée par une B-Spline.

Pour ce qui est du troisième critère d'évaluation « le temps de calcul », il est difficile de donner une estimation précise de cet algorithme. Le temps de calcul pour cette approche est de beaucoup inférieur à celui des deux algorithmes précédents, favorisant ainsi le choix de cet algorithme comme étant une solution intéressante afin de résoudre notre

problème. Mais ce temps de calcul n'est pas facilement prévisible puisqu'il dépend de la quantité et de la grandeur des obstacles. Plus nombreux sont les obstacles, plus grand sera le temps de calcul.

Le principal avantage de cette approche par rapport aux deux dernières est, qu'au moyen de la validation, il est possible de fournir au robot une partie du chemin à suivre. Durant ce temps, le robot peut également calculer, au moyen de la planification locale, les variations à effectuer au chemin afin d'éviter les obstacles. De plus, si le robot est dirigé vers un autre point d'intérêt, le calcul pourra être arrêté ou mis de côté, jusqu'à l'accomplissement de la nouvelle tâche.

Pour finir l'évaluation de la performance des solutions proposées, nous pouvons dire, que la troisième approche obtient la première place. Elle pourra donc être considérée comme la meilleure solution parmi d'autres afin de résoudre le problème défini au chapitre 1.

Dans la section suivante, nous présenterons une série d'idées employées afin de rendre les algorithmes plus rapides et plus performants.

3.1.4 Dans la recherche de la rapidité

Le premier élément à être pris en considération afin de réduire le temps de calcul est la discrétisation du chemin. Pour ce faire, nous devons établir un pas maximal pouvant varier entre le 50% et 80% de la longueur du robot. Ceci permet d'analyser la présence d'obstacles dans le chemin et avoir une valeur approximative de la fonction de coût $f(\lambda)$ sans avoir besoin d'évaluer chacun des points du chemin.

Pour déterminer la grandeur du pas, il est nécessaire d'abord de trouver la longueur totale du chemin. De cette façon, nous pouvons calculer facilement la valeur du pas exprimée en fonction de λ.

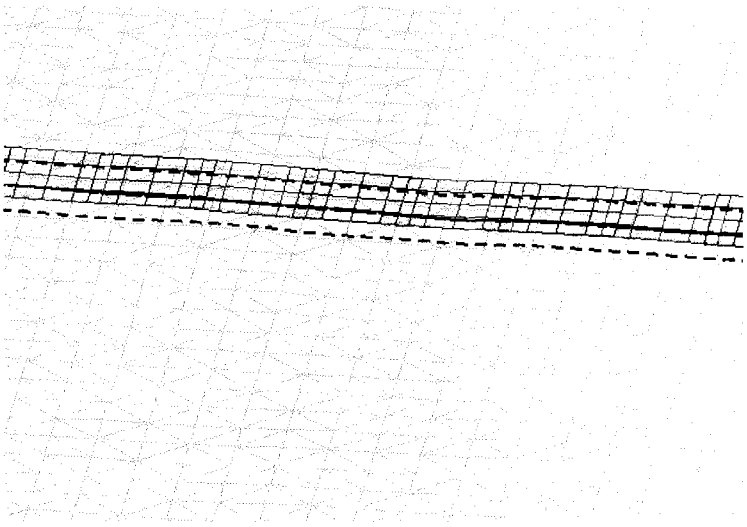


FIG. 3.1 Discrétisation du chemin

$$\delta\lambda = l \times p\%/Longueurduchemin \tag{3.1}$$

La figure 3.1 illustre une partie du chemin où nous avons placé le robot afin de démontrer l'aire occupée par ce dernier lors de chaque incrément de $\delta\lambda$.

Le principal problème relié à cette approximation, est que pour les deux premières méthodes, il faudra donc limiter l'aire de recherche afin de ne pas allonger trop le chemin et ainsi risquer de laisser des morceaux du chemin sans les analyser. L'autre option envisageable serait de redéfinir le pas lorsque la longueur du chemin aura atteint un seuil.

Pour la troisième approche, nous n'avons pas ce problème, puisque les chemins possibles entre les nœuds de la grille sont du même ordre de grandeur.

Devant effectuer de multiples recherches de données dans les immenses matrices Tr et D, la fonction de détection d'obstacles devient la principale cause du temps de calcul jugé trop long. Le deuxième pas dans la recherche de la rapidité attaque donc cette problématique. La solution a été de hiérarchiser les sous-fonctions qui forment la fonc-

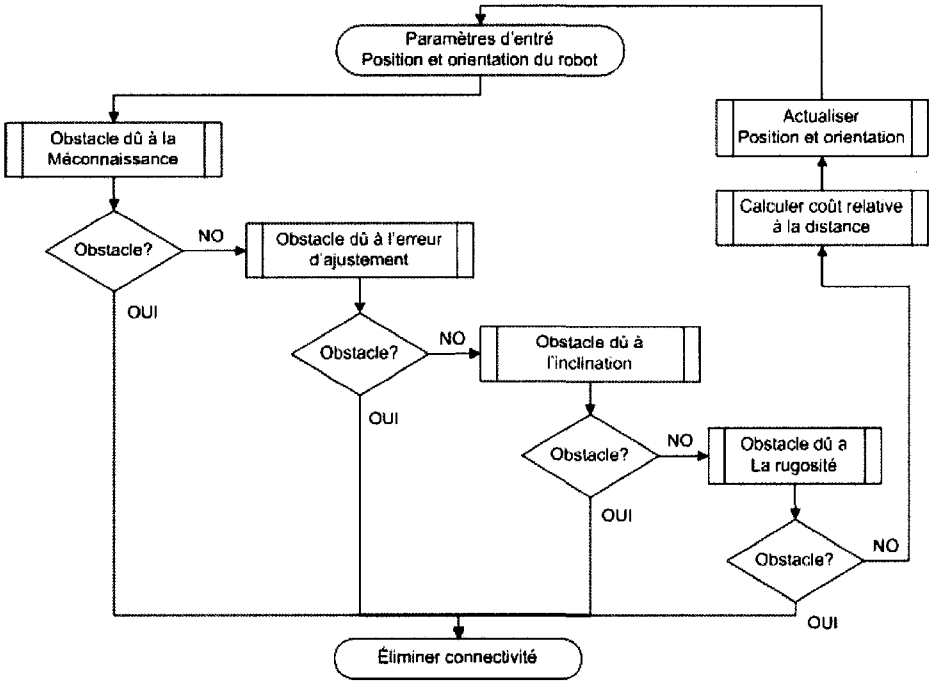


FIG. 3.2 procédure de hiérarchisation de la fonction de détection d'obstacle

tion de détection d'obstacles.

Donc, la fonction de détection d'obstacle présente une série d'étapes à franchir avant de considérer l'aire occupée par le robot comme libre d'obstacles. Chaque étape consiste à déterminer la présence ou non de chacun des différents types d'obstacles. L'ordre d'exécution de ces étapes sera assigné selon la quantité de temps requis pour effectuer l'identification. Cette procédure de hiérarchisation est illustrée à la figure 3.2.

Dans cet ordre d'idées, la première étape consiste à identifier si pour la configuration actuelle du robot, le terrain est connu. Sinon, cette aire est déclarée comme obstacle et nous continuons avec l'analyse d'une autre configuration. Si oui, nous poursuivons avec la deuxième étape.

La deuxième étape porte sur une analyse dans la détection d'obstacles jamais mentionnée. Elle est par conséquent l'analyse de l'erreur d'approximation de l'équation du

plan du robot, celle-ci est exprimée comme la distance entre le plan qui représente la base du robot et la position de chaque roue. Cette distance peut être considérée comme distance maximale permise pour la suspension du robot. Si cette distance est plus grande qu'un seuil, il s'agit donc d'un obstacle. Alors, une valeur de pénalité est calculée pour cette configuration. Sinon, cette configuration est libre d'obstacle selon la deuxième étape, donc une troisième étape sera exécutée. L'idée d'analyser l'erreur est simple et peut être interprété comme étant le robot dans une aire avec une rugosité élevée.

Dans la troisième étape, nous revenons à l'identification d'obstacle due à l'inclinaison du terrain, tel que présenté dans la section 2.2.2. Donc si la configuration du robot est libre d'obstacle, la quatrième et dernière étape est exécutée. Sinon, cette configuration est marquée comme étant un obstacle et une valeur de pénalité sera calculée.

La quatrième et la plus gourmande en temps et en ressources du système, elle est l'étape où nous identifions les obstacles dus à la rugosité du terrain. Cette étape devient la dernière inspection effectuée à l'aire occupée par le robot pour déterminer si elle est libre d'obstacles. Pourquoi est-elle aussi lente à exécuter ? Le principal facteur est la recherche de tous les points qui sont dessous et autour du robot dans les immenses matrices Tr et D, qui représentent le terrain. De plus, il y a toutes les opérations à effectuer avec ces données afin d'identifier la présence d'un obstacle. Si après avoir franchi cette étape la configuration actuelle du robot est libre d'obstacle, nous continuons avec la configuration suivante. Sinon, une valeur de pénalité sera calculée.

Afin de pouvoir utiliser cette procédure de hiérarchisation dans les deux premières approches, il sera nécessaire de remplacer $f_3(\lambda)$ et $f_4(\lambda)$ par une valeur fixe dans le cas où il y a présence d'obstacle, sans tenir compte du type d'obstacle qui est détecté.

Par contre, dans la troisième approche ; celle de la programmation dynamique, l'implémentation de ce pas représente une économie non négligeable en regard au temps de

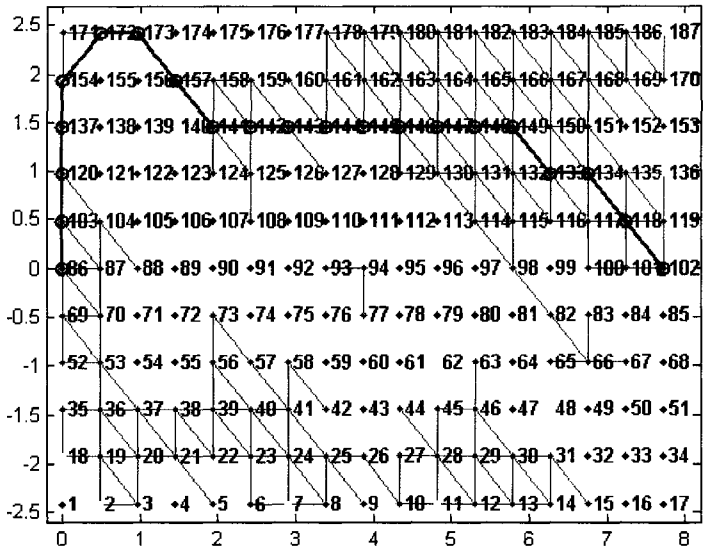


FIG. 3.3 Connectivité entre nœuds de la grille

calcul. Afin de profiter de cette économie, certaines modifications ont été réalisées. Par exemple, lorsqu'un obstacle est détecté, la connectivité entre les nœuds de la grille disparaît instantanément. En conséquence, la présence d'un obstacle dans un des segments de droite qui relie deux nœuds, fait que celui-ci soit éliminé et que le chemin local soit dirigé vers une autre direction. La figure 3.3 montre une grille de recherche générée par notre algorithme. Comme nous pouvons le constater, il y a un grand nombre de nœuds dont la connectivité avec son entourage est limitée ou inexistante.

Le troisième pas développe l'idée suivante. Si le problème est la taille des matrices Tr et D alors, il faut trouver le moyen de les rendre plus petites, afin d'accélérer la recherche des points d'intérêt. Un des moyens d'abaisser la taille de ces matrices, consiste à réduire la résolution de la carte, avec pour conséquence la perte d'information sur la topographie du terrain. Cette perte d'information sur le terrain peut occasionner une mauvaise détection d'obstacles et compromettre la sécurité du robot. Alors, une deuxième alternative s'offre à nous. Celle de laisser la résolution du terrain la plus haute possible et de subdiviser la carte en différents secteurs. Ceci réduira la taille des matrices Tr et D.

Encore une fois, lorsqu'une nouvelle approche est implémentée, il y a des points à sacrifier et des compromis à faire. Par exemple, si nous définissons une aire de recherche du chemin autour du chemin initial, nous réduisons le temps de calcul, mais nous limiterons la recherche dans uniquement dans la région choisie. Malgré la réduction de la taille des matrices, lorsqu'il s'agit de chemins définis sur de longues distances, les dimensions des matrices Tr et D restent encore énormes et le temps de calcul n'est que subtilement réduit. Donc, l'idée de diviser le terrain en secteur avec une aire proportionnelle à la dimension du robot semble une bonne idée. De cette façon, nous pourrons identifier la présence d'obstacles rapidement dans un secteur et migrer vers le secteur adjacent tout en suivant le chemin. Cette idée semble très attirante. Cependant, elle est seulement rentable si on possède un administrateur de cartes, lequel découpe et fournis les secteurs à être traités à notre algorithme de recherche. Le besoin de posséder un administrateur de cartes commence à être évident, lorsque nous sommes en train de perdre le temps épargné dans la détection d'obstacles, en découpant la carte.

Dans l'approche qui utilise la programmation dynamique comme outil pour la planification locale, nous avons décidé de couper le morceau de la carte équivalente à la dimension de la grille de recherche. Cette approche représente la meilleure performance en temps de calcul. L'unique problème se présente lorsque la grille de recherche commence à être très grande, alors le temps utilisé pour le découpage et l'identification d'obstacles commence à être aussi élevé.

3.2 Exemples

Présentons maintenant, un ensemble d'exemples représentatifs des trois algorithmes. La plupart seront dédiés à présenter les résultats de la troisième méthode, qui est de loin le meilleur choix entre toutes.

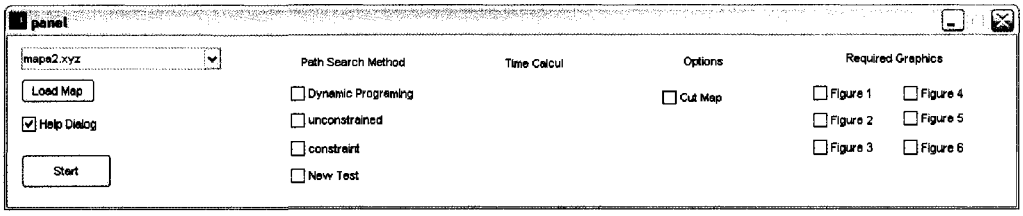


FIG. 3.4 Interface graphique

D'abord, nous allons présenter des exemples de chacune des solutions proposées. Par la suite, une comparaison entre chacune d'elles et finalement une série d'exemples de la solution qui utilise la programmation dynamique. L'idée principale sera de montrer comment l'algorithme est capable d'éviter les différents types d'obstacles présents sur la carte.

La machine sur laquelle les essais ont été faits est un ordinateur portable doté d'un processeur Intel Centrino Core Duo T2300 @ 1.660 Ghz, avec 1024 MB de mémoire RAM et Windows XP comme système d'exploitation.

Les différents algorithmes ont été codés et éprouvés sur Matlab R2006a dont une interface graphique permet le choix entre les différentes cartes, le choix de méthode, le choix entre la création ou la charge d'un chemin initial et aussi des options pour permettre ou non la visualisation de certains graphiques pouvant être générés. Un aperçu de cette interface graphique est illustré dans la figure 3.4.

Le modèle du terrain est une carte fournie par l'agence spatiale canadienne de sa « Mars Yard » (voir section 1.2.1). Nous allons travailler sur deux cartes. La première est un modèle aérien de la totalité du terrain où il est possible de visualiser les différents types de surfaces auxquelles le robot sera soumis. Cette carte utilise un maillage triangulaire régulier. La deuxième est un modèle d'une partie du terrain capté à partir du robot dont le maillage triangulaire est irrégulier. Les figures 3.5 et 3.6 présentent les deux cartes ainsi que les différents types de surfaces.

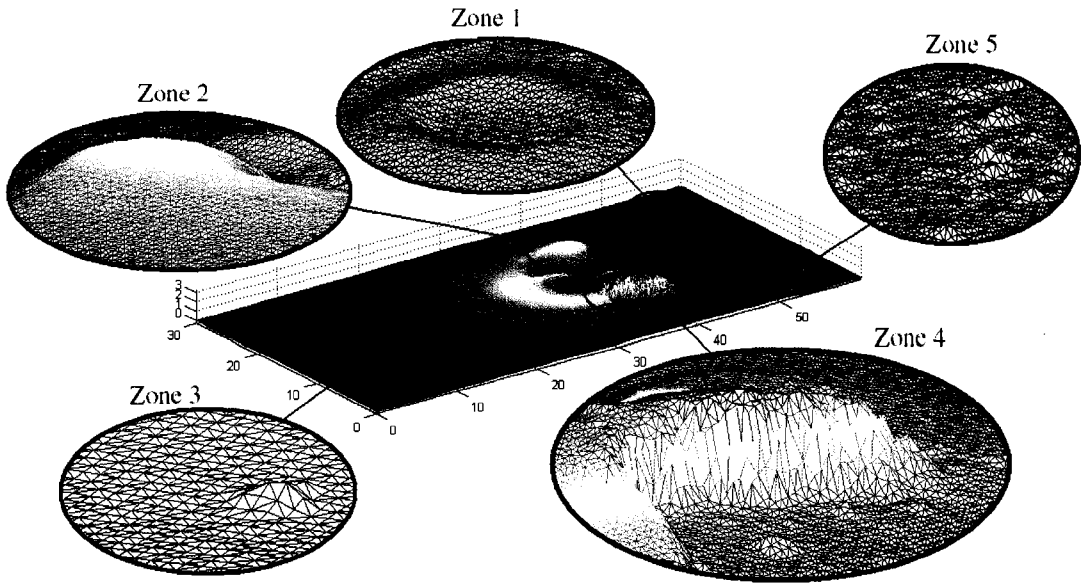


FIG. 3.5 Vue aérienne du terrain de preuve

La figure 3.5 illustre la totalité du terrain utilisé pour les simulations où nous avons extrait cinq différentes zones. La zone 1 nous montre la présence des cratères dans le terrain. La zone 2 montre une partie d'une colline d'élévation modérée et des versants en pente douce de différente inclinaison. La zone 3 a été sélectionnée pour illustrer une grande région plate avec la présence de quelques roches de différente grandeur. La zone 3 est la topographie la plus simple où le robot rencontre le moins de problèmes pour ses déplacements, ce qui n'est pas le cas pour la zone 4 où nous trouvons une région abrupte, rocheuse et escarpée avec des falaises. Cette zone peut être une des plus dangereuses pour le robot, il doit l'éviter en tout temps. Finalement, nous avons la zone 5 qui est aussi une région plate, mais à la différence de la zone 3, il y a un grand nombre d'obstacles présents, ce qui fait de cette région un cauchemar pour la planification de chemins.

Dans la figure 3.6 nous avons sélectionné 3 zones représentatives des différentes surfaces présentes sur le terrain. La zone 1 montre une partie d'une surface hautement rugueuse qui appartient à la base d'une région abrupte. La zone 2 montre un cratère et la zone 3

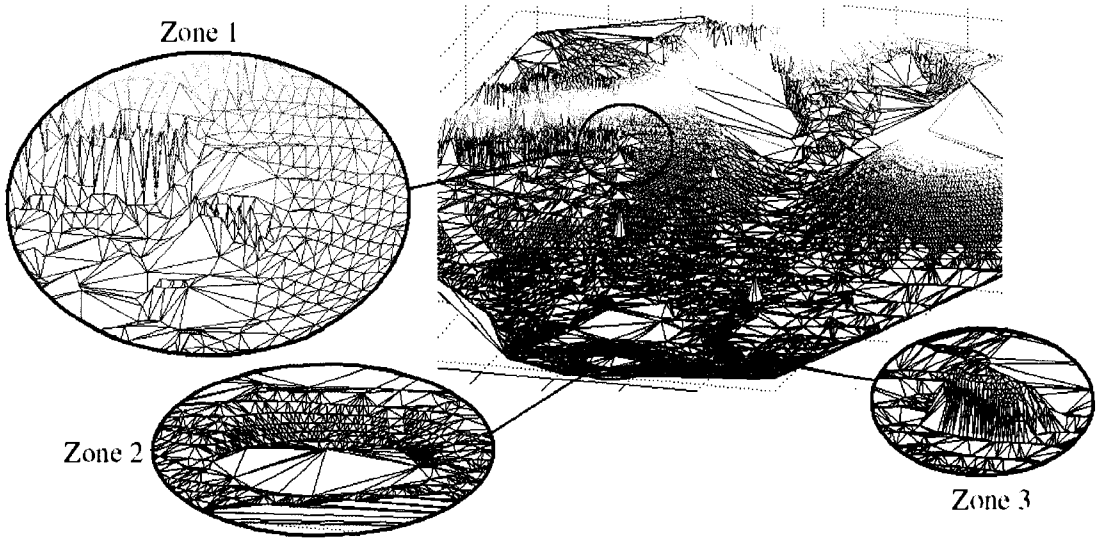


FIG. 3.6 Terrain de preuve vue à partir du robot

montre une grosse roche au milieu d'une région plate.

Pour l'ensemble des exemples, le chemin initial sera défini par la courbe continue de couleur noire. Lorsque le chemin final sera identifié par la courbe continue de couleur rouge. Les courbes parallèles et pointillées représenteront la trajectoire des roues du robot tant à gauche qu'à droite.

3.2.1 Approche 1. Avec l'optimisation sans contraintes.

À la figure 3.7 nous présentons un exemple pour notre première approche. Pour cet exemple, un chemin initial est tracé sur une zone peu rugueuse. Pas très loin du point de départ, nous voyons que le chemin traverse un obstacle. De plus, nous remarquons que le chemin initial présent deux bosses qui pourront être éliminées.

Après avoir exécuté notre algorithme, nous obtenons comme réponse un chemin qui

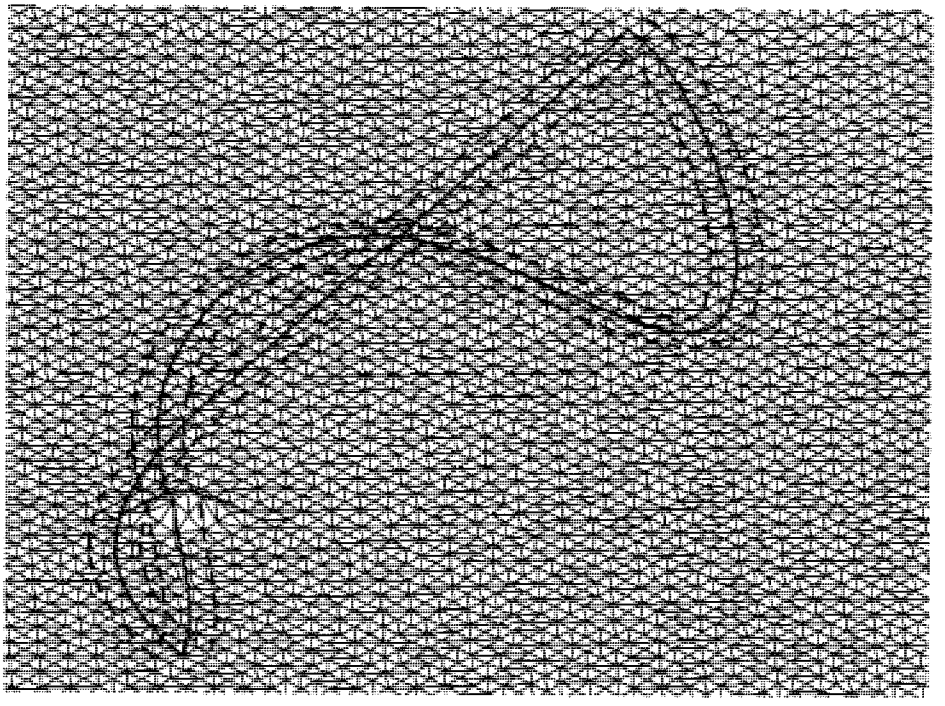
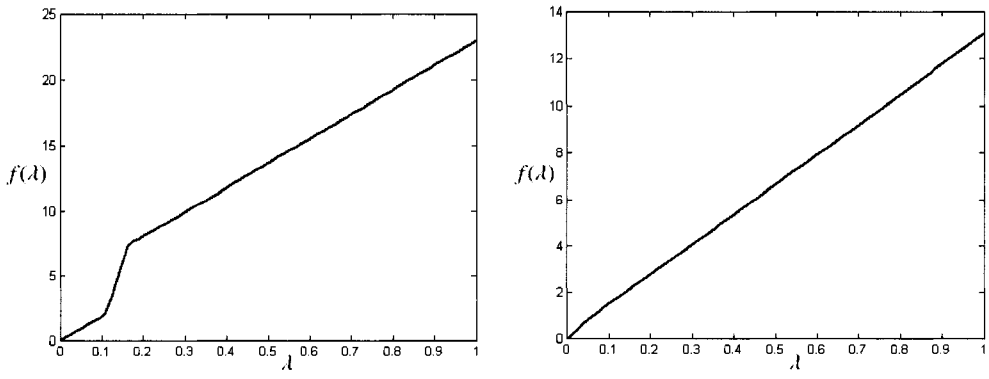


FIG. 3.7 Planification du chemin avec l'optimisation sans contraintes

contourne l'obstacle en éliminant les deux bosses. Avec cet exemple nous pouvons mettre en évidence l'action des différentes composantes de la fonction de coût choisie. Tout d'abord l'algorithme trouve dans une direction de recherche, un chemin qui ne présente pas d'obstacles ce qui diminue notablement la valeur de $f(\lambda)$. Par la suite il continue à minimiser cette fonction dans cette direction, en réduisant la longueur et la courbure du chemin, ce qui fait que l'on ait à la fin un chemin droit tout de suite après avoir contourné l'obstacle.

L'exemple illustre aussi la présence d'un minimum local de la fonction de coût. Lorsque l'algorithme, dès les premières itérations, rencontre un chemin dont la fonction est beaucoup plus faible, il dirige donc la recherche du chemin à la gauche de l'obstacle. Il est évident que s'il essaie de retourner vers la droite il rencontrera encore une fois l'obstacle, ce qui aura pour conséquence que notre algorithme continuera à chercher une solution à la gauche de l'obstacle et qu'il n'arrivera pas à obtenir un chemin complètement droit,



(a) (b)

FIG. 3.8 Fonction de coût : (a) chemin initial ; et, (b) chemin final.

même si ce dernier est possible dans ce cas en particulier.

Les figures 3.8(a) y 3.8(b) présentent la valeur de la fonction de coût pour le chemin initial et le chemin final respectivement. En faisant une comparaison entre les deux graphiques, nous constatons que pour le chemin final nous avons la fonction de coût atteint une valeur inférieure à celle du chemin initial et qu'elle croît d'une façon plus uniforme au long du chemin.

Nous pouvons noter aussi à la figure 3.8(a) un fort changement de la pente entre les valeurs 0.108 et 0.162 de λ, ce qui met en évidence la présence d'un obstacle sur cette portion du chemin. Pour ce qui est de la fonction de coût pour le chemin final, nous pouvons constater dans la figure 3.8(b) que la fonction de coût croît plus rapidement pour λ entre 0 et .096 avec une pente égale à 15.05 lorsque pour λ entre 0.096 et 1 la pente est de 12.86. Ce changement est attribué à la présence de la courbe au début du chemin final.

Le temps de calcul requis pour trouver cette solution a été de 1335.2 sec. Avec un chemin initial de 18.26 mètres.

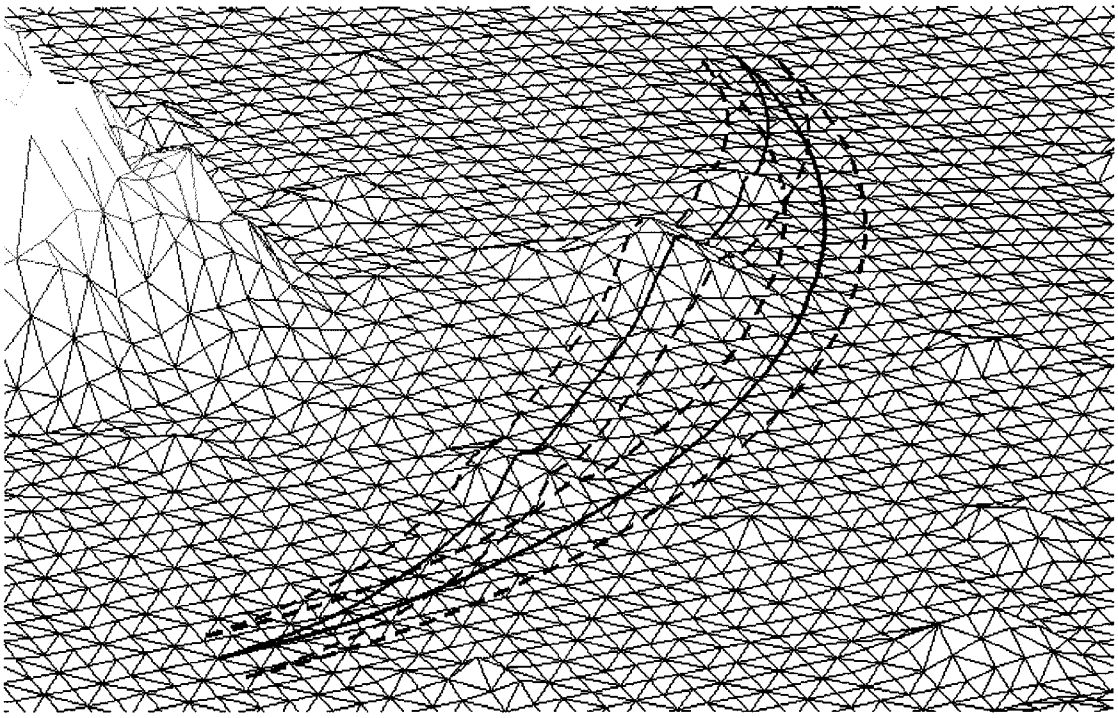


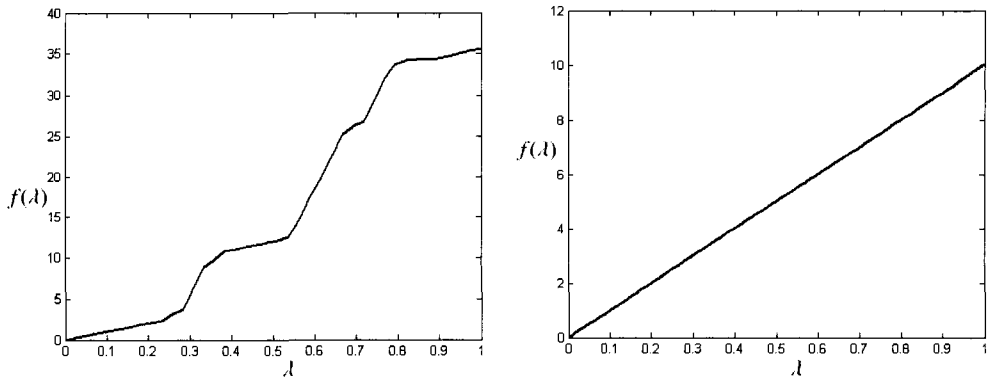
FIG. 3.9 Planification du chemin avec l'optimisation avec contraintes

3.2.2 Approche 2. Avec l'optimisation avec contraintes.

Pour notre deuxième approche, nous avons obtenu une solution semblable à la première. Nous avons choisi un chemin initial dans une zone peu rugueuse où nous pouvons apprécier la présence de trois obstacles tout au long du chemin. Après avoir exécuté notre deuxième algorithme, nous constatons à la figure 3.9 que le chemin final fourni par l'algorithme est libre d'obstacle et faisable par le robot.

D'une façon similaire à l'exemple fourni à la première approche, les figures 3.10(a) et 3.10(b) illustrent les fonctions de coût respectives pour le chemin initial et final. Nous constatons aussi l'influence des obstacles dans la valeur de la fonction de coût pour les intervalles de λ $[0.28, 0.33]$, $[0.54, 0.67]$, $[0.72, 0.79]$.

Dans le cas de réussite, les deux premières approches peuvent donner comme résultat



(a) (b)

FIG. 3.10 Fonction de coût : (a) chemin initial ; et, (b) chemin final.

la même réponse et un temps de calcul avec le même ordre de grandeur. Par contre, au niveau de certitude, nous pouvons être certains que les chemins finaux fournis pour la deuxième approche, seront toujours libres d'obstacles. La grande différence entre les deux approches se retrouve dans le cas d'échec. Pour la première approche nous aurons toujours un chemin final pour lequel la fonction de coût est au minimum dans le voisinage, mais nous ne savons pas si ce chemin est faisables ou pas. Par contre, avec la deuxième approche nous savons que dans le cas d'échec, nous n'aurons pas de chemin final. Il faudrait donc chercher une autre solution.

3.2.3 Approche 3. Avec la programmation dynamique.

Nous avons dit tout au long de ce rapport, que notre troisième approche est celle avec la meilleure performance. Avec l'exemple illustré à la figure 3.11, nous voulons mettre en évidence la bonne performance de notre algorithme. L'objectif avec cet exemple est de mettre au défi l'algorithme. Alors, nous avons donc choisi un chemin initial, lequel passe par plusieurs zones interdites. Dans un premier temps, le chemin emprunte une surface plate et libre d'obstacle, par la suite, le chemin traverse la colline ou nous rencontrons la

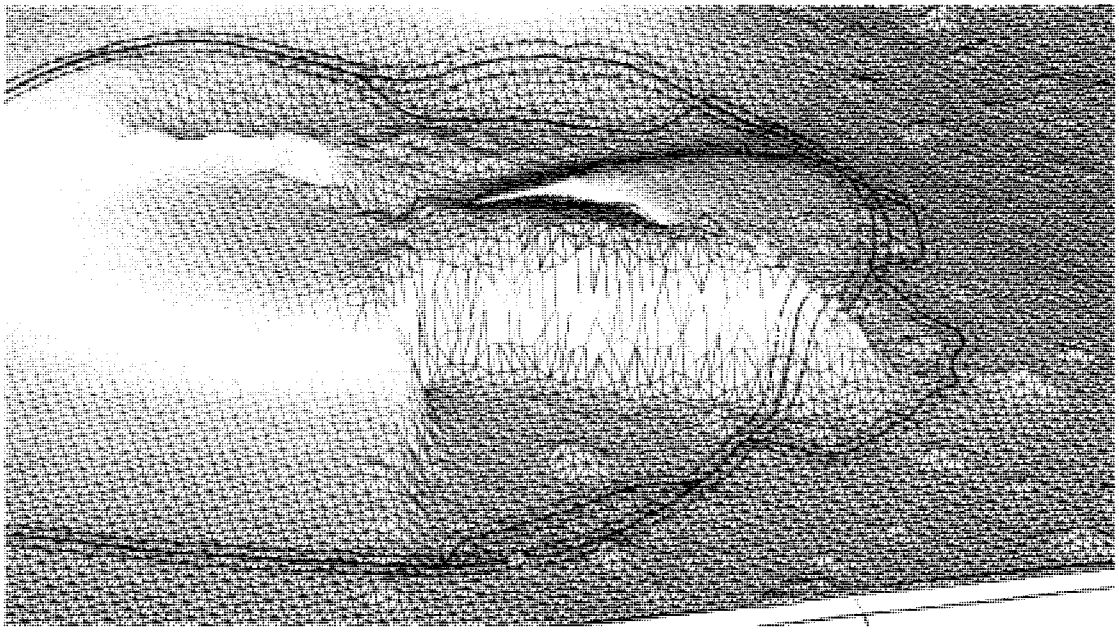


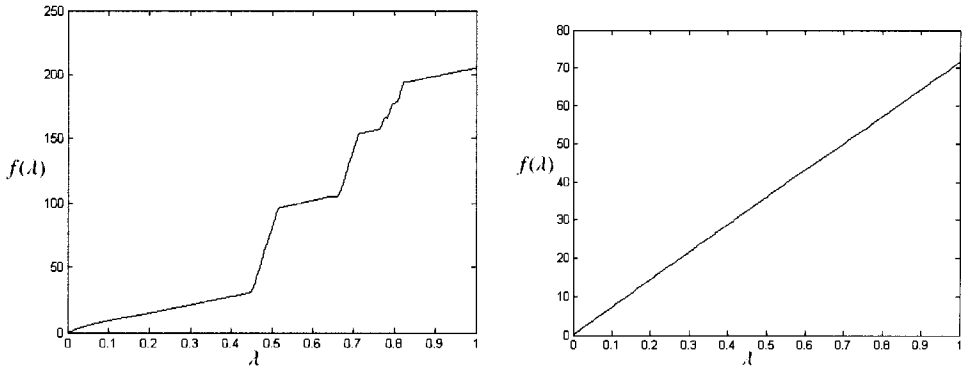
FIG. 3.11 Planification du chemin avec la programmation dynamique. Exemple 1

première zone interdite dû à l'inclinaison. Vient ensuite un ensemble de zones interdites de moyenne et haute rugosité, en plus d'une falaise prononcée. Finalement, le chemin prend fin dans une zone plate.

Comme nous pouvons le constater à la figure 3.11, le chemin final résultant après l'exécution de notre algorithme a changé radicalement dans les secteurs ne pouvant être traversés par le robot. Par contre, il n'y a aucune modification dans les trajets du chemin qui ne présentaient pas de danger pour le robot.

La figure 3.12(a) représente la valeur de la fonction de coût pour notre chemin initial. En regardant ce graphique, il est possible de distinguer les différentes zones franchies par le chemin initial dont nous avons parlé antérieurement. Les secteurs interdits sont localisés où les forts changements de la pente de la fonction de coût ont lieu. Pour le reste du chemin, la croissance de celle-là est uniforme.

La figure 3.12(b), représente quant à elle, le graphique associé à la fonction de coût le



(a) (b)

FIG. 3.12 Fonction de coût : (a) chemin initial ; et, (b) chemin final.

long du chemin final. La croissance uniforme de la valeur de la fonction de coût nous indique la non-présence d'obstacles dans ce chemin. Dû à la longueur du chemin, il est difficile d'apprécier l'influence de la courbure.

Nous aussi avons testé l'algorithme avec la représentation du terrain par un maillage triangulaire non uniforme pris à partir du robot. Comme dans les exemples précédents, la figure 3.13 illustre le chemin initial et le chemin final. Les figures 3.14(a) et 3.14(b) montrent quant à elles, respectivement la valeur de $f(\lambda)$ pour ces chemins. Le comportement l'algorithme est similaire au cas avec maillage régulier.

3.2.4 Comparaisons

Dans cette section nous allons présenter des exemples illustrant plusieurs aspects dont nous avons discutés dans la section 3.1. Nous allons montrer l'influence de la taille de la carte et de sa résolution sur le temps requis pour trouver le chemin désiré. Aussi, nous ferons une comparaison entre les méthodes proposées.

Pour le premier exemple, nous prenons comme chemin initial celui présenté antérieurement

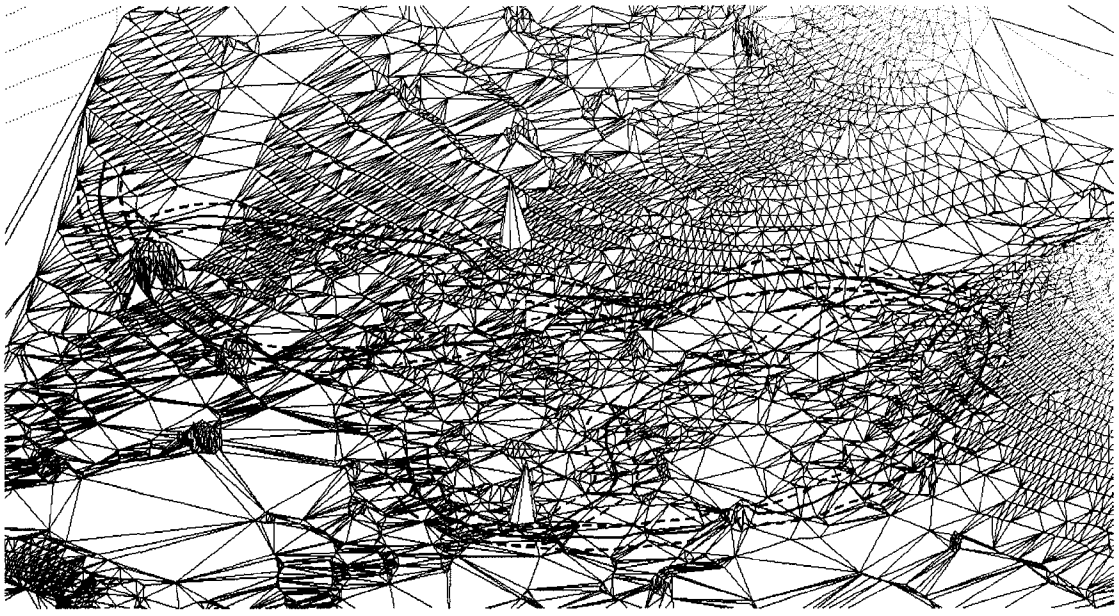
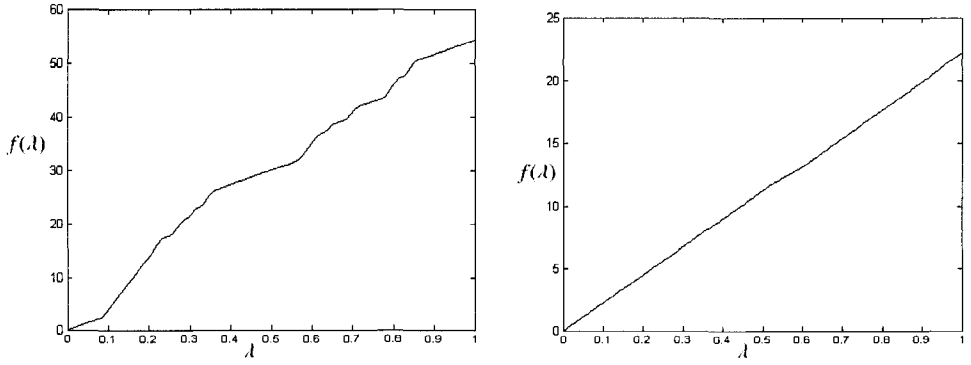


FIG. 3.13 Planification du chemin avec la programmation dynamique. Exemple 2



(a) (b)

FIG. 3.14 Fonction de coût : (a) chemin initial ; et, (b) chemin final.

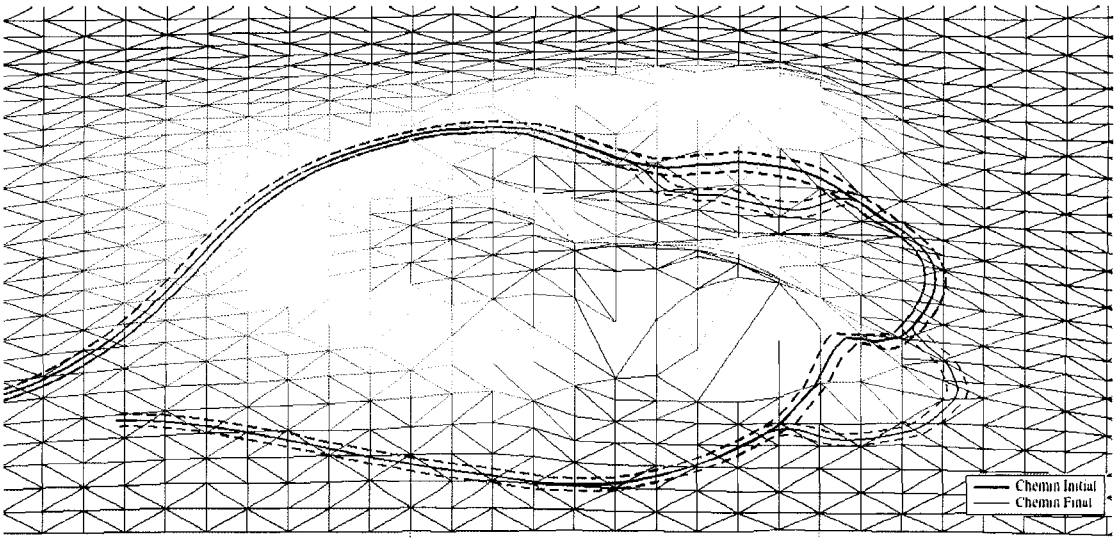


FIG. 3.15 Influence de la résolution de la carte

et illustré à la figure 3.11. La différence est que nous avons utilisé la même carte avec une résolution plus basse afin de mettre en évidence l'influence de la résolution sur le temps de calcul. Ainsi, nous pourrons comparer l'influence de la perte d'information de la topographie du terrain sur le chemin final. Pour l'exemple de la figure 3.11 nous avons utilisé une carte avec résolution de 0,2 mètre, dont les dimensions des matrices D et Tr sont $[45300 \times 3]$ et $[89700 \times 3]$ respectivement. Pour l'exemple de la figure 3.15 nous avons utilisé une carte avec une résolution de 1 mètre, dont les dimensions des matrices D et Tr sont $[1891 \times 3]$ et $[3600 \times 3]$ respectivement.

D'après la table de temps, nous observons qu'il y a une économie de temps non négligeable. Il est aussi évident que l'information relative aux obstacles de moyenne et de petite taille a été complètement perdue. Notre algorithme n'est donc plus capable de les détecter.

Résolution	Temps de validation (sec.)	Temps de calcul (sec.)
Haute (0.2m)	92.31	451.78
Basse (1m)	3.73	48.01

Notre deuxième exemple cherche à démontrer l'influence de la grandeur de la carte utilisée. Nous avons repris l'exemple présenté dans la section 3.2.1 figure 3.7, où nous avons extrait une partie de la carte pour faire la recherche du chemin. Ensuite, nous avons effectué la même procédure, mais en utilisant la carte complète du terrain. Évidemment, le chemin final sera le même, mais le temps alloué pour la recherche est beaucoup plus important. Dans la table de temps, nous pouvons constater l'énorme différence de temps requis pour fournir la même réponse. Donc, dans l'absence d'un administrateur de cartes, il est beaucoup plus rentable de faire le découpage de celle-ci avant de commencer la recherche.

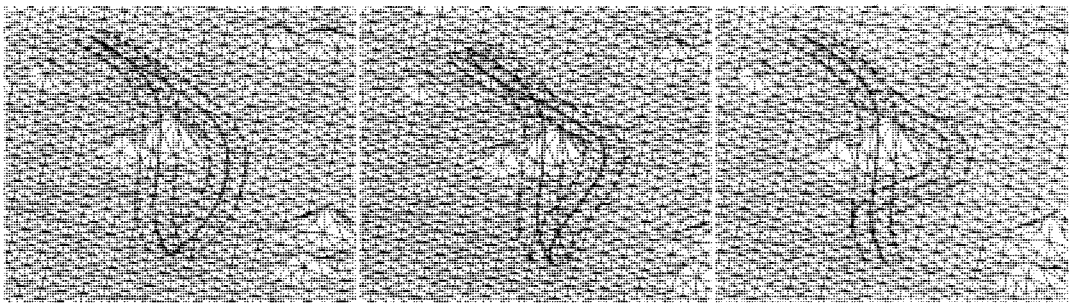
	Temps de calcul (sec.)
Avec découpage	1335.2
Sans découpage	3068.1

L'algorithme qui utilise la programmation dynamique est muni d'une fonction qui fait le découpage de l'aire occupée par la grille avant de faire les calculs nécessaires pour effectuer la recherche. Nous avons donc, une économie de temps de calcul très appréciable. Par exemple, si nous enlevons cette fonction de l'algorithme, le temps de calcul nécessaire pour trouver le chemin correspondant à l'exemple illustré avec la figure 3.11 à passé de 451.78 secondes à 1039.3 secondes.

	Temps de calcul (sec.)
Avec découpage	451.78
Sans découpage	1039.3

3.2.4.1 Comparaisons des méthodes

Afin de pouvoir faire une comparaison entre les méthodes, nous avons soumis le même chemin initial aux trois algorithmes correspondants. Pour l'analyse quantitative, nous



(a) (b) (c)

FIG. 3.16 Comparaison entre les méthodes : (a) Avec l'optimisation sans contraintes ; (b) Avec l'optimisation avec contraintes ; et, (c) Avec la programmation dynamique.

utiliserons le temps de calcul et la valeur de la fonction de coût du chemin final comme mesure de performance.

Pour notre premier exemple de comparaison, nous avons pris un chemin initial assez simple pour garantir la réussite des trois méthodes, il s'agit d'un chemin d'une dizaine de mètres qui traverse un obstacle. Les résultats sont représentés dans la figure 3.16.

	Méthodes de recherche		
Temps (sec.)	Sans contraintes	Avec contraintes	Prog. Dynamique
Coupure	44	44.8	NA
Validation	NA	NA	9.3
Recherche	282.4	328.3	66.4
Temps total (sec.)	326.4	373.1	75.7

Notre deuxième exemple est beaucoup plus simple, il s'agit d'un chemin initial libre d'obstacle que nous faisons passer par les trois algorithmes. Cet exemple met en évidence l'avantage de la validation du chemin et l'influence des sous-fonctions de la fonction de coût. La figure 3.18 montre les résultats de ce deuxième exemple.

D'après les résultats, nous constatons que pour les deux premières approches, les che-

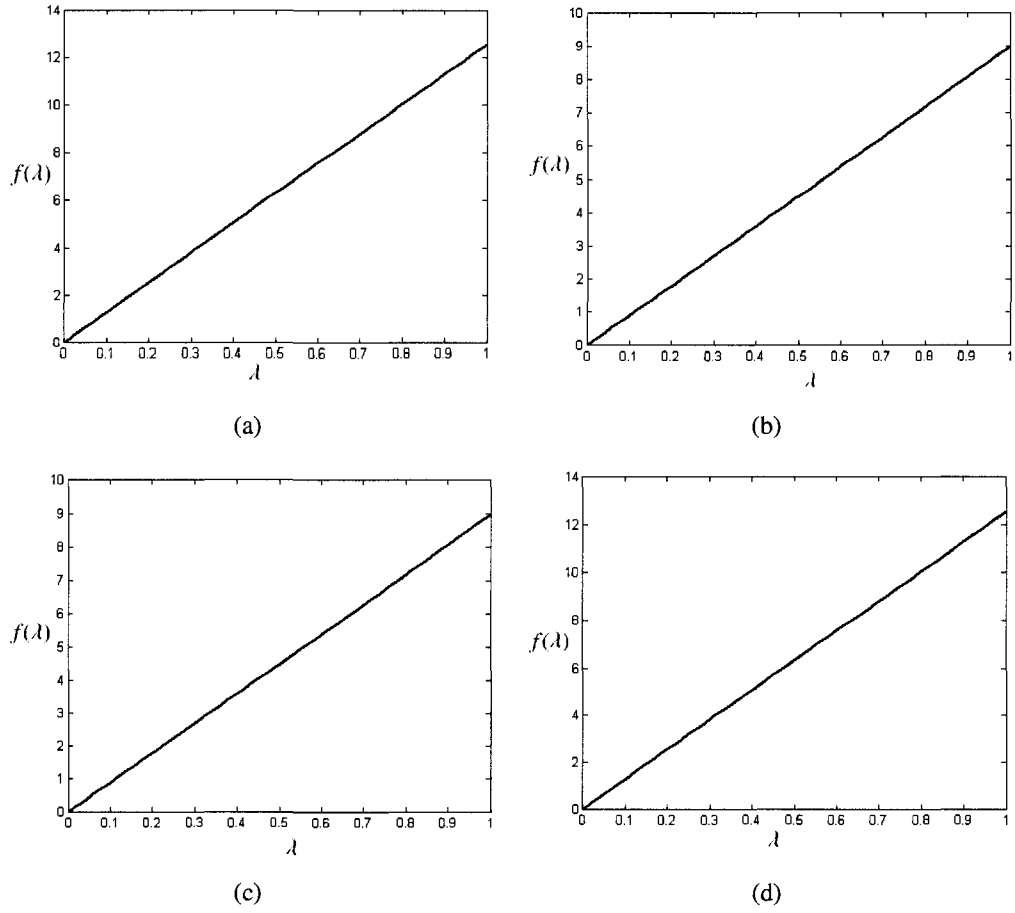


FIG. 3.17 Fonction de coût associée aux méthodes : (a) du Chemin initial ; (b) du Chemin final approche 1 ; (c) du Chemin final approche 2 ; et, (d) du Chemin final approche 3.

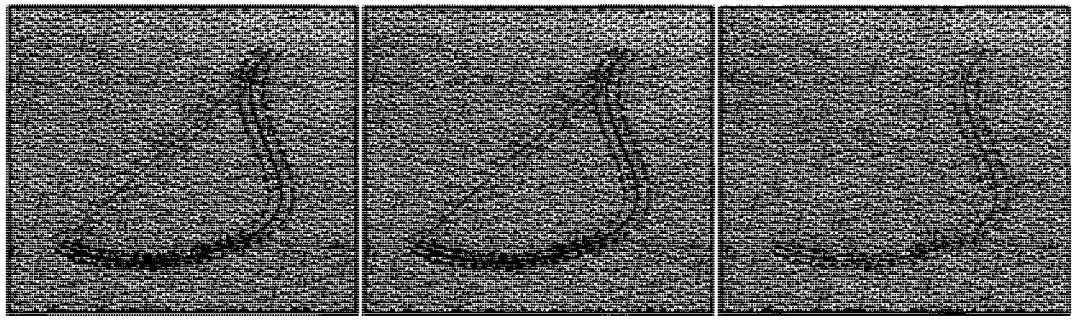


FIG. 3.18 Comparaison entre les méthodes : (a) Avec l'optimisation sans contraintes ; (b) Avec l'optimisation avec contraintes ; et, (c) Avec la programmation dynamique.

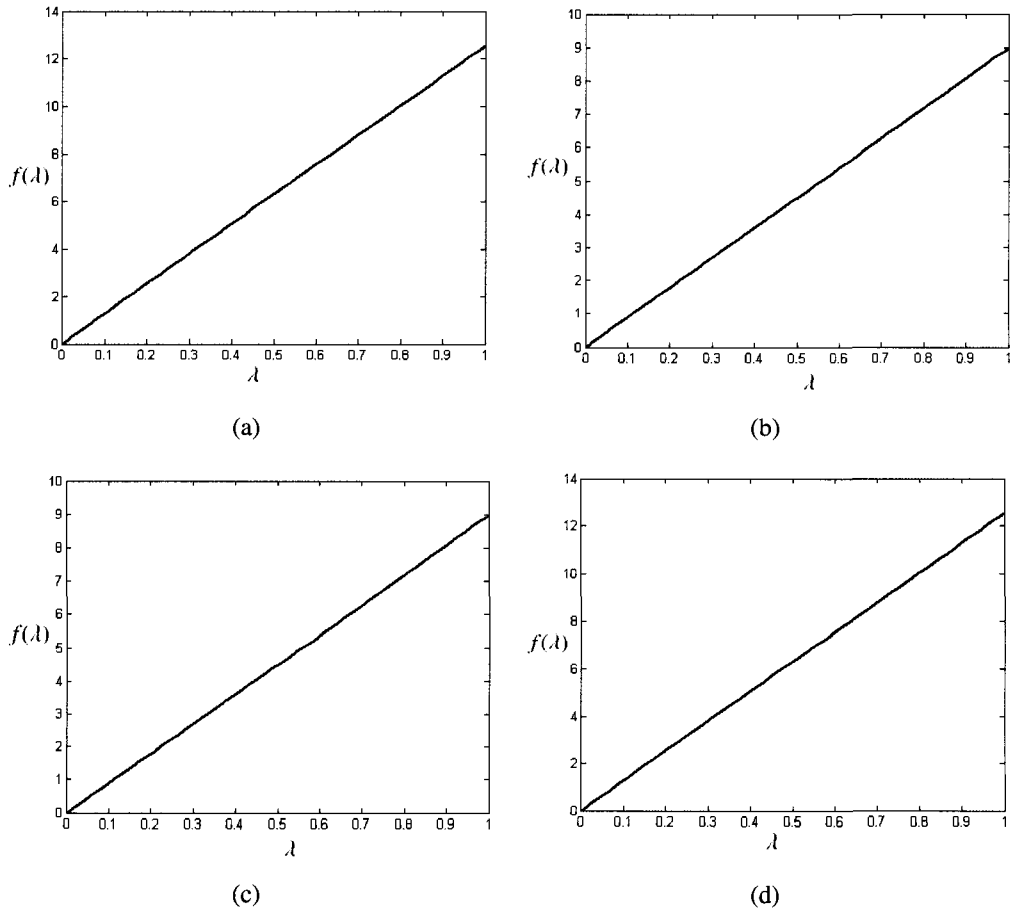


FIG. 3.19 Fonction de coût associée aux méthodes : (a) du Chemin initial ; (b) du Chemin final approche 1 ; (c) du Chemin final approche 2 ; et, (d) du Chemin final approche 3.

mins résultants sont identiques et ils sont devenus des lignes droites, ce qui est prévu, puisque le chemin le plus court et le moins courbe, entre deux points dans l'espace libre, est une ligne droite. Par contre, avec notre troisième approche nous obtenons comme réponse le même chemin initial, ceci est aussi attendu, car l'algorithme a validé le chemin et il n'a pas trouvé d'obstacles. Comme nous l'avons déjà dit, le chemin de base est généré en utilisant l'algorithme de recherche du chemin le plus court de Dijkstra et malgré le fait qu'il pourrait avoir quelques irrégularités, il sera proche d'une ligne droite après l'interpolation. Donc, la validation du chemin devient encore une fois un choix intéressant.

Maintenant, si nous regardons la table de temps, nous avons obtenu que pour la troisième approche, l'algorithme n'a utilisé que 18.9 secondes pour faire la validation et pour nous offrir comme résultat que le chemin initial est complètement faisable par le robot. Comparativement aux autres deux algorithmes qui ont pris 687.5 et 640.8 secondes respectivement pour nous fournir aussi un chemin faisable par le robot.

	Méthodes de recherche		
Temps (sec.)	Sans contraintes	Avec contraintes	Prog. Dynamique
Coupure	69.6	71.1	NA
Validation	NA	NA	18.9
Recherche	687.5	640.8	0
Temps total (sec.)	757.1	711.9	18.9

3.3 Études futures

À partir de cet travail et en utilisant les mêmes idées afin de fournir une solution viable pour ce type de mission, on pourrait continuer à approfondir la matière. Même si nous sommes satisfaits du travail réalisé jusqu'à maintenant, il faut reconnaître qu'il y a en-

core plusieurs aspects que nous pourrions améliorer. Dans cette section, nous allons d'abord énumérer une série d'idées à implanter afin de rendre l'algorithme de planification locale avec la programmation dynamique beaucoup plus attrayant afin de lui permettre d'être sélectionné comme planificateur dans une mission d'exploration planétaire. Par la suite, nous proposerons deux cas d'étude qui pourraient être complémentaires à ce travail de recherche.

Dans le but d'améliorer la performance et la fiabilité de notre algorithme de planification locale, les idées suivantes devraient être considérés.

- L'optimisation de l'algorithme et son implantation sur une plateforme différente de Matlab. Nous avons utilisé Matlab pour valider notre modèle dû à la grande quantité d'outils mathématiques, à la rapidité de l'implémentation et aux interfaces graphiques disponibles. Malgré ses vertus, l'utilisation de Matlab pourra être remplacée par des langages de programmation de plus bas niveau avec un code adapté à nos besoins. De plus, le code des algorithmes présentés comme solution au problème, sont sous-optimaux et l'utilisation de l'interface graphique demande des ressources supplémentaires.

- L'implémentation d'un administrateur de cartes serait un atout. De cette façon l'algorithme de planification sera dédié à la recherche du chemin et à l'identification d'obstacles sans avoir perdu du temps avec le découpage de cartes et la définition d'aires de recherche. L'administrateur de cartes devra être en mesure de fournir dans un délai établi, la section de la carte du terrain sur laquelle l'algorithme de planification local sera exécuté. Dans la section 3.2, vous avez pu apprécier l'économie de temps de calcul lors de l'utilisation des cartes réduites et le temps requis pour les extraire.

- L'implémentation d'un algorithme d'analyse et de décision qui permet de changer les objectifs dans le cadre de la planification locale. C'est à dire, l'algorithme de planifi-

cation locale a, par défaut, des valeurs et des paramètres de recherche, par exemple ; la grandeur maximale de la grille, la distance pour déterminer PI et PF, la distance entre les nœuds de la grille, etc..

L'objectif de cet algorithme de décision est de donner une certaine intelligence à notre algorithme de planification. Par exemple, il pourrait savoir : quand les points PI et PF sont très proches de l'obstacle ; quand le robot est en face d'un obstacle de forme concave ; quand il faut agrandir ou diminuer la grille et dans quels secteurs et quand arrêter la recherche, etc..

Pour illustrer quelques fonctions que l'algorithme de décision pourrait posséder, nous allons présenter des graphiques des différentes grilles générées par l'algorithme. Ces derniers nous aideront à comprendre et à définir les fonctions que nous avons énoncées précédemment.

- Le premier cas est le plus simple, il s'agit de savoir si les points choisis comme point de départ PI ou point d'arrivée PF sont très proches de l'obstacle. Dans cette situation, les nœuds de la grille qui représentent à PI et PF n'auront aucune connectivité avec les nœuds voisins. Si cela se produit, il faudra placer donc ceux-ci à une distance plus grande que prévu. Comme nous le disions précédemment, ce cas est le plus simple, puisqu'il s'agit de vérifier la connectivité d'un seul nœud et il pourra être identifié dès les premières itérations. La figure 3.20 illustre cette situation.

- Le deuxième cas pourra être vu comme la généralisation du cas précédent. Il aura pour fonction de détecter si le robot est au milieu d'un obstacle de forme concave. La raison pour laquelle nous disons que ce cas est une généralisation du premier cas, est que pour reconnaître cette sorte de situation il faudra analyser le voisi-

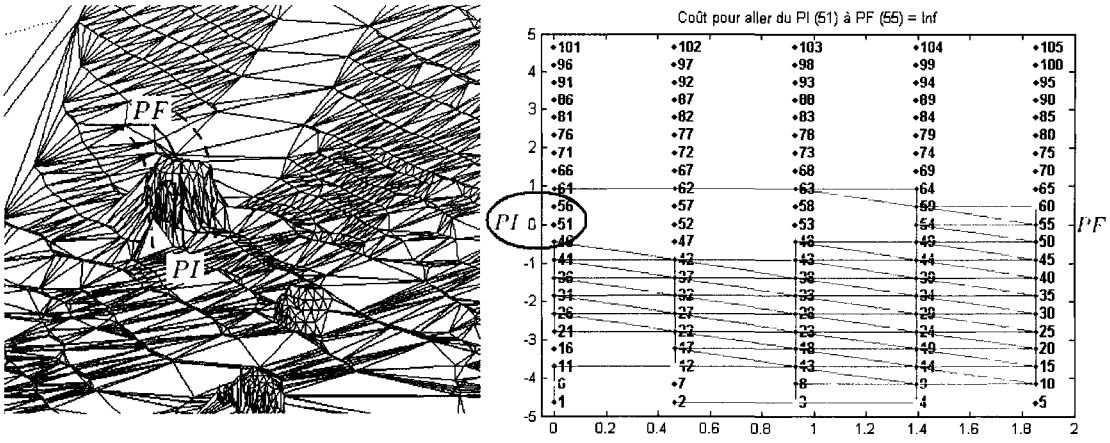


FIG. 3.20 Point initial PI très proche

nage du PI et détecter quand ce voisinage reste complètement sans possibilité de connexion avec son entourage. Lorsqu'on rencontre cette situation, la solution est plus complexe. Il faudrait recourir aux théories de résolution du problème de cycle limite (Xu et al., 1998) où l'implantation d'un mur virtuel pourrait délimiter l'aire occupée par cet obstacle (Ordónez, 2006).

À la figure 3.21 nous pouvons apprécier le phénomène décrit ci-dessus. La partie supérieure de la figure 3.21 montre les points PI et PF, lesquels nous voulons relier pour définir un chemin. également nous pouvons apprécier la forme bien définie de l'obstacle entre les deux points. Si nous effectuons une recherche telle que décrite dans la section 2.2, nous obtenons une grille de recherche telle qu'illustrée dans la partie inférieure de la figure 3.21, où nous avons encerclé le voisinage du point PI pour souligner le fait que le point PI reste entouré par l'obstacle et la possibilité d'une connexion entre PI et PF est nulle. On a agrandi la grille de recherche, cependant une solution ne sera atteinte puisque le voisinage de PI est complètement isolé.

– Le troisième cas est relié à la capacité de savoir, en analysant la grille, quand il

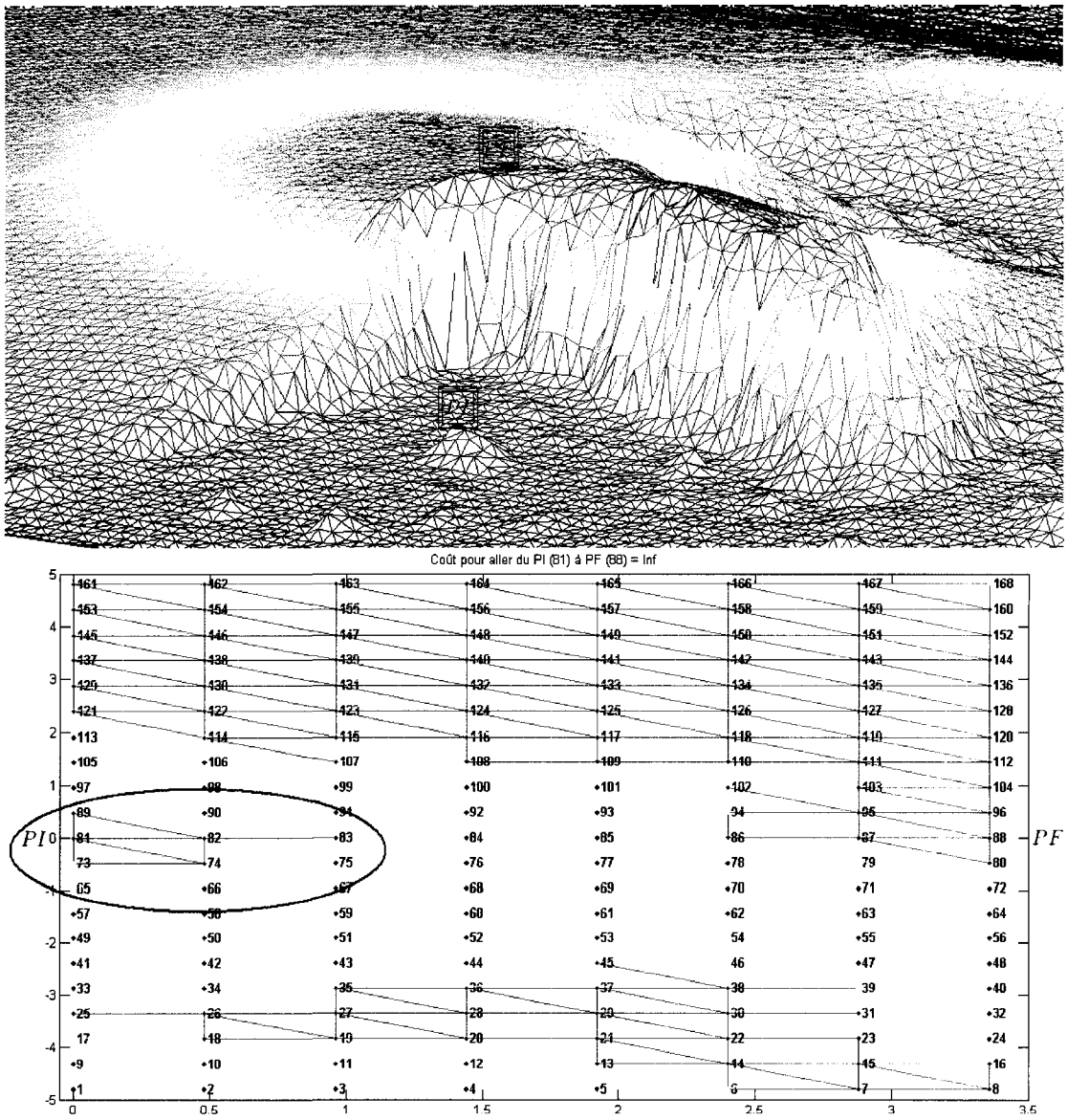


FIG. 3.21 Robot en face d'un obstacle en forme concave

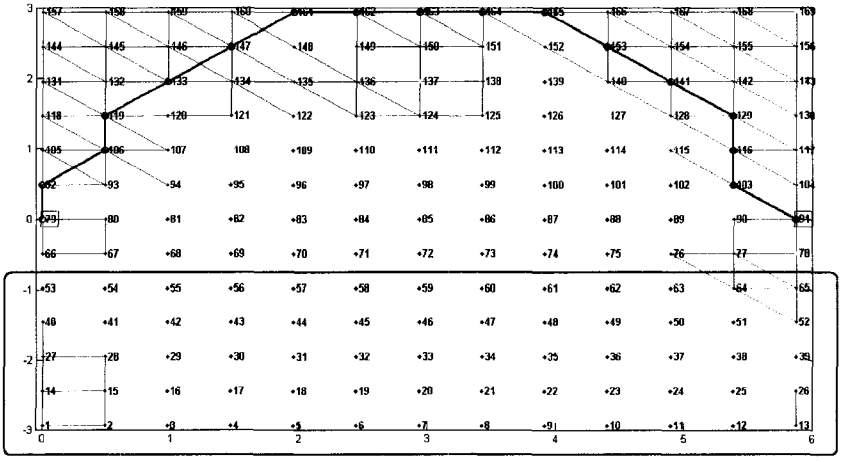


FIG. 3.22 Priorisassions des aires de recherche

faut arrêter d'agrandir la grille de recherche dans une direction car la probabilité de trouver un chemin, qui lie les points PI et PF, est minimal. Il pourrait aussi avoir la capacité de prioriser la recherche dans une aire spécifique où la probabilité de trouver un chemin rapidement est plus grande.

Dans la figure 3.22 nous avons une grille de recherche qui a permis de trouver le chemin entre les points PI et PF afin de contourner l'obstacle, mais lorsque la grille de recherche croit d'une façon symétrique, nous constatons avoir perdu beaucoup de temps en cherchant un chemin dans la zone sélectionnée. Il est évident que dans cette direction, la recherche ne portera pas fruit car dans cette zone, la connectivité entre les nœuds, est presque inexistante.

Maintenant, nous proposons deux cas d'étude. Nous comptons utiliser que la fonction de détection d'obstacles sur deux algorithmes.

Un premier algorithme sera celui de planification de chemin sur de longues distances afin d'obtenir le chemin désiré libre d'obstacles, tout en tenant en compte, de la géométrie

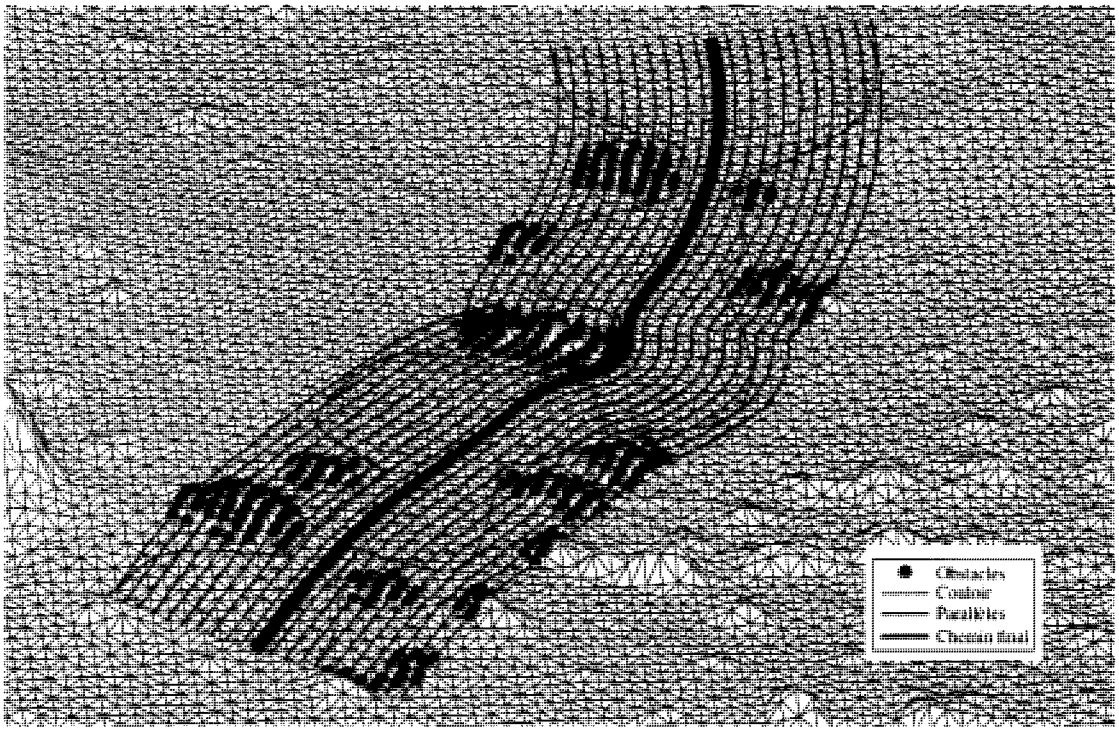


FIG. 3.23 Le Couloir

du robot. La partie la plus intéressante de ce problème sera la stratégie de négociation de l'algorithme pour diriger la recherche vers les zones où la probabilité de trouver le chemin désiré est plus grande, sans avoir besoin de faire une analyse globale du terrain.

Le deuxième algorithme sera l'implémentation d'un algorithme qui permet de définir un couloir autour du chemin final. La finalité est que n'importe quel chemin défini dans ce couloir, sera sécuritaire pour le robot. Aussi, le fait d'avoir connaissance de ce couloir, permettra de connaître la marge d'erreur maximale du robot tout au long du chemin. Cette même stratégie pourra être prise en considération lors de la génération du chemin. Dans l'ouvrage présenté en (Berglund et al., 2003) il est proposé une méthode pour définir le chemin avec courbure minimale dans une enveloppe ou un couloir.

La figure 3.23 illustre un premier essai afin de trouver ce couloir. Il s'agit simplement de chemins parallèles au chemin final libre d'obstacles, où les régions interdites ont été

identifiées.

CONCLUSION

Nous avons présenté trois approches différentes qui donnent une solution à notre problème de planification de chemin. Une fonction d'identification d'obstacles a été mise en place afin de permettre aux différents algorithmes de pouvoir les éviter. Cette fonction tient compte tant de la géométrie du robot comme de ses limitations physiques.

Malgré que nous ayons décidé de proposer les trois approches comme solution, nous pouvons conclure que la troisième approche est sélectionnée, selon les critères d'évaluation, comme étant la solution la plus viable.

Les deux premières solutions proposées sont limitées en matière de performance, elles présentent des problèmes dans la réussite, elles ne fonctionnent que pour des cas simples et sur de courtes distances. De plus, le temps requis pour le calcul d'une solution est inacceptable. En analysant les faiblesses de ces dernières, nous avons donc cherché un autre type de solution visant résoudre notre problème d'une façon plus efficace.

La troisième approche part de l'hypothèse qu'une bonne partie du chemin initial peut être traversé par le robot sans risque de collision. Pour ce qui est des parties qui présentent des problèmes pour le robot, une planification locale serait effectuée afin de rendre le chemin complètement faisable. Cette approche est capable de fournir le chemin désiré, si possible, dans un délai acceptable et avec une certitude supérieure. Le processus de validation permet d'assigner au robot une trajectoire partielle à parcourir, pendant que l'algorithme redéfinit les zones problématiques.

Lorsque nous partons d'un chemin tracé pour le planificateur principal du robot, il est possible de penser que les obstacles à éviter seront en général peu présents et qu'ils seront des obstacles d'une grandeur « moyenne » et qu'il ne s'agit pas de grosses formations rocheuses, montagnes, etc. Si c'est le cas, les obstacles seront évités efficacement,

mais sinon notre algorithme prendra beaucoup plus de temps et il serait souhaitable d'implanter des fonctions de décisions qui feront de celui-ci, l'algorithme le plus adéquat pour cette tâche.

Les différents algorithmes ont été testés sur une représentation mathématique du terrain ce qui nous a permis d'évaluer sa performance. Pour ce qui est du troisième algorithme, nous l'avons testé sur différents types de surfaces avec des résultats convaincants.

RÉFÉRENCES

Bakambu, J. N., Allard, P., and Dupuis, E. (2006). 3d reconstruction of environments for planetary rover autonomous navigation. *The 3rd Canadian Conference on Computer and Robot Vision (CRV'06)*, page 61.

Barber, C. B., Dobkin, D. P., and Huhdanpaa, H. (1996). The quickhull algorithm for convex hulls. *ACM Trans. Math. Softw.*, **22**(4), 469–483.

Berglund, T., Jonsson, H., and Soderkvist, I. (2003). An obstacle-avoiding minimum variation b-spline problem. *Proceedings of the 2003 International Conference on Geometric Modeling and Graphics*, pages 156–161.

De Santis, R. M. (2005). *Commande des systèmes robotiques : cours ELE6207.* Presses Internationales Polytechnique, Montreal,QC, 3 edition.

D'Errico, J. (2006). Fminsearchcon. Mathworks-fileexchange.

Ellis, S. P. (2002). Fitting a line to three or four points on a plane. *SIAM Rev.*, **44**(4), 616–628.

García, M. A. (1995). Massively parallel approximation of irregular triangular meshes with g1 parametric surfaces. *Parallel Algorithms for Irregularly Structured Problems, Lecture Notes in Computer Science*, **980**, 217–230.

Gemme, S., Bakambu, J. N., Allard, P., and Rekleitis, I. (2005). 3d reconstruction of environments for planetary rover autonomous navigation. *The Second Canadian Conference on Computer and Robot Vision (CRV'05)*, pages 594–601.

Green, A. R., Rye, D., and Durrant-Whyte, H. (2006). Planning for vehicles in unstructured environments. *ARC Centre of Excellence in Autonomous Systems.*

Kirk, D. E. (2004). *Optimal Control Theory An Introduction.* Dover Publications, Inc., Mineola, NY.

Lagarias, J., Reeds, J. A., Wright, M. H., and Wright, P. E. (1998). Convergence properties of the nelder-mead simplex method in low dimensions. *SIAM Journal of Optimization*, **9**(1), 112–147.

Latombe, J.-C. (1991). *Robot Motion Planning*. Kluwer Academic Publishers, Boston,MA.

LaValle, S. M. (2006). *Planning Algorithms*. Cambridge University Press, Cambridge, U.K. Available at http ://planning.cs.uiuc.edu/.

Ordónez, C. (2006). Design and implementation of a limit cycle negotiation strategy for robot navigation. Master's thesis, THE FLORIDA STATE UNIVERSITY.

Page, D., Koschan, A., and Abidi, M. (2006). Linking feature lines on 3d triangle meshes with artificial potential fields. *Proc. IEEE 3rd International Symposium on 3D Data Processing, Visualization and Transmission 3DPVT*.

Seraji, H. (1999). Traversability index : A new concept for planetary rovers. *Robotics and Automation, 1999. Proceedings. 1999 IEEE International*, **3**, 2006–2013.

Shiller, Z. (2000). Obstacle traversal for space exploration. *Proceedings 2000 IEEE International Conference on Robotics and Automation*.

Shiller, Z. and Chen, J. (1990). Optimal motion planning of autonomous vehicles in three dimensionalterrains. *Proceedings 1990 IEEE International Conference on Robotics and Automation*, **1**, 198–203.

The MathWorks, I. (2005). Fitting an orthogonal regression using principal components analysis. Published with MATLAB® 7.2.

Xu, W., Tso, S., and Lu, Z. (1998). A virtual target approach for resolving the limit cycle problem in navigation of a fuzzy behavior-based mobile robot. *Proceedings of the 1998 IEEE/RSJ Intl. Conference on Intelligent Robots and Systems*, pages 44–49.

YAMAMOTO, M., IWAMURA, M., and MOHRI, A. (1998). Time-optimal motion in the planning presence of skid-steer of obstacles mobile robots. *Proceedings of the 1998 IEEE/RSJ Intl. Conference on Intelligent Robots and Systems*.

Yokokohji, Y., Chaen, S., and Yoshikawa, T. (2007). Evaluation of traversability of wheeled mobile robots on uneven terrains by a fractal terrain model. *Advanced Robotics*, **21**, 121–142.